Visual Recognition Using Deep Neural Networks on Abstract and Decomposed Data

BERGISCHE
UNIVERSITÄT
WUPPERTAL

Dissertation

University of Wuppertal
School of Mathematics and Natural Science

submitted by **Annika Mütze, M. Sc.**
for the degree of Doctor of Natural Sciences (Dr. rer. nat.)

Supervisor	Prof. Dr. Hanno Gottschalk
Co-Supervisor	PD Dr. Matthias Rottmann

Wuppertal, 29.02.2024

Acknowledgments

First, I would like to express my deepest gratitude to my supervisors Hanno Gottschalk and Matthias Rottmann since I could not have undertaken this journey without them. They gave me the opportunity to work on the projects which form the basis of this thesis. Furthermore, they supported me constantly throughout my time as doctoral candidate and guided me in the broad research field. I benefitted greatly from their knowledge and experience and enjoyed our frequent discussions.

Furthermore, this thesis would not have been possible without financial support. This work was funded by the German Federal Ministry for Economic Affairs and Climate Action within the project "KI Delta Learning", grant no. 19A19013Q and within the project "KI Data Tooling", grant no. 19A20001E. I thank the consortium for the successful cooperation and interesting discussions which broaden my research horizon. Moreover, I gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS [123] at Jülich Supercomputing Centre (JSC). Furthermore, I thank Hannah L. and Hannah B. for assisting in setting up the classification experiments in Chapter 4 as well as Natalie for her efforts contributing to the results in Chapter 5.

I thank my colleagues of the Applied Computational Mathematics AI Lab and former BUW-KI team for the pleasant time we had together. Sharing knowledge, data, ideas, tool tips and code snippets as well as our discussions in our reading club and in the hallway led to an active and enjoyable research exchange. Especially, I would like to thank Antonia and Svenja for the great time we had together, not only by sharing an office. I am also grateful to all who improved this thesis by proofreading and giving constructive comments and support for the design.

Lastly, I would like to thank my family, my close friends and faithful companions. I would like to express my gratitude to everyone who supported me throughout this chapter of my life. Your unwavering belief in my abilities has been a great source of encouragement. I also would like to thank my faithful companion who I lost during that time. He gave me faith whenever I was struggling. In addition, my family enabled me to take the path to science and supported my decisions that led me to where I am today.

Foreword

The writing style chosen for this thesis is the first person plural. Thereby, we adopt the reader including writing style which is commonly used for writing in Mathematics and Computer Science research articles.

The work presented in Chapter 4 was widely taken word-by-word from the publications

- A. Mütze, M. Rottmann, and H. Gottschalk, *Semi-Supervised Domain Adaptation with CycleGAN Guided by Downstream Task Awareness*, in Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2023) - Volume 5: VISAPP, SciTePress, 2023, pp. 80–90
- ——, *Semi-supervised Task Aware Image-to-Image Translation*, in International Joint Conference on Computer Vision, Imaging and Computer Graphics, vol. 2103 of Communications in Computer and Information Science (CCIS), Springer Nature Switzerland, Cham, to appear

The latter is not published at the time of printing this thesis but will appear as a chapter in a book in the CCIS Series published by Springer. All contribtuions made in the listed publications were done by myself under supervision of Hanno Gottschalk and Matthias Rottmann. Both publications are merged and adapted to the notation of the thesis in Chapter 4.

Remarks. As we outline in the introduction, deep learning is a rapidly emerging research area and technology which surpass human performance in specific tasks. Despite its potential to improve and facilitate daily life, it is important not to overlook the potential risks associated with technology that is involved in far-reaching decisions such as job positions and crime accusation, as it can have a significant impact on our daily lives [54, 120]. In addition, the deep learning technology in general can be misused. There is a potential risk that authoritarian regimes or profit-driven companies make use of the technology to conduct surveillance and discriminate against individuals, whether intentionally or unintentionally. The necessity of an informed use of large deep learning models is recently formulated in an open letter signed by over 30,000 people: “Powerful

AI systems should be developed only once we are confident that their effects will be positive and their risks will be manageable." [3]. The solution to this is most probable multifaceted, but careful data management and sophisticated data collection strategies, rather than collecting everything, should most likely be part of it [23]. This includes awareness of potentially biases either harming or discriminating marginalized groups and minorities or more general biases based on naturally arising cues in images like shape or texture. Deep neural networks are statistical models which learn from data. That means, even with the best model architecture and training algorithms a model is only as good as the data it was trained on.

All developed methods in the context of this thesis aim to give more insight into the behavior of neural networks and to overcome limitations like annotation cost or offer opportunities for a potentially life-saving testing environment due to the use of simulations. Furthermore, this thesis contributes to a better understanding of the influence of image cues on the behavior of deep neural networks for visual recognition tasks. With our research we provide a bit more insight into neural networks and thereby provide methods to investigate semantic segmentation models on abstract and decomposed data.

Contents

Chapter 1

Introduction

Deep Learning is a branch of machine learning and a subfield of artificial intelligence (AI) that deals with specific parametric models which learn from data, so-called (artificial) neural networks, inspired by the functionality of the human brain. The first models mimicking the human brain's structure of interconnecting neurons were not yet considered deep. They date back to the mid of the last century when, among other works [170], Frank Rosenblatt introduced the concept of a perceptron [226]. Later, perceptrons with multiple layers [227], known as deep feedforward neural networks, and perceptrons trained by stochastic gradient descent to optimize the model's parameters were proposed [8]. They were able to classify non-linearly separable patterns. Nearly a decade later, the publication and application of backpropagation for training neural networks was a key driver for the development of neural networks [145, 160, 229]. The foundation of the success of deep learning models for image processing was laid in the late 1990s by introducing deep convolutional neural networks trained with backpropagation [146]. Nevertheless, deep learning was and is computationally expensive and depends on the amount and quality of available data which all was limited at that time. More recently, deep learning has been boosted by the accessibility of more powerful hardware. Training neural networks in parallel on graphics processing units (GPUs) allows for larger models and research on a larger scale. Coincidently, due to the digitalization of the daily life more and more data is collected and aggregated to large datasets.

Deep learning in general has a wide range of applications [236] including but not limited to autonomous driving [108], medicine [112, 176], natural language processing [2, 15], speech recognition [85], cybersecurity [188], agriculture [6], recommendation systems [19], process or quality monitoring [199] and surveillance [304]. If the task can be solved by learning (complex) patterns from data, neural networks are most likely a good model choice. Particularly, deep learning has shown remarkable results in various computer vision tasks [92, 134, 142, 161, 218] including perception and visual recognition

tasks. The aim of computer vision is to algorithmically extract (high level) information from images, videos (sequences of images) or other visual sensors like LIDAR point clouds to make decisions about real world objects or scenes [248]. Previously, classical approaches like the Roberts cross operator (1963) [221] or Hough transformations (1962) [104] for identifying edges or patterns in images were common practice. Research on computer vision inspiring deep learning dates back to at least 1983 when the international conference on computer vision and pattern recognition (CVPR) was held in Washington DC [1] for the first time and Ballard et al. claimed that vision task which can easily be solved by humans should be solvable by parallel algorithms inspired by the human visual cortex [17]. The actual breakthrough happened 2012 when Alex Krizhevsky et al. demonstrated the dominating performance of convolutional neural networks trained on multiple GPUs for image classification [142]. Since then, a large variety of model architectures were proposed. After the VGG-Net family [255] demonstrated that convolutional neural networks can be trained with a depth of 16 to 19 layers, different model architectures were proposed to further increase the depth while preserving learnability [92, 263]. Besides the trend of going deeper, lean models for limited hardware were also investigated [105, 234, 265]. Recently, it has been demonstrated that transformer models which have shown outstanding results in natural language processing tasks may be adapted for vision tasks as well [161]. Besides classification, common tasks in computer vision are object detection (object classification including the localization of the object enclosing bounding box) [326] and semantic, instance or panoptic segmentation [175] which requires classification on pixel-level.

A typical pipeline to solve recognition tasks with neural networks include the following five steps: data acquisition $\rightarrow$ labeling $\rightarrow$ data pre-processing $\rightarrow$ model training or inference $\rightarrow$ prediction. The first three aspects are data related, whereas the model training incorporates most of the code, the model (architecture) and algorithmic aspects of solving the problem. Thus, the success of deep neural networks bases on two pillars, the model aspects and the data (assuming sufficient computational power). In the last decades, AI evolution in computer vision was mainly driven by architecture design and improving on benchmarks rather than data quality [55]. This is often referred to model-centric AI in contrast to a data-centric view. In the study "Everyone wants to do the model work, not the data work" Sambasivan et al. demonstrated that "data is the most under-valued and de-glamorised aspect of AI" [233] even though negative downstream effects arise from poor, biased or insufficient data [192, 233].

Deep generative models are another branch of deep learning research that focuses more on data. These models learn a mapping from a stochastic input to a data point drawn from a certain distribution. Thus learning the joint distribution over a dataset. This allows to generate or sample new data from the learned distribution. Work on classical probabilistic models, the predecessors of deep generative models, has been studied since the mid 1980s [98]. However, the advent of efficiently learning joint distributions using deep neural networks dates back to 2014 when variational autoencoders showed success in generating images [133, 219]. Data generation and particularly image generation

gained more attention due to the pioneering work of Goodfellow et al. about generative adversarial networks (GANs) [79]. The evolution of deep generative models thereby benefited from the development of deep learning models for discriminative tasks and improvements in the training procedures [184]. Several other methods like flow-based, autoregressive, energy-based and diffusion models have been proposed in diverse variations [273]. Each method has its own application and limitations. As a consequence, the choice of model should depend on the task at hand. Particularly GANs have gained popularity due to their simplicity and the neat idea of implicitly learning the data distribution by a generator. This led to a wide range of publications about stabilizing the training process and improving the image quality from 32×32 pixel to high resolution images [11, 79, 127, 148, 177, 193, 209, 232, 238, 288].

Despite the success of deep neural networks which even surpass human performance in specific tasks, e.g., in fine-grained class discrimination [91], or in fooling humans with generated images [127, 190], deep learning models are still an active field of research and leave improvement potential for both laboratory conditions and real world applications [173, 254].

Label Shortage. The availability of high-quality, annotated data remains a challenge as well as the selection of a suitable dataset for a specific task or problem. When a sensor setup is given, recording data is often simple whereas the data selection as well as the labeling of the data is difficult [121], time-consuming [47], costly [217] and error-prone [192, 203, 228]. In general, for recognition tasks labeling or annotating is required to define the ground truth (GT) of the task. Particularly, there is a need of a large, representative and correctly labeled dataset when applying deep learning to safety-critical applications like autonomous driving, diagnostics or medical imaging. Both domains come with their own difficulties ending in a shortage of labeled data. Annotation in a medical context requires expert knowledge which is extremely expensive. Clinical datasets are often small. As a result, the neural network does not generalize well to unseen data [154]. In autonomous driving, a detailed and encompassing understanding of the environment is necessary to use machine perception to maneuver a car safely. To this end, semantic segmentation is often used to classify each pixel in the image leading to an annotation time of about 90 min per image for an experienced annotator [47]. However, to make deep learning models accessible to a broader user group and more applications, new methods need to be developed to enable training models with limited amount of or no labeled data. To reduce the cost or amount of labeling, respectively, weakly and semi-supervised learning approaches are proposed. Weak labels serve as a coarse approximation of fine-grained annotations. For example, object enclosing bounding boxes can serve as 'noisy' labels for semantic segmentation [53, 128]. Bounding box annotations are assumed to be faster than segment annotations on pixel-level. Semi-supervised learning deals with datasets which contain a large fraction of unlabeled data and only a small fraction of annotated data [280]. Nowadays, typical semi-supervised learning strategies

are variants of self-training methods with pseudo labels. These pseudo labels are generated by the neural network and iteratively included into the training process, oftentimes weighted by the uncertainty of their prediction. Thereby, the model performance and also the pseudo labels are improved step-by-step [118, 150, 306]. Another learning technique encompasses a two stage training process. An unsupervised pre-training is used to initialize suitable network parameters. The parameters are then fine-tuned on a small labeled dataset [280]. However, these methods come with some limitations. The success of semi-supervised learning presumes a certain smoothness of the data distribution which is not necessarily given in real world settings [41]. Even though weak labels seem to reduce annotation cost, they can still be intractable [277].

In parallel to research on learning strategies that address the label shortage, computer simulations have made progress towards creating photorealistic scenes, especially in the automotive context [59, 220, 224, 296]. An advantage of computer simulations is that semantic labels can often be recorded simultaneously for free. Furthermore, simulations enable the construction of various scenarios[1] which are rare or life-threatening in the real world. In addition, theoretically, it provides the opportunity to reproduce scenarios under different conditions such as different weather and lighting conditions, allowing for a diverse and curated train and test setup. Particularly, testing and retraining on safety-critical corner cases can help to improve the robustness of the neural network [138]. For that reason, the availability of simulations reduces the severity of label or data shortage. As a consequence, well-performing segmentation models can be trained which will likely generalize well to unseen scenarios within the same domain.

Besides, generative models can serve as data generators. This is particularly useful when only limited data is available. By learning the data generating distribution by a GAN for example, the generator can be used to sample new data points within the distribution. However, this data comes without labels. Conditional GANs [177] offer the opportunity to generate class-specific data but also need labels during training. In summary, the source of data is not limited to only real-world sensor data since computer simulations or generative models can serve as data sources as well. However, deep neural networks suffer under domain shifts, i.e., the performance of the neural network drops when evaluated on a different domain than it was trained on.

Domain Shift. Neural networks for recognition tasks learn the conditional class probability based on a labeled dataset where each sample pair is assumed to be drawn independently from the joint data generating distribution. When this distribution changes at test time the performance of the neural network drops [185] due to a domain shift. Domain adaptation methods aim to mitigate this performance gap [48]. The adaptation can be applied on different (non-exclusive) levels: input [100], feature [277] or

[1] Nevertheless, creating complex scenarios or rare assets requires some expertise in computer graphics and might need a non-negligible amount of time.

output [317, 325] level. To shift the visual appearance of a scene, image-to-image translation methods can be used. These methods learn a pixel-to-pixel mapping between two image domains which transfers one scene representation into another, e.g., from summer to winter [114]. Recently, GAN-based methods have shown promising translation results [114, 320]. Changing the appearance of the scene, while preserving the overall content makes image-to-image translation a strong candidate for domain adaptation approaches at the input level. However, these models lack task awareness for the downstream task under domain shift.

Benefitting From Abstract Data. In this thesis, we address the challenge of limited availability of labeled data by taking advantage of a synthetic domain in which it is easier to obtain labels, thereby improving label efficiency [185]. Transferring the problem to a more abstract domain, such as a simulation or sketches instead of real-world images, simplifies the image representation and makes it easier to collect labels. As a consequence, this approach allows for the development of high-performing neural networks with potentially leaner architecture [192]. Furthermore, in the case of simulations our approach can benefit from the earlier noted advantages, such as training and testing on life-threatening (corner) cases. While drawing advantages of labeled data in an abstract domain, we put up with a domain shift. We propose a new semi-supervised task aware image-to-image translation method for semantic segmentation tasks which aims to mitigate the gap. Domain adaptation in general is an active research area and has received increasing attention as evidenced by the increasing number of publications in the last five years [242]. Most research is done for unsupervised domain adaptation when no labels are available in the target domain, i.e., the domain at test time. If a few labels from the target domain are available, one speaks of semi-supervised domain adaptation. Apart from our method, there has been little research in this area [242]. Most state-of-the-art methods in unsupervised domain adaptation focus on hybrid methods adapting the downstream task network on multiple levels and combining several learning objectives. This, however, makes it more difficult to understand the influences of different features. In contrast, our method is a modular approach which not only aims to mitigate the domain shift but also allows for tuning the individual components of the method. Moreover, the developed method in Chapter 4 addresses the domain shift challenge from a more restricted data-centric perspective, asking how the training data can be improved under a fixed model. Additionally, this method offers the possibility for a visual inspection of which image features are relevant to the semantic segmentation model on abstract representations for accurate predictions. To balance the lack of information due to missing labels and the cost of labeling, an extended version of our method for classification tasks was developed in the context of this thesis [186]. The basic method has access to a random subset of images which are provided with labels to come up with task awareness. In contrast, we show that our method benefits when informative data points are actively queried in the real domain which are then labeled and help to attain task awareness.

Model Insight by Image Decomposition. Using deep learning models in safety-critical applications requires an understanding of any shortcomings in a dataset and of the trained model. As a consequence, an additional challenge of neural networks is that their decision-making process needs to be transparent for humans in uncertain situations. In general, deep neural networks are considered as 'black boxes'. Both users and developers lack insight into the model's behavior and decision process for certain (potentially devastating) predictions. Different research fields have focused on improving the traceability of decisions. For example, explainable AI aims to obtain transparency of parts of the neural network, to learn which meaning a certain network part has in the decision-making process and to generate human-readable and understandable explanations for a prediction [303]. On the other hand, the credibility of the neural network's prediction is examined by uncertainty quantification measures [107]. By incorporating components for estimating prediction uncertainty, an (un)certainty measure can be applied to the model's output. This allows for the identification of potential false predictions and the implementation of corrective actions. Among other applications, uncertainty-based methods are often used for active sample selection. Therefore, we took advantage of these approaches in [186].

The dataset has a non-negligible impact on the performance of the trained neural network. A major concern are biases in datasets since they lead to biases in the decision-making process. The term *bias* has different meanings in the machine learning literature [64, 178]. In the following, we refer to a bias in a dataset when it causes the model to prefer one aspect over another. For example, we encounter a biased dataset and model when the pedestrian prediction accuracy differs based on the skin color of the people [295]. Equally, we encounter a bias if certain image features like a watermark, the background or texture is more relevant to the prediction than the actual object or scene [20, 72, 315]. Biases in dataset are often not obvious but deep neural networks learn to exploit unwanted features leading to shortcut learnings which hamper generalization [60, 70]. Among others, understanding the influence of cues, image features, which naturally arise in images like shape, texture and color are crucial for an informed use of neural networks. A large debate started since the pioneering work of Geirhos et al. stating that ImageNet pre-trained models are texture biased as their prediction relies more on texture than on shape [44, 52, 72, 111]. Furthermore, it was found, that the source of bias is diverse [95] but the dataset including its label strategy has significant influence on the neural network decision process [156]. Most of the research done so far is concerned with neural networks trained on classification tasks and often refer to ImageNet as pre-training dataset. Thereby, the approaches often lack the transferability to more complex tasks like semantic segmentation. Furthermore, the analysis bases on conflicting cues which disregards a study on what can be learned from a standalone cue. To this end, we introduce a new method in Chapter 5 which enables the analysis of the decomposed cue influence in a semantic segmentation network. The work strives to achieve an in-depth analysis of what neural networks can learn from certain cues naturally arising in images. Therefore, we propose a method to decompose the image

into its main cues: color, texture and shape. While we could take advantage of existing methods for the shape extraction, we developed a new method which allows to extract the texture from a semantic segmentation dataset and to create a new segmentation task for training only on this particular cue. Based on this, an in-depth analysis on the influence of different cues was conducted, revealing insight into the behavior of semantic segmentation networks.

In summary, this thesis addresses two major questions:

How can a deep neural network for visual recognition tasks benefit from data in an abstract domain when facing label shortage in a complex domain?

and

Which cues in an image encode the most accessible information for a neural network to solve a semantic segmentation task?

To this end, we developed a semi-supervised task aware image-to-image (active) domain adaptation method as well as a strategy to decompose images into their basic cues followed by an in-depth study of the cue influences.

Structure. In the following two chapters (Chapters 2 and 3), we give an overview on deep learning and deep generative learning to provide the theoretical basis for the Chapters 4 and 5. Furthermore, Chapters 2 and 3 define the notation which is used throughout this thesis. We start in Section 2.1 by introducing the concept of deep (convolutional) neural networks and define mathematically the different components of neural networks which are used in later chapters. We introduce the general learning process and give a short overview on statistical learning theory which provides the theoretical foundations. In Section 2.3, we describe two applications of deep learning in the context of visual perception: classification and semantic segmentation. For both, we provide an overview over relevant neural network architectures used in this thesis and define commonly used evaluation methods. As deep learning models are statistical models, they need sufficient data to generalize well. Methods which address the challenge when only a limited amount of data is available are presented in Section 2.4. While Chapter 2 focuses on discriminative models applied to computer vision tasks, Chapter 3 deals with deep generative models. After giving a general introduction into the topic, we focus on generative adversarial networks in Section 3.1. These networks are a special form of deep generative models. We conclude the chapter with an application of generative models, the so-called image-to-image translation, which plays a central role in the context of this thesis. In Chapter 4 our semi-supervised task aware image-to-image translation approach is presented. We start by giving a short introduction before reviewing related works. A detailed description of the developed method is then given in Section 4.3.

This includes the description of our extension to an active domain adaptation method in Section 4.3.5. Detailed experiments on different datasets are given in Section 4.4 for the basic method and in Section 4.4.2 for the extended version. We conclude the chapter by summarizing our contributions and ideas for further research directions. One of these directions is addressed by our work described in Chapter 5 in which we study the influence of different image cues in depth. This chapter is structured in the same way. After an introduction to the latest insights into biases in deep convolutional neural networks and the actual state of the art, we describe our method to decompose an image into its basic cues: color, texture and shape in Section 5.3. In Section 5.4 we propose a late fusion approach which enables us to analyze the cue influence on pixel-level for complementary cues. Section 5.5 discusses our study on the influence of different cues. Finally, we summarize our findings in Section 5.6. We conclude this thesis in Chapter 6 by summarizing our main contributions and insights as well as discussing the topic in a broader context along with future research directions.

Chapter 2

Fundamentals

This chapter introduces the key terms, methods, and concepts in deep learning that serve as the foundation for the work in subsequent chapters. We start with describing neural networks and how to train them - the core part of deep learning. Afterwards, we focus on the application of neural networks to computer vision tasks and give an overview of methods which address the challenge of how to adequately train neural networks with limited amount of data. Besides laying the theoretical foundations, we define the notation used throughout this thesis. To easily follow the notation, we additionally provide a list of notations at the end of this thesis (page 166).

2.1 Deep Neural Networks

Neural networks are parametric models which can be represented as a graph. This structure consists of (a variety of) compute nodes which are connected via weighted edges. A neural network architecture is referred to a graph realization with a fixed number of nodes per dedicated node type and their interactions by edges. Most of the edges have adjustable weights which are learned during a data-based training procedure. Processing data that the model has not necessarily seen before after training completion is referred to as model inference. In this thesis, we focus on the application of neural networks to computer vision and image synthesis tasks. In the following, we introduce the most common neural network types and components of feedforward neural networks used in computer vision and image generation. Particularly, we focus on fully connected layers in Section 2.1.1 and convolutional layers in Section 2.1.2. In Section 2.2 we explain how neural networks can adapt to a non-explicit data generating distribution based on training data drawn from that distribution. This includes an overview on theoretical foundations of statistical learning, a decomposition of the errors made by a statistical model (Section 2.2.2) and an outline about the universal approximation property of deep

neural networks in Section 2.2.3. Finally, we give a detailed description of the iterative training process in Section 2.2.4 and introduce some methods to stabilize training in Section 2.2.5.

2.1.1 Feedforward Networks

(Artificial) neural networks (NNs) are loosely motivated by the neuron structure of the human brain where plenty of neurons are connected in a complex network. These neurons are activated if they get an electrical signal above their activation threshold. Formerly inspired by neuroscience, an artificial neuron with $m \in \mathbb{N}$ inputs given by the vector $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathcal{X} = \mathbb{R}^m$ and one output $y \in \{0, 1\}$ can be modelled by a **perceptron** $f_{\boldsymbol{\theta}}$ which maps the input to the output by an affine transformation (pre-activation) followed by an activation function [113]

$$
\begin{aligned}
z = f_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \sum_{i=1}^{m} w_i x_i - b = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - b \quad \triangleleft \text{ affine transformation (pre-activation)} \\
y = a_{\texttt{step}}(z) &= a_{\texttt{step}}(f_{\boldsymbol{\theta}}(\boldsymbol{x})), \qquad \triangleleft \text{ perceptron} \quad (2.1) \\
\text{with } a_{\texttt{step}}(z) &= \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}. \qquad \triangleleft \text{ activation function}
\end{aligned}
$$

Thereby, the perceptron is understood as a parametric statistical model $f_{\boldsymbol{\theta}}$ with an adaptable weight vector $\boldsymbol{w} \in \mathbb{R}^m$ and bias parameter $b \in \mathbb{R}$ grouped to $\boldsymbol{\theta} = (\boldsymbol{w}, b)$. The activation function $a_{\texttt{step}} : \mathbb{R} \to \mathbb{R}$ is used to threshold the pre-activation. The perceptron gives only the positive feedback 1 if $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$ is larger than the bias term b. A schematic representation of a perceptron with m inputs is shown in Figure 2.1. Rosenblatt showed that this simple model can be used to solve binary classification of linearly separable problems [226]. The aim of the perceptron thereby is to learn an approximation of a function $f^* : \mathbb{R}^m \to \{0, 1\}$ based on a set of $N \in \mathbb{N}$ input-output pairs $\mathcal{D} = \{\boldsymbol{x_n}, f^*(\boldsymbol{x_n}) = y_n\}_{n=1}^N$, the so-called training dataset. To this end, the learnable parameters $\boldsymbol{\theta}$ are iteratively adapted in a training process to approximate the function f^*. We provide a description of the training concept as well as the corresponding learning theory in Section 2.2.

In order to solve more complex tasks, multiple perceptrons can be combined in a graph structure. Combining multiple perceptrons lead to a **fully connected** (FC) layer, where each input neuron[2] has a weighted connection to all output neurons. Let $\boldsymbol{w_i}$ be the weight vector of the i-th neuron, with $i \in \{1, \ldots, m_l\}$, $m_l \in \mathbb{N}$, then we define a weight matrix $W_l = (\boldsymbol{w_1}^{\mathrm{T}}, \ldots, \boldsymbol{w_{m_l}}^{\mathrm{T}})$ where the rows of the matrix are given by the weighted connections of the m_l neurons. Given an input vector $\boldsymbol{x} = (x_1, \ldots, x_{m_{l-1}}) \in \mathbb{R}^{m_{l-1}}$,

[2] For simplicity, we adapt the wording 'neuron' from the literature when not referring to a stand-alone perceptron.

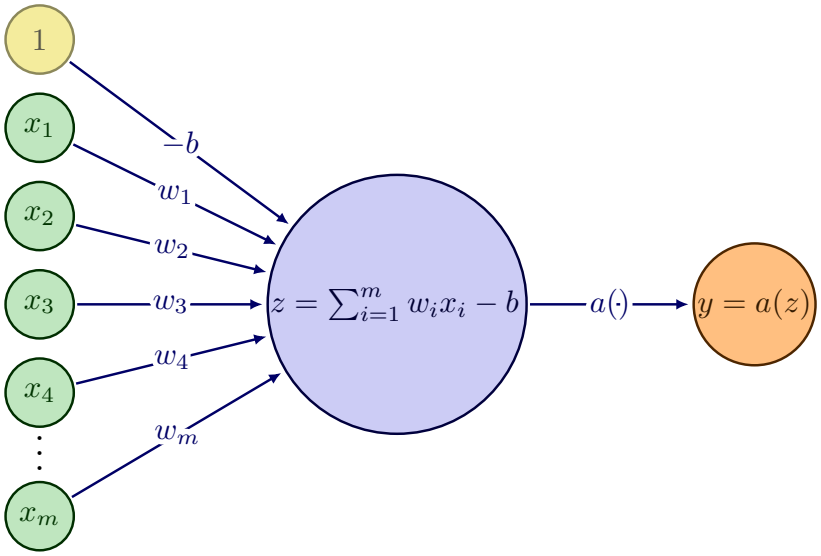


Figure 2.1: Schematic visualization of a perceptron.

$m_{l-1} \in \mathbb{N}$, a weight matrix $W_l \in \mathbb{R}^{m_l \times m_{l-1}}$ and a bias vector $\boldsymbol{b} \in \mathbb{R}^{m_l}$, a fully connected layer maps $\boldsymbol{x}$ to the pre-activation vector

$$\boldsymbol{z} = W_l \boldsymbol{x} + \boldsymbol{b} = \sum_{i=1}^{m_l} \sum_{j=1}^{m_{l-1}} w_{i,j}^l x_j + b_i \in \mathbb{R}^{m_l} \tag{2.2}$$

where $w_{i,j}^l$ defines the weight between x_j and z_i. The number of output neurons m_l is denoted as the **width** of a fully connected layer.

The **multilayer perceptron** (MLP) is a composition of $L \in \mathbb{N}$ fully connected layers with possibly different widths $m_l, l = 1, \ldots, L$, forming a weighted directed acyclic graph. This structure implies that data flows from the input to the output without loops or backward connections. For this reason, MLPs are a special case of so-called **feedforward neural network**. Connecting perceptrons in this way leads to a compositional statistical model $f_{\boldsymbol{\theta}}$ where all learnable parameters $W_l, \boldsymbol{b_l}$ for $l = 1, \ldots, L$ are summarized in the parameters $\boldsymbol{\theta}$. Since the composition of affine linear maps is still an affine linear map, the expressiveness of such a model is limited. To this end, an activation function $a_l : \mathbb{R}^{m_l} \to \mathbb{R}^{m_l}$, $l = 1, \ldots, L$ is applied to the pre-activation vector $\boldsymbol{z_l}$. In addition to the step function $a_{\texttt{step}}$ in Equation (2.1), other activation functions can be utilized to obtain the activated layer output based on the pre-activation vector. We introduce common activation functions, including non-linear ones, in Section 2.1.1.1. To also solve non-linear separable problems the non-linearity of the activation functions applied to the pre-activation vector is essential as we see in Section 2.2.3. With this, we can mathematically define an MLP as the following mapping from an input vector

$\boldsymbol{x} \in \mathbb{R}^{m_0}$ to an activated output.

$$f_{\boldsymbol{\theta}} : \mathbb{R}^{m_0} \to \mathbb{R}^{m_L}, \boldsymbol{x} \mapsto f_{\boldsymbol{\theta}}(\boldsymbol{x}) = f^L_{\boldsymbol{\theta_L}} \circ \ldots \circ f^2_{\boldsymbol{\theta_2}} \circ f^1_{\boldsymbol{\theta_1}}(\boldsymbol{x}) \tag{2.3}$$

where each $f^l_{\boldsymbol{\theta}_l}$, $l \in \{1, \ldots, L\}$ is defined by

$$\mathbb{R}^{m_{l-1}} \ni \boldsymbol{x} \mapsto f^l_{\boldsymbol{\theta_l}}(\boldsymbol{x}) = a_l(W_l \boldsymbol{x} + \boldsymbol{b_l}) \overset{(2.2)}{=} a_l(\boldsymbol{z_l}) =: \boldsymbol{h_l} \in \mathbb{R}^{m_l} \tag{2.4}$$

Assuming that the activation functions a_l are measurable for $l = 1, \ldots, L-1$, a feedforward neural network $f_{\boldsymbol{\theta}}$ can be understood as a composition of measurable functions and an output activation function a_L in the last layer. The output activation can be seen as a task defining head. It can for example rescale the values of the last measurable function in the composition to fit within the range of $(0, 1)$ and sum up to 1 such that they represent probabilities. A probabilistic view on task defining data is common in deep learning as data is often noisy and therefore understood as realizations of random variables. The layers can be structured into three types. Thereby, $f^1_{\boldsymbol{\theta_1}}$ is called **input layer**, $f^L_{\boldsymbol{\theta_L}}$ is the **output layer** and all layers in between are **hidden layers** with hidden states $\boldsymbol{h_l}$. The output layer needs to solve a dedicated task based on the data, e.g., estimating the (conditional) probability distribution of the data. In contrast, the hidden layers have no prespecified input-output relation. Their task is to transform the input $\boldsymbol{x}$ to facilitate the output generation of the last layer, e.g., by finding patterns or discriminative features in the data. For that reason, the hidden state vectors are also called feature vectors. We define the **depth** of a feedforward neural network by the number of layers with hidden states[3]. An MLP with depth 2 and arbitrary width m_l per layer l is depicted in Figure 2.2.

2.1.1.1 Activation Functions

To enlarge the expressiveness of MLPs, an activation function $a : \mathbb{R}^m \to \mathbb{R}^m$ is applied to the pre-activation $\boldsymbol{z_l} = W_l \boldsymbol{x} + \boldsymbol{b_l}$ for $l = 1, \ldots, L$. Most commonly, a is associated with a scalar-function $a : \mathbb{R} \to \mathbb{R}$ which is applied componentwise. For serving as suitable activation function, a should be easy to compute, differentiable and non-linear. The differentiability is needed to use gradient-based optimization algorithms in the training process to learn the parameters of the neural network. A detailed description of this process is presented in Section 2.2.4. Representative examples for activation functions are

- *Heaviside step function*: The Heaviside step function has already been introduced in Equation (2.1) and is a quite old and simple activation function, which is inspired by neurons in the human brain. It only fires when a certain threshold is

[3]The number of layers is not consistently defined in the literature. In this thesis, we count each function $f^l_{\boldsymbol{\theta}}$ as one layer. Thus, a feedforward neural network with L layers has one input state (input vector), $L-1$ layers with hidden states and 1 output state and therefore a depth of $L-1$.

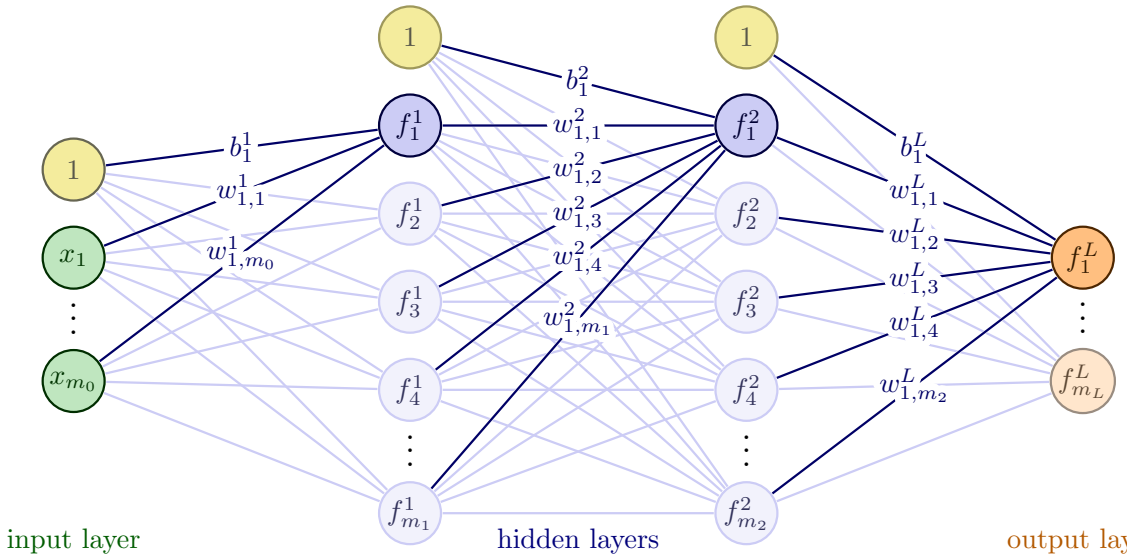


Figure 2.2: Schematic visualization of an MLP with depth of 2 and varying width m_l per layer. The computation of the first node in each layer is highlighted.

surpassed. This function is 0 for negative inputs and fires, i.e., is equal to 1, for values greater or equal to zero

$$a_{\texttt{step}} : \mathbb{R} \to \mathbb{R},\ z \mapsto \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}. \tag{2.5}$$

MLPs with this activation function could firstly solve the so-called XOR problem, i.e., learning the exclusive OR of two binary inputs, which was not possible with perceptrons as the data of this problem is not linearly separable [183]. However, due to the zero gradients and the jump discontinuity this activation function is not well suited for gradient-based optimization.

- *Sigmoid-like function*: A sigmoid function can be seen as smooth approximation of the Heaviside step function. Its property of being differentiable everywhere was the reason for its wide use as non-linear activation function for neural networks. The most prevalent functions are the logistic function

$$a_{\texttt{logistic}} : \mathbb{R} \to \mathbb{R},\ z \mapsto \frac{1}{1 + e^{-z}} \tag{2.6}$$

and its rescaled and shifted equivalent, the hyperbolic tangent function

$$a_{\texttt{tanh}} : \mathbb{R} \to \mathbb{R},\ z \mapsto 2 \cdot a_{\texttt{logistic}}(2z) - 1 = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{2.7}$$

Despite their differentiability, which makes them useful for gradient-based opti-

mization, the derivatives of sigmoid-like functions saturate and gradients start to diminish when moving away from $z = 0$. Small gradients can lead to difficulties in the training process such as slowing down the learning progress. Activation functions with gradients in $(0, 1)$ suffer from ever smaller gradients due to the compositional structure of the layers. We refer to Section 2.2.4 for the details on calculating the gradients. As a consequence, the feedback how to update the parameters get lost through the layers when these activations are used in the hidden layers of deeper neural networks. Especially the logistic function yields small gradients as its derivative is bounded by $\frac{1}{4}$. Furthermore, due to the need to evaluate exponential functions, the computational cost of this activation function is higher than that of e.g., the ReLU function which we introduce next. However, the logistic function is a suitable activation for the output layer when predicting a probability over a binary output since it maps to $(0, 1)$.

- *Rectified Linear Unit* (ReLU) [75]: Due to the mentioned training instabilities the sigmoid functions were widely replaced by the ReLU activation function which is defined as the piecewise linear function

$$a_{\texttt{ReLU}} : \mathbb{R} \to \mathbb{R},\ z \mapsto \max\{z, 0\}. \tag{2.8}$$

  This activation function is non-linear but preserves most of the positive properties of a linear function, e.g., non-saturating and consistent gradients with a value of 1 for all positive input values. In addition, the function value as well as the derivative can be computed at low cost compared to sigmoid functions. Therefore, it is nowadays the commonly used activation function in MLPs and convolutional neural networks which we introduce in Section 2.1.2. Nevertheless, due to the piecewise definition, it is not differentiable in zero and has a gradient of 0 for negative input values. This may appear questionable in the context of gradient-based optimization at first glance. However, evaluating the gradient at zero is extremely unlikely, and thus, this case can be neglected. In practice, due to computer precision, evaluation of exactly zero is handled by the one-sided derivative.

- *Leaky ReLU* (LReLU): A leaky version of ReLU was introduced by Maas et al. [167] as an easily computable function that, unlike the ReLU function, has no zero gradients.

$$a_{\texttt{LReLU}_\alpha} : \mathbb{R} \to \mathbb{R},\ (z) \mapsto \begin{cases} z & \text{if } z \geq 0 \\ \alpha \cdot z & \text{if } z < 0 \end{cases} \tag{2.9}$$

  Later on it was proposed to not fix but parametrize and learn the slope for the negative piece. This activation function is called parametric ReLU (PReLU) [91].

- *Rectified Power Unit* (RePU): As both Leaky ReLU's and ReLU's second derivatives are zero, RePU activation functions with positive integer $\beta \geq 2$ are used

when information about the curvature is needed [249].

$$a_{\texttt{RePU}} : \mathbb{R} \to \mathbb{R},\ z \mapsto \max\{z, 0\}^{\beta} = \begin{cases} z^{\beta} & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}. \tag{2.10}$$

For the special case when $\beta = 2$ the activation function is called ReQU and is the function with the smallest integer β with a continuous second derivative. For smaller β already, introduced functions can be recovered. For $\beta = 1$, this corresponds to ReLU and for $\beta = 0$ the step function is restored.

- *Gaussian Error Linear Unit* (GELU): The GELU activation functions introduced by Hendrycks and Gimpel in 2016 [94] became established later but is nowadays the preferred choice in transformer architectures (see Section 2.3.1.2). GELU is defined as follows:

$$a_{\texttt{GELU}} : \mathbb{R} \to \mathbb{R},\ z \mapsto z \cdot \frac{1}{2}\left[1 + \texttt{erf}\left(\frac{z}{\sqrt{2}}\right)\right], \tag{2.11}$$

where $\texttt{erf}$ denotes the (Gauss) error function defined by $\texttt{erf}(z) = {}^{2}/_{\sqrt{\pi}} \int_0^z e^{-t^2} dt$. In practice the error function is approximated and the approximated GELU is given by

$$a_{\texttt{GELU}}(z) \approx z \cdot \frac{1}{2}\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}\left(z + 0.044715 z^3\right)\right]\right) \tag{2.12}$$

In contrast to activations in hidden units there are often output specific activations used in the output layer. As introduced before, the logistic function can be used to achieve a binary probability distribution over two output neurons. The softmax activation function can be used to model the distribution of a discrete random variable with C possible realizations. Given unnormalized log probabilities $\boldsymbol{z} = (z_1, \ldots, z_C)$ from the last layer, a proper (categorical) probability distribution where all elements sum up to 1 and each element is greater or equal to zero can be computed by the function

$$a_{\texttt{softmax}} : \mathbb{R}^C \to \mathbb{R}^C,\ a_{\texttt{softmax}}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{c=1}^{C} e^{z_c}} \quad \text{for } i = 1, \ldots, C. \tag{2.13}$$

MLPs can theoretically approximate a wide range of function classes as we see in Section 2.2.3. However, they suffer from a large number of trainable parameters when e.g., high resolution images are used as inputs, as each input pixel has a weighted connection to the neurons in the first hidden layer [146]. Furthermore, the local correlations between pixels, inherent to an image topology, are discarded by MLPs and need to be learned [146].

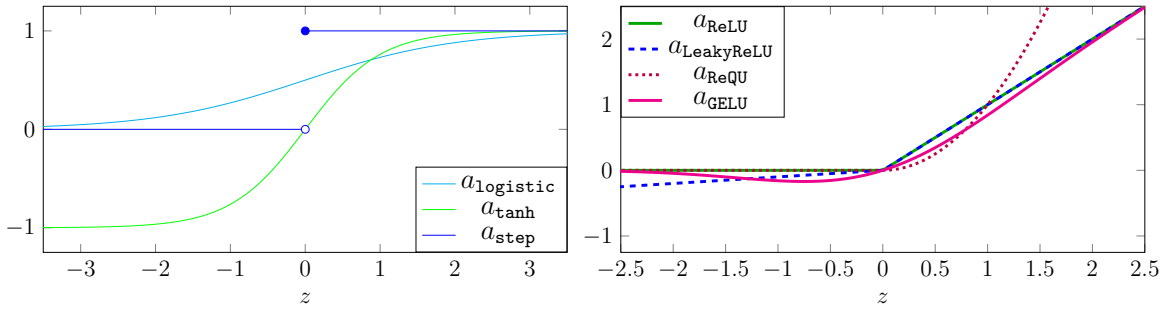


Figure 2.3: Overview of different activation functions.

2.1.2 Convolutional Neural Networks

In order to exploit the topology of grid structures, e.g., 2D-pixel arrangement for images or 1D arrays for time series data, a special kind of feedforward neural network, the **convolutional neural network** (CNN), was proposed [146]. By restricting the receptive field of a neural network to local features and by sharing weights, the information of the image topology is preserved while using distinctly fewer parameters [146]. Furthermore, a convolutional neural network makes use of the fact that the exact pixel positions might be irrelevant for a certain feature since the features like contours or color of an object do not change regardless of the object's position in an image. A CNN is a composition of different layer types. Each CNN consists of at least one convolutional layer (Section 2.1.2.1) and usually one pooling layer (Section 2.1.2.2). The convolutional layer is motivated by a discrete convolution operation, whereas the second condenses information by downsampling. To allow non-linear transformations of the features, an activation (cf. Section 2.1.1.1) follows each convolutional layer. In general, blocks with convolutional layers with activation are followed by pooling layers at certain depths. Depending on the task of the network a fully connected layer can be used at the end to map the spatially arranged features to a prediction of C classes. The following sections are based on [78, 88, 113] and [153].

2.1.2.1 Convolutional Layer

Convolutional layers are motivated by the mathematical convolution operation and aim for sparse connectivity and parameter sharing. Furthermore, they lead to equivariance with respect to translations of the input. Both, the fully connected layer and the convolutional layer have learnable weights and biases and calculate a dot product between the weights and the input. In contrast to the fully connected layer where each neuron is connected to all input neurons, in a convolutional layer a kernel with a support covering only a small region of the input reduces the number of learnable elements (weights). Furthermore, due to convolving a kernel with small support the parameter amount does not depend on the input size [122]. Kernels have been widely used in computer vision

before deep learning has shown superior results on vision tasks. In the past, kernels were handcrafted and convolved with an image to extract particular features, for example edges. See Figure 2.4 for an example of a kernel detecting vertical edges. The idea of using learnable convolutions was firstly presented by LeCun [145]. Visualizing the learned kernels in the first layer of a CNN shows that some of those handcrafted filters seem to be useful for feature extraction although learned from data by the neural network [142, Figure 3].

In the context of neural networks, we focus on the discrete convolution. For signal processing, functions are given by a finite amount of scalar data points and can be represented as m-dimensional vectors $\boldsymbol{\psi} = (\psi_1, \dots, \psi_m) \in \mathbb{R}^m$ and $\boldsymbol{\kappa} = (\kappa_1, \dots, \kappa_{k_x}) \in \mathbb{R}^{k_x}$, $m, k_x \in \mathbb{N}$. In theory, the discrete convolution is calculated over sequences in $\mathbb{R}^{\mathbb{Z}}$ and $\boldsymbol{\psi}$ and $\boldsymbol{\kappa}$ can be represented by a function with finite support in $\mathbb{Z}$. Let $\boldsymbol{\kappa}$ have finite support on the index set $\{1, \dots, k_x\} \subsetneq \mathbb{Z}$ and $\boldsymbol{\psi}$ on $\{1, \dots, m\} \subsetneq \mathbb{Z}$ then the following finite sum is calculated in practice leading to non-zero vector elements for the 2-nd to $(m + k_x)$-th entries:

$$(\boldsymbol{\kappa} * \boldsymbol{\psi})_t = \sum_{i=1}^{k_x} \kappa_i \cdot \psi_{t-i}. \tag{2.14}$$

In machine learning libraries, the mathematical convolution is rarely used, and the more efficiently implementable ([122]) **cross-correlation**

$$(\boldsymbol{\kappa} * \boldsymbol{\psi})_t = \sum_{i=1}^{k_x} \kappa_i \cdot \psi_{i+t}, \qquad \text{for } t \in \{0, \dots, m - k_x\} \tag{2.15}$$

is more common. The cross-correlation differs in such a way that the kernel is not flipped with respect to the input function. Therefore, the operation is not commutative anymore in contrast to mathematical convolutions. As the kernel is learned, a flipped cross-correlation, i.e., a convolution, could be learned instead. Furthermore, the combination of convolutional layers with other operations does not necessarily commute in general. As a consequence, omitting this property does not interfere with the learning problem. Therefore, 'convolution' and 'cross-correlation' are used interchangeably in the machine learning context[4].

The main results in this thesis are concerned with applications to vision problems and therefore the focus in this section as of now is on images. We represent an image by a 3D-tensor $\Phi \in \mathbb{R}^{m \times h \times w}$ where w and h define the spatial width and height of the image and m is the number of channels, e.g., 3 for a color image (red, green and blue channel) and 1 for grayscale images. For the sake of clarity, we confine ourselves to $m = 1$ and consider $\Phi \in \mathbb{R}^{h \times w}$ for now. Referring to the integral kernel, a discrete **kernel** tensor $K \in \mathbb{R}^{k_y \times k_x}, k_x, k_y \in \mathbb{N}$ is convolved with the image Φ. The 2D-convolution of a kernel

[4]We follow this convention and call both operations 'convolution' if the kernel flip is not relevant.

K and an image Φ is defined by

$$(K * \Phi)_{y,x} = \sum_{i \in \mathbb{Z}} \sum_{j \in \mathbb{Z}} K_{y-i,x-j} \cdot \Phi_{i,j} \tag{2.16}$$

and its corresponding cross-correlation is given by

$$(K * \Phi)_{y,x} = \sum_{i \in \mathbb{Z}} \sum_{j \in \mathbb{Z}} K_{i,j} \cdot \Phi_{y+i,x+j}, \tag{2.17}$$

where $\Phi_{y,x} = 0$ if $y \notin \{1, \dots, h\}$ or $x \notin \{1, \dots, w\}$ as well as $K_{i,j} = 0$ if $i \notin \{1, \dots, k_y\}$ or $j \notin \{1, \dots, k_x\}$. For notational simplicity and as it is common implementation practice, we assume that the kernel K is quadratic and $k \coloneqq k_x = k_y \in 2\mathbb{N}+1$ is odd. Commonly kernels are used with $k \in \{1, 3, 5, 7, 11\}$. With this, the calculation of a single cross-correlation element[5]

$$(K * \Phi)_{y,x} = \sum_{i=1}^{k} \sum_{j=1}^{k} K_{i,j} \cdot \Phi_{y+i-1,x+j-1} \tag{2.18}$$

reduces to $\mathcal{O}(k^2)$ operations for $y \in \{1, \dots, h-k+1\}$ and $x \in \{1, \dots, w-k+1\}$. The convolution can be thought of as a kernel sliding across the image. For the sliding window approach, the kernel is slid over the spatial axes of the image with a **stride** $(s_y, s_x) \in \mathbb{N}^2$ which indicates the number of pixels the kernel is moved along the horizontal and vertical spatial dimension of the image before the next convolution is computed. In general the stride is set to $s \coloneqq s_x = s_y = 1$, i.e., for each pixel the discrete convolution is calculated based on Equation (2.18). A convolution with stride larger than 1 is named **strided convolution** and leads to the coarser evaluation

$$(K *_s \Phi)_{y,x} = \sum_{i=1}^{k} \sum_{j=1}^{k} K_{i,j} \cdot \Phi_{s_y(y-1)+i-1,s_x(x-1)+j-1}, \tag{2.19}$$

for $x = 1, \dots, w - s_x(k-1)$ and $y = 1, \dots, h - s_y(k-1)$ of the input and therefore to an output of reduced spatial dimension. For a 6×6 grayscale image the 2D-convolution with a kernel of size $(3, 3)$ is shown in Figure 2.4. It visualizes that the convolution preserves the spatial arrangement of the pixels and extracts local features.

With this we define a **convolutional layer** with a single kernel $K \in \mathbb{R}^{k \times k}$ as the linear mapping

$$\mathrm{Conv} : \mathbb{R}^{h \times w} \to \mathbb{R}^{h' \times w'}, \tag{2.20}$$

such that

$$\mathrm{Conv}(\Phi) = K * \Phi + b\mathbf{1}_{h' \times w'}, \tag{2.21}$$

where $b \in \mathbb{R}$ is a bias term and $\mathbf{1} \in \mathbb{R}^{h' \times w'}$ a matrix where all elements are equal to one.

[5] The shift of 1 in the formula derives from starting counting from 1 for the first element in the matrices.

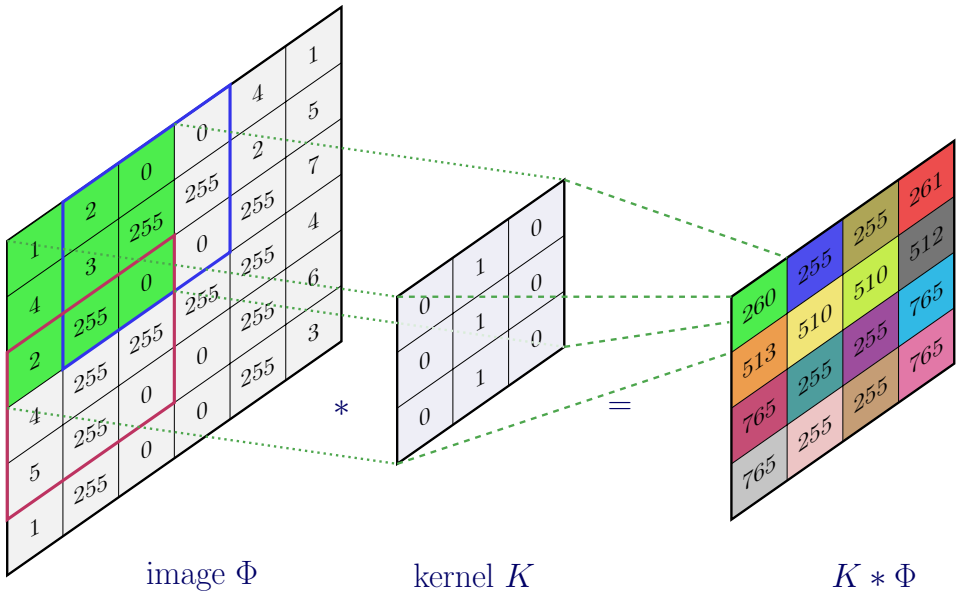


Figure 2.4: Illustration of a 2D-convolution of a 6×6 image Φ and a kernel K of size $(3, 3)$. The kernel slides over the image with a stride of 1. The first element of $K * \Phi$ is given by $1 \cdot 0 + 2 \cdot 1 + 0 \cdot 0 + 4 \cdot 0 + 3 \cdot 1 + 255 \cdot 0 + 2 \cdot 0 + 255 \cdot 1 + 0 \cdot 0 = 2 + 3 + 255 = 260$. As no padding is applied the kernel activation map $K * \Phi$ has a slightly reduced resolution. The filter value structure aims to find vertical edges. Therefore, the kernel activation is high at the pixels colored in purple, gray, cyan and pink.

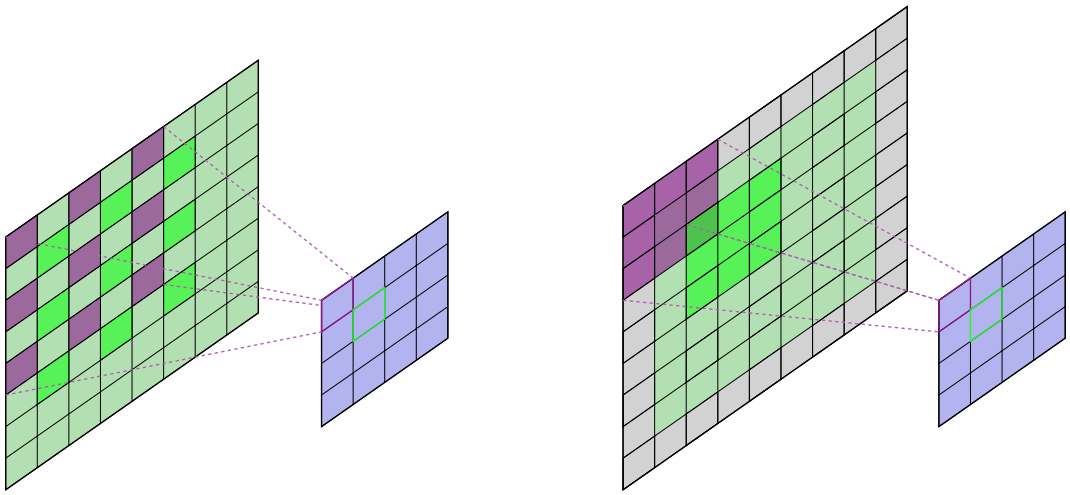
The operation of the kernel on each spatial input position (cf. Equation (2.18)) leads to a 2-dimensional activation map, showing the response of the kernel operation. Additionally, one bias value is added to the activation map forming the so-called **feature-map**[6].

In contrast to strided convolutions, for **dilated** (or **atrous**) convolutions, the kernel is split at a certain *rate* leading to a holey (*french*: à trous) kernel. This can be understood as upsampling the kernel by a factor of r and inserting zeros between kernel elements. Thereby, r resembles the sampling rate of the input feature map. Given an input image Φ the resulting feature map of a two-dimensional dilated convolution with rate r and a kernel K of size (k, k) is given by

$$(K *_r \Phi)_{y,x} = \sum_{i=1}^{k} \sum_{j=1}^{k} K_{i,j} \cdot \Phi_{y+r(i-1),x+r\cdot(j-1)}, \tag{2.22}$$

for $x = 1, \ldots, w - k^{\text{dil}} + 1$ and $y = 1, \ldots, h - k^{\text{dil}} + 1$. Thereby, $k^{\text{dil}} = k + (k-1)(r-1)$ defines the size of the holey kernel K_{dil} if the holes had been realized with zero entries. Convolutions introduced so far are dilated convolutions with rate $r = 1$ as they do not contain holes. Enlarging the rate r does neither enlarge the number of parameters

[6]Feature-map and kernel activation map are sometimes used interchangeably in the literature. We follow this convenience when a distinction is not necessary for the understanding.



(a) Dilated convolution with rate $r = 2$. The kernel still has $3 \cdot 3$ learnable parameters but non-neighboring pixel can interact due to the holey structure.

(b) Strided convolution with stride $s = 2$. The spatial dimension reduction is moderated by a padding with $\rho = 1$, visualized by the gray border pixels surrounding the green input pixels.

Figure 2.5: Schematic illustration of convolution types. The operation of the different 3×3 kernels on the input is visualized in purple and light green exemplary.

nor the computational cost as the number of learnable parameters of the dilated kernel $K \in \mathbb{R}^{k \times k}$ are equal to the enlarged kernel K_{dil} with size $(k^{\text{dil}}, k^{\text{dil}})$. A visualization of this concept is shown in Figure 2.5a.

Independently of the choice of stride or dilation, the output resolution would be reduced compared to an unmodified input, as we consider operations on tensors with a fixed size in Equation (2.18) rather than considering the convolution over $\mathbb{Z}$ with finite support. As a consequence, the operation is not defined for pixels at the border of the input. To keep the original input resolution the input tensor can be enlarged by ρ_y and ρ_x pixels at the borders of the input, such that $(K * \Phi)_{y,x}$ can be calculated for each position (y, x) in the original input. For a kernel of size (k_y, k_x), this is achieved for $\rho_x = \lfloor \frac{k_x}{2} \rfloor$ and $\rho_y = \lfloor \frac{k_y}{2} \rfloor$. This method is called **padding**. The most intuitive padding method is *zero padding* (enlarging the input tensor with zeros) which is motivated by the theory where we associate the vector with a discrete function with finite support. Additional padding methods include [88]

- *constant padding*: padding the input with a constant value (includes *zero padding*),
- *replication padding*: repeating the border values,
- *cyclic or periodic padding*: the input is repeated periodically, i.e., values are given by the input modulo its shape,
- *reflection padding*: starting at the original image border the values of the tensor are mirrored.

The general shape of a resulting feature map can be expressed by

$$\begin{aligned} w' &= \left\lfloor \frac{w - k_x + 2\rho_x + s_x}{s_x} \right\rfloor, \\ h' &= \left\lfloor \frac{h - k_y + 2\rho_y + s_y}{s_y} \right\rfloor, \end{aligned} \tag{2.23}$$

depending on the spatial dimension (k_y, k_x) of K, the stride s of the kernel evaluation and possible padding ρ_y, ρ_x in the y and x direction. A strided convolution with padding is depicted in Figure 2.5b.

Layers introduced so far reduce the spatial dimension of the input or keep the dimension constant if padding is applied. In contrast, a **transposed convolution** or **fractionally-strided convolution**[7] allows to learn an increased output [62]. A transposed convolution of an input $\Phi \in \mathbb{R}^{h \times w}$ and a kernel $K \in \mathbb{R}^{k \times k}$ yields a $(h + k - 1) \times (w + k - 1)$ feature-map

$$K *_{\mathrm{T}} \Phi. \tag{2.24}$$

The name is inspired by the representation of convolutions by a matrix multiplication. Convolutions can be written in terms of $\tilde{K}\bar{\mathbf{\Phi}}$ with a sparse matrix $\tilde{K}$ defined by the kernel and a flattened input vector $\bar{\mathbf{\Phi}}$ by concatenating the input row by row. For a 2×2 kernel

$$K = \begin{pmatrix} K_{1,1} & K_{1,2} \\ K_{2,1} & K_{2,2} \end{pmatrix} \tag{2.25}$$

and a 3×3 input, the weight matrix is given by

$$\tilde{K} = \begin{pmatrix} K_{1,1} & K_{1,2} & 0 & K_{2,1} & K_{2,2} & 0 & 0 & 0 & 0 \\ 0 & K_{1,1} & K_{1,2} & 0 & K_{2,1} & K_{2,2} & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{1,1} & K_{1,2} & 0 & K_{2,1} & K_{2,2} & 0 \\ 0 & 0 & 0 & 0 & K_{1,1} & K_{1,2} & 0 & K_{2,1} & K_{2,2.} \end{pmatrix} \tag{2.26}$$

Reshaping $\tilde{K}\bar{\mathbf{\Phi}} = ((\tilde{K}\bar{\mathbf{\Phi}})_1, (\tilde{K}\bar{\mathbf{\Phi}})_2, (\tilde{K}\bar{\mathbf{\Phi}})_3, (\tilde{K}\bar{\mathbf{\Phi}})_4)^{\mathrm{T}}$ into

$$\begin{pmatrix} (\tilde{K}\bar{\mathbf{\Phi}})_1 & (\tilde{K}\bar{\mathbf{\Phi}})_2 \\ (\tilde{K}\bar{\mathbf{\Phi}})_3 & (\tilde{K}\bar{\mathbf{\Phi}})_4 \end{pmatrix} \tag{2.27}$$

lead to the same result as $K * \Phi$. The transposed convolution is obtained by multiplying with the transposed matrix $\tilde{K}^{\mathrm{T}}$. Reshaping the result $\tilde{K}^{\mathrm{T}}\bar{\mathbf{\Phi}}$ leads to $K *_T \Phi$. The transposed convolution can be calculated as easy as a standard convolution. Its forward pass (multiplying the input by $\tilde{K}^{\mathrm{T}}$) resembles the backward pass of a convolution since the gradient of $\tilde{K}\bar{\mathbf{\Phi}}$ with respect to $\bar{\mathbf{\Phi}}$ is $\tilde{K}^{\mathrm{T}}$. Transposed convolution or more often fractionally-strided convolution in this context can also be defined with stride. Thereby,

[7] Even though this operation is sometimes called deconvolution in literature, it is worth noting that this operation does not implement the inverse of a convolution. Therefore, the name is misleading.

an upsampling of factor s resembles a strided convolution with stride $\frac{1}{s}$ which can be understood as the backward pass of a convolution of stride s.

Convolutional layers are not restricted to single-channel inputs as introduced so far. Let $\boldsymbol{\Phi} = (\Phi_1, \ldots, \Phi_m) \in \mathbb{R}^{m \times h \times w}$ be a multichannel input with $m \in \mathbb{N}$, e.g., $m = 3$ for an image with RGB channels. To enable convolving multichannel inputs, the kernel is expanded by m channels, such that $\boldsymbol{K} \in \mathbb{R}^{m \times k \times k}$. We get a **multichannel convolution** by summing over all single-channel convolutions

$$\mathrm{Conv}(\boldsymbol{\Phi}, \boldsymbol{K}) = \sum_{u=1}^{m} K_u * \Phi_u + b\mathbf{1}_{h' \times w'} \in \mathbb{R}^{h' \times w'}. \tag{2.28}$$

Thereby, the operation reduces the channel dimension m while preserving the spatial input resolution (according to resolution calculation given in Equation (2.23)). In addition, multiple features can be extracted from the same input by stacking $m'(\in \mathbb{N})$ different kernels to form a kernel tensor $\boldsymbol{K} \in \mathbb{R}^{m' \times m \times k \times k}$ leading to $\mathrm{Conv}(\boldsymbol{\Phi}, \boldsymbol{K}) \in \mathbb{R}^{m' \times h' \times w'}$ such that

$$[\mathrm{Conv}(\boldsymbol{\Phi})]_t = \sum_{u=1}^{m} K_{u,t} * \Phi_u + b_t \mathbf{1}_{h' \times w'}, \quad t = 1, \ldots, m' \tag{2.29}$$

This leads to a feature map with m' channels each representing one specific kernel activation map of a multichannel convolution.

In practical applications, CNNs consist of several composed multichannel convolutions with varying but small kernel sizes, including **$(\mathbf{1} \times \mathbf{1})$-convolutions** [157], i.e., convolutions with kernel $\boldsymbol{K} \in \mathbb{R}^{m' \times m \times 1 \times 1}$. These allow to calculate a weighted feature accumulation per pixel and by composing these layers allows for learning across channels. Thereby, the channel dimension is changed from m to m'. As a consequence, it is often used for channel dimension reduction. It is sometimes called channel pooling layer since it modifies the channel dimension but preserves the spatial dimensions. A (1×1)-convolution can be understood as an MLP with one hidden layer operating on the features (channels) of a single pixel. It is applied in the ResNet model which is a widely used component in classification and semantic segmentation models (see Section 2.3.1.1).

2.1.2.2 Pooling Layer

Besides convolution operations, a CNN often incorporates pooling operations. Pooling is a deterministic, parameter-free operation which condenses information of a $p_x \times p_y$ pixel grid $M \in \mathbb{R}^{p_x \times p_y} \subsetneq \mathbb{R}^{h \times w}$ to a single value. It is used to reduce the input resolution and increases the robustness against noise and small object translations. The most commonly used pooling methods are **max-pooling** (maximum-pooling)

$$\mathrm{pool}_{\texttt{max}} : \mathbb{R}^{p_x \times p_y} \to \mathbb{R}, M \mapsto \mathrm{pool}_{\texttt{max}} = \max_{i,j} M_{i,j} \tag{2.30}$$

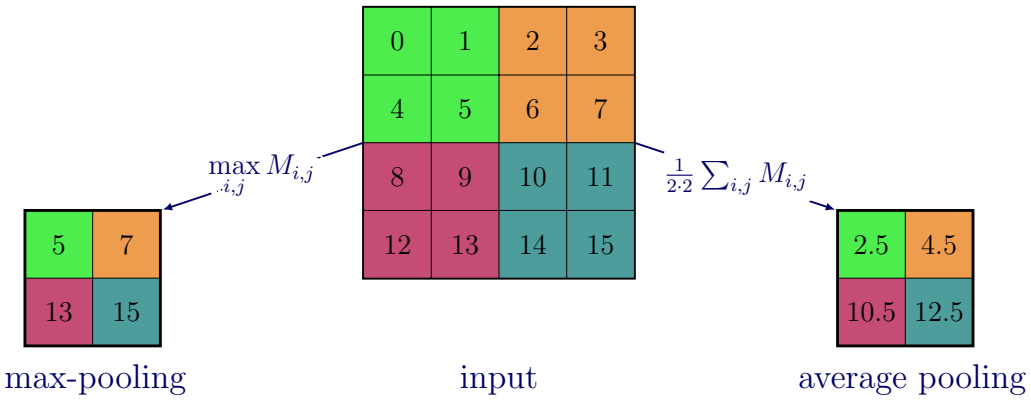


Figure 2.6: Schematic illustration of a 2×2 max (left) and average (right) pooling with a stride of 2. This halves the spatial input resolution in both dimensions from 4×4 to 2×2.

and **average pooling**

$$\text{pool}_{\texttt{avg}} : \mathbb{R}^{p_x \times p_y} \to \mathbb{R}, M \mapsto \text{pool}_{\texttt{avg}}(M) = \sum_{i=1}^{p_x} \sum_{j=1}^{p_y} \frac{1}{p_x p_y} M_{i,j}. \tag{2.31}$$

Max-pooling emphasizes high values and therefore selects the maximal feature within one grid cell. In contrast, average pooling smooths regions leading to a reduced noise sensitivity of the neural network but, on the other side, might also blur edges. The general practice is to split the image in regular 2×2 grid cells, thus $p_x = p_y = 2$, and apply the pooling operation on each of the 2×2 sub-grids. Referring to the association of a sliding window across the image, pooling can be understood as a local operation on a $p_y \times p_x$ grid which is slid across the image with a stride of (p_y, p_x) (pooling size) so that the image resolution is quartered. A visualization of the operation is depicted in Figure 2.6. A special case is **global average pooling** where the average is taken over each element in the spatial dimension of the feature map leading to a map from $\mathbb{R}^{m \times h \times w}$ to $\mathbb{R}^{m \times 1 \times 1}$. If m is set to C, the amount of categories in a classification task, it can be interpreted as a feature map resembling the confidence per category (when fed into a softmax layer). According to Lin et al. average pooling "is easier to interpret and less prone to overfitting than traditional fully connected layers" [157] as the final feature maps need to correlate to the C classes and the layer has no learnable parameter. In addition, global average pooling leads to an increased translation invariance of features as the feature map is averaged along the spatial dimension.

2.1.2.3 Characteristics of CNN

In fully connected layers each neuron in layer l has access to the activation of all neurons in the previous layer $l-1$. In contrast, convolutional layers, where a kernel with size (k_y, k_x), $k_x \ll w$ and $k_y \ll h$ is used to extract feature information, enforce a *sparse*

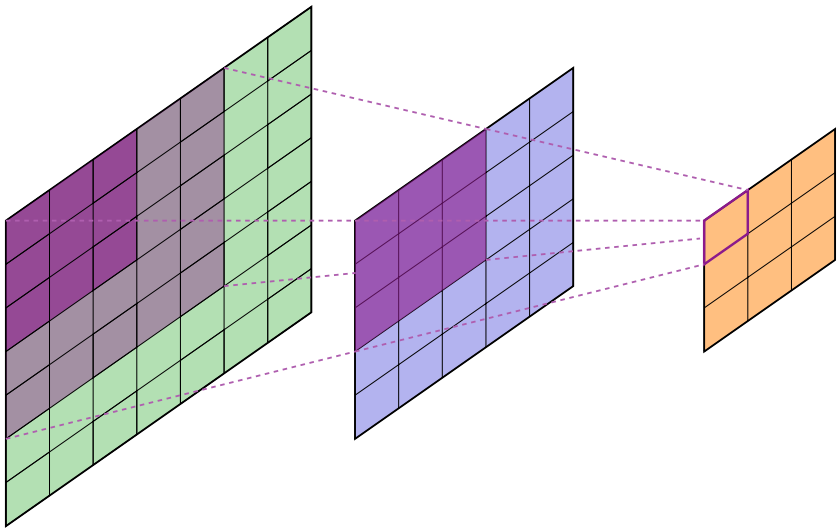


Figure 2.7: A CNN with two convolutional layers each with a 3×3 kernel. The kernel size is visualized in purple. The effective receptive field of a neuron in the last feature map (one example is highlighted in purple in the orange grid) expands with the depth of the CNN. In the previous layer (blue) the receptive field is as wide as the kernel. Due to the compositional structure of the layers, the effective receptive field in the input (green map on the left) is enlarged to 5×5.

interaction between neurons from one layer to the next. The number of neurons in the input image which have an impact on a single neuron in a subsequent layer or feature map is called **receptive field** or field of view. In the first layer the receptive field coincides with the size of the kernel. Composing multiple convolutional layers effectively increases the receptive field. This is due to the fact that in each layer features are extracted and aggregated by the convolution such that $k_y^l \times k_x^l$ of the previously calculated features are convolved in layer l. Given a neuron in layer l, the effective receptive field $(\mathrm{rf}_x^1(z_l), \mathrm{rf}_y^1(z_l))$ can iteratively be calculated for the horizontal spatial dimension by

$$\begin{aligned} \mathrm{rf}_x^l(z_l) &= k_x^l \\ \mathrm{rf}_x^{l-1}(z_l) &= \mathrm{rf}_x^l(z_l) k_x^{l-1} - (\mathrm{rf}_x^l(z_l) - s^l)(k_x^{l-1} - s^{l-1}). \end{aligned} \tag{2.32}$$

The vertical spatial dimension of the effective receptive field can be calculated analogously by replacing x by y. A scheme of the receptive field for two chained convolutional layers is visualized in Figure 2.7. In addition to chaining convolutional layers, the pooling operation (cf. Equation (2.30) or Equation (2.31)) is introduced to further enlarge the receptive field by reducing the resolution of the feature map. In addition, dilated convolutions (cf. Equation (2.22)) enable to enlarge the receptive field without reduced spatial resolution in the feature map. They are therefore valuable for dense feature maps which are needed for semantic segmentation (see Section 2.3.2).

As the same kernel is used on the entire image with $k_x \ll w$ and $k_y \ll h$, convolutional

layers have distinctly fewer learnable parameters ($k_x \cdot k_y \cdot m \cdot m' + m'$ biases) compared to fully connected layers where the learnable weight tensor W would be in $\mathbb{R}^{w \cdot h \cdot m \times w \cdot h \cdot m'}$ as each input neuron is connected to each output neuron plus $w \cdot h \cdot m'$ biases. Furthermore, the learnable weights are shared across the image due to the sliding window approach, in which the same kernel weights are used at each location of the image. In contrast, each neuron pair in a fully connected layer has a separate weight. Therefore, in CNNs a typical feature like vertical edges can be extracted with one and the same kernel at each image position leading to a distinct reduction of parameters.

Another useful characteristic is **translation equivariance**. That means, for a convolutional layer f with input $\Phi \in \mathbb{R}^{h \times w}$ and a translation mapping $T_{\boldsymbol{v}} : \mathbb{R}^{h \times w} \to \mathbb{R}^{h \times w}$ with $(T_{\boldsymbol{v}}(\Phi))_{y,x} = \Phi_{y-v_1, x-v_2}$ for $\boldsymbol{v} = (v_1, v_2) \in \mathbb{Z}^2$ applies[8]

$$T_{\boldsymbol{v}}(f(\Phi)) = f(T_{\boldsymbol{v}}^{-1}(\Phi)). \tag{2.33}$$

This is a desired property as it justifies that the use of the same kernel at different image positions leads to the same features. Furthermore, it was shown in [116, 4.4.1] that linear shift equivariant operators in two dimensions can be parameterized by convolution operators.

2.2 Learning from Data

In the last section, we gave an overview of neural network types including several building blocks. Due to the adjustable parameters $\boldsymbol{\theta}$, e.g., $\boldsymbol{\theta} = (W, \boldsymbol{b})$ or $\boldsymbol{\theta} = (K, \boldsymbol{b})$, neural networks fall into the category of parametric models. In deep learning, it is a common assumption that the data generating distribution is given by a parametric distribution which we aim to estimate with the help of neural networks. The parameters $\boldsymbol{\theta}$ can be *learned* such that the model best approximates the (unknown) distribution based on concepts from the statistical learning theory which we introduce in this section following [81, 82, 183]. The notation is mostly inspired by [82]. In Section 2.2.3, we then give an overview of which function spaces can be universally approximated with neural networks and review the sources of errors of a statistical model in Section 2.2.2. After the excursion into statistical learning theory, we explain how the parameters of the neural network are adapted in a so-called training process with the backpropagation algorithm as its core component in Section 2.2.4 and Section 2.2.4.2.

[8] This equivalently applies for multichannel input by keeping the channel fixed and applying the translation only in the spatial dimensions.

2.2.1 Statistical Learning

In contrast to optimization problems where an explicit function is minimized or maximized, there are several tasks in computer vision where the task solving function, e.g., a labeling function is unknown. When solving real world problems in computer vision, often the task is defining by data rather than by an explicit function. Furthermore, the given data is usually noisy. As a consequence, it is more common to assume that the task is defined by a data generating distribution rather than a deterministic function. Statistical learning deals with estimating the unknown distribution μ from experience, often given in terms of data $\mathcal{D}_N$ by fitting a model $\hat{\mu}$ with respect to some divergence measure[9] $\mathfrak{d}$

$$\mathfrak{d}(\mu \| \hat{\mu} \circ \mathcal{D}_N) \overset{N\to\infty}{\longrightarrow} 0 \text{ in probability} \tag{2.34}$$

for an increasing amount of data samples. In the following, we assume that the model is a parametric model $\hat{\mu} = \mu_{\boldsymbol{\theta}}$ defined by a neural network. As an example, there is no explicit function mapping known, which maps any structured pixel grid showing real world animals to their species. However, we can collect data in form of photographs of specific animals, like horses or cats, such that we have data tuples with an image and its category which we call **label** or annotation. Statistical learning theory provides us with the theoretical fundamentals that under adequate conditions we have stochastic guarantees that we can approximate the underlying data generating distribution if we sample often enough.

In the context of computer vision tasks considered in this thesis, we focus on supervised learning. In the supervised setting we observe data pairs $z = (\boldsymbol{x}, y) \in \mathcal{X} \times \mathcal{Y}$ of images $\boldsymbol{x}$ and their corresponding label or value y. Let $\mathcal{Y} = \{1, \dots, C\}$ denote the label space[10] and $\mathcal{X} = \mathbb{R}^d$ ($d \coloneqq m \times h \times w$) the image space. To understand the setup from a probabilistic perspective, we model the data by random variables $Z = (X, Y) : (\Omega, \mathcal{A}, \mathbb{P}) \to \mathcal{X} \times \mathcal{Y}$ which are distributed according to $P_{X,Y}$, the joint probability measure on $\mathcal{X} \times \mathcal{Y}$ with the Borel-σ-algebra $\mathcal{B}_{X,Y}$. The input of the statistical model follows the marginal distribution P_X of $P_{X,Y}$ and the corresponding label follows the conditional distribution $P_{Y|X}$ with conditional probability density function $p_{Y|X}$. The observed data is then modelled as[11]

$$\mathcal{D} \coloneqq \mathcal{D}_N \coloneqq (Z_1, \dots, Z_N) \sim P_{X,Y}^{\otimes N}. \tag{2.35}$$

Thereby we assume that our data can be modeled by independent identically distributed (i.i.d.) random variables. Supervised learning then describes the task to estimate the conditional data generating distribution of Y given X by a statistical model based on

[9] A divergence is a mapping $\mathfrak{d} : \mathcal{M}_1(\mathbb{R}^d) \times \mathcal{M}_1(\mathbb{R}^d) \to [0, \infty]$ with $\mathfrak{d}(\mu \| \nu) = 0$ iff $\mu = \nu$, where $\mathcal{M}_1(\mathbb{R}^d)$ denotes the set of non-negative probability measures over $(\mathbb{R}^d, \mathfrak{B}(\mathbb{R}^d))$ as the measurable space.

[10] The setup can equally be shown for a continuous label space $\mathcal{Y} = \mathbb{R}^{m'}, m' \in \mathbb{N}$ by replacing discrete measures or densities by their continuous counterparts.

[11] For the sake of readability, we use the symbol $\mathcal{D}$ for a dataset both when considering the vector of random variables (X_i, Y_i) and their realization $(\boldsymbol{x_i}, y_i)$. The difference gets clear by the notation of the data points.

the data $\mathcal{D}$. Throughout this thesis, neural networks serve as a basis for this statistical model. We refer to a neural network where the output activation a_L is the identity mapping as $\varphi_{\boldsymbol{\theta}} : \mathcal{X} \to \mathbb{R}^C$. We can recover a neural network with non-trivial output activation function by $f_{\boldsymbol{\theta}} = \phi \circ \varphi$, where $\phi : \mathbb{R}^C \to \Xi \subseteq \mathbb{R}^C$ denotes a suitable activation function or any other suitable function such that $\Xi \ni \xi = (\phi \circ \varphi)(x) \mapsto p(\cdot|\xi) \in \mathcal{M}_1^+(\mathcal{Y})$ is continuous with respect to an adequate divergence measure in $\mathcal{M}_1^+(\mathcal{Y})$, the set of positive probability measures. In the following we set $\phi = a_{\texttt{softmax}}$ to model a parametric conditional probability density function defined by

$$p_{\boldsymbol{\theta}}(y|x) = (a_{\texttt{softmax}} \circ \varphi_{\boldsymbol{\theta}}(\boldsymbol{x}))_y. \tag{2.36}$$

Based on this definition, we define a set of feasible candidate models with respect to a dataset $\mathcal{D}_N$

$$\mathcal{H} \coloneqq \{p_{\boldsymbol{\theta}} : p_{\boldsymbol{\theta}}(y|\boldsymbol{x}) = (a_{\texttt{softmax}} \circ \varphi_{\boldsymbol{\theta}}(\boldsymbol{x}))_y : \boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^\tau\}, \quad \tau \in \mathbb{N} \tag{2.37}$$

which we refer to as hypothesis space. With a certain knowledge about the task we could also restrict the hypothesis space to specific types of architectures, e.g., $\mathcal{H}^{\text{CNN}} \subseteq \mathcal{H}$ where we restrict the hypothesis space to the specific network type of CNNs. In contrast to unsupervised learning where explicitly or implicitly the data generating distribution is estimated, the aim of supervised learning in the context of deep learning is to learn the network parameters $\boldsymbol{\theta}$ with $p_{\boldsymbol{\theta}} \in \mathcal{H}$, such that

$$p_{\boldsymbol{\theta}} \approx p_{Y|X}. \tag{2.38}$$

The unsupervised case can be modeled with the same theory since supervised learning can be understood as unsupervised learning with a fixed factorization of the data-space $\mathcal{X} \times \mathcal{Y}$. When we have learned the joint distribution we can model the conditional distribution by marginalizing with respect to X. Since the dimension of the label space $\mathcal{Y}$ is usually smaller than the image space $\mathcal{X}$, learning the conditional distribution directly in a supervised manner is often easier than learning $P_{X,Y}$.

To determine the parameters of the neural network, it is necessary to define the quality of the approximation and develop a strategy for identifying optimal parameters based on that measure. To evaluate the approximation quality, we build upon the Kullback-Leibler divergence $\mathfrak{d}_{\text{KL}}$. Given two probability distributions $\mu, \nu \in \mathcal{M}_1^+(\mathcal{Y})$ which are both absolute continuous with respect to the same measure ϱ, i.e., there exist p_μ, p_ν such that $d\mu(y) = p_\mu(y)\, d\varrho(y)$ and $d\nu(y) = p_\nu(y)\, d\varrho(y)$, the **Kullback-Leibler divergence** is defined by

$$\begin{aligned} \mathfrak{d}_{\text{KL}}(p_\mu || p_\nu) =& \mathbb{E}_\varrho \left[p_\mu(y) \log \left(\frac{p_\mu(y)}{p_\nu(y)} \right) \right] \\ =& \underbrace{\mathbb{E}_\varrho \left[p_\mu(y) \log(p_\mu(y)) \right]}_{\text{neg. (differential) entropy}} \underbrace{- \mathbb{E}_\varrho \left[p_\mu(y) \log(p_\nu(y)) \right]}_{\text{(differential) cross entropy}}. \end{aligned} \tag{2.39}$$

Based on this definition, we can define the distance between the two conditional probability density functions $p_{Y|X}, p_{\boldsymbol{\theta}}$ with respect to $\mathfrak{d}_{\mathrm{KL}}$ by taking the expectation with respect to the marginal distribution P_X of X to compare probability densities on $\mathcal{Y}$

$$\begin{aligned}\mathfrak{D}_{1,\mathrm{KL}}\big(p_{Y|X}\big\|p_{\boldsymbol{\theta}}\big) &= \mathbb{E}_{X\sim P_X}\left[\mathfrak{d}_{\mathrm{KL}}\Big(p_{Y|X}\big(\,\cdot\,|X\big)\big\|p_{\boldsymbol{\theta}}\big(\,\cdot\,|X\big)\Big)\right] \\ &= \mathbb{E}_{(X,Y)\sim P_{X,Y}}\left[\log\left(\frac{p_{Y|X}(Y|X)}{p_{\boldsymbol{\theta}}(Y|X)}\right)\right] \\ &= \mathbb{E}_{(X,Y)\sim P_{X,Y}}\left[\log\big(p_{Y|X}(Y|X)\big)\right] - \mathbb{E}_{(X,Y)\sim P_{X,Y}}\left[\log\big(p_{\boldsymbol{\theta}}(Y|X)\big)\right].\end{aligned} \tag{2.40}$$

The function

$$\mathcal{L}: \mathcal{H} \to \mathbb{R}_{\geq 0}, p_{\boldsymbol{\theta}} \mapsto \mathfrak{D}_{1,\mathrm{KL}}\big(p_{Y|X}\big\|p_{\boldsymbol{\theta}}\big), \tag{2.41}$$

which maps a candidate from the hypothesis space to its quality of approximation is called **loss** or **risk function**. Assuming that the realizability assumption $p_{Y|X} \in \mathcal{H}$ is met in combination with the fact[12] that

$$\mathfrak{d}_{\mathrm{KL}}\big(p_{Y|X}(\cdot|X)\big\|p_{\boldsymbol{\theta}}(\cdot|X)\big) = 0 \Leftrightarrow p_{Y|X}(\cdot|X) = p_{\boldsymbol{\theta}}(\cdot|X), \tag{2.42}$$

minimizing the loss function would result in an optimal approximation if $p_{Y|X}$ was known[13]. However, the true distribution is unknown in practice. The only information we have about the conditional distribution $p_{Y|X}$ is given by the training samples which are drawn according to $P_{X,Y}$. As a consequence, the best approximation we can think of is to reduce the error made on the training samples. To this end, the loss function is replaced by an **empirical risk function**, i.e., a sequence of functions, measurable in $(\mathcal{X}\times\mathcal{Y})^N$ and differentiable in $\mathcal{H}$

$$\hat{\mathcal{L}} = \{\hat{\mathcal{L}}_N : \mathcal{H}\times(\mathcal{X}\times\mathcal{Y})^N \to \mathbb{R}_{\geq 0}\}_{N\in\mathbb{N}} \tag{2.43}$$

which converges under adequate conditions to the true loss function in probability when enlarging the dataset. We define

$$\begin{aligned}\hat{\mathcal{L}}_N(p_{\boldsymbol{\theta}}, \mathcal{D}_N) &= \frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{\theta}; f_{\boldsymbol{\theta}}(X_n), Y_n) \\ &= \frac{1}{N}\sum_{n=1}^{N}\log\left(\frac{p_{Y|X}(Y_n|X_n)}{p_{\boldsymbol{\theta}}(Y_n|X_n)}\right) \\ &= \frac{1}{N}\sum_{n=1}^{N}\log\big(p_{Y|X}(Y_n|X_n)\big) - \frac{1}{N}\sum_{n=1}^{N}\log\big(p_{\boldsymbol{\theta}}(Y_n|X_n)\big)\end{aligned} \tag{2.44}$$

[12] The proof of Equation (2.42) bases on the properties of the log function and Jensen's inequality. It can be found in e.g., [183, Theorem 6.2.1.].

[13] If $p_{Y|X}$ was known, we could simply use it rather than approximating it.

for which holds

$$\hat{\mathcal{L}}_N(p_{\boldsymbol{\theta}}, \mathcal{D}_N) \to \mathfrak{D}_{1,\mathrm{KL}}\big(p_{Y|X} \big\| p_{\boldsymbol{\theta}}\big) \text{ for } N \to \infty \tag{2.45}$$

by the law of large numbers. As a consequence, the optimal parameters for a model with respect to the so-called **empirical risk minimization** principle are defined by

$$\hat{\boldsymbol{\theta}} \in \underset{\boldsymbol{\theta}\in\Theta: p_{\boldsymbol{\theta}}\in\mathcal{H}}{\operatorname{argmin}} \hat{\mathcal{L}}_N(p_{\boldsymbol{\theta}}, \mathcal{D}_N). \tag{2.46}$$

As the first summand is not dependent on $\boldsymbol{\theta}$, it is enough to minimize

$$\hat{\mathcal{L}}_N^{\mathrm{NLL}}(p_{\boldsymbol{\theta}}, \mathcal{D}_N) = -\frac{1}{N}\sum_{n=1}^{N} \log\big(p_{\boldsymbol{\theta}}\big(Y_n|X_n\big)\big), \tag{2.47}$$

which is the averaged conditional negative log-likelihood and a standard loss in supervised learning (see Sections 2.3.1 and 2.3.2). The conditional negative log-likelihood can be derived from the principle of **maximum likelihood estimation**. It is an often applied principle in deep learning where the parameters of the parametric model are estimated in such a way that the training data gets assigned the highest probability. The maximum likelihood estimator, i.e., the parameters which maximize the likelihood of a dataset $\mathcal{D} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\}$ under a model $p_{\boldsymbol{\theta}}$, satisfies

$$\hat{\boldsymbol{\theta}}_{\mathrm{MLE}} = \underset{\boldsymbol{\theta}\in\Theta_N: p_{\boldsymbol{\theta}}\in\mathcal{H}}{\operatorname{argmax}} \prod_{n=1}^{N} p_{\boldsymbol{\theta}}(y_n|\boldsymbol{x_n}). \tag{2.48}$$

This is equivalent to maximizing the logarithm of the product since the logarithm is a monotone function. As a consequence, the optimization problem reduces to the sum over each data point

$$\hat{\boldsymbol{\theta}}_{\mathrm{MLE}} = \underset{\boldsymbol{\theta}\in\Theta_N: p_{\boldsymbol{\theta}}\in\mathcal{H}}{\operatorname{argmax}} \sum_{n=1}^{N} \log(p_{\boldsymbol{\theta}}(y_n|\boldsymbol{x_n})), \tag{2.49}$$

which is numerically more stable to compute. By negation, the problem can equivalently be considered as a minimizing problem, leading to the noted **conditional negative log-likelihood** ($\mathtt{NLL}(\mathcal{D}|\boldsymbol{\theta})$) in Equation (2.47)

$$\hat{\boldsymbol{\theta}}_{\mathrm{MLE}} = \underset{\boldsymbol{\theta}\in\Theta_N: p_{\boldsymbol{\theta}}\in\mathcal{H}}{\operatorname{argmin}} \underbrace{\left(-\sum_{n=1}^{N} \log(p_{\boldsymbol{\theta}}(y_n|\boldsymbol{x_n}))\right)}_{\mathtt{NLL}(\mathcal{D}|\boldsymbol{\theta})}. \tag{2.50}$$

2.2.2 Error Decomposition

Even though we have seen strategies to statistically estimate the true conditional density function from data, the practice involves multiple sources of errors. The error in ERM

learning decomposes into a model error (ε_{model}), an estimation or learning error (ε_{learn}) and a sampling error (ε_{sampl}). The error made by the estimator $p_{\hat{\boldsymbol{\theta}}}$ with respect to the true distribution $p_{X|Y}$ and $\mathfrak{D}_{1,\mathrm{KL}}$ can be upper bounded by

$$0 \leq \mathfrak{D}_{1,\mathrm{KL}}\left(p_{Y|X} \middle\| p_{\hat{\boldsymbol{\theta}}}\right) \leq \varepsilon_{model} + \varepsilon_{learn} + 2\varepsilon_{sampl}. \tag{2.51}$$

The **model error**

$$\varepsilon_{model} = \inf_{p_{\boldsymbol{\theta}} \in \mathcal{H}} \mathfrak{D}_{1,\mathrm{KL}}\left(p_{Y|X} \middle\| p_{\boldsymbol{\theta}}\right) \tag{2.52}$$

describes the minimal error achievable with a fixed hypothesis space $\mathcal{H}$. This error is independent of the training sample amount [246]. The error is also known as approximation error since it links to the universal approximation property of a hypothesis space. In Section 2.2.3 we show that neural networks in theory have the capacity to express or approximate to any degree of accuracy a wide range of function classes. This gives insights on how the model error can be controlled. If the realizability assumption is satisfied, the error equals to zero. Mis-specification of the hypothesis space by restricting, e.g., the architecture of the neural network class and therefore ending up in a poor model design choice, leads to $\varepsilon_{model} > 0$. Depending on the distribution to approximate, enlarging the hypothesis space can diminish the error at the cost of the complexity of the model.

Additional sources of errors are **estimation errors** or learning errors during the training procedure denoted by

$$\varepsilon_{learn} = \mathcal{L}_{\mathcal{D}_N}(p_{\hat{\boldsymbol{\theta}}}, \mathcal{D}_N) - \min_{p_{\boldsymbol{\theta}} \in \mathcal{H}} \mathcal{L}_{\mathcal{D}_N}(p_{\boldsymbol{\theta}}, \mathcal{D}_N). \tag{2.53}$$

In theory this error vanishes for learning algorithms which exactly minimize the empirical risk. However, in practice these errors can for example arise from gradient estimation in stochastic gradient decent (see Section 2.2.4.1) and optimizer-specific errors like convergence to a local minimum. Lastly, statistical errors, referred to as **sampling error**(ε_{sampl}), occur due to the limited representativity of the true data generating function by training samples and violation of theoretical assumptions like the independence of the training samples. The error is given by

$$\varepsilon_{sampl} = \sup_{p_{\boldsymbol{\theta}} \in \mathcal{H}} \left| \mathcal{L}(p_{\boldsymbol{\theta}}) - \mathcal{L}_{\mathcal{D}_N}(p_{\boldsymbol{\theta}}, \mathcal{D}_N) \right|. \tag{2.54}$$

The error depends on the model complexity and the size of the training dataset. Since we approximate the true loss $\mathcal{L}$ with a surrogate loss defined by training samples (the empirical risk function), the sampling error $\varepsilon_{\text{sampling}}$ is linked to the conditions on the convergence of the empirical risk function to the true loss function for an increasing amount of data samples. However, a uniform convergence is needed to control the error. If the hypothesis space satisfies the so-called uniform convergence property [246, Def.4.3], an arbitrary small sampling error can be obtained when increasing the amount of data.

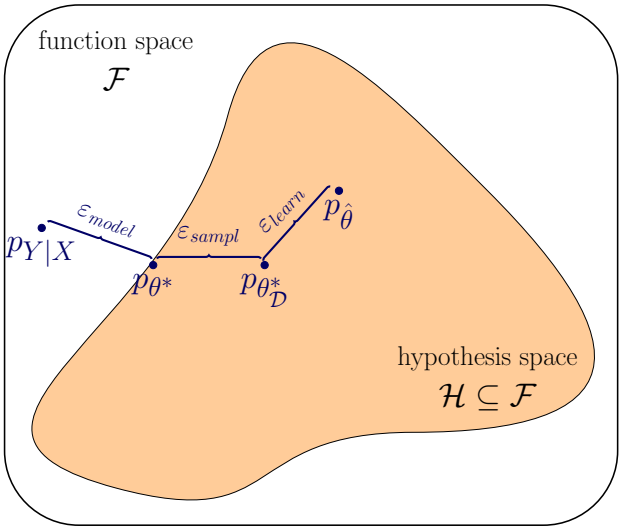


Figure 2.8: The error made compared to the true distribution $p_{Y|X}$ can be split into the model error (ε_{model}), the sampling error ($\varepsilon_{sampling}$) and the estimation error (ε_{learn}). The first arises when the model does not lie within the hypothesis space $\mathcal{H}$. We refer to $p_{\theta^*} \in \mathcal{H}$ to the optimal model within the hypothesis space. As we optimize the parameters with respect to the minimal error on the training samples $\mathcal{D}$ the best model is $p_{\theta^*_{\mathcal{D}}}$. Additionally, we face an error due to the learning algorithm, leading to the best estimator $p_{\hat{\theta}}$ achievable with a learning algorithm.

Inspired by the visualization in [107] the different error sources are depicted in Figure 2.8.

Given a fixed model the sampling error diminishes with the increase of the amount of independently drawn training samples from the data generating distribution. However, when increasing the model complexity to reduce the model error, the sampling error most likely increases due to overfitting, i.e., the model adapts strongly to the data rather than learning the true conditional density function. This roots in the statistical approximation of the loss function. Since the loss function is only defined on samples, it fosters potential overfitting to the training samples disregarding performance on unseen data. Therefore, a trade-off between model complexity, which defines the expressiveness, and the amount of training samples is needed for a well generalizing model. [78, 246]. This is sometimes referred to bias-variance trade-off [208] or bias-complexity trade-off [246]. The bias in both cases denotes the inductive bias which is included by defining a hypothesis space. By restricting the model to a specific function class or, e.g., network architecture, certain assumptions on the true distribution are made. The variance term refers to the diversity of models when trained on different datasets drawn from the same distribution. A huge variance is observed when the models over adapt to data since training a neural network on different dataset would lead to different optimal parameters if the model over-adapted to the data, although both aim to approximate the

same distribution. To monitor the phenomenon of overfitting, data is usually split into two or three disjoint and equally distributed subsets – a training set, a validation set and an optional test set. The first one is used to estimate $\hat{\boldsymbol{\theta}}$. The second is used to validate the generalization capability of the trained model. It is used to observe overfitting and builds the decision basis which model performs best during the training process (details on the training process are described in the next section). Furthermore, the validation split allows to choose between models generalizability potential when changing hyperparameters of the architecture or the training process itself. The test set is used to evaluate the performance of the finally selected model to have an independent set which was not used for training or used for model selection (otherwise an adaptation to the validation set is possible). The latter can be discarded if no hyperparameter tuning is done. We introduce regularization methods to mitigate overfitting in Section 2.2.5.

2.2.3 Universal Approximation

This section examines the capability of neural networks to approximate certain function classes. Neural networks are parametric models which have proven to approximate diverse real-world functions very well [37,142]. In the following we give a short overview over the theoretical foundations of this phenomenon. We define a sequence of increasing hypothesis spaces $\{\mathcal{H}_q\}_{q\in\mathbb{N}}$ where $\mathcal{H}_q \subseteq \mathcal{H}_{q+1}$ with

$$\mathcal{H}_q = \{p_{\boldsymbol{\theta}} : p_{\boldsymbol{\theta}}(y|x) = (a_{\texttt{softmax}} \circ \varphi_{\boldsymbol{\theta}}(x))_y : \boldsymbol{\theta} \in \Theta_q \subseteq \mathbb{R}^{\tau_q}\}. \tag{2.55}$$

The different hypothesis spaces can be understood as variations of the network architecture, e.g., variation of width or depth. Additionally, we assume that $\varphi_{\boldsymbol{\theta}} : \mathcal{K} \to \mathbb{R}^C$, denoting a neural network with linear output layer, is defined on a compact set $\mathcal{K} \subseteq \mathbb{R}^d$. We further define the function space defined by those neural networks by

$$\mathcal{F}_{\mathcal{H}} = \{\varphi_{\boldsymbol{\theta}} : \mathcal{K} \to \mathbb{R}^C : \boldsymbol{\theta} \in \Theta_q \subseteq \mathbb{R}^{\tau_q}\}. \tag{2.56}$$

In the following we assume that the true conditional density function $p_{Y|X}$ is parameterized by a function $\varphi \in \mathcal{F}$ such that $p_{Y|X}$ is given by

$$p_{Y|X}(y|(\phi \circ \varphi)(\boldsymbol{x})) \tag{2.57}$$

for all $x \in \mathcal{K}$. Thereby $\mathcal{F}$ denotes a function space, e.g., $\mathcal{F}$ could be the space of continuous function on $\mathcal{K}$. This composite view on the parametric models allows to reduce the complex approximation problem of conditional probability densities to an approximation problem on function spaces of functions from $\mathbb{R}^d$ to $\mathbb{R}^C$. We say, $\mathcal{F}_{\mathcal{H}}$ possesses the **universal approximation property** with respect to $\mathcal{F}$ and the norm $\|\cdot\|_\infty$ if for all $\varphi \in \mathcal{F}$

$$\inf_{\boldsymbol{\theta}\in\Theta_q} \|\varphi - \varphi_{\boldsymbol{\theta}}\|_\infty \overset{q\to\infty}{\longrightarrow} 0. \tag{2.58}$$

Under adequate continuity assumption with respect to $\mathfrak{D}_{1,\mathrm{KL}}$ it holds

$$\varepsilon_{model} \overset{q\to\infty}{\longrightarrow} 0 \tag{2.59}$$

if $\mathcal{F}_{\mathcal{H}}$ possesses the universal approximation property with respect to $\mathcal{C}(\mathcal{K}, \mathbb{R}^C)$, the function space of continuous functions from $\mathcal{K}$ to $\mathbb{R}^C$. We refer to [82, Prop. 16.2] for the proof. In universal approximation theory the focus is on the existence of such an approximation rather than the question if a training procedure is capable to find such an approximation.

It has been shown that fully connected neural networks with a single hidden layer are universal approximators[14] for continuous functions on the unit cube under certain assumptions [50, 102]. In detail, let $\mathcal{K} = [0,1]^d \subseteq \mathbb{R}^d$ be the compact unit cube. Let $\mathcal{F}_{\mathcal{H}}$ be the set of fully connected neural networks with one hidden layer with width $w_N \in \mathbb{N}$ and a linear output layer without a bias term such that $w_N \to \infty$ as $N \to \infty$ and let $\boldsymbol{\theta} \in \Theta_N = \mathbb{R}^{(w_N+1)\times d + w_N}$. The activation function $a : \mathbb{R} \to \mathbb{R}$ is assumed to be continuous and sigmoidal[15]. Cybenko proved in [50] that then

$$\mathcal{F}_{\mathcal{H}} \text{ possesses the universal approximation property with respect to } \mathcal{C}(\mathcal{K}, \mathbb{R}).$$

This result can be generalized to $\mathcal{C}(\mathcal{K}, \mathbb{R}^{m'})$, $m' \in \mathbb{N}$, by stacking multiple multilayer perceptrons of the denoted form to approximate the function per component. Furthermore, it has been shown by Cybenko that $\mathcal{F}_{\mathcal{H}}$ defined as above but with a bounded measurable sigmoidal activation function $a : \mathbb{R} \to \mathbb{R}$ possesses the universal approximation property with respect to $\mathcal{F} = L^1(\mathcal{K})$ and $\|\cdot\|_{L^1}$. With other words, any Borel measurable function in $\mathbb{R}^d$ can be approximated almost everywhere by a single hidden layer neural network with increasing width and bounded measurable sigmoidal activation function [50]. Replacing the activation function in $\mathcal{F}_{\mathcal{H}}$ by a ReLU (more generally for non-polynomial locally bounded piecewise continuous) activation function, it was shown by Leshno et al. in the early 1990s that this network class holds the universal approximation property with respect to $F = \mathcal{C}(\mathcal{K}, \mathbb{R})$ [151].

The choice of the activation function for a neural network thereby is crucial for its expressiveness. If we consider an MLP where all activation functions are set to the identity mapping, a linear model based on matrix vector multiplication is left over. As a consequence, only (affine) linear functions can be represented. Thus, the non-linearity of the activation functions is essential for the capability to represent non-linear functions.

These results assert that single hidden layer MLPs have enough expressiveness for a wide range of function classes but the width of the neural networks might be impracticable or infeasibly large. Furthermore, the statements lack a relation between a maximal acceptable model error ε_{model} and the number of neurons, the width and the depth of a

[14] i.e., $\mathcal{F}_{\mathcal{H}}$, where $\varphi_{\boldsymbol{\theta}}$ is the mentioned neural network class, possesses the universal approximation property

[15] We call a function $\sigma : \mathbb{R} \to \mathbb{R}$ sigmoidal iff $\lim_{x\to-\infty} \sigma(x) = 0$ and $\lim_{x\to\infty} \sigma(x) = 1$.

neural network architecture. Yarotsky showed in a constructive manner that MLPs with ReLU activations can approximate 'relatively smooth' functions to any desired degree of accuracy [307]. Particularly, upper bounds on the depth and amount of neurons and therefore the width in a neural network with respect to a given error ε were established. This is mathematically formulated in the following theorem by Yarotsky [307]. Let $\mathcal{F}_{\beta,d} = \{\varphi \in W^{\beta,\infty}([0,1]^d) : \|\varphi\|_W^{\beta,\infty} \leq 1\}$ denote the functions in the unit ball of the $W^{\beta,\infty}([0,1]^d)$-Sobolev space [4]. For any d, β and $\varepsilon \in (0,1)$ there is a ReLU network architecture $\{\varphi_{\boldsymbol{\theta}}\}_{\boldsymbol{\theta} \in \Theta_{d,\beta,\varepsilon}}$ that

1. is capable of expressing any function from $\mathcal{F}_{\beta,d}$ with error ε, i.e.

$$\forall \varphi \in \mathcal{F}_{\beta,d}\ \exists \hat{\boldsymbol{\theta}} \in \Theta_{d,\beta,\varepsilon} : \|\varphi - \varphi_{\hat{\boldsymbol{\theta}}}\|_\infty < \varepsilon \tag{2.60}$$

2. has depth of at most $c \cdot (\log(1/\varepsilon) + 1)$ and at most $c\varepsilon^{-d/\beta}(\log(1/\varepsilon) + 1)$ weights or neurons, respectively, with some constant $c = c(d, \beta)$.

Note, that $\Theta_{d,\beta,\varepsilon}$ is fixed for all φ, i.e., the number of neurons does not vary. This theorem demonstrates that the model complexity (number of neurons) depends on the inverse of the approximation error. In addition, it was shown that for a given ε a ReLU network $\varphi_{\boldsymbol{\theta}}$ with fixed depth L and parameters in $\Theta_{L,\varepsilon}$ needs at least $c\varepsilon^{-1/(2(L-2))}$ neurons with some constant $c = c(\varphi, L) > 0$ to approximate a non-linear two-times differentiable function $\varphi \in \mathcal{C}^2([0,1]^d)$ with continuous extension of the second derivatives on the boundary, i.e.,

$$\forall \varphi \in \mathcal{C}^2([0,1]^d)\ \exists \hat{\boldsymbol{\theta}} \in \Theta_{L,\varepsilon} : \|\varphi - \varphi_{\hat{\boldsymbol{\theta}}}\|_\infty < \varepsilon. \tag{2.61}$$

From these two theorems it follows that ReLU neural networks with unbounded depth need fewer neurons than neural networks with a fixed depth to express the same smooth function class.

Beside the investigations on MLPs also universal approximation statements can be made for convolutional neural networks on finite pixel grids with periodic boundary conditions. Yarotsky proved in [308] that translation equivariant maps from $\mathbb{R}^{m_1 \times h \times w}$ to $\mathbb{R}^{m_2 \times h \times w}$ can be approximated by a convolutional neural network with a single layer (without pooling) with respect to $\|\cdot\|_\infty$. To mitigate constraints on the functions which can be approximated, the setup provides that the kernel can have the same size as the input to not restrict the field of view. For approximation statements with respect to deep convolutional neural networks with pooling on infinite dimensional spaces we refer to [308, Sec. 3.3.].

2.2.4 Training Process

In Section 2.2.1 we introduced the concept of empirical risk functions. These functions provide the basis for learning the parameters of a neural network in a training process.

The details of the training process are discussed based on [78] in the following section. As above, we consider a supervised learning problem. In the following we consider neural networks of the form $f_{\boldsymbol{\theta}} : \mathcal{X} \to (0,1)^C$, $\boldsymbol{x} \mapsto (a_{\texttt{softmax}} \circ \varphi)(\boldsymbol{x})$ with learnable parameters $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^\tau$ constraint by the architecture of the neural network. To be able to train a neural network, data $\mathcal{D}$ from a data model $\{(X_i, Y_i)\}_{i=1}^N \sim P_{X,Y}$ is observed. We call $\mathcal{D} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\}$ with $(\boldsymbol{x_i}, y_i) \in \mathcal{X} \times \mathcal{Y}$ the training dataset where $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y} = \{1, \ldots, C\}$, $C \in \mathbb{N}$. An (empirical) loss function

$$\mathcal{L}_{\mathcal{D}} : \Theta \to \mathbb{R}_{\geq 0}, \boldsymbol{\theta} \mapsto \frac{1}{N} \sum_{i=1}^{N} L(f_{\boldsymbol{\theta}}(\boldsymbol{x_i}), y_i) \tag{2.62}$$

with respect to the dataset $\mathcal{D}$ is a member of the empirical risk function $\{\hat{\mathcal{L}}_N\}_{N=1}^{\infty}$. Since the parametric model $p_{\boldsymbol{\theta}}$ as defined in Equation (2.36) is completely described by the parameters of the neural network $f_{\boldsymbol{\theta}}$ and a fixed dataset $\mathcal{D}$ is considered, it is sufficient to define the loss on Θ rather than on the hypothesis and sample space (cf. Equations (2.43) and (2.44)). To visualize the dependence on the dataset we replace N with $\mathcal{D}$ in the notation. The negative log-likelihood

$$\mathcal{L}_{\mathcal{D}}^{\mathrm{NLL}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \underbrace{\left(-\log\left(f_{\boldsymbol{\theta}}(\boldsymbol{x_i})_{y_i}\right)\right)}_{=L(f_{\boldsymbol{\theta}}(\boldsymbol{x_i}), y_i)}, \tag{2.63}$$

serves as an example of a standard loss function. The training objective is to solve the non-linear optimization problem

$$\min_{\boldsymbol{\theta} \in \Theta} \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}), \tag{2.64}$$

leading to the optimal parameters with respect to the empirical loss

$$\hat{\boldsymbol{\theta}} \in \operatorname*{argmin}_{\boldsymbol{\theta} \in \Theta} \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}). \tag{2.65}$$

This is done in an iterative process by a gradient-based optimization.

2.2.4.1 Stochastic Gradient Descent

To find a (local) minimum of the loss function, different forms of gradient decent algorithms [69] are usually used in deep learning contexts. It is an iterative method where in each iteration the direction of the steepest descent is used to update the current iteration value. For a real valued function $l : \mathbb{R}^\tau \to \mathbb{R}, \boldsymbol{\theta} \mapsto l(\boldsymbol{\theta})$ and randomly chosen initial parameters $\boldsymbol{\theta}^{(0)} \in \Theta$, the parameter update rule for the gradient descent algorithm is defined by

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta^{(i)} [\nabla_{\boldsymbol{\theta}^{(i)}} l(\boldsymbol{\theta}^{(i)})]\big|_{\boldsymbol{\theta} = \boldsymbol{\theta}^{(i)}}. \tag{2.66}$$

The parameter η^i is the so-called **learning rate** and controls the step length in each iteration.

As we consider an empirical loss with respect to the dataset $\mathcal{D}$ as minimization objective (Equation (2.64)), the gradient with respect to all training data points needs to be calculated. Computing the exact gradient would require to load the whole dataset. In most of the use cases, this is impossible due to hardware memory limitations and leads most likely to overfitting to the training samples [78]. Alternatively, the gradient can be computed with respect to a single data point. For a single data point $(\boldsymbol{x}, y)$ and a randomly initialized parameter $\boldsymbol{\theta} = \boldsymbol{\theta}^{(0)}$ the update rule of stochastic gradient descent based on the loss $L(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$ of a model $f_{\boldsymbol{\theta}}$ is given by

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta^{(i)} [\nabla_{\boldsymbol{\theta}} L(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y)]\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(i)}}. \tag{2.67}$$

Stochastic gradient descent is likely to be unstable when the gradient is calculated with respect to a single data point as the descent direction can change drastically in each iteration. As a consequence, the gradient is replaced by an estimate $\hat{\nabla}_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}}$ based on a subset of data points, so-called batches. In practice this is realized by randomly splitting the training data into multiple disjoint (mini-)**batches** of size $N_{\mathcal{B}}$ with $N \equiv 0 \mod N_{\mathcal{B}}$

$$\mathcal{B}_i \subseteq \mathcal{D} \text{ such that } \dot{\bigcup_i} \mathcal{B}_i = \mathcal{D} \text{ and } |\mathcal{B}_i| = N_{\mathcal{B}}. \tag{2.68}$$

An unbiased estimator of the gradient can be obtained by averaging the point-wise gradient with respect to the randomly sampled subset $\mathcal{B} \subseteq \mathcal{D}$ [78]

$$\hat{\nabla}_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{D}} = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{B}} \coloneqq \frac{1}{N_{\mathcal{B}}} \sum_{n=1}^{N_{\mathcal{B}}} \nabla_{\boldsymbol{\theta}} L(f_{\boldsymbol{\theta}}(\boldsymbol{x_n}), \boldsymbol{y_n}). \tag{2.69}$$

If $N \bmod N_{\mathcal{B}} \neq 0$, often the last batch which contains less than $N_{\mathcal{B}}$ data points is dropped during training. Alternatively, the gradient estimate based on the last batch is computed on a smaller amount of data points. The advantage of minibatch stochastic gradient descent (SGD) is that the compute-time is only dependent on the batch size and does not scale with the size of the training dataset. After iterating once over all batches and therefore over the whole dataset which we call **epoch**, a new batch partition is sampled for the next epoch. In practice the neural network performance benefits from training multiple epochs on resampled data.

The overall training stops based on time or performance conditions. A time-based stopping criterion means that the training stops when the neural network was trained for a predefined amount of iterations (number of batches) or epochs (number of times the complete dataset was processed). Training can also be terminated, if a certain target performance on a validation dataset has been reached or the loss function saturates on either the training or validation data for multiple consecutive iterations or epochs. The stopping criteria can also be combined such that the training stops when one of the

criteria is met. If the performance-based criterion is met before the amount of predefined iterations or epochs is reached, it is referred to as **early stopping**, often used in active learning experiments (see Section 2.4.4.1) to moderate the training time.

In contrast to optimization in general, in the context of statistical learning and especially deep learning the learning rate needs to be reduced over time to ensure a stable training. This is due to the fact that the gradient is estimated with the help of a batch of size $N_B \ll N$. As a consequence, the gradient value is noisy and in contrast to general optimization does not necessarily vanish in consecutive steps near a minimum. Multiple learning rate schedules can be defined mostly depending on the total amount of epochs $\mathfrak{ep}$. Some examples are [201]:

- linear decay: $\eta^{i+1} = 1 - \frac{\max(0, i+1-N\cdot\mathfrak{ep})}{(\text{decay}+1)}$
- polylinear decay, e.g., $\eta^{i+1} = \left(\frac{1-\eta^i}{N\cdot\mathfrak{ep}}\right)^{0.9}$
- exponential decay: $\eta^{i+1} = \alpha \cdot \eta^i,\ \alpha \in \mathbb{R}$
- reduce on plateau (reduce the learning rate when a validation metric stagnates for a number of iterations).

The first one we use for CycleGAN training (Chapter 3) whereas we use the polylinear decay for training semantic segmentation networks (Section 2.3.2).

The loss landscape $\{(\boldsymbol{\theta}, \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta})) \in \Theta \times \mathbb{R} : \boldsymbol{\theta} \in \Theta\}$ of neural networks is often non-convex and ragged which leads to frequently changing descent directions. An approach which increases the training speed is to smooth the update by incorporation of the previously calculated gradients. A momentum term v is introduced to keep following the past gradient directions and thereby stabilize the training (direction). It accumulates the past gradients by an exponentially decaying moving average

$$\begin{aligned}
\boldsymbol{v}^{(i+1)} &= \alpha \boldsymbol{v}^{(i)} - \eta^{(i)} [\nabla_{\boldsymbol{\theta}} \mathcal{L}_B(\boldsymbol{\theta})]\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(i)}} \\
\boldsymbol{\theta}^{(i+1)} &= \boldsymbol{\theta}^{(i)} + \boldsymbol{v}^{(i+1)} \\
&= \boldsymbol{\theta}^{(i)} + \alpha \boldsymbol{v}^{(i)} - \eta^{(i)} [\nabla_{\boldsymbol{\theta}} \mathcal{L}_B(\boldsymbol{\theta})]\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(i)}} \\
&= \boldsymbol{\theta}^{(i)} - \eta^{(i)} [\nabla_{\boldsymbol{\theta}} \mathcal{L}_B(\boldsymbol{\theta})]\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(i)}} - \sum_{j=1}^{i} \alpha^j \eta^{(i-j)} [\nabla_{\boldsymbol{\theta}} \mathcal{L}_B(\boldsymbol{\theta})]\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(i-k)}}.
\end{aligned} \tag{2.70}$$

Thereby $\alpha \in [0, 1]$ models the decay rate of the previously calculated gradients. SGD with momentum was firstly introduced by [206]. Multiple variants of SGD where published since its extensive usage for deep learning problems. Especially algorithms with adaptive learning rates for each individual parameter θ_j such as RMSProp [270] with and without momentum, AdaGrad [61] and Adam [132] improved the training progress. The latter, named after "adaptive moments", as it uses the first and second moments of the gradients to estimate the best descent direction, is used in this thesis. Even though

the learning rate is adapted individually for each parameter, Loshchilov and Hutter suggest combining it with a global learning rate scheduler [164]. We follow this suggestion and apply learning rate schedulers for all our trainings throughout the thesis.

2.2.4.2 Backpropagation

All previously mentioned optimization algorithms make use of the loss function's gradient. Calculating the gradient is not straight forward as there is no closed form of the loss function due to the structure of the neural network it depends on. Therefore, calculating the gradient is realized by the so-called reverse mode of algorithmic or automatic differentiation [83] which dates back to the 1970s [160]. The core idea behind this method is to exploit the chain rule of differentiation for compositional functions. Algorithmically formulated calculations can often be expressed by a composition of multiple elementary subfunctions like addition, multiplication, exponential or trigonometric functions each having simple derivatives. The derivative of the original compositional function can then be expressed by the composition of the partial derivatives of the elementary functions. Given two differentiable functions $g : \mathbb{R}^m \to \mathbb{R}, \boldsymbol{y} \mapsto g(\boldsymbol{y})$ and $f : \mathbb{R}^t \to \mathbb{R}^m, \boldsymbol{x} \mapsto f(\boldsymbol{x})$ the chain rule for derivates states that for

$$z = g(f(\boldsymbol{x})) = g(\boldsymbol{y}) \tag{2.71}$$

the derivative is given by

$$\partial_{\boldsymbol{x}}(g \circ f)(\tilde{\boldsymbol{x}}) = \partial_{\boldsymbol{y}} g\big|_{\boldsymbol{y}=f(\tilde{\boldsymbol{x}})} \cdot \partial_{\boldsymbol{x}} f\big|_{\boldsymbol{x}=\tilde{\boldsymbol{x}}}. \tag{2.72}$$

The vector elements are defined by

$$\partial_{\boldsymbol{x}}(g \circ f)_i = \frac{\partial(g \circ f)}{\partial x_i} = \frac{\partial z}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \text{ for } i = 1, \ldots, t. \tag{2.73}$$

Thereby, $\partial_{\boldsymbol{x}} f$ denotes the Jacobian matrix in the variable $\boldsymbol{x}$ to clarify according to which variable the differentiation is done. For scalar or tensor-valued functions, we use the same notation to describe the vector or tensor whose entries are defined by the partial derivatives.

In the context of deep learning an algorithm called **backpropagation algorithm** grew in popularity, which was independently published by Rumelhart et al. in 1986 [229]. The algorithm calculates algorithmically derivatives for layered models and exploits the chain rule to propagates derivatives from the output layer back to the input layer. The aim is to calculate the partial derivative of the loss function with respect to the learnable parameters $\boldsymbol{\theta}$

$$\partial_{\boldsymbol{\theta}} L(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}). \tag{2.74}$$

For a fixed point $\tilde{\boldsymbol{\theta}}$ in the parameter space Θ the gradient breaks down into two main components

$$\partial_{\boldsymbol{\theta}} L(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y})\big|_{\boldsymbol{\theta}=\tilde{\theta}} = \partial_{f_{\boldsymbol{\theta}}} L(f_{\boldsymbol{\theta}}, \boldsymbol{y})\big|_{f_{\boldsymbol{\theta}}=f_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{x})} \cdot \partial_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x})\big|_{\boldsymbol{\theta}=\tilde{\boldsymbol{\theta}}}, \tag{2.75}$$

the derivative of L with respect to the output of the neural network and the derivative of the neural network with respect to the learnable parameters $\boldsymbol{\theta}$. The first component can be easily computed as long as the loss function has an explicit derivative with respect to the neural network output and a forward pass through the neural network has been done to evaluate the derivative at this point. In most cases we even know a closed-form expression of the first component. Due to the structure of a neural network, the underlying function of the second component breaks down into further subfunctions. As a consequence, the chain rule can be applied consecutively from the output layer to the input layer.

In the following, we have a closer look into that derivative calculation of the second component for a fully connected neural network $f_{\boldsymbol{\theta}}$ with learnable parameters $\boldsymbol{\theta} = (W_1, \dots W_L, \boldsymbol{b_1}, \dots, \boldsymbol{b_L})$. The general idea also applies to all other layers such as convolutional layers. The calculation can be realized by general algorithmic differentiation or see [78] for, e.g., the identification of convolutional layers with fully connected layers. As shown in Equations (2.3) and (2.4) $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is of the form

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = f^L_{\boldsymbol{\theta_L}} \circ \dots \circ f^2_{\boldsymbol{\theta_2}} \circ f^1_{\boldsymbol{\theta_1}}(\boldsymbol{x}), \tag{2.76}$$

and the output of layer l is given by

$$f^l_{\boldsymbol{\theta_l}}(\boldsymbol{h_{l-1}}) = a_l(W_l \boldsymbol{h_{l-1}} + \boldsymbol{b_l}) = a_l(\boldsymbol{z_l}) = \boldsymbol{h_l} \quad \text{for } l = 1, \dots, L. \tag{2.77}$$

Backpropagating gradients starts at the output, layer L. We can iteratively calculate the derivative with respect to the previous layer. The derivative of a layer l with respect to the output $\boldsymbol{h_{l-1}}$ of the previous layer is given by:

$$\begin{aligned} \partial_{\boldsymbol{h_{l-1}}} f^l_{\boldsymbol{\theta}_l}(\boldsymbol{h_{l-1}})\big|_{\boldsymbol{h_{l-1}}=\tilde{\boldsymbol{h}}_{\boldsymbol{l-1}}} &= \operatorname{diag}\left(a_l'(\boldsymbol{z_l})\right) \cdot \partial_{\boldsymbol{h_{l-1}}} \boldsymbol{z_l} \\ &= \operatorname{diag}\left(a_l'(\boldsymbol{z_l})\right) \cdot \partial_{\boldsymbol{h_{l-1}}} (W_l \boldsymbol{h_{l-1}} + \boldsymbol{b_l}) \\ &= \operatorname{diag}\left(a_l'(\boldsymbol{z_l})\right) \cdot W_l \\ &=: \boldsymbol{\zeta_l} \end{aligned} \tag{2.78}$$

The derivative a' of the componentwise applied activation function $a : \mathbb{R} \to \mathbb{R}$ can be computed easily. As the activation function is applied element-wise in the forward pass, also the derivative can be applied element-wise. This is realized by a diagonal matrix $\operatorname{diag}\left(a_l'(\boldsymbol{z_l})\right)$ where the diagonal is given by $a_l'(\boldsymbol{z_l})$.

Due to the composition and the feedforward structure of the neural network the gradients with respect to $\boldsymbol{\theta_l}$ depend only on the gradients of later layers (layers j with $j > l$). The derivative of the neural network with respect to $\boldsymbol{\theta_l}$ at some fixed point $\tilde{\boldsymbol{\theta}}_{\boldsymbol{l}}$ can be

expressed by

$$\begin{aligned}
&\partial_{\boldsymbol{\theta_l}} f_{\boldsymbol{\theta}}(\boldsymbol{x})\big|_{\boldsymbol{\theta_l}=\tilde{\boldsymbol{\theta}}_{\boldsymbol{l}}} \\
&= \partial_{\boldsymbol{h_{L-1}}} f^L_{\tilde{\boldsymbol{\theta}}_{\boldsymbol{L}}}(\boldsymbol{h_{L-1}})\big|_{\boldsymbol{h_L}=f^L_{\tilde{\boldsymbol{\theta}}_{\boldsymbol{L}}}(\boldsymbol{h_{L-1}})} \cdots \partial_{\boldsymbol{h_l}} f^{l+1}_{\tilde{\boldsymbol{\theta}}_{\boldsymbol{l+1}}}(\boldsymbol{h_l})\big|_{\boldsymbol{h_{l+1}}=f^{l+1}_{\tilde{\boldsymbol{\theta}}_{\boldsymbol{l+1}}}(\boldsymbol{h_l})} \cdot \partial_{\boldsymbol{\theta_l}} f^l_{\boldsymbol{\theta_l}}(\boldsymbol{h_{l-1}})\big|_{\boldsymbol{\theta_l}=\tilde{\boldsymbol{\theta}}_{\boldsymbol{l}}} \\
&= \boldsymbol{\zeta_L} \cdots \boldsymbol{\zeta_{l+1}} \cdot \partial_{\boldsymbol{\theta_l}} f^l_{\boldsymbol{\theta_l}}(\boldsymbol{h_{l-1}})\big|_{\boldsymbol{\theta_l}=\tilde{\boldsymbol{\theta}}_{\boldsymbol{l}}}.
\end{aligned} \tag{2.79}$$

With this, it remains to derive a formula for the calculation of $\partial_{\boldsymbol{\theta_l}} f^l_{\boldsymbol{\theta_l}}\big|_{\boldsymbol{\theta_l}=\tilde{\boldsymbol{\theta}}_{\boldsymbol{l}}}$ which differs for W_l and $\boldsymbol{b_l}$. We get the derivative with respect to the bias term $\boldsymbol{b_l}$ by

$$\partial_{\boldsymbol{b_l}} f^l_{\tilde{\boldsymbol{\theta}}_{\boldsymbol{l}}}(h_{l-1})\big|_{\boldsymbol{b_l}=\tilde{\boldsymbol{b}}_{\boldsymbol{l}}} = \partial_{\boldsymbol{b_l}} \left(a(W_l \boldsymbol{h_{l-1}} + \boldsymbol{b_l})\right)\big|_{\boldsymbol{b_l}=\tilde{\boldsymbol{b}}_{\boldsymbol{l}}} = \operatorname{diag}\left(a'(W_l \boldsymbol{h_{l-1}} + \tilde{\boldsymbol{b}}_{\boldsymbol{l}})\right) \tag{2.80}$$

The derivatives with respect to W_l are given by[16]

$$\begin{aligned}
&\partial_{W_l} f^l_{(W_l,\boldsymbol{b_l})}(h_{l-1})\big|_{W_l=\tilde{W}_l} \\
&\quad = \partial_{W_l} \left(a_l(W_l \boldsymbol{h_{l-1}} + \boldsymbol{b_l})\right)\big|_{W_l=\tilde{W}_l} \\
&\quad = \partial_{\boldsymbol{z_l}} a_l(\boldsymbol{z_l})\big|_{\boldsymbol{z_l}=\tilde{\boldsymbol{z}}_{\boldsymbol{l}}} \cdot \partial_{W_l}(W_l \boldsymbol{h_{l-1}} + \boldsymbol{b_l})\big|_{W_l=\tilde{W}_l}.
\end{aligned} \tag{2.81}$$

The component k, i, j of the derivate with respect to W_l can be written down by using the notation $[\cdot]_k$ to denote the k-th entry in a vector or likewise $[\cdot]_{i,j}$ for the i, j-th entry in a matrix.

$$\begin{aligned}
\frac{\partial [f^l_{(W_l,\boldsymbol{b_l})}(h_{l-1})]_k}{\partial [W_l]_{i,j}} &= \frac{\partial [a\left(\sum_{m=1}^{m_{l-1}} [W_l]_{km} [\boldsymbol{h_{l-1}}]_m + [\boldsymbol{b_l}]_k\right)]_k}{\partial [W_l]_{i,j}} \\
&= a'([\boldsymbol{z_l}]_k) \cdot \frac{\partial}{\partial [W_l]_{i,j}} \left(\sum_{m=1}^{m_l - 1} [W_l]_{km} [\boldsymbol{h_{l-1}}]_m + [\boldsymbol{h_{l-1}}]_m \right) \\
&= a'([\boldsymbol{z_l}]_k) \cdot \sum_{m=1}^{m_{l-1}} \frac{\partial [W_l]_{km}}{\partial [W_l]_{i,j}} [\boldsymbol{h_{l-1}}]_m \\
&= a'([\boldsymbol{z_l}]_k) \cdot \sum_{m=1}^{m_{l-1}} \delta_{ki} \delta_{mj} [\boldsymbol{h_{l-1}}]_m \\
&= a'([\boldsymbol{z_l}]_k) \cdot \delta_{ki} [\boldsymbol{h_{l-1}}]_j
\end{aligned} \tag{2.82}$$

for $k, i = 1, \ldots, m_l$ and $j = 1, \ldots, m_{l-1}$ where δ_{mj} denotes the Kronecker delta with $\delta_{mj} = 1$ if $m = j$ and $d_{mj} = 0$ if $m \neq j$. These expressions can likewise be computed for tensor-valued input.

The advantage of backpropagation is that parts of the computation can be saved to memory and reused during the derivative calculation. This holds for the internal states

[16] We denote with $\partial_{\boldsymbol{z_l}} a_l(\boldsymbol{z_l})$ the derivative of the vector valued activation function cf. Section 2.1.1.1.

from the forward pass like $\boldsymbol{h_l}$, as well as redundant computations for the derivatives. As the loss maps to $\mathbb{R}$ and due to the associativity of matrix multiplication starting from the last layer, the storage and computational effort can be reduced by storing and calculating the resulting vector of the gradient-Jacobian-product instead of the matrix-matrix products of the Jacobian matrices between $\boldsymbol{\zeta_l}$, as those get extremely huge in neural networks with millions of parameters. With this, all derivatives can be computed with minimal recalculation effort and with one forward and one single backward pass through the network. Backpropagation can be seen as a special type of reverse accumulation mode of algorithmic differentiation [78]. As a consequence, deep learning libraries like PyTorch [201] use automatic differentiation engines (torch.autograd) to calculate derivatives of diverse neural network architectures.

2.2.5 Training Stabilization by Normalization and Regularization Layers

Despite the undeniable performance of neural networks in practice, the success is not guaranteed. Deep neural networks suffer from training instabilities [74, 92, 99, 110]. For convex-Lipschitz-bounded loss functions it can be shown that stochastic gradient descent converges to the minimum of the loss function [246, Cor. 14.12]. However, stochastic gradient descent suffers from training instabilities when the loss landscape is non-convex and ragged, which is likely to be observed in real world applications. If the loss is non-convex stochastic gradient descent is prone to iterate towards local minima which potentially are far from the optimum. As a consequence, the estimation error increases. The layered structure of neural networks allows to compute the gradients by calculating the chain rule (cf. Section 2.2.4.2). However, gradients may vanish for earlier layers if deeper layers have small gradients since the small gradients are multiplied and therefore the error signal decays exponentially. This results in limited feedback for earlier layers and a slow or stagnating training behavior [99]. Besides vanishing gradients, exploding gradients can be encountered if the propagated gradients are large. This leads to an unstable training since the weights of earlier layers change distinctly in each iteration. Several activation functions (cf. Section 2.1.1.1) were proposed to mitigate these problems [74]. Furthermore, it has been shown in [74] that the initialization of the parameters $\boldsymbol{\theta}$ is crucial for a stable training. A normalized initialization is suggested to hamper vanishing gradients. In Section 2.3.1.1, we introduce a neural network architecture which addresses the problem by incorporating so-called skip connections which allow bypassing gradient information to earlier layers [282]. In addition, deep neural networks especially with several fully connected layers are sensitive to changes in the input distribution. This holds for the input layer as well as for the hidden layers. The training progress is slow if the averaged input over the complete training set is shifted away from zero [147]. This limits the optimization process to straightly approach a minimum and leads to 'zigzag' behavior [147]. On input level this shift is mitigated

by **(input) normalization**, i.e., the empirical mean and standard deviation of the entire training dataset is used to linearly transform the data leading to a standardized input [110]. Training on standardized inputs lead to faster convergence and practice has demonstrated that it is crucial for the network performance in training and at inference time as well. The sensitivity to the shift in the distribution expands to the hidden layers in such a way that the input to a layer l changes due to the iteratively updated parameters of the previous layers $l-1,\dots,1$ (see Section 2.2.4.1 for details on the training process). To stabilize the training, **batch normalization** layers [110] which aim to standardize the pre-activation states of the feature maps in each training iteration, can be included into the neural network architecture. Similar to the input normalization the mean and variance is calculated to reparametrize the layer outputs such that their distribution has mean zero and a variance of one. However, the standardization is calculated with respect to the minibatch and since it is a neural network layer, it is taken into account during gradient calculation. As a consequence, the input distribution of the next layer does not change between the update steps, leading to a faster training which is less sensitive to the parameter initialization. In addition, it leads to much smoother loss landscapes which facilitates training [109]. As standardization constrains the layer capacity, two learnable parameters $\boldsymbol{\varsigma}, \boldsymbol{\varpi} \in \mathbb{R}^m$ are introduced to enable to revert the standardization if needed. Let $\boldsymbol{z} \in \mathbb{R}^{N_\mathcal{B} \times m}$ denote the batched pre-activation outputs of a layer and let[17]

$$\boldsymbol{\mu_\mathcal{B}} = \frac{1}{N_\mathcal{B}} \sum_{i=1}^{N_\mathcal{B}} \boldsymbol{x_i} \tag{2.83}$$

and

$$\boldsymbol{\sigma^2_\mathcal{B}} = \frac{1}{N_\mathcal{B}} \sum_{i=1}^{N_\mathcal{B}} (\boldsymbol{x_i} - \boldsymbol{\mu_\mathcal{B}})^2 \tag{2.84}$$

be the empirical mean and a biased empirical variance of the batch $\mathcal{B}$, respectively. By scaling and shifting we obtain the normalized input value

$$\hat{\boldsymbol{z}}_\boldsymbol{i} = \frac{\boldsymbol{z_i} - \boldsymbol{\mu_\mathcal{B}}}{\sqrt{\boldsymbol{\sigma^2_\mathcal{B}} + \boldsymbol{\varepsilon}}}, \tag{2.85}$$

where $\boldsymbol{\varepsilon}$ is a regularization value to prevent numerical instabilities of the fraction. With this we define the batch normalization

$$\mathrm{BN}^\mathcal{B}_{\boldsymbol{\varsigma},\boldsymbol{\varpi}} : \mathbb{R}^{N_\mathcal{B} \times m} \to \mathbb{R}^{N_\mathcal{B} \times m}, \boldsymbol{z_i} \mapsto \boldsymbol{\varsigma} \odot \hat{\boldsymbol{z}}_\boldsymbol{i} + \boldsymbol{\varpi}, \tag{2.86}$$

where $\boldsymbol{\varsigma} \odot \hat{\boldsymbol{z}}_\boldsymbol{i}$ denotes element-wise multiplication (Hadamard product). In order not to face a shift in distributions when switching from training to inference (evaluating or using the model after training with fixed weights) the moving average of the mean and

[17] The following operations are understood element-wise.

the standard deviation during training is calculated based on an update rate $\alpha \in \mathbb{R}$ [109]

$$\begin{aligned} \boldsymbol{\mu}_{\texttt{avg}} &= \boldsymbol{\mu}_{\texttt{avg}} + \alpha(\boldsymbol{\mu}_{\mathcal{B}} - \boldsymbol{\mu}_{\texttt{avg}}) \\ \boldsymbol{\sigma}_{\texttt{avg}} &= \boldsymbol{\sigma}_{\texttt{avg}} + \alpha \left(\sqrt{\frac{1}{N_{\mathcal{B}}} \sum_{i=1}^{N_{\mathcal{B}}} (\boldsymbol{x_i} - \boldsymbol{\mu}_{\mathcal{B}})^2 + \boldsymbol{\varepsilon}} - \boldsymbol{\sigma}_{\texttt{avg}} \right) \end{aligned} \tag{2.87}$$

and used to renormalize [109] the samples at inference time

$$\boldsymbol{y} = \boldsymbol{\varsigma} \frac{\boldsymbol{z} - \boldsymbol{\mu}_{\texttt{avg}}}{\boldsymbol{\sigma}_{\texttt{avg}}} + \boldsymbol{\varpi}. \tag{2.88}$$

Batch normalization can be applied to any affine transformation which is followed by a non-linear activation function [110]. For examples for convolutional neural networks batch normalization is done per channel, calculating the mean and the variance with respect to the spatial and batch dimension. This preserves the translation equivariance of features which is a key element of CNNs. However, batch normalization is less suited for small batches and batches containing dependent data. To this end, normalization techniques which do not depend on the batch dimension are proposed like [109] or layer normalization. In the context of transformer architectures, introduced in Section 2.3.1.2, layer normalization [14] is much more common. It estimates the normalization statistics with respect to the channel and spatial dimensions for each batch element independently.

Another challenge is that training on a limited dataset bears an increased risk of overfitting to the data if the model has oversized capacity. This phenomenon can be mitigated by incorporating specific regularization techniques into the training process. To balance the bias-complexity trade-off either the dataset can be enlarged or the model complexity reduced. The first is addressed by methods in Section 2.4 via e.g., data augmentation. Penalizing the complexity of the model can be achieved by adding a regularization term $\mathfrak{C}(\boldsymbol{\theta})$ to the loss which, e.g., constrains the neural network parameters $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^\tau, \tau \in \mathbb{N}$ [24].

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}; \gamma) = \mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}) + \gamma \mathfrak{C}(\boldsymbol{\theta}) \tag{2.89}$$

$\gamma \in \mathbb{R}_{\geq 0}$ weights the importance of the penalty term. The most common regularization is L^2 regularization also known as **weight decay**

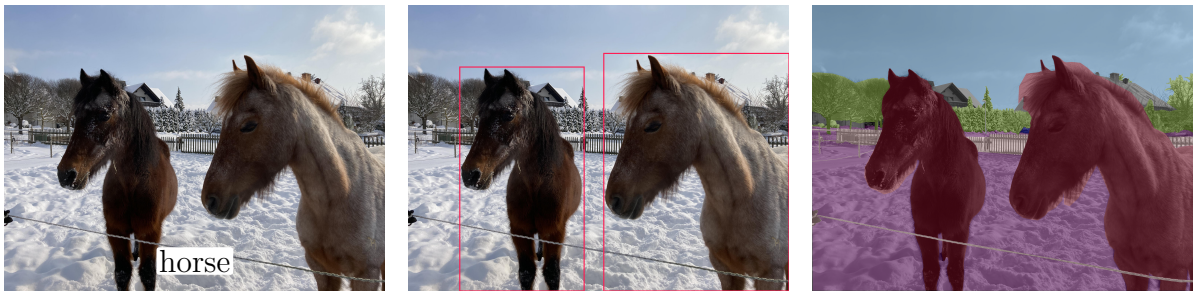
$$\mathfrak{C}(\boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{\theta}\|_2^2 = \frac{1}{2} \sum_{i=1}^{\tau} |\theta_i|^2. \tag{2.90}$$

The L^1 regularization defined by

$$\mathfrak{C}(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum_{i=1}^{\tau} |\theta_i| \tag{2.91}$$

is another often applied parameter regularization. Weight decay leads to a more smooth loss landscape which reduces overfitting by restricting the model to smoother functions and therefore stabilizes training [208, Sec. 9.1.2]. In contrast, L^1 regularization enforces sparsity on the parameters and thereby reduces the model complexity. Both regularization methods include a model bias to leverage the bias-variance trade-off introduced in Section 2.2.2.

Regularization can also be introduced on neural network layer basis. Removing ('dropping') randomly with a chance of $\mathfrak{p} \in (0, 1)$ a unit and its weighted connections to other units in the neural network during training is a stochastic regularization technique called **dropout** [259]. This leads to slimmed networks whose architecture changes randomly. Similar to the batch norm layer, dropout is only applied during training. By weighting the outputs by $\frac{1}{1-\mathfrak{p}}$ at training time, the layer resembles an identity mapping at inference time. This implementation differs slightly from the initial paper where the authors proposed to weight the weights of the neural network with $\mathfrak{p}$ during inference. Dropout has proven to support model performance as well as regularization by reducing overfitting [259].



(a) classification (b) object detection (c) semantic segmentation

Figure 2.9: The image shows different kinds of object recognition tasks by showing the ground truth label, box or mask respectively. Classification (a) aims to assign a class label to the entire image whereas object detection also adds a localization component by an enclosing bounding box (b). Semantic segmentation refines the localization on pixel-level leading to a semantic segmentation mask for annotation (c).

2.3 Deep Learning for Images

Deep learning is applied to a wide range of application areas. Particularly, in computer vision neural networks are the state-of-the-art method to extract high level information from images. A typical task solved by neural networks is object recognition. Recognition can be further split into multiple subtasks each with its own peculiarities. An example for three tasks with different granularity of object localization is shown in Figure 2.9. These tasks range from classifying an object or a scene (see Section 2.3.1), detecting (multiple) objects and their localization by an object-enclosing bounding box to classifying pixel-wise the content of the scene leading to semantic segmentation (see Section 2.3.2). Further information which can be extracted from images concerns the distance (in form of depth maps), pose or, e.g., sentiment, of an object of interest. A second branch of applications related to images is image synthesis which is formally not part of computer vision as its goal is to create images rather than extracting information from a sensor-based image. Nonetheless, deep neural networks are state-of-the-art in this category as well. Mostly those tasks are solved by deep generative models (see Chapter 3). Tasks include for example image generation, image enhancement like super resolution, image denoising, image-to-image translation and content aware image manipulation. In the next sections, we introduce the tasks addressed in this thesis in detail and present typical neural network architectures as well as evaluation metrics.

We start by fixing some notation. The input space $\mathcal{X}$ is set to the domain of images $\mathcal{X} = [0,1]^{m \times h \times w}$ represented by a pixel grid with $m \in \{1,3\}$ channels (1 for gray scale images and 3 for the red, green and blue channel of colored images), width $w \in \mathbb{N}$ and height $h \in \mathbb{N}$. For this reason, each pixel in the spatial dimension can be indexed by a tuple (i,j). We denote the pixel index set by

$$\mathcal{I} \coloneqq \{(i,j) | i = \{1,\dots,h\}, j = \{1,\dots,w\}\}\,. \tag{2.92}$$

2.3.1 Classification

The most basic of the tasks mentioned is image classification as it neglects the position of the object in an image. The aim is to categorize an image by assigning one of $C \in \mathbb{N}$ previously defined classes. Therefore, we define $\mathcal{Y} = \{1, \dots, C\}$ as the label space where each number is associated with a class, e.g., '1 = horse', '2 = zebra', '3 = apple'. A dataset $\mathcal{D}$ for a classification tasks consists of $N \in \mathbb{N}$ tuples $(\boldsymbol{x_n}, y_n) \in \mathcal{X} \times \mathcal{Y}$, $n = 1, \dots, N$ drawn from a data generating distribution $P_{X,Y}$. The task is to approximate the conditional probability $p_{Y|X}$ over the classes. The approximation is done by learning a neural network $f_{\boldsymbol{\theta}}$. By using a fully connected layer with softmax activation as output layer and setting the number of output neurons to the number of classes, a categorical probability distribution

$$p_{\boldsymbol{\theta}}(y_n|\boldsymbol{x_n}) = f_{\boldsymbol{\theta}}(\boldsymbol{x_n})_{y_n} \in (0,1) \text{ with } \sum_{c=1}^{C} f_{\boldsymbol{\theta}}(\boldsymbol{x})_c = 1 \tag{2.93}$$

is obtained. With this, the neural network

$$f_{\boldsymbol{\theta}} : \mathcal{X} \to (0,1)^C, \boldsymbol{x} \mapsto (f_{\boldsymbol{\theta}}(\boldsymbol{x})_1, \dots, f_{\boldsymbol{\theta}}(\boldsymbol{x})_C) \tag{2.94}$$

yields a so-called (softmax) confidence score per class. The prediction

$$\hat{c}(\boldsymbol{x}|\boldsymbol{\theta}) = \underset{c \in \{1,\dots,C\}}{\operatorname{argmax}} f_{\boldsymbol{\theta}}(\boldsymbol{x})_c \tag{2.95}$$

is set to the class with the highest probability for the given input. This assigns the discrete class label $\hat{c}(\boldsymbol{x}|\boldsymbol{\theta})$ to the input image $\boldsymbol{x}$.

To train the model, the empirical cross entropy loss with respect to our neural network $f_{\boldsymbol{\theta}}$ and the dataset $\mathcal{D}$

$$\mathcal{L}_{\mathcal{D}}^{\mathrm{CE}}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} \underbrace{\log(f_{\boldsymbol{\theta}}(\boldsymbol{x})_y)}_{=:L^{\mathrm{CE}}(f_{\boldsymbol{\theta}}(\boldsymbol{x}),\boldsymbol{y})}, \tag{2.96}$$

serves as training objective. This function is just the same as the negative log-likelihood introduced in Equation (2.47). For the practical implementation the label $y = c'$, $c' \in \{1, \dots, C\}$, is converted to a one-hot-encoded vector $\boldsymbol{y} \in \{0,1\}^C$ with $y_{c'} = 1$ for the true class and $y_c = 0$ for $c \neq c'$. Hence, we can formulate

$$L^{\mathrm{CE}}(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) = \boldsymbol{y}^{\mathrm{T}} \log(f_{\boldsymbol{\theta}}(\boldsymbol{x})). \tag{2.97}$$

With this method, images which display one (or multiple) object(s) from the same class dominating the presented scene can be classified if the class is given in the predefined label list $\mathcal{Y}$. Popular datasets for image classification tasks range from handwritten digits

(MNIST), where 10 classes $(0, \ldots, 9)$ need to be distinguished, to fine granular hierarchically categorized objects which aim to cover most of the nouns in the English languages (ImageNet [56]). Tiny images like the 32×32 grayscale images in the MNIST dataset can be classified with the help of MLPs [45]. Nevertheless, for more complex tasks more sophisticated architectures mostly consisting of at least some convolutional layers were needed. Nowadays, vision transformers (Section 2.3.1.2) dominate CNNs [311, 312] but CNNs catch up with accuracy if their capacity is enlarged (see e.g., ConvNeXt [162] and large scale foundation models like ImageIntern [286]).

The general setup of convolutional classification networks consists of a backbone, which extracts general features, and a classification head on top which uses this latent feature representation to estimate the categorical probability distribution over the classes. During feature extraction the backbone reduces the spatial dimension of the feature maps, e.g., through pooling operations, while traversing the depth of the neural network. With this, the information gets more and more compressed during the forward pass through the network. The importance of well suited features was highlighted by Girshick et al. [73]. Besides of convolutional layers, transformer layers in the backbone have shown success in solving classification tasks [161]. The classification head consists typically of one or very few fully connected layers where the last layer consists of C neurons. For CNNs the output needs to be flattened to yield a one dimensional vector as input for the classifier head. Alternatively, the backbone can return C feature maps and the classifier head solely performs a global average pooling (Section 2.1.2.2) on each of them to reduce the spatial dimension and compressing the image features. Additionally, the two concepts can be combined such that the global average pooling is used to vectorize the output of the backbone (which does not necessarily have C feature maps) and fully connected layer(s) are used for classification (see e.g., ResNet in Section 2.3.1.1).

The first neural network, showing that neural networks containing convolutional layers can achieve good results in image classification tasks, was LeNet, introduced by LeCun [146]. A depiction of LeNet as prototypical CNN structure for classification is shown in Figure 2.10. With the help of multi-GPU training and ReLU activations Krizhevsky et al. [142] demonstrated with AlexNet that much wider and a few layers deeper CNNs can be trained. AlexNet won the ImageNet Large Scale Visual Recognition Challenge in 2012 with great margin to the second-best submission[18]. Thereby depth was a key component of their success, and they speculated that more performant GPUs and bigger datasets further increase the performance [142]. Thereafter, CNN-based backbones became wider and deeper, e.g., VGG16/VGG19 [255] and GoogLeNet [263] but just stacking more layers does not necessarily improve but even degrade the network performance [92]. As introduced in Section 2.2.5 neural networks suffer from training instabilities which often get worse with deeper layers.

[18] `https://image-net.org`

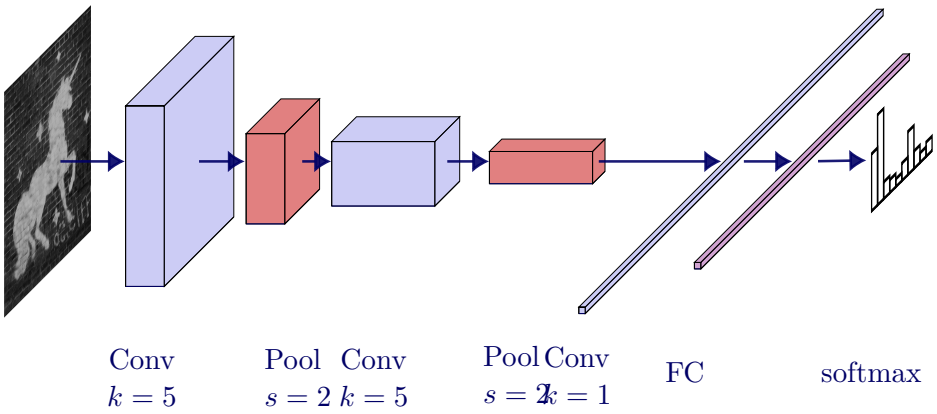


Figure 2.10: Prototypical illustration of a CNN for classification. Boxes denoted with 'Conv' represent the feature map of a convolution operation of a $k \times k$ kernel. Red boxes depict the feature map of pooling operations with a stride s. Convolution and pooling operations are followed by an either 1×1 convolution or a flattening to serve as an input for a fully connected layer and lastly a linear output layer predicting a probability distribution over the class IDs via a softmax activation.

2.3.1.1 Residual Neural Networks

Residual neural networks aim to overcome this phenomenon by learning residual connections rather than approximating a direct mapping in a hidden layer. The description of the method follows the original paper [92]. The idea behind residual neural networks is that if a shallow network can learn a task with a certain accuracy, a deeper network with the same performance can be constructed by stacking multiple identity mappings for all layers deeper than the shallow network. However, these neural networks show worse performance than their shallow counterparts. Assuming the function to approximate with the neural network $f_{\boldsymbol{\theta}}$ is of the form $f = H^1 \circ \cdots \circ H^L$, He et al. [92] proposed to better learn a residual mapping

$$F^i(\boldsymbol{x}) \coloneqq H^l(\boldsymbol{x}) - \boldsymbol{x}, \quad l \in \{1, \ldots, L\} \tag{2.98}$$

which makes it easier to learn an identity mapping. This is due to the fact that the weights of F only need to be pushed to zero. F is realized by a small neural network. With Equation (2.98) the mapping H^l can be replaced by

$$H^l(\boldsymbol{x}) = F^l(\boldsymbol{x}) + \boldsymbol{x} \tag{2.99}$$

The implementation of H^l is done by a **residual layer**[19] which consists of the neural network for F^l and a shortcut bypass, often called **skip connection**, in form of an

[19] In the original paper the subnetwork is denoted as residual block since it consists of multiple layers. To distinguish between blocks of multiple of those subnetworks and the subnetwork itself we refer to the subnetwork approximating $F^l(\boldsymbol{x}) + \boldsymbol{x}$ as residual layer.

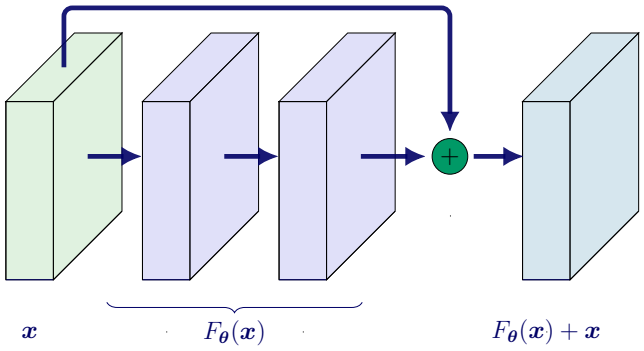


Figure 2.11: Schematic representation of a residual layer consisting of chained convolutional layers forming $F_{\boldsymbol{\theta}}(\boldsymbol{x})$ which are bypassed by an identity mapping x. Addition of the input and the residual forms the layer output $F_{\boldsymbol{\theta}}(\boldsymbol{x}) + \boldsymbol{x}$.

identity mapping which is added to the output $F^l(\boldsymbol{x})$. A depiction of the basic residual layer is shown in Figure 2.11.

Based on this concept, a family of network architectures was proposed. The overall architecture of **ResNets** draws inspiration from previous neural network architectures, such as VGG nets. There exist multiple versions (ResNet18, ResNet34, ResNet50, ResNet101 and ResNet152) of ResNets differing mostly in the number of layers and therefore in parameter size and depth. All versions have in common that the information of the ResNet input is primarily extracted and downsampled by a 7×7 strided convolution with stride $s = 2$ and 64 output channels. The spatial dimension of the input is further halved by a 3×3 pooling again with a stride of 2. The pooling is followed by 4 ResNet blocks with different amount of residual layers depending on the version. In the following we introduce the structure of ResNet18 which is depicted in Figure 2.12. We further investigate the differences with respect to ResNet50 as those two are used in this thesis. The layer constellation of the other versions can be found in [92, table 1]. In ResNet18 all four blocks are identically constructed. Each block consists of two residual layers, each built of two consecutive convolutional layers with 3×3 kernels bypassed by an identity mapping as shown in Figure 2.11. Starting with the second block, a downsampling is realized by using a convolution with stride $s = 2$ for the first convolution of each of the four blocks. Downsampling in the first block is already realized by the pooling. This leads to an overall spatial dimension reduction by a factor of 32. To realize skip connections between layers with reduced spatial dimension either zero padding or a linear projection can be used. The latter is given by a learnable projection matrix W such that the residual layer output turns into

$$y = F_{\boldsymbol{\theta}}(\boldsymbol{x}) + W\boldsymbol{x}. \tag{2.100}$$

Furthermore, starting with the second block, the number of channels stays constant within a block but is increased by a factor of 2 at the beginning of the next block. After

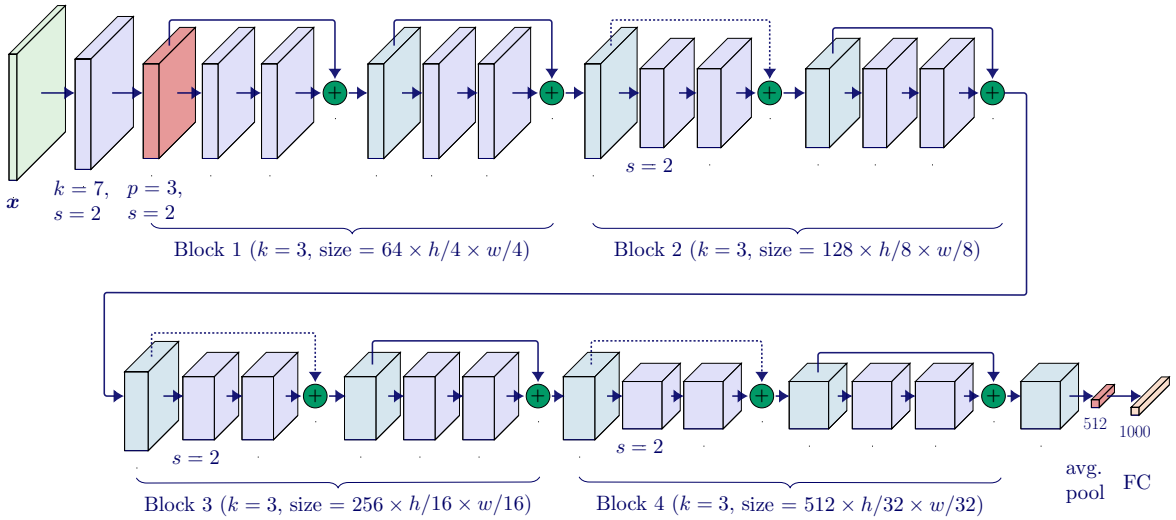


Figure 2.12: ResNet18, consists of 4 ResNet blocks each containing 2 residual layers with a 2-layer convolutional neural network $F_{\boldsymbol{\theta}}(\boldsymbol{x})$ to approximate the residual. Dashed skip connections connect feature maps with different spatial dimensions. This is realized by either zero padding or by a linear projection matrix W such that $F_{\boldsymbol{\theta}}(\boldsymbol{x}) + W\boldsymbol{x}$ is calculated. The size of the intermediate feature maps is constant within a block for ResNet18 and noted by 'size' in the figure. Red colored layers are pooling layers. Spatial dimension reduction is additionally achieved by strided convolutions, denoted by $s = 2$. Classification is conducted by a fully connected layer with exemplary 1,000 classes.

the four blocks in all ResNet versions, a global average pooling is used to compress and flatten the features. Lastly, it is followed by a fully connected layer of size C with softmax activation to yield a categorical probability distribution over the label space.

For deeper versions, e.g., ResNet50, a bottleneck structure is used to approximate the residual. Therefore, the channel dimension at first is reduced by a (1×1)-convolution, then features are calculated by a (3×3)-convolutional layer and at the end the channel size is upsampled again by a (1×1)-convolution. Starting in the first block the residual layer consists of a $(64 \times 1 \times 1)$-convolution, followed by a $(64 \times 3 \times 3)$-convolution and ending with a $(256 \times 1 \times 1)$-convolution. The residual layers are concatenated a varying number of times per block. Thereby, the channel amount is increased by a factor of 2 at the beginning of each of the four blocks but kept constant within the repetitions of the residual layers. For ResNet50 these building blocks are concatenated 3 times for the first block, 4 times for the second, 6 times for the third and 3 times for the forth. Together with the input and output layers, 50 layers are composed to form ResNet50.

The advantage of ResNet and its skip connections is that the parameter amount remains identical to a simple stacked CNN, but the training error does not suffer from deeper neural networks. Due to the skip connections ResNet there exists multiple paths of varying length between a neuron in the output layer and the input layer. Thus, ResNets

can be understood as an ensemble of multiple moderate deep neural networks instead of one extreme deep neural network [282]. To this end, the effective path length of the gradients is much shorter than the actual depth of the ResNet [282]. In addition, the complexity in terms of multiply-add operations is significantly reduced compared to, e.g., VGG which has even fewer layers. Nevertheless, ResNet architectures achieve a higher accuracy on ImageNet than VGG models [92]. With the help of skip connections it was possible for the first time to easily train deep neural networks with over 100 layers. Thereby the results of the authors confirm the hypothesis of 'the deeper, the better', as long as enough data is available and the problem has a certain complexity, cf. Section 2.2.2. Beyond classification, the ResNet architecture has proven to serve as a universal backbone in multiple vision tasks such as object detection [158] or semantic segmentation [37].

2.3.1.2 Transformers

The second model type which has shown superior performance in image classification in the recent years bases on transformer blocks. The use of transformer models for image classification is motivated by their success in natural language processing, e.g., Bert [57], GPT series [29, 196, 210, 211], initially introduced by Vaswani et al. in 2017 [281]. To allow for inputs of variable length, e.g., a text in natural language processing, a sequence of so-called tokens is used instead. Tokens are the smallest considered unit of a text or an input in general. The granularity of the tokenization is defined by a tokenization function. As an example, a sentence can be decomposed on word or character level leading to an input sequence of words and punctuations in the first case [316].

In an encoding step the information from the input sequence $x_1, \ldots, x_N$ is compressed into a continuous lower dimensional so-called embedding or **latent space** $\mathbb{R}^t$ and enriched by a positional encoding to encode the position of the token within the input sequence. A learned convex combination of the input token embedding vectors is calculated based on the similarity between the individual token embedding pairs. This concept is called attention mechanism and is explained in detail in the next paragraph as it is a key component of transformer models. The encoder consists of multiple of these attention-based blocks. The output of a transformer is generated sequentially by a so-called decoder which also consists of transformer blocks operating on the t-dimensional embedding space. In more detail, the encoded representation as well as previously generated outputs serve as input and the attention-weighted convex combination thereof is calculated. This is fed to a fully connected layer with softmax activation to generate output probabilities over, e.g., a dictionary $\mathcal{Y}$, to predict the next token, e.g., a word in a sentence. This aims to consecutively generate an output sequence $y_1, \ldots, y'_m$ with variable length m' and $y_i \in \mathcal{Y}$.

The idea of attention can be made more tangible by the following association. Think of a database which stores (key, value)-pairs, e.g., an online telephone book with entries

(name, number). With the help of a query, e.g., 'Peter', we can ask for the number of a specific person by its name. That means, that if the query meets a key we get the value, the number, returned [316]. In the example the key and values are fixed, and the attention is drawn with respect to whether a key exists or not. In the case of transformers the attention and thereby the interaction between input tokens is captured by learned **self-attentions**. The following detailed description is based on [281], but we follow the notation of [313]. Given N token embedding vectors $\boldsymbol{z_i} \in \mathbb{R}^t$ which are stacked column-wise to a matrix $Z = (\boldsymbol{z_1}, \dots, \boldsymbol{z_N}) \in \mathbb{R}^{t \times N}$ the key, value and query can be defined by learnable weight matrices $W_K \in \mathbb{R}^{d \times t}$, $W_V \in \mathbb{R}^{d_V \times t}$ and $W_Q \in \mathbb{R}^{d \times t}$ such that

$$\begin{aligned} W_K Z &= (W_K \boldsymbol{z_1}, \dots, W_K \boldsymbol{z_N}) \in \mathbb{R}^{d \times N} && \triangleleft \text{ key} \\ W_V Z &= (W_V \boldsymbol{z_1}, \dots, W_V \boldsymbol{z_N}) \in \mathbb{R}^{d_V \times N} && \triangleleft \text{ value} \\ W_Q Z &= (W_Q \boldsymbol{z_1}, \dots, W_Q \boldsymbol{z_N}) \in \mathbb{R}^{d \times N} && \triangleleft \text{ query.} \end{aligned} \tag{2.101}$$

The projection dimension $d \in \mathbb{N}$ thereby can be chosen arbitrarily and is part of the architecture design choice. The attention mechanism is based on three steps. First the similarity between $W_Q Z$ and $W_K Z$ is calculated. This can be done additively [15] or multiplicatively via a (scaled) dot-product [165,281] between the affine translated query and key vectors. The latter is defined by

$$\texttt{sim}(W_Q Z, W_K Z) = \frac{(W_Q Z)^{\mathrm{T}} W_K Z}{\sqrt{d}} \in \mathbb{R}^{N \times N}. \tag{2.102}$$

It is followed by a column-wise softmax activation to yield a predictive distribution

$$\alpha_{W_Q, W_K}(Z) = a_{\texttt{softmax}}\left(\frac{(W_Q Z)^{\mathrm{T}} W_K Z}{\sqrt{d}}\right). \tag{2.103}$$

The scaling $1/\sqrt{d}$ is used to control the size of the dot-product to yield meaningful gradients and therefore stabilizes the training process. Furthermore, the dot-product can be implemented much more efficient compared to additive attention [281].

As $W_K \neq W_Q$ in general, the attention weights of two embedding vectors given by $\alpha_{W_Q, W_K}(Z)_{i,j}$ and $\alpha_{W_Q, W_K}(Z)_{j,i}$ for $i \neq j$ are not necessarily symmetric such that the impact of a token can be unidirectional. This weighting is then used to calculate a convex combination of the embedded value vectors $V = W_V Z$:

$$\mathrm{Attn}_{W_Q, W_K, W_V}(Z) = V \cdot \alpha_{W_Q, W_K}(Z) = W_V Z \cdot a_{\texttt{softmax}}\left(\frac{(W_Q Z)^{\mathrm{T}} W_K Z}{\sqrt{d}}\right) \in \mathbb{R}^{d_V \times N}.$$

This form of attention is called self-attention as the key, query and value all base on the same input and its embedding Z.

To increase the capacity of the transformer, **multi-head self attention** modules are

used. Multiple different projection matrix triples $\{(W_K^i,\ W_Q^i, W_V^i)\}_{i=1}^{m_h}$ are learned in parallel and fused as a convex combination to yield the final result. Often the projection dimension are chosen to be equal, i.e., $d_v = d$, to compute all heads in parallel [183]. To not increase the computational complexity, the projection dimension d is divided by $m_h \in \mathbb{N}$, the number of parallel computed so-called attention heads. Thus, m_h projections in the d_h-dimensional space with $d_h = \frac{d}{m_h}$ are learned. This permits the neural network to learn from different representations and different positions of the input tokens. Accompanied by a skip connection as known from ResNet (cf. Section 2.3.1.1) and corresponding weight matrices $\{W_O^i\}_{i=1}^{m_h}$ with $W_O^i \in \mathbb{R}^{t\times d_h}$ for weighting each attention head and mapping back to the embedding dimension t, the multi-head self attention module is formally defined by

$$\mathrm{MHSA}_{\boldsymbol{\theta}}(Z) = Z + \sum_{i=1}^{m_h} W_O^i \cdot \mathrm{Attn}_{W_Q^i, W_K^i, W_V^i}(Z) \in \mathbb{R}^{t\times N}. \tag{2.104}$$

Thereby all learnable weight matrices are grouped in $\boldsymbol{\theta}$ for the sake of readability.

With this the complete **transformer block** which is a mapping $\mathcal{T}_{\boldsymbol{\theta}} : \mathbb{R}^{t\times N} \to \mathbb{R}^{t\times N}$ can be formulated. It consists of two sub-layers each bypassed with a skip connection, known from the ResNet architecture, and followed by a layer normalization (cf. Section 2.2.5). The first submodule is the above introduced multi-head self attention module. The second is a 2-layer MLP with m_1 hidden nodes and a ReLU activation. Let $W_1 \in \mathbb{R}^{m_1\times t}$, $W_2 \in \mathbb{R}^{t\times m_1}$ be the learnable weight matrices of the MLP and $\boldsymbol{b_1} \in \mathbb{R}^{m_1}$, $\boldsymbol{b_2} \in \mathbb{R}^t$ the biases stacked N times to form a matrix $B_1 = (\boldsymbol{b_1}, \dots, \boldsymbol{b_1}) \in \mathbb{R}^{m_1\times N}$ and $B_2 = (\boldsymbol{b_2}, \dots, \boldsymbol{b_2}) \in \mathbb{R}^{t\times N}$. With this, we define the general transformer block

$$\mathcal{T}_{\boldsymbol{\theta}}^{m_h, d_h, m_1}(Z) \coloneqq \mathrm{MHSA}_{\boldsymbol{\theta}}(Z) + W_2 \cdot a_{\texttt{ReLU}}\left(W_1 \cdot \mathrm{MHSA}_{\boldsymbol{\theta}}(Z) + B_1\right) + B_2 \in \mathbb{R}^{t\times N}. \tag{2.105}$$

2.3.1.3 Vision Transformers

When applying transformers to vision tasks, the tokenization is crucial. Pixel-wise attention, which seems the straight forward transfer, does not scale well for images with a reasonable resolution as the compute complexity is quadratic in the amount of pixels. This problem is mitigated by using image patches as tokens [58]. An image $\boldsymbol{x} \in \mathbb{R}^{m\times h\times w}$ is divided in quadratic patches of $m \times p \times p$ pixels with $p < w, h$. Those patches are flattened to one-dimensional vectors. With a linear projection to a t-dimensional embedding space and equipped with a positional encoding to encode the position of the patch in the original image, the flattened vectors serve as input sequence of length $N = h \cdot w/p^2$ for the transformer block. To solve a classification problem the sequence is enlarged by an additional token, the class-token, whose embedding is learned as well and its aim is to encode all information such that the image label can be predicted. In contrast to the transformer block introduced in the last paragraph, vision transformers (ViT [58]) use a GeLU activation function (cf. Equation (2.11)) in the MLP submodule.

Furthermore, only the encoder part of the transformer is used. The classification is realized by a classifier head in form of an MLP on top of the class-token output.

Vision transformers need plenty of data to perform comparable to CNNs. Dosovitskiy et al. demonstrated in [58] that transformers surpass CNN performance for training datasets with 14 to 300 million images. On ImageNet (1.2 million training images) ResNet-based CNNs still were better in 2020 [136]. This was changed with data-efficient image transformers (DeiT) [275] and tokens-to-token vision transformers (T2T-ViT) [311]. One reason for the need of data is the lack of inductive bias which CNNs have due to their ability of preserving spatial relations. Vision transformers need to completely learn spatial relations as the positional encoding does only encode the position of the patches but not a two-dimensional pixel arrangement [58]. Another difficulty of transformers is its sensitivity to hyperparameters including initialization [275]. This problem is for example addressed by adding regularization techniques (cf. Section 2.2.5) and data augmentation methods (cf. Section 2.4.1). In [187], Naseer et al. study the intriguing properties of vision transformers and compare them against CNNs. One key difference between CNNs and vision transformers is the receptive field. Since the kernel tensors applied to an input share the weights across all positions in the image, CNNs are said to be content-independent [187]. In contrast, vision transformers have a content-dependent dynamic receptive field due to the global patch-wise self-attention [187].

2.3.1.4 Evaluation Metrics

The loss allows to optimize the parameters, but it lacks interpretability with respect to the test samples. To measure the performance of a classifier (independent of its architecture) most commonly the accuracy is evaluated. It follows the most native way to measure the performance of a classification task by counting the amount of correctly classified data samples and setting it in relation to the total amount of evaluated data points. Formally, the accuracy of a neural network $f_{\boldsymbol{\theta}}$ over a dataset $\mathcal{D} = \{\boldsymbol{x_i}, y_i\}_{i=1}^{N}$ with predictions $\hat{c}(\boldsymbol{x}|\boldsymbol{\theta})$ is given by

$$\mathbf{acc} = \frac{1}{N} \sum_{i=1}^{N} \delta_{\hat{c}(\boldsymbol{x_i}|\boldsymbol{\theta}),y_i}. \tag{2.106}$$

A **confusion matrix** gives insight into the mis-classifications. In a confusion matrix the rows refer to the true class labels and the columns refer to the prediction. Using a $C \times C$ matrix, we can monitor which class the classifier predicted relative to the true class. A perfectly separating classifier would result in a diagonal matrix. Misclassifications are noted on off-diagonal elements. When the dataset is unbalanced the accuracy might be a misleading measure for the classifier performance. The classifier might learn to always predict the dominating class. This results in a reasonably good accuracy, but the performance of the underrepresented classes is catastrophic. When considering a binary

classification task alternative evaluation methods exist. In binary classification the dataset consists of data points which have either label 1 (positive / Pos) or 0 (negative / Neg)

$$\begin{aligned} \text{Pos} &= |\{(\boldsymbol{x}, y) \in \mathcal{D} : y = 1\}| \\ \text{Neg} &= |\{(\boldsymbol{x}, y) \in \mathcal{D} : y = 0\}|. \end{aligned} \tag{2.107}$$

There are four potential outcomes for the prediction of the classifier. The classifier can predict a data point correctly as positive (TP), falsely as positive (FP), falsely as negative (FN) or correctly as negative (TN). The resulting sets for a neural network $f_{\boldsymbol{\theta}}$ with prediction $\hat{c}(\boldsymbol{x}|\boldsymbol{\theta})$ are given by

$$\begin{aligned} \text{TP} &= |\{(\boldsymbol{x}, 1) \in \mathcal{D} : \hat{c}(\boldsymbol{x}|\boldsymbol{\theta}) = 1\}| \\ \text{FN} &= |\{(\boldsymbol{x}, 1) \in \mathcal{D} : \hat{c}(\boldsymbol{x}|\boldsymbol{\theta}) = 0\}| \\ \text{FP} &= |\{(\boldsymbol{x}, 0) \in \mathcal{D} : \hat{c}(\boldsymbol{x}|\boldsymbol{\theta}) = 1\}| \\ \text{TN} &= |\{(\boldsymbol{x}, 0) \in \mathcal{D} : \hat{c}(\boldsymbol{x}|\boldsymbol{\theta}) = 0\}|. \end{aligned} \tag{2.108}$$

The accuracy is then equivalently defined by

$$\textbf{acc} = \frac{\text{TP} + \text{TN}}{\text{Pos} + \text{Neg}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{TN} + \text{FP}}. \tag{2.109}$$

Task specific metrics like

$$\textbf{specificity}\left(\frac{\text{TN}}{\text{Neg}}\right),\ \textbf{precision}\left(\frac{\text{TP}}{\text{TP} + \text{FP}}\right),\ \textbf{sensitivity/recall}\left(\frac{\text{TP}}{\text{Pos}}\right) \tag{2.110}$$

and/or a look at the confusion matrix can be evaluated when the dataset is imbalanced.

2.3.2 Semantic Segmentation

Assigning only one category to an entire image is insufficient if multiple objects of different classes are visible or a broader understanding of the entire scene is important including the location of the objects. As mentioned earlier, this is particularly important in the context of perception in autonomous driving. Semantic segmentation aims to segment the image into regions which share the same semantics. Specifically, each pixel in an image is assigned the category of the object or class it belongs to. As for classification, the label categories are predefined as $\{1, \ldots, C\}$. Given an input image $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^{c \times h \times w}$, a semantic segmentation mask

$$y \in \mathcal{Y} \coloneqq \{1, \ldots, C\}^{h \times w} \quad \text{with } y_{i,j} \in \{1, \ldots, C\} \tag{2.111}$$

serves as label for $\boldsymbol{x}$. The problem can be understood as a pixel-wise classification problem. As a consequence, we obtain a function usually represented by a neural network

$$f_{\boldsymbol{\theta}} : \mathcal{X} \to [0,1]^{C \times h \times w} \text{ with } f_{\boldsymbol{\theta}}^{i,j}(\boldsymbol{x}) \in [0,1]^C \tag{2.112}$$

describing the pixel-wise conditional probability distribution. Per pixel $(i,j) \in \mathcal{I}$ the most probable class can be determined by applying the argmax-function to $f_{\boldsymbol{\theta}}^{i,j}$, yielding the prediction

$$\hat{c}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) = \underset{c \in \{1,\dots,C\}}{\operatorname{argmax}} (f_{\boldsymbol{\theta}}^{i,j}(\boldsymbol{x})_c). \tag{2.113}$$

With the help of the pixel-wise class predictions, segments with the same semantics, e.g., object type, are formed. To properly define a segment in an image, we associate the image with a pixel adjacency graph $\mathcal{G} = (\mathcal{I}, \mathcal{E})$ where each node represents a pixel $(i,j) \in \mathcal{I}$. The edges are defined by the neighboring relation of pixels and their class label. If neighboring pixels are assigned the same class, they are connected with an edge. The set of neighbors for non-border pixels is defined by

$$\mathcal{N}_8(i,j) \coloneqq \{(i, j \pm 1)\} \cup \{(i \pm 1, j)\} \cup \{(i \pm 1, j \pm 1)\}. \tag{2.114}$$

For the border pixels, all index combinations which lead to indices not in $\mathcal{I}$ are excluded from the set. We say, two vertices (i,j) and (k,l) are connected by an edge $e_{(i,j),(k,l)}$ if

$$(k,l) \in \mathcal{N}_8((i,j)) \text{ and } \hat{c}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) = \hat{c}_{k,l}(\boldsymbol{x}|\boldsymbol{\theta}). \tag{2.115}$$

This graph can be split into connected components, i.e., non-connected simple subgraphs where each pixel has the same class label. A segment in an image can then be defined as a connected component in the graph.

As loss function serves the empirical pixel-wise cross entropy

$$\mathcal{L}_{\mathcal{D}}^{\text{CE-pix}}(\boldsymbol{\theta}) = -\frac{1}{N} \cdot \frac{1}{h \cdot w} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} \sum_{(i,j) \in \mathcal{I}} L^{\text{CE}}(f_{\boldsymbol{\theta}}^{i,j}(\boldsymbol{x}), y_{i,j}) \tag{2.116}$$

which is the cross entropy, known from the classification setup (cf. Equation (2.96)), averaged over each pixel. Even though it seems to be a straight forward extension of the classification task, semantic segmentation is much more complex and comes with its own peculiarities like computational complexity and fine-grained information extraction. In a classification dataset, it is quite easy to ensure that each class is equally often represented in the dataset by construction. For semantic segmentation this is harder to achieve as not all classes need to occur equally often (in terms of pixel count). For example, in urban traffic scenes, the roads, buildings and the sky are typically the most prominent elements, while e.g., pedestrians and bicycles occupy a relatively small portion of the scene in terms of pixels. An imbalance in the pixel amount per class results in a loss biased towards the dominating classes as the average over the pixels is calculated. To

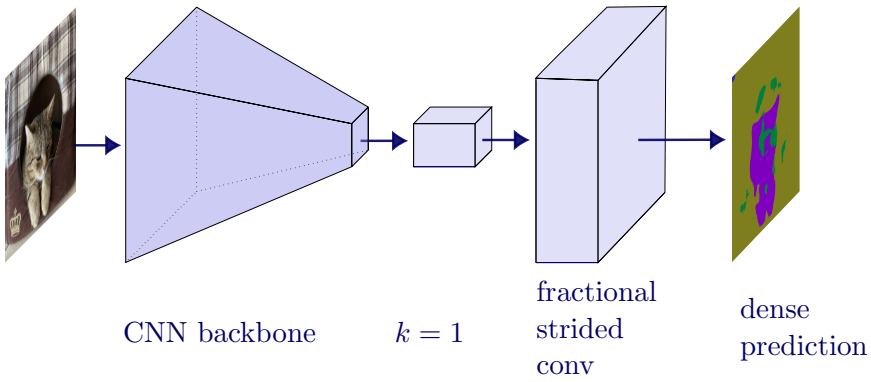


Figure 2.13: The schematic illustration shows a FCN consisting of a CNN backbone with convolution and pooling layers. The output is mapped to a tensor of C channels via a (1×1)-convolution and upsampled with a strided convolution with learnable weights to recover the spatial input resolution. By applying a softmax activation in the output layer and calculating the maximal argument, we achieve a dense prediction map.

foster the network to also learn the underrepresented classes, a class weighting based on the pixel-wise class distribution of the training data label masks is often included into the loss.

Similar to the classification architecture, neural networks for semantic segmentation follow the concept of a feature extracting backbone followed by a task specific head which estimates a pixel-wise class probability in this case. Both convolutional and transformer-based architectures have been found to be effective at tackling semantic segmentation [37, 300]. We introduce both architecture types in the following. CNNs as introduced in Section 2.3.1 have proven to be useful for classification tasks. But due to the fully connected layers in the classification heads of the presented architectures, they cannot cope with varying input resolutions and yield a vector-valued output whereas a mask of input size is demanded for semantic segmentation.

2.3.2.1 Fully Convolutional Networks (FCNs)

Fully convolutional networks [163] aim to overcome these limitations. Features are extracted as it was done for classification with a CNN backbone. To achieve a dense feature map with an estimated probability distribution over C classes, all fully connected layers in the classification head are replaced by convolutions and for the last layer the number of feature map channels is transformed to the number of classes. This can be realized by (1×1)-convolutions (cf. Section 2.1.2.1). At the end, upsampling is done to recover the input resolution. In Figure 2.13 a schematic illustration of a FCN is depicted. An important factor of success of convolutional architectures is the information compression due to resolution reduction while moving to deeper layers. Using backbones which incorporate resolution reduction layers like pooling or strided convolutions lead to a final

output feature map with subsampled resolution compared to the input image. As a dense feature map is required for a pixel-wise classification, upsampling is performed to restore the original input resolution leading to a per-pixel class distribution. Upsampling can be non-parametric for example by nearest neighbor or bilinear upsampling. This is used in the output layer to allow for faster training [36]. Alternatively, upsampling can be learned via fractionally-strided convolutions (cf. Equation (2.24)) which is often done in earlier layers. With this, an end-to-end training is possible and an input of any spatial size can be fed into the FCN. Furthermore, pre-trained classifier backbones can be used to limit the training effort to fine-tuning the segmentation head. We explain the concept of pre-training in more depth in Section 2.4.3.1. A variety of backbones (in varying sizes depending on the complexity of the task) often pre-trained on ImageNet can be used for FCNs [163].

Despite fractionally-strided convolutions, FCNs with subsampling operations lead to fuzzy object boundaries when upsampling to the input resolution. On the other hand pooling is a key component to compress information and enlarges the effective field of view. Adding skip connections (cf. Section 2.3.1.1) from earlier layers with higher resolution to the upsampled output layers allows fusing information on a fine and coarse level [163]. The idea of using skip connections between different feature-map resolution was extended by the U-Net architecture [223]. Each resolution step in the downsampling path has a skip connection to the corresponding feature-map in the upsampling path. A subsequent convolutional layer learns to combine the information from the earlier layer and the upsampled layer. Firstly, information is compressed step-by-step using convolutional layers and pooling operations in a downsampling path. It follows a symmetric upsampling path where the pooling operations are replaced with fractionally-strided convolutions. The convolution layers in the upsampling path help to learn context information relevant for later layers. This path expands the feature-maps symmetrically to the downsampling, leading to an architecture which resembles a U-shape.

2.3.2.2 Deeplab

A computationally effective alternative to achieve a dense and accurate feature map prediction is proposed by the deeplab architecture family. Deeplab is a special type of FCN, following the concept of using a classifier backbone, where all fully connected layers are replaced by equivalent convolutions, followed by a modified semantic segmentation head. It evolved from version 1 [35] over 2 [36], and 3 [37] to version 3+ [39]. In the following we refer to the version by deeplabv1 to deeplabv3+. The architecture design aims at a dense feature prediction, taking into account that objects occur on multiple scales and adds a method to improve on the localization of objects which is neglected in classification. The dense feature prediction is achieved by replacing the downsampling and convolutions in later layers by atrous convolutions with different rates (Equation (2.22)). As introduced in Section 2.1.2, atrous convolutions can enlarge the effective field of view without increasing the number of parameters to learn. In addition, the different rates

can be used to explicitly select the resolution at which features are computed. The atrous convolutions are supplemented by bilinear upsampling to achieve the resolution of the final feature activation map. To further refine the prediction boundaries and the localization of the objects, deeplabv1 and v2 include a post-processing with a fully connected conditional random field (CRF) [140]. Conditional random fields model a conditional prediction by a graphical model of random variables. This graphical model enables to take context of neighbors defined by the graph structure into account. As the prediction of a pixel is correlated to its neighbors, CRFs based on the output of the neural network can learn to refine the predicted segmentation mask. However, it has been shown that conditional random fields as post-processing is obsolete from version 3 on. One reason for that might be that in contrast to version 1 and 2 in version 3 the ground truth is not subsampled during training and therefore finer feedback is given by the gradients. To capture features of objects or image information at different scales, feature maps can be sub- or upsampled at different scales and processed in parallel, which is known as spatial pyramid pooling [90]. Starting with deeplabv2 **atrous spatial pyramid pooling** (ASPP), which calculates multiple atrous convolutions with different rates in parallel to probe the image with different effective field of views, was introduced. This approach was improved in deeplabv3 by adding batch normalization (cf. Section 2.2.5) and by incorporating a global view on the image (features) by a global average pooling (cf. Section 2.1.2.2) alongside atrous (3×3)-convolutions at three different rates and a (1×1)-convolution (cf. Section 2.1.2.1) for a cumulative view on the different feature maps per pixel. The authors suggest different rate combinations depending on the backbone and size of the feature map. We use the rate $r = 12$, $r = 24$, and $r = 36$ for the three stages as implemented in torchvision. These 5 feature maps are computed in parallel on the same feature map, concatenated afterwards and followed by an additional (1×1)-convolution and a dropout layer (Section 2.2.5) with a dropout probability of 0.5. To achieve a dense feature map with class activations, the ASPP module is fed to a (3×3)-convolution to learn feature combinations from different scales followed by a (1×1)-convolution with the width of the number of classes. Except for the very last (1×1)-convolution all layers are augmented with a batch normalization layer (Section 2.2.5) between the convolution and the ReLU activation layer. Together that defines the deeplabv3 head[20]. The full architecture of deeplabv3 with a ResNet backbone is sketched in Figure 2.14.

Deeplabv3+ extended the segmentation head of deeplabv3 by a decoder which uses low-level features from the backbone and the output of the deeplabv3 upsampled with a factor of 4 instead of 8. This leads to a typical encoder-decoder structure. The decoder refines the compressed features of the encoder by a few convolutional layers and lastly upsamples bilinearly the refined feature map to the full image resolution. In this thesis the latter two versions of deeplab are extensively used with a ResNet backbone in various sizes and with atrous convolutions in the last two blocks.

[20]The description follows the implementation [38] of the authors which deviates slightly from their description in the paper, e.g., the dropout layer is omitted in the paper.

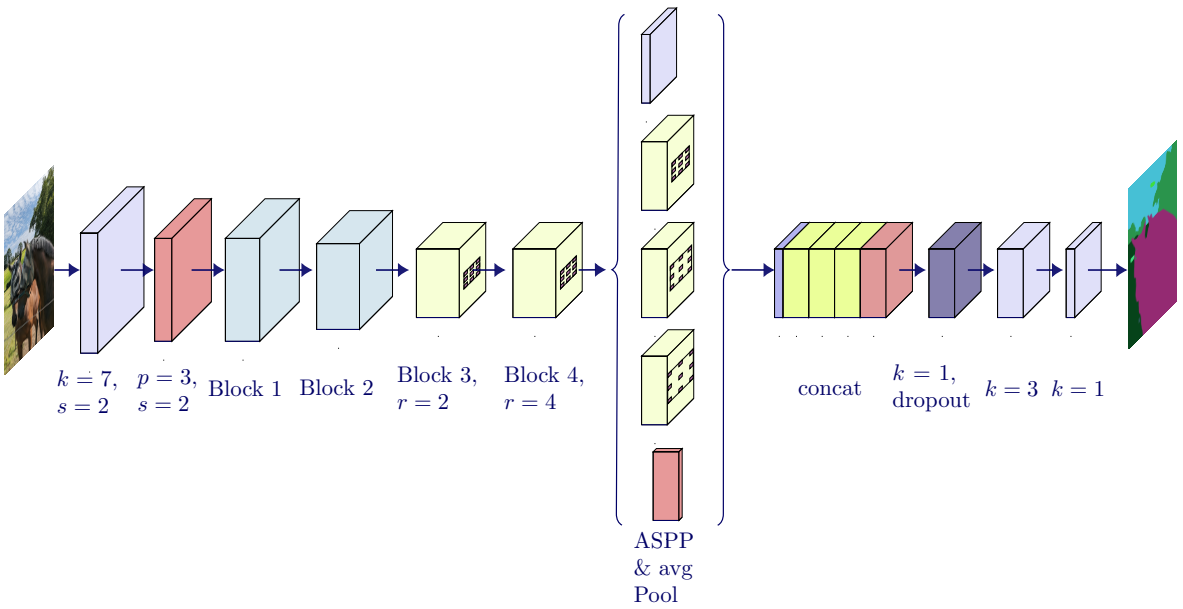


Figure 2.14: Deeplabv3 architecture with ResNet backbone. In the last two ResNet blocks, the downsampling is replaced by atrous convolution (yellow) with rate $r = 2$ and $r = 4$ respectively. The deeplabv3 head consists of an atrous spatial pyramid pooling (ASPP) module with global average pooling to capture the image features (in braces). All 5 operations are calculated in parallel. The results are concatenated and feature combinations are learned by a (1×1)-convolution with dropout (violet) and a regular 3×3-convolution. The last layer projects the extracted features to a probability distribution over the C classes per pixel. Via upsampling a dense feature map is achieved.

2.3.2.3 Vision Transformers for Semantic Segmentation

Besides convolutional semantic segmentation models, also transformer models exist for semantic segmentation. Even though vision transformers like ViTs [58] have proven success for image classification, they are not directly suitable as general purpose backbone for other computer vision tasks. This is partly due to their low resolution feature map conditioned by the patch size (16×16 for ViT). Dividing the image into a regular grid of patches enables vision transformers to reduce the computational complexity compared to the number of pixels (cf. Section 2.3.1.3). However, their computational complexity is quadratic in the number of patches and due to a fixed patch size dependent on the image size. To overcome these limitations, Swin transformers [161] follow a hierarchical setup combined with a shifted window approach. For the shifted window approach the image is divided into a fixed number of windows. A window contains $M \times M$ patches. In contrast to ViTs where the self-attention is calculated with respect to all patches, self attention is computed locally, meaning only within one window. This leads to linear compute time in the input image size. To also allow for cross window information flow, the windows are shifted from transformer block to transformer block

by half of the window size in all spatial dimensions. For self-attention, shifting the windows is computationally more efficient than a sliding window approach as known from CNNs. This is due to the fact that memory access cannot be handled in a similarly efficient way [161]. Furthermore, the self attention is extended with a relative position bias [212] replacing the deterministic positional encoding formerly added to the token embeddings. In addition, Swin transformers profit from hierarchically merging neighboring patch tokens and projecting them to a lower dimension when going deeper in the neural network. With this they can mimic the different resolution level of typical CNN backbones like ResNet. As a consequence, Swin transformers are the first transformers which can replace ResNet backbones while being computationally efficient. This makes swin transformers suitable as general purpose backbone for different computer vision tasks. A fully transformer-based model, meaning that backbone and decoder consist of transformer blocks, for semantic segmentation was introduced with Segmenter [260]. In contrast to CNN-based models, transformers have the capability to learn global context in the first layers but lack the inductive bias of the local arrangement of image pixels. To overcome the immense need of labeled data, which is rare in semantic segmentation as annotation on pixel level is needed, classification tasks are used to pre-train the transformer model such that fine-tuning to the semantic segmentation task can be achieved with a "moderate sized dataset" [260]. Additionally, transformers with a small number of parameters can be used. For example, SegFormer [300] was proposed as a simple and efficient transformer architecture for semantic segmentation. SegFormer is a hierarchical transformer with multiscale feature output which is aggregated by a lightweight multilayer perceptron decoder allowing to combine global and local attention. The design of the transformer encoder was inspired by ResNet. It consists of four blocks each containing a number of transformer blocks with an increasing channel dimension and reduced spatial dimension in each of the four blocks while moving deeper in the neural network. By merging overlapping patches a hierarchical structure of feature maps is achieved and enables the extraction of fine and coarse features on different scales. One key contribution is that SegFormer avoids a positional encoding by introducing a 3×3-convolution ($\mathrm{Conv}_{3\times3}$) in the linear layer of in the transformer block such that Equation (2.105) transforms to

$$\mathcal{T}_\theta^{m_h,d_h,m_1}(Z) \coloneqq \mathrm{MHSA}_\theta(Z) + W_2 \cdot a_{\texttt{GELU}}\left(\mathrm{Conv}_{3\times3}\left(W_1 \cdot \mathrm{MHSA}_\theta(Z) + B_1\right) + B_2\right).$$

Positional-encoding-free transformers allow for input resolution change at test time which is important as the resolution during training and test often differs for semantic segmentation. The claimed efficiency is achieved by adopting the efficient self-attention introduced by [287]. Furthermore, the architecture benefits from its small fully connected decoder which has been shown to be enough to fuse the multiscale feature maps and predicting the class map. Similar to ResNet, SegFormer comes with differently sized encoder models ranging from B0 to B5. With this SegFormer represents a family of lean transformer-based segmentation models with reasonable parameter-performance trade-offs and adequate inference speed. Besides single task transformers, Mask2Former [43]

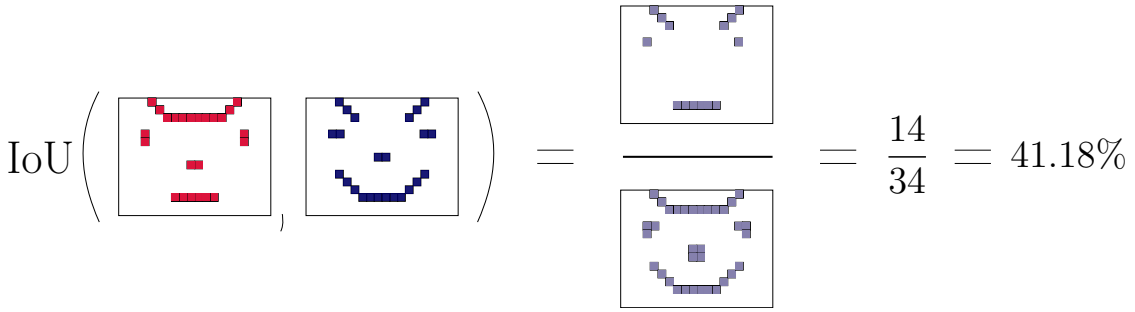


Figure 2.15: The IoU of the blue and red pixel sets is calculated by counting the pixels of their intersection and divide it by the number of pixels in the unified set.

is a transformer architecture which aims to solve multiple vision tasks at once: instance segmentation (pixel-wise segmentation of objects where each object is distinguished by a different object ID), semantic segmentation and panoptic segmentation (pixel-wise classification for background or static classes combined with instance segmentation for objects). Despite being a universal architecture, Mask2Former outperforms specialized architectures on all tasks [43].

2.3.2.4 Evaluation Metrics

Semantic segmentation is evaluated in terms of pixel accuracy or mean intersection over union (mIoU) [115]. The first is a direct transfer of the accuracy calculated for classification tasks (Equation (2.109)) to the pixel-wise classification task. Therefore, the percentage of correctly classified pixels per image is computed. As for classification, this metric is sensitive to class imbalance. Considering a binary segmentation problem where 95% of the pixels correspond to class 0 and 5% correspond to class 1, a model predicting always class 0 yields an accuracy of 95% even though it has not generalized well to class 1. The **intersection over union** also known as Jaccard index [115] tries to mitigate this bias. As the name suggests, the ratio between the intersection and the union of two countable sets S_1, S_2 is calculated

$$\mathrm{IoU}(S_1, S_2) := \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}. \tag{2.117}$$

To make it easy to count the elements in a set, e.g., pixels, the visualization of the metric is shown in Figure 2.15 for two pixel art images in red and blue, respectively.

For semantic segmentation the IoU can be represented using the number of correctly

and wrongly predicted pixels per class. We define the sets

$$\begin{aligned}
\mathrm{TP}_{\boldsymbol{\theta}}(c', \boldsymbol{x}, \boldsymbol{y}) &\coloneqq |\{(i,j) \subseteq \mathcal{I} \,:\, \hat{c}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) = c' = y_{i,j}\}| \quad \triangleleft \text{ correctly classified as class } c' \\
\mathrm{FP}_{\boldsymbol{\theta}}(c', \boldsymbol{x}, \boldsymbol{y}) &\coloneqq |\{(i,j) \subseteq \mathcal{I} \,:\, \hat{c}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) = c' \neq y_{i,j}\}| \quad \triangleleft \text{ falsely classified as class } c' \\
\mathrm{FN}_{\boldsymbol{\theta}}(c', \boldsymbol{x}, \boldsymbol{y}) &\coloneqq |\{(i,j) \subseteq \mathcal{I} \,:\, \hat{c}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) \neq c' = y_{i,j}\}| \quad \triangleleft \text{ omitted to classify as class } c'
\end{aligned}$$

for a pair $(\boldsymbol{x}, \boldsymbol{y})$ of input and ground truth and a class $c' \in \{1, \dots, C\}$. With this we can define the IoU per class over the entire dataset $\mathcal{D}$ which contains $N \in \mathbb{N}$ pairs $(\boldsymbol{x_n}, \boldsymbol{y_n})$ of input and ground truth

$$\mathrm{IoU}_{\mathcal{D},\boldsymbol{\theta}}(c) = \frac{1}{N} \sum_{n=1}^{N} \frac{\mathrm{TP}_{\boldsymbol{\theta}}(c, \boldsymbol{x_n}, \boldsymbol{y_n})}{\mathrm{TP}_{\boldsymbol{\theta}}(c, \boldsymbol{x_n}, \boldsymbol{y_n}) + \mathrm{FP}_{\boldsymbol{\theta}}(c, \boldsymbol{x_n}, \boldsymbol{y_n}) + \mathrm{FN}_{\boldsymbol{\theta}}(c, \boldsymbol{x_n}, \boldsymbol{y_n})}. \tag{2.118}$$

Thus, the IoU is the ratio of pixels where the prediction and ground truth coincide and all pixels such that $\hat{c}_{i,j}(\boldsymbol{x}|\boldsymbol{\theta}) = c'$ or[21] $y_{i,j} = c'$. The IoU is often denoted in percent. To achieve a metric equally incorporating all classes, the **mean intersection over union** (mIoU)[22]

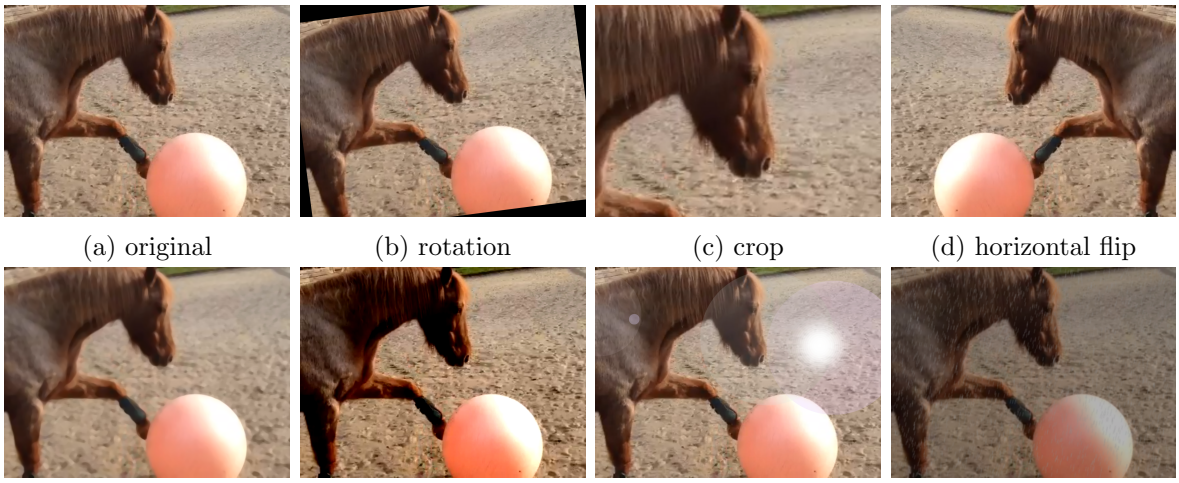
$$\mathrm{mIoU}(\mathcal{D}) = \frac{1}{C} \sum_{c=1}^{C} \mathrm{IoU}_{\mathcal{D},\boldsymbol{\theta}}(c) \tag{2.119}$$

is calculated by taking the mean with respect to the classes. As a consequence, disregarding a class leads to a distinct reduction of the metric value. Considering the binary segmentation problem from above where 95% of the pixel correspond to class 0 and 5% correspond to class 1, a model predicting always class 0 achieves an mIoU of 47.5% in contrast to the pixel accuracy of 95%.

2.4 Overcoming Data Limitation

The success of deep neural networks for vision tasks heavily depends on the amount and the quality of the (labeled) data. In many real life applications data is rare and/or expensive to annotate. Especially annotating images on pixel-level detail as needed for semantic segmentation is time-consuming, costly and likely to incorporate errors [47, 228]. Nevertheless, for understanding complex scenes like street scenes with multiple vulnerable road users, perception on a detailed level is crucial. Several concepts were proposed ranging from label coarsening over exploiting unlabeled data to extracting advantages from additional domains and datasets. In the next sections we cover the basics of the approaches to compensate the limited data amount relevant for the understanding of this thesis.

[21] Here we mean the logical 'or' which is non-exclusive.
[22] For notation simplicity we omit the dependency on the neural network.



(a) original (b) rotation (c) crop (d) horizontal flip

(e) median blur (f) contrast-brightness (g) sun flare artifacts (h) artificial rain

Figure 2.16: Image augmentations. First row shows geometric augmentations whereas the second row depicts augmentations on pixel-level. Sun flare and artificial rain are automotive specific augmentations which aim to cover a wide range of weather scenarios. Augmentations were done with the albumentation library [30].

2.4.1 Data Augmentation

To enlarge the dataset without additional annotation cost, **data augmentation** is a common practice [30]. Therefore, the input and where necessary its annotation is modified randomly in each epoch. Augmentations can be categorized into geometric and pixel-level augmentations. Typical geometric augmentation methods are random cropping, scaling, rotating, flipping and non-linear geometric distortions [146]. Pixel-level augmentations refer to color- or texture-based augmentations ranging from color space transformations, color jittering [142] and contrast normalization to adding noise or blur to the input [253]. A prior knowledge about the task and its invariance to augmentations is needed to choose appropriate modifications to the input. In street scenes mostly horizontal flipping is used since vertical flipping – flip the upper with the lower part of the image – leads to unrealistic scenes. For example the sky is most likely in the upper half of an image and the street on the lower part and not vice versa. Similarly, in digit recognition a rotation of 180 degree changes the semantic of a 6 and a 9. This technique additionally leads to more diversity in the dataset as it differs from epoch to epoch which makes it an implicit regularization method to mitigate overfitting [30, 142]. Furthermore, neural networks trained with input augmentation are more robust to unseen data [241]. Examples of augmentation methods are visualized in Figure 2.16. In practice, several of the presented augmentation methods are combined followed by the input normalization as described in Section 2.2.5. The realization of the transformation takes place at the time of data loading.

2.4.2 Weak and Semi-supervised Learning

If unlabeled data is available, weakly labeling the data can be an alternative to limit the labeling cost. Thereby, weak labels serve as a rough approximation of the true labels. By e.g., using bounding box annotations for semantic segmentation tasks, the labeling cost is reduced from pixel accurate segments to rectangular boxes. With the help of region proposal methods multiple candidate segmentation masks are generated. By choosing the one which overlaps the most with the bounding box annotation a segmentation mask with estimated label is achieved and can be used for training the semantic segmentation network. In an iterative learning process the segmentation is refined by updating the label estimates for the proposals and updating the network parameters [53].

In practice, labeling a small subset of data in a fine manner is feasible. However, the amount of labeled data is not enough to train a well generalizing model in a supervised manner (cf. Section 2.2.2). If we can assume that the marginal distribution P_X contains information about the conditional distribution $P_{Y|X}$, semi-supervised learning methods can be applied to take advantage from the unlabeled data [280]. The loss with respect to a labeled dataset $\mathcal{D}$ and an unlabeled dataset $\mathcal{U}$ can be defined by

$$\mathcal{L}_{\mathcal{D}\cup\mathcal{U}} = \gamma_{\mathcal{D}}\mathcal{L}_{\mathcal{D}} + \gamma_{\mathcal{U}}\mathcal{L}_{\mathcal{U}}, \tag{2.120}$$

where $\gamma_{\mathcal{D}}, \gamma_{\mathcal{U}}$ are scalar weighting factors which may depend on the training iteration and $\mathcal{L}_{\mathcal{D}}$ and $\mathcal{L}_{\mathcal{U}}$ are a supervised and unsupervised loss function, respectively [41]. The success of semi-supervised learning presumes a certain smoothness and manifold structure of the data distribution. That means, data points which are close in the data space should share the same label with a high probability. Furthermore, it is assumed that the data within a manifold shares the same label and the manifolds or clusters are separable by low-density regions [41]. Modern semi-supervised learning techniques involve self-training methods that use (learned) pseudo labels generated by the neural network. These labels are iteratively included in the training process, often weighted by the prediction's uncertainty, to enhance the model's performance and, consequently, the pseudo labels [118, 150, 306]. Often these methods need a warm-up phase to assure an adequate quality of the pseudo-labels. This can be modeled with the help of the weighting factors $\gamma_{\mathcal{D}}, \gamma_{\mathcal{U}}$. Besides self-training, a two-stage training is a common strategy when limited labeled data is available. Starting with an unsupervised pre-training good initial network parameters can be achieved to then fine-tune to a specific task on a small amount of labeled data. Albeit the literature lacks uniform categorization of this method, we understand it as a semi-supervised method. It can be modeled by setting $\gamma_{\mathcal{D}} = 0$ and $\gamma_{\mathcal{U}} = 1$ for the pre-training and switching their values for the fine-tuning phase. We discuss this method in more detail in the context of transfer learning (see Section 2.4.3.1).

2.4.3 Transfer Learning

When considering specific problems, often only a small dataset $\mathcal{D}_\mathcal{T} \subseteq \mathcal{X}_T \times \mathcal{Y}_T$ is available. Nevertheless, a larger dataset $\mathcal{D}_\mathcal{S} \subseteq \mathcal{X}_S \times \mathcal{Y}_S$ stemming from a potentially different distribution might exist which can help to learn general features. The idea of transfer learning is to transfer 'knowledge' from the source domain and the task solved therein to help estimate the predictive function in a target domain [197]. According to the taxonomy in [137] a learning problem has three components in which it can differ: the feature space $\mathcal{X}$, the label space $\mathcal{Y}$ and the distribution p_S given by the marginal distribution p_X according to which each $\boldsymbol{x}$ in the training dataset was drawn and the conditional probability distribution $p_{Y|X}$ for each tuple $(\boldsymbol{x}, y)$ in the training dataset $\mathcal{D}_\mathcal{S}$. We define a domain $\mathcal{S}$ as the triple

$$\mathcal{S} = (\mathcal{X}_S, \mathcal{Y}_S, p_S) \tag{2.121}$$

where p_S is the joint probability defined by $p_S(\boldsymbol{x}, y) = p_{Y|X}(y|\boldsymbol{x})p_X(\boldsymbol{x})$. As introduced in Section 2.2.1 the predictive distribution (conditional probability distribution) is learned by a neural network $f_{\boldsymbol{\theta}}^S$ on the source dataset $\mathcal{D}_\mathcal{S}$ and $f_{\boldsymbol{\vartheta}}^T$ on the target dataset $\mathcal{D}_\mathcal{T}$. In later chapters we omit the parameter dependency in the notation and write f_S or f_T for the sake of readability.

Transfer learning is used when the learning problems in the source and target domain differ in at least one of the components. If the learning problems differ only in the distribution, i.e., $p_S \neq p_T$ but $\mathcal{X}_S = \mathcal{X}_T$ and $\mathcal{Y}_S = \mathcal{Y}_T$ are identical, the method to achieve this transfer is named **domain adaptation** which we introduce in Section 2.4.4. A new domain adaptation method developed as part of this thesis is described in Chapter 4.

2.4.3.1 Pre-Training

An intuitive example how knowledge can be transferred is when only the label space differs, i.e, $\mathcal{Y}_S \neq \mathcal{Y}_T$. General features to identify specific patterns can be learned on a much broader task. In a subsequent step, only the fine peculiarities of the target label space need to be learned. In other words, when the input data for two tasks, a source task and a target task, is similar, e.g., naturalistic RGB images, neural networks solving a target task can draw advantage from neural networks trained for the source task.

To this end, firstly, a neural network $f_{\boldsymbol{\theta}}^S$ is trained on the source dataset D_S to solve the source task, e.g., classification of a broad range of objects. This is called **pre-training** and leads to a model $f_{\hat{\boldsymbol{\theta}}}^S$. Assuming that (part of) the features learned by $f_{\hat{\boldsymbol{\theta}}}^S$ are general enough for the target distribution, the parameters of $f_{\hat{\boldsymbol{\theta}}}^S$ can support learning the target task. To solve the target task, e.g., a fine-grained classification, a second neural network $f_{\boldsymbol{\vartheta}}^T$ with the same base architecture is used to solve

$$\underset{\boldsymbol{\vartheta}}{\operatorname{argmin}} \mathcal{L}_{\mathcal{D}_T}(\boldsymbol{\vartheta}) + \gamma \mathtt{d}(f_{\boldsymbol{\vartheta}}^T - f_{\hat{\boldsymbol{\theta}}}^S) \tag{2.122}$$

in a so-called **fine-tuning** step [184, 20.5.1.1]. With $\mathtt{d}(f^T_{\boldsymbol{\vartheta}} - f^S_{\hat{\boldsymbol{\theta}}})$ we denote any appropriate distance measure between the functions represented by the neural networks. A possible measure could be the difference in the parameters $\hat{\boldsymbol{\theta}}$ and $\boldsymbol{\vartheta}$. The regularization is weighted with a scalar $\gamma \in \mathbb{R}_{\geq 0}$. In practice the regularization is often done by sharing all but the head weights with $f^S_{\hat{\boldsymbol{\theta}}}$. Pre-training therefore can be understood as a good initialization for the network parameters of $f^T_{\boldsymbol{\vartheta}}$ to simplify the training [208]. Several approaches are known for fine-tuning. Starting from the weights of the pre-trained model, one common approach is to 'freeze' the weights of all but the head layers and only train the task specific head on top of that model. Freezing the weights means that these weights are treated as fixed values rather than variables whose gradients are calculated during backpropagation. Fixing the weights ensures that the general features are not discarded due to the task but task specific combinations of the features and their hierarchical dependency are learned with the help of the remaining learnable weights. Nevertheless, this might be a too strict assumption on the similarity of the two feature spaces. As a consequence, a different approach is to fine-tune all weights but with a distinct smaller learning rate for the backbone parameters than for the head which is trained from scratch, i.e., starting from randomly initialized weights. However, the parameter subset to fine-tune depends on the task to solve. In the context of domain adaptation, it was shown in [124] that fine-tuning of earlier layers have more influence when faced with non-semantic shifts, e.g., lighting conditions. In contrast, parameters of later layers have more influence on semantic related features. To this end, fine-tuning earlier layers can be more reasonable in certain applications.

As described in Section 2.3.2, neural networks for semantic segmentation follow the concept of a feature extracting backbone combined with a task specific head. Many backbones for classification are also suitable for semantic segmentation tasks and therefore act as general purpose backbones, e.g., ResNet. A common practice is to first train the backbone on an easier task like classification where a huge amount of labeled data is available to learn basic features. The primary dataset for acquiring fundamental features of naturalistic objects and therefore serving as a source dataset $\mathcal{D}_S$ is ImageNet[23] [183]. After pre-training, the model is fine-tuned on the specific dataset (e.g., a semantic segmentation dataset of urban street scenes) to solve the actual task.

Even though pre-training (in particular ImageNet pre-training) is a widely used strategy, it may not be as effective for certain applications. This is especially true if the similarity requirement of the input domains does not hold, as for example for medical images [7,213]. Furthermore, the experiments of He et al. suggest that ImageNet pre-training for object detection leads to a convergence speed up in early training stages but does neither necessarily have a positive influence on the target accuracy nor on overfitting prevention compared to training neural networks from scratch sufficiently long. Therefore, the authors question the extensive use of ImageNet pre-training [89].

[23] More precisely, a curated subset of 1,000 classes and approximately 1.2 million training images which was created for the ImageNet Large Scale Visual Recognition Challenge [230]. It is a challenging benchmark in image classification and the data is widely used for training or to compare methods.

2.4.3.2 Self-supervised Learning

Besides the supervised pre-training discussed so far, neural networks can be pre-trained in an unsupervised manner. One method to learn general features from unlabeled data, is to define a pretext task, e.g., colorizing grayscale images or playing image jigsaw puzzles. The labels of this task, so-called pseudo labels, can be automatically generated based on the data. By solving the pretext task a descriptive representation and general features of the data are learned. This technique is better known as **self-supervised learning** and has proven successful for several vision tasks [322]. A detailed review on this topic can be found in [119]. In contrast to self-training (see Section 2.4.2), self-supervised learning is task agnostic and therefore does not need any task-specific labels [41].

2.4.4 Domain Adaptation

When the marginal distributions of the source and target domain differ distinctly, neural networks often fail to generalize to the new domain even though they achieve excellent performance on the source domain. When the learning problem only differs on distribution level (marginal and/or conditional), this phenomenon is called domain gap or domain shift and roots in the shift in distribution between the two domains [137]. A distinct shift in the input distribution is for example given when considering real versus computer generated data while both stem from the same feature space $\mathcal{X}_S = \mathcal{X}_T = [0,1]^{m \times h \times w}$ – the space of images. Given a labeled source domain $\mathcal{S} = (\mathcal{X}_S, \mathcal{Y}_S, p_S)$ and a target domain $\mathcal{T} = (\mathcal{X}_T, \mathcal{Y}_T, p_T)$ domain adaptation (DA) methods aim to mitigate the performance gap of neural networks trained on a subset $\mathcal{D}_\mathcal{S} \subseteq \mathcal{S}$ and evaluated on a subset $\mathcal{D}_\mathcal{T} \subseteq \mathcal{T}$.

To understand the root causes of the domain shift, we recap that $(\boldsymbol{x}, y)$ can be associated by a random variable (X, Y) distributed according to $P_{X,Y}$ with density $p_{X,Y}$. The relation between the joint, marginal and conditional distribution is given by

$$p_{X,Y}(\boldsymbol{x}, y) = p_{Y|X}(y|\boldsymbol{x}) p_X(\boldsymbol{x}) \tag{2.123}$$

or likewise

$$p_{X,Y}(\boldsymbol{x}, y) = p_{X|Y}(\boldsymbol{x}|y) p_Y(y). \tag{2.124}$$

The latter representation reveals a prior shift if the class-conditional distribution is identical in both domains, i.e., $p_S(\boldsymbol{x}|y) = p_T(\boldsymbol{x}|y)$, but the class distribution differs, i.e., $p_S(\boldsymbol{y}) \neq p_T(\boldsymbol{y})$. This shift occurs if for example a class appears much more often in one domain as in the other. Modeling the joint distribution via the first representation, a covariate shift can be the cause of the neural network's performance drop. That is, the posterior distribution is identical ($p_S(y|\boldsymbol{x}) = p_T(y|\boldsymbol{x})$) but the data distribution differs ($p_S(\boldsymbol{x}) \neq p_T(\boldsymbol{x})$). Covariate shifts are often associated with texture, brightness

or contrast variation between the domains. Lastly there can be a shift in concepts. Thus, the data distribution is unchanged but the posterior distribution differs between the domains. This is the case if objects of class c in one domain have label c' with $c' \neq c$ in the other domain. The taxonomy in transfer learning including domain adaptation is not consistent in literature and has changed over time. We follow the categorization in [137]. In practice the domain shift is generally not caused by a single shift but by a combination of two or more.

The question of learnability of domain adaptation has been addressed by Ben-David et al. [22]. They examine the necessary assumptions for successfully learning domain adaptation regarding the correlation between the two distributions. The key finding is that the dissimilarity of the data distribution needs to be small as well as there must exist a hypothesis which has low error on both domains. Furthermore, they investigated "a uniform convergence learning bound for algorithms which minimize convex combinations of empirical source and target errors" [22]. Kouw and Loog list a few additional relations which can be exploited to achieve generalizability to the target domain [137].

Domain adaptation is mostly done when there are little or even no labels available for the target domain. In this thesis, we focus on domain adaptation for semantic segmentation where labels are much more expensive to obtain than for classification. We distinguish between three major types of domain adaptation depending on the label amount available in the target domain: unsupervised (UDA; no labels available), semi-supervised (SSDA; a few labels available) or supervised (SDA; labels exist for all training samples) [271]. Domain adaptation is a very active and developing research area where most publications in context of domain adaptation focus on UDA as no additional labeling effort is needed in the target domain. For an overview over the variety of publication see for example [49, 242, 271].

Adaptation can be done in different stages of the neural network. Ranging from input-level adaptation [26, 63, 100, 182, 272], over feature-level adaptation [101, 126, 129, 305] to output-level adaptation via, e.g., self-training [129, 171, 277, 278, 283, 305, 317, 325]. A visualization of the three levels is depicted in Figure 2.17. Input level adaption can also be done offline, meaning that the input transformation is not part of the model training but has been done in advance. The adaptation is not limited to a single stage, and often it is difficult to map the approaches to only one of the categories like [129, 305]. Approaches which act on multiple levels are sometimes called 'hybrid models'. A study on hybrid models was done in [242].

UDA has been shown to still have a distinct gap to target performance. Many approaches aim to align the features in the source and target domain. Since no labels are available the alignment is done regardless of the class ID. Nevertheless, a few labels in the target domain might lead to more performance gain with only little annotation cost [250]. In a **semi-supervised domain adaptation** setup, methods have the potential to draw advantage from a random but mostly very small subset of the target data which is provided with labels. A direct transfer of unsupervised approaches reveals that

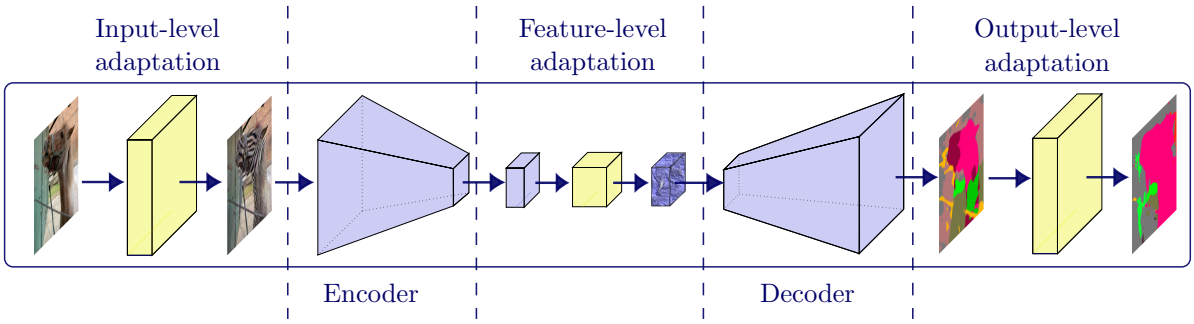


Figure 2.17: The different adaptation levels in domain adaptation are depicted in the style of the visualization of [271]. Adaptation can be done in each single stage or in multiple subsequent stages and is realized by the yellow drawn adaptation modules. If adaptation is done only in one stage the other modules would act as identity mappings. The input-level adaptation in the figure was achieved by a CycleGAN [324]. Prediction results are yield by a deeplabv3 with ResNet101 backbone trained on ADE20k with the help of the MMSegmentation library [179].

they might fail to learn discriminative features [256]. In the context of classification several approaches are proposed. This problem is for example addressed by inter-domain contrastive alignment [256] or optimizing a minimax loss on the conditional entropy of unlabeled data and the task loss [231]. However, publications addressing SSDA for semantic segmentation [40, 42, 185, 291] are still limited. The method in [185] was developed in the context of this thesis and is explained in detail in Chapter 4. We also give a short overview over the other mentioned methods in Section 4.2.

2.4.4.1 Active Learning Under Domain Shift

Instead of using a small randomly sampled but fixed labeled set in the target domain as it is the case in SSDA, in active domain adaptation (ADA) investigations are made on strategies asking for the annotation of likely informative data points, image regions or pixels in the target domain. In an iterative manner, the additional annotation is included into the training dataset and the model under domain shift is retrained. The aim is to achieve a sample-efficient domain alignment [207]. These methods are often transferred from classical active learning (AL) strategies [207].

Active learning aims to reduce the labeling cost by asking only for the labels of informative samples. This describes a cycle of training, querying unlabeled data, annotating them by an oracle and retraining on an enlarged dataset. The general concept is visualized in Figure 2.18. The unlabeled data for which labels can be queried, can be loosely grouped into three categories [244]. The data might be stream-based, i.e., data is shown sequentially and the active learner can decide to query the data for annotation or discard it. The data can be synthesized by the active learner, e.g., by a generative model (see Section 3.1) or most commonly the data is given as a large pool. In this thesis we focus on the latter since in real world applications it is often easy to collect a large amount of

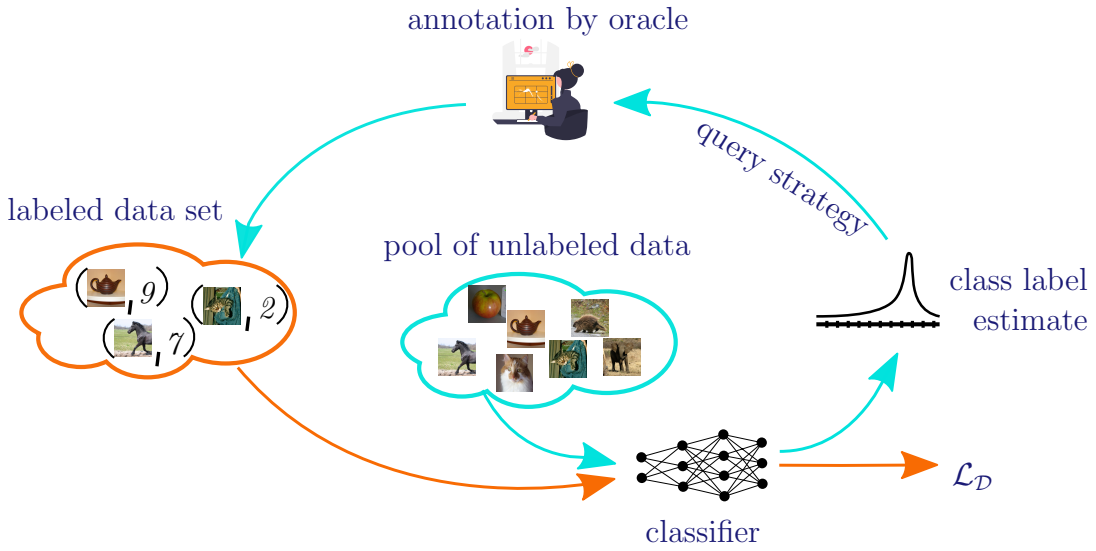


Figure 2.18: The active learning cycle consists of four steps. Training a neural network on a (small) labeled dataset, querying data from the unlabeled pool based on, e.g., model uncertainty, annotating it by an oracle, enlarging the labeled dataset and restarting the cycle by newly training the neural network.

data by, e.g., recording scenes but annotating is expensive. In the following we describe the cycle for pool-based AL in more detail on the basis of [244]. Therefore, we assume having a large unlabeled dataset $\mathcal{U}_0 \subseteq \mathcal{X}$. The cycle starts by training a neural network $f_{\boldsymbol{\theta}}^{(0)}$ on a small labeled dataset $\mathcal{D}_0 = \{(\boldsymbol{x_1}, \boldsymbol{y_1}), \ldots, (\boldsymbol{x_{N_0}}, \boldsymbol{y_{N_0}})\} \subseteq \mathcal{X} \times \mathcal{Y}$.

The key component in active learning is the so-called **query strategy** and the corresponding **acquisition function** applied to the unlabeled data. On its basis the informativeness of data points is measured and samples to annotate are chosen. Often the acquisition function depends on $f_{\boldsymbol{\theta}}^{(j)}$ the actual trained neural network in the j-th, $j \in \mathbb{N}$, cycle iteration. Let $\mathcal{U}_j$ denote the pool of unlabeled data in iteration j. Query strategies include concepts of query-by-committee [21,245], diversity sampling [243] and uncertainty sampling [68,244]. We make the acquisition more tangible for a multi-class classification setup with uncertainty-based sampling. One way to measure the uncertainty of a neural network is to consider the softmax-confidence score as defined in Equation (2.94). Possibilities to query with respect to the confidence are choosing the data point with the least confidence

$$\boldsymbol{x}^*_{\text{LC}} = \underset{\boldsymbol{x} \in \mathcal{U}_j}{\operatorname{argmax}} \left(1 - \max_i (f_{\boldsymbol{\theta}}^{(j)}(\boldsymbol{x})_i)\right), \tag{2.125}$$

the minimal margin between the prediction $\hat{c}(\boldsymbol{x}|\boldsymbol{\theta})$ and the second most confident class

$$\boldsymbol{x}^*_{\text{MARGIN}} = \underset{\boldsymbol{x} \in \mathcal{U}_j}{\operatorname{argmin}} \left(\hat{c}(\boldsymbol{x}|\boldsymbol{\theta}) - \underset{i \in \{1,\ldots,C\} \setminus \hat{c}(\boldsymbol{x}|\boldsymbol{\theta})}{\operatorname{argmax}} f_{\boldsymbol{\theta}}^{(j)}(\boldsymbol{x})_i\right), \tag{2.126}$$

or with the highest entropy [247]

$$\boldsymbol{x}^*_{\mathrm{ENT}} = \underset{\boldsymbol{x}\in\mathcal{U}_j}{\operatorname{argmax}} -\sum_{i=1}^{C} f_{\boldsymbol{\theta}}^{(j)}(\boldsymbol{x})_i \log f_{\boldsymbol{\theta}}^{(j)}(\boldsymbol{x})_i. \tag{2.127}$$

It is possible to query one data point $\boldsymbol{x} \in \mathcal{U}_j$ or a batch $\mathcal{B} = \{\boldsymbol{x_1}, \ldots, \boldsymbol{x_{N_\mathcal{B}}}\} \subseteq \mathcal{U}_j$ of $N_\mathcal{B} \in \mathbb{N}$ data points. The selected samples are then annotated by an oracle which can, e.g., be a human specialist, e.g., a doctor, or a superior neural network. For studies on AL concepts, the oracle is often simulated by holding back the ground truth data until samples are selected for annotation. The data points chosen for annotation are removed from the unlabeled pool

$$\mathcal{U}_1 = \mathcal{U}_0 \setminus \mathcal{B} \text{ and more generally } \mathcal{U}_{j+1} = \mathcal{U}_j \setminus \mathcal{B} \tag{2.128}$$

and instead the newly annotated data $\mathcal{B}_{\boldsymbol{y}} = \{(\boldsymbol{x_1}, \boldsymbol{y_1}), \ldots, (\boldsymbol{x_{N_\mathcal{B}}}, \boldsymbol{y_{N_\mathcal{B}}})\}$ is included into the labeled dataset

$$\mathcal{D}_1 = \mathcal{D}_0 \cup \mathcal{B}_{\boldsymbol{y}} \text{ and more generally } \mathcal{D}_{j+1} = \mathcal{D}_j \cup \mathcal{B}_{\boldsymbol{y}} \text{ for } j \geq 0. \tag{2.129}$$

The enlarged dataset is used to retrain the neural network from scratch, leading to a model $f_{\boldsymbol{\theta}}^{(j+1)}$ which is now used to calculate the updated acquisition function. This cycle is repeated until a certain performance is reached or a predefined amount of label budget is used up.

Even though **active domain adaptation** (ADA) is inspired by classical AL, ADA is more complex since the model, which often serves as source of data point selection, is exposed to a domain shift. As a consequence, it is potentially less reliable. Furthermore, the selection should help the network to adapt to the target domain rather than to the source domain. That means, not only the predictive uncertainty but also the targetness of the samples is important [301]. Thereby, the targetness describes how representative or relevant the samples are to cover the target distribution. Several approaches were published with respect to ADA such as [66, 207, 214, 261, 299]. Thereby the approaches investigate different query strategies and adaptation approaches ranging from uncertainty-weighted clustering to active adversarial domain adaptation, where adversarial domain alignment is combined with different query strategies to mitigate the domain shift. An overview on ADA in semantic segmentation was published by Csurka et al. [49]. The extension of our SSDA method [185] to an ADA method is presented in Section 4.3.5. Our experiments in Section 4.4.2 demonstrate that actively sampling informative data points for a semi-supervised image-to-image generator leads to a notable performance gain.

Chapter 3

Generative Learning

In the previous chapter we focused on discriminative models. Thus, a function $f_{\boldsymbol{\theta}} : \mathcal{X} \to [0,1]^{|\mathcal{Y}|}$ was learned which approximates a conditional probability distribution over the label space $\mathcal{Y}$ and is used to classify an input by assigning the ID of the class with the highest probability. However, besides learning the conditional probability distribution $p_{Y|X}(y|\boldsymbol{x})$ or more generally a conditional probability measure $\nu_{|X}$ on $\mathcal{Y}$ to classify inputs $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ given by a random variable X, one could also ask for the data generating distribution μ_{data} of X. That means, we are interested in a generative model which models the probability distribution of a random variable with values in $\mathcal{X}$ describing the data generating process. Let $\mathcal{D}$ be a dataset where each sample was independently drawn with respect to μ_{data}. Then, generative learning aims to train a generative model on $\mathcal{D}$ which is capable of producing new data that is also approximately distributed according to μ_{data} and thus approximately distributed as the training data [113].

A distribution can be modelled in an explicit or implicit manner. For explicit models or sometimes called prescribed probabilistic models, an explicit parametric specification of the distribution is given which allows calculating the corresponding (log-)likelihood function with parameters $\boldsymbol{\theta}$ [180]. New data points can be sampled via, e.g., Monte-Carlo methods [9]. Examples for explicit probabilistic models are (Gaussian-)Mixture Models [184, chapter 29.2] in the generative setting but also classification models as introduced in Section 2.3, where we used the maximum likelihood principle to optimize the parameters of a discriminative model in a training process (cf. Equation (2.48)). In contrast, for implicit models no explicit likelihood function is given, and a stochastic process is used to generate samples following a certain distribution $\hat{\mu}$. The stochastic process is often realized by feeding stochastic input to a deterministic parametric function

$$g_{\boldsymbol{\theta}} : \mathcal{Z} \to \mathcal{X}, g_{\boldsymbol{\theta}}(\boldsymbol{z}) = \hat{\boldsymbol{x}}, \tag{3.1}$$

As $g_{\boldsymbol{\theta}}$ generates data by transforming stochastic input into the data space $\mathcal{X}$, it is often

called generator function, yielding samples distributed according to a distribution $\hat{\mu}$

$$g_{\boldsymbol{\theta}}(Z) = \hat{X} \sim \hat{\mu}, \qquad Z \sim \nu, \tag{3.2}$$

where ν is a probability measure on a latent space $\mathcal{Z}$. The training objective is to learn parameters $\hat{\boldsymbol{\theta}}$ such that $\mathfrak{d}(\mu_{\text{data}}, \hat{\mu})$ is minimal for some divergence measure $\mathfrak{d}$. How well two distributions are aligned can for example be measured by the Kullback-Leibler divergence introduced in Equation (2.39) or by the **Jensen-Shannon divergence**

$$\mathfrak{d}_{\text{JS}}(\mu_{\text{data}}, \hat{\mu}) \coloneqq \frac{1}{2}\left[\mathfrak{d}_{\text{KL}}\left(\mu_{\text{data}} \middle\| \frac{\mu_{\text{data}} + \hat{\mu}}{2}\right) + \mathfrak{d}_{\text{KL}}\left(\hat{\mu} \middle\| \frac{\mu_{\text{data}} + \hat{\mu}}{2}\right)\right]. \tag{3.3}$$

Without loss of generality, we assume that the probability measure μ_{data} is absolutely continuous with respect to the Lebesgue measure λ. Therefore, we aim to approximate a density function p_{data} with $d\mu_{\text{data}} = p_{\text{data}} d\lambda$. Assuming that ν is absolutely continuous with respect to the Lebesgue measure λ with density function q, the estimated density $p_{g_{\boldsymbol{\theta}}}$ on the data space $\mathcal{X}$ can be modeled as the derivative of the cumulative distribution function over the set of transformed data under $g_{\boldsymbol{\theta}} : \mathbb{R}^t \to \mathbb{R}^d$ [180, 184] with

$$p_{g_{\boldsymbol{\theta}}}(\boldsymbol{x}) = \frac{\partial}{\partial x_1} \cdots \frac{\partial}{\partial x_{d_2}} \int_{\{g_{\boldsymbol{\theta}}(\boldsymbol{z}) \leq \boldsymbol{x}\}} q(\boldsymbol{z}) d\boldsymbol{z}. \tag{3.4}$$

However, $p_{g_{\boldsymbol{\theta}}}(\boldsymbol{x})$ is intractable and hard to compute for implicit models. To this end, a new learning principle is used to learn the optimal parameters for $g_{\boldsymbol{\theta}}$ which avoids an explicit density representation of $\hat{\mu}$ [82,180]. This is done by comparing the distributions solely based on a set of generated data points and samples from the true data generating distribution in form of samples in a training dataset. The parameters $\boldsymbol{\theta}$ of the generator function $g_{\boldsymbol{\theta}}$ are then optimized with respect to the feedback of the sample comparison.

Generative models are used in many vision-related applications, not limited to data generation, e.g., for enlarging training datasets, but also used for in-painting [310], image translation [114, 324] as well as generating counterfactuals for model explanations [176] and out-of-distribution robustness [237]. Knowledge on the data distribution (explicitly or implicitly) allows the generation or sampling of new data points from that distribution. Furthermore, knowing the data distribution μ_{data} can also help in the discriminative setting to access uncertainty about the data space when considering the joint distribution from the perspective of a factorization in the marginal distribution and a conditional probability measure [195, 273].

In this thesis, we focus on generative adversarial networks which belong to the class of implicit generative models. We introduce the concept of generative adversarial networks and show how to compare distributions solely based on sample sets from two distributions in the next section. Those are an adequate choice as our primary aim is to generate high quality images and select specific latent spaces $\mathcal{Z}$ but not having the need of an explicit distribution specification.

3.1 Generative Adversarial Networks

Generative adversarial networks (GANs) belong to the class of implicit distribution approximators. This means that, GANs enable data synthesis without being restricted to a family of parametric density functions. Instead, they implicitly implement a distribution by providing a generator that can be used to sample data. We consider two probability spaces $(\Omega_S, \mathcal{A}_S, \nu)$ and $(\Omega_T, \mathcal{A}_T, \mu_{\text{data}})$ with σ-algebras $\mathcal{A}_S$, $\mathcal{A}_T$ and a probability measure $\nu : \mathcal{A}_S \to [0,1]$ on Ω_S, which is simple to sample, and an unknown probability measure $\mu_{\text{data}} : \mathcal{A}_T \to [0,1]$ on Ω_T. The generator is a measurable function

$$g : (\Omega_S, \mathcal{A}_S) \to (\Omega_T, \mathcal{A}_T), \tag{3.5}$$

i.e., for all $A_T \in \mathcal{A}_T$

$$g^{-1}(A_T) = \{\omega_S \in \Omega_S | g(\omega_S) \in A_T\} \in \mathcal{A}_S. \tag{3.6}$$

With the help of the measurable function g the measurable space $(\Omega_T, \mathcal{A}_T)$ can be endowed with the pushforward measure $g_*\nu$, such that

$$g_*\nu(A_T) \coloneqq \nu(g^{-1}(A_T)) \in \mathcal{A}_S \quad \forall A_T \in \mathcal{A}_T. \tag{3.7}$$

As a consequence, the generator can generate samples $g(Z) \sim g_*\nu$ from a random variable $Z \sim \nu$ which can be sampled by a random number generator. To approximate the unknown measure μ_{data} on $(\Omega_T, \mathcal{A}_T)$ the generator in a GAN is learned in combination with a so-called discriminator

$$D : \Omega_T \to (0,1) \tag{3.8}$$

which estimates the probability of ω_T stemming from μ_{data} rather than from $g_*\nu$. With the help of the concatenated mapping $D \circ g$, the generator can learn from the feedback of the discriminator. As a consequence, the discriminator enables a comparison between the two measures without the need of explicit likelihood functions. In the following, we explain the concept in more detail.

3.1.1 Vanilla GAN

Generative adversarial networks were introduced by Goodfellow et al. [79] in 2014. The two mappings g and D are realized by two neural networks $G_{\boldsymbol{\theta}}$ and $D_{\boldsymbol{\vartheta}}$ with learnable parameters $\boldsymbol{\theta}$ and $\boldsymbol{\vartheta}$. The input space $(\Omega_S, \mathcal{A}_S)$ of the generator is set to a latent space $\mathcal{Z}$, e.g., $\mathcal{Z} = \mathbb{R}^t, t \in \mathbb{N}$ with a probability measure ν from which it is easy to draw samples. An example for such a probability distribution is the uniform distribution over $[0,1]^t$. The generator maps into the space of images $\mathcal{X} = [0,1]^d, d = m \times h \times w$. The discriminator is defined on $(\mathcal{X}, \mathcal{A}_{\text{data}}, \mu_{\text{data}})$, where μ_{data} is the true data generating distribution and therefore the learning target of $G_{\boldsymbol{\theta}*}\nu$.

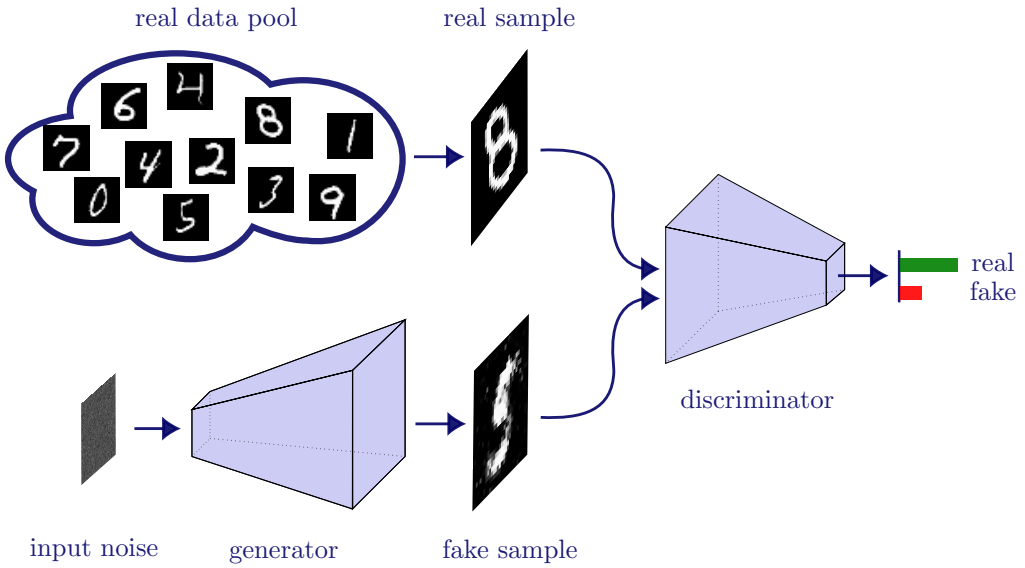


Figure 3.1: The schematic illustration shows a GAN consisting of a generator which generates fake samples from noise and a discriminator. The aim of the discriminator is to distinguish between the sources of its input. Therefore, images drawn from the true data generating distribution and fake, i.e., generated, images are fed to the discriminator during training. With the help of the feedback of the discriminator, the generator learns how to better fool the discriminator to iteratively approximate the true distribution by the pushforward measure.

The neural networks are trained with adversarial aims. The task of the generator $G_{\boldsymbol{\theta}} : \mathcal{Z} \to \mathcal{X}$ is to synthesize data from noise, which is indistinguishable from samples drawn from the unknown data generating distribution μ_{data}. The discriminator $D_{\boldsymbol{\vartheta}} : \mathcal{X} \to (0, 1)$ learns to classify the source of the data samples in order to reveal synthesized (fake) data. The source of data can be the training dataset $\mathcal{D}$, where each sample is distributed according to μ_{data}, i.e.,

$$X \sim \mu_{\text{data}} \tag{3.9}$$

or based on the generator output, i.e.,

$$\hat{X} = G_{\boldsymbol{\theta}}(Z) \sim G_{\boldsymbol{\theta}*}\nu, \tag{3.10}$$

where $Z \sim \nu$ is a random noise vector. A schematic description of a GAN is shown in Figure 3.1.

Considering the discriminator as classifier, we can apply the training methods from classification tasks. When keeping $\boldsymbol{\theta}$ fixed, the negative binary cross entropy loss

$$V^{BCE}(D_{\boldsymbol{\vartheta}}) = \mathbb{E}_{X \sim \mu_{\text{data}}}[\log D_{\boldsymbol{\vartheta}}(X)] + \mathbb{E}_{Z \sim \nu}[\log(1 - D_{\boldsymbol{\vartheta}}(G_{\boldsymbol{\theta}}(Z)))] \tag{3.11}$$

can be maximized by learning the parameters $\boldsymbol{\vartheta}$ such that the discriminator assigns high probability to data points coming from the data generation distribution (first summand)

and low probability to synthesized data (second summand) [273]. Note, that in this formulation we aim to maximize the 'loss' as it is understood as the value of the objective rather than a loss as introduced in Section 2.2.1 where the cross entropy is minimized (cf. Equation (2.64)). At the same time the generator aims to fool the discriminator by improving its output such that $D_{\boldsymbol{\vartheta}}$ assigns high probability to $G_{\boldsymbol{\theta}}(\boldsymbol{z})$. Therefore, $G_{\boldsymbol{\theta}}$ is trained with respect to minimizing the loss

$$\mathcal{L}(G_{\boldsymbol{\theta}}) = \mathbb{E}_{X\sim\mu_{\text{data}}}[\log D_{\boldsymbol{\vartheta}}(X)] + \mathbb{E}_{Z\sim\nu}[\log(1 - D_{\boldsymbol{\vartheta}}(G_{\boldsymbol{\theta}}(Z)))]. \tag{3.12}$$

This leads to a minimax game

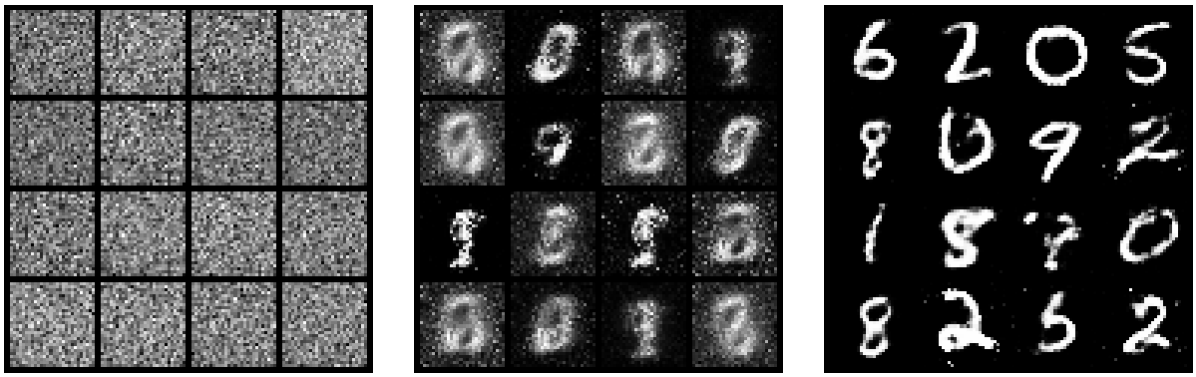
$$\min_{G_{\boldsymbol{\theta}}} \max_{D_{\boldsymbol{\vartheta}}} \underbrace{\mathbb{E}_{X\sim\mu_{\text{data}}}[\log D_{\boldsymbol{\vartheta}}(X)] + \mathbb{E}_{Z\sim\nu}[\log(1 - D_{\boldsymbol{\vartheta}}(G_{\boldsymbol{\theta}}(Z)))]}_{\mathcal{L}^{\text{GAN}}(D_{\boldsymbol{\vartheta}}, G_{\boldsymbol{\theta}})} \tag{3.13}$$

between the two players. In practice, we approximate the expected values empirically with the arithmetic mean over a finite training dataset $\mathcal{D} = \{\boldsymbol{x_n}\}_{n=1}^N \subseteq \mathcal{X}$ where each data point is drawn with respect to μ_{data} and a finite number of samples $\{\boldsymbol{z_n}\}_{n=1}^N \subseteq \mathcal{Z}$ drawn independently from ν, $N \in \mathbb{N}$. Under adequate conditions on the hypothesis spaces for G_θ and D_ϑ this is a decent approximation due to the uniform law of large numbers. The two objectives are adversarial and depend on the parameters of both players. Therefore, the loss in Equation (3.13) is also called **adversarial loss**. As a consequence, the optimal solution is a point where none of the players could improve if the adversary does not change its parameters [80]. This point is called local Nash equilibrium in game theory [216].

To approximate the local Nash equilibrium in practice the two neural networks are trained alternately in the known iterative manner based on training batches (cf. Section 2.2.4). Therefore, the objective is reformulated with respect to a training dataset of size N

$$\hat{\mathcal{L}}^{\text{GAN}}(D_{\boldsymbol{\vartheta}}, G_{\boldsymbol{\theta}}) = \frac{1}{N}\sum_{n=1}^{N}[\log D_{\boldsymbol{\vartheta}}(\boldsymbol{x_n})] + \frac{1}{N}\sum_{n=1}^{N}[\log(1 - D_{\boldsymbol{\vartheta}}(G_{\boldsymbol{\theta}}(\boldsymbol{z_n})))]. \tag{3.14}$$

The discriminator is trained for j iterations with a fixed generator $G_{\hat{\boldsymbol{\theta}}}$. This leads to a discriminator $D_{\hat{\boldsymbol{\vartheta}}}$. Thereafter, the generator is trained for one iteration with the updated but fixed discriminator $D_{\hat{\boldsymbol{\vartheta}}}$. Due to the concatenated mapping $D_{\boldsymbol{\vartheta}} \circ G_{\boldsymbol{\theta}}$ the parameters of the generator are updated during backpropagation with the feedback from the discriminator. This procedure is repeated until a final amount of training steps is reached. If the discriminator $D_{\hat{\boldsymbol{\vartheta}}}$ distinguishes the data source remarkably well early in the training phase, this often leads to vanishing gradients. This results in little feedback for the generator how to improve to fool its adversary. Consequently, it is often sufficient to set $j = 1$ [79]. In Figure 3.2 we show 16 handwritten digits which are generated using a GAN. The GAN learns to transform the noise into handwritten digits iteratively. After a few epochs of training the generator learned to concentrate white pixels in the center



(a) Input noise (b) insufficiently trained (c) fully trained

Figure 3.2: Training stages of a GAN trained on MNIST, a dataset of handwritten digits.

of the image but the pixel blobs barely resemble realistic digits as shown in Figure 3.2b. After training for 198 epochs, this GAN generates diverse and clearly recognizable digits (Figure 3.2c).

In the following, we assume that both probability measures are absolutely continuous with respect to the same measure λ and have positive density functions. That means, they both can be expressed with the help of a positive probability density function by

$$dG_{\boldsymbol{\theta}*}\nu = p_{G_{\boldsymbol{\theta}}} d\lambda \tag{3.15}$$

and

$$d\mu_{\text{data}} = p_{\text{data}} d\lambda. \tag{3.16}$$

Particularly, let λ be the Lebesgue measure for all the following investigations. Goodfellow et al. showed in [79], that theoretically, meaning generator $G_{\boldsymbol{\theta}} = G$ and discriminator $D_{\boldsymbol{\vartheta}} = D$ have infinite capacity and are not restricted to a limited parametric model class, the Nash equilibrium of Equation (3.13) is given by $p_G = p_{\text{data}}$. That means, that the generator adequately mimics the true data distribution. The main reason for this is that minimizing the generator loss equals minimizing the Jensen-Shannon distance between p_G and p_{data}. To see this, we first study the objective in the inner loop (the discriminator) with a fixed but arbitrary generator G. Rewriting the training objective (3.11) with respect to p_G as defined in Equation (3.15) leads to

$$\begin{aligned}
&\sup_D \mathbb{E}_{X\sim\mu_{\text{data}}}[\log D(X)] + \mathbb{E}_{Z\sim\nu}[\log(1 - D(G(Z)))] \\
&= \int_{\mathcal{X}} [\log D(\boldsymbol{x})] p_{\text{data}}(\boldsymbol{x}) d\lambda(\boldsymbol{x}) + \int_{\mathcal{X}} [\log(1 - D(G(\boldsymbol{x})))] p_G(\boldsymbol{x}) d\lambda(\boldsymbol{x}) \\
&= \int_{\mathcal{X}} \left([\log D(\boldsymbol{x})] p_{\text{data}}(\boldsymbol{x}) + [\log(1 - D(G(\boldsymbol{x})))] p_G(\boldsymbol{x}) \right) d\lambda(\boldsymbol{x})
\end{aligned} \tag{3.17}$$

The integrand has the form

$$y \mapsto a\log(y) + b\log(1-y), \qquad a, b \in \mathbb{R}. \tag{3.18}$$

As the objective in Equation (3.17) is maximal if its integrand is maximal, it is enough to solve

$$0 = \frac{\partial(a\log(y) + b\log(1-y))}{\partial y} = \frac{a}{y} - \frac{b}{1-y} \quad \Rightarrow y^* = \frac{a}{a+b} \tag{3.19}$$

and to see that

$$\frac{\partial^2(a\log(y) + b\log(1-y))}{\partial y^2} = -\frac{a}{y^2} - \frac{b}{(1-y)^2} \leq 0. \tag{3.20}$$

Therefore, for all $a, b > 0$, $y^* = \frac{a}{a+b}$ is the global maximum of Equation (3.18) for $y \in (0,1)$. Given that p_{data} and p_G are probability densities and $D(\mathcal{X}) \subseteq (0,1)$ we can conclude that

$$D_G^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})} \tag{3.21}$$

is the optimal discriminator for G up to ν null sets [79]. Given the optimal discriminator the second training objective reduces to

$$\begin{aligned}
\mathcal{L}(G) =& \max_D \mathcal{L}^{\text{GAN}}(D,G) = \mathbb{E}_{X\sim\mu_{\text{data}}}[\log D_G^*(X)] + \mathbb{E}_{X\sim G_*\nu}[\log(1 - D_G^*(X))] \\
=& \mathbb{E}_{\boldsymbol{x}\sim\lambda(\boldsymbol{x})}\left[\log\left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right) p_{\text{data}}(\boldsymbol{x})\right] \\
&+ \mathbb{E}_{\boldsymbol{x}\sim\lambda(\boldsymbol{x})}\left[\log\left(1 - \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right) p_G(\boldsymbol{x})\right] \\
\overset{*}{=}& \mathbb{E}_{\boldsymbol{x}\sim\lambda(\boldsymbol{x})}\left[\log\left(\frac{2p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right) p_{\text{data}}(\boldsymbol{x})\right] \\
&+ \mathbb{E}_{\boldsymbol{x}\sim\lambda(\boldsymbol{x})}\left[\log\left(\frac{2p_G(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}\right) p_G(\boldsymbol{x})\right] - \log(4), \\
=& \mathfrak{d}_{\text{KL}}\left(p_{\text{data}}(\boldsymbol{x}) \middle\| \frac{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}{2}\right) + \mathfrak{d}_{\text{KL}}\left(p_G(\boldsymbol{x}) \middle\| \frac{p_{\text{data}}(\boldsymbol{x}) + p_G(\boldsymbol{x})}{2}\right) - \log(4)
\end{aligned} \tag{3.22}$$

which needs to be minimized with respect to G. In the second last equation $(*)$ we utilize a zero addition with $\pm\log(4)$ and exploit the fact that

$$\log(4) = \mathbb{E}_{\boldsymbol{x}\sim\lambda(\boldsymbol{x})}(\log(2)) + \mathbb{E}_{\boldsymbol{x}\sim\lambda(\boldsymbol{x})}(\log(2)). \tag{3.23}$$

With this the objective in Equation (3.22) can be rewritten as Jensen-Shannon distance:

$$\mathcal{L}(G) = -\log(4) + 2 \cdot \mathfrak{d}_{\text{JS}}(p_{\text{data}} || p_G). \tag{3.24}$$

Since $\mathfrak{d}_{\mathrm{JS}}$ is a divergence

$$\mathfrak{d}_{\mathrm{JS}}(p_{\mathrm{data}}||p_G) \geq 0 \text{ and } \mathfrak{d}_{\mathrm{JS}}(p_{\mathrm{data}}||p_G) = 0 \Leftrightarrow p_{\mathrm{data}} = p_G \tag{3.25}$$

and thus $\mathcal{L}(G)$ is minimal for

$$p_{\mathrm{data}} = p_G \tag{3.26}$$

up to ν null sets with a global minimum of $-\log(4)$. Plugging $p_{\mathrm{data}} = p_G$ in Equation (3.21) leads to the optimal discriminator $D^*_{G^*}(\boldsymbol{x}) = \frac{1}{2}$, thus assigning each source equal probability as it cannot distinguish between them.

Assuming μ_{data} and $G_*\nu$ have positive density functions, the estimation error made by this approach can be decomposed into a model error of the generator

$$\varepsilon_{G,\mathrm{model}} \coloneqq \inf_{G \in \mathcal{H}^G} \mathfrak{d}_{\mathrm{JS}}(\mu_{\mathrm{data}}, G_*\nu) \tag{3.27}$$

a model error of the discriminator

$$\varepsilon_{D,\mathrm{model}} \coloneqq \sup_{G \in \mathcal{H}^G} \inf_{D \in \mathcal{H}^D} \mathcal{L}^{\mathrm{GAN}}(D^*_G, G) - \mathcal{L}^{\mathrm{GAN}}(D, G) \tag{3.28}$$

and a sampling error

$$\varepsilon_{\mathrm{sample}}(N) \coloneqq \sup_{D \in \mathcal{H}^D,\, G \in \mathcal{H}^G} \left| \mathcal{L}^{\mathrm{GAN}}(D, G) - \hat{\mathcal{L}}^{\mathrm{GAN}}(D, G) \right| \tag{3.29}$$

where $\mathcal{H}^G$ or $\mathcal{H}^D$ denotes a hypothesis space of candidate functions for the generator or the discriminator, respectively. The error decomposes into

$$\mathfrak{d}_{\mathrm{JS}}(\mu_{\mathrm{data}}, \hat{G}_*\nu) \leq \varepsilon_{G,\mathrm{model}} + \varepsilon_{D,\mathrm{model}} + 2\varepsilon_{\mathrm{sample}}(N) \tag{3.30}$$

as proven in [82, Theo. 24.2]. Asatryan et al. further showed in [12] that the generator model error vanishes if Hölder regularities are assumed for the density function p_{data}. Let $p_{\mathrm{data}} = \frac{d\mu_{\mathrm{data}}}{d\lambda}$ be in $\mathcal{C}^{k,\alpha}([0,1]^d)$ for some $\alpha \in (0,1]$ and $k \geq 1$. Here, $\mathcal{C}^{k,\alpha}([0,1]^d)$ denotes the space of k-times differentiable α-Hölder functions [4, 1.29]. Thus, these are all functions $f \in \mathcal{C}^k((0,1)^d)$ with a uniformly continuous k-th derivative on $(0,1)^d$ such that

$$\|f\|_{\mathcal{C}^{k,\alpha}([0,1]^d)} \coloneqq \max_{|\beta| \leq k} \sup_{\boldsymbol{x} \in [0,1]^d} |D_\beta f(\boldsymbol{x})| + \max_{|\beta| = k} \sup_{\substack{\boldsymbol{x}, \tilde{\boldsymbol{x}} \in (0,1)^d \\ \boldsymbol{x} \neq \tilde{\boldsymbol{x}}}} \frac{|D_\beta f(\boldsymbol{x}) - D_\beta f(\tilde{\boldsymbol{x}})|}{|\boldsymbol{x} - \tilde{\boldsymbol{x}}|^\alpha} < \infty, \tag{3.31}$$

where $\beta \in \mathbb{N}^d$ is a multi-index with $|\beta| = \sum_{i=1}^d \beta_i$ and $D_\beta f(\boldsymbol{x}) \coloneqq \frac{\partial^{|\beta|} f(\boldsymbol{x})}{\partial x_1^{\beta_1} \cdots \partial x_d^{\beta_d}}$. Assume additionally that p_{data} is strictly bounded away from zero, i.e.,

$$\epsilon \coloneqq \min_{\boldsymbol{x} \in [0,1]^d} p_{\mathrm{data}}(\boldsymbol{x}) > 0. \tag{3.32}$$

Under these assumptions[24], there exists a bijective generator

$$G \in \mathcal{C}^{k,\alpha}([0,1]^d) \tag{3.33}$$

with

$$\|G\|_{\mathcal{C}^{k,\alpha}} \leq \mathfrak{K} \text{ and } \|G^{-1}\|_{\mathcal{C}^{k,\alpha}} \leq \hat{\mathfrak{K}} \tag{3.34}$$

for sufficiently large $\mathfrak{K}, \hat{\mathfrak{K}} > 0$, such that

$$\mu_{\text{data}} = G_* \nu. \tag{3.35}$$

This implies that $\varepsilon_{\text{model}} = 0$. We refer to [12] for the proof. In practice, the assumption of Hölder generators and discriminators is not necessarily fulfilled by default, as both are implemented by neural networks with a fixed network architecture. Nevertheless, neural networks have been proven to be universal approximators (see Section 2.2.3). This means that the Hölder functions can be approximated to any degree of precision given neural networks with enough capacity in terms of width and depth. Despite the theoretical realizability of an optimal generator as stated in Equation (3.35), numerical issues arise when training GANs.

In contrast to the training of vision tasks presented in Section 2.3, GAN training is often slow, unstable and convergence of the training is difficult to assess even after hundreds of iterations [273]. The alternating learning strategy makes it difficult to interpret the adversarial loss [273] as the objective function changes with each update of the generator or of the discriminator. Furthermore, GAN training is sensitive to hyperparameters and a balanced neural network performance during the optimization process is needed to have meaningful gradients to update the adversarial player [232]. Convergence to the Nash equilibrium is not guaranteed in practice and often the performance of generator and discriminator oscillates without reaching a fix point [174]. One source of convergence difficulties is the highly non-convex optimization problem such that the learning algorithms get stuck in local minima. A frequently encountered problem during GAN training is **mode collapse** [77]. This means the generator learns images which are indistinguishable for the discriminator, but the results of the generator do not cover the entire data generating distribution but only one or limited modes of it. The extreme case is that the generator learns one perfect standard image and has no variety although receiving stochastic input. As a more general example when we consider handwritten digits from 0 to 9 as training samples, a GAN is mode collapsed when its output variety is limited to one or a few (potentially always differently looking) specific digit(s). Concepts to encourage training stabilization include, e.g., noise injection to the discriminator input [10, 65] and different kinds of minibatch discrimination. The latter can be accomplished by tracking a running average over previous model parameters [232] or

[24] The assumptions can be fulfilled by regularizing the input data with noise $\mathfrak{N} \sim \mathcal{N}(0, \epsilon^2 \mathbb{1})$. In detail according to [12], a mapping $X \mapsto (X + \mathfrak{N}) \bmod \mathbb{1}$ is applied to the data which resembles embedding $[0,1]^d$ in $\mathbb{R}^d$, convolving it with noise drawn from the normal distributed with covariance $\epsilon^2 \mathbb{1}$ and re-projecting to $[0,1]^d$.

buffering previously generated images to let the discriminator compare against a history of generated images rather than constantly changing images [251].

3.1.2 GAN Variants

In the 'vanilla' version by Goodfellow et al., the neural networks for the generator and discriminator are MLPs. Later those networks were replaced by, e.g., CNNs with (fractionally-strided) convolutions (cf. Section 2.1.2 and Equation (2.24)) and normalization layers (Section 2.2.5) as well as different activation functions Section 2.1.1.1. Firstly, these adaptations were implemented by the **DCGAN** [209]. Later architectures proved successful in other vision tasks had been adopted like the VGG architecture [169]. Besides multiple architecture suggestions, also different loss functions were proposed. For the **Wasserstein GAN** [11] and its variations [86] a critic instead of a discriminator was proposed to tackle the problem of vanishing gradients by using the Wasserstein-1 distance as loss which opens a theoretical link to optimal transport theory. The advantage of using the Wasserstein-1 distance as a loss is that it provides meaningful gradients even when the two distributions have no common support. Apart from considering a critic motivated from optimal transport, also other losses motivated by additional divergence measures were proposed [193]. In the **Least Square GAN** (LSGAN) [169] for example the cross entropy loss is replaced by a least square loss to mitigate the vanishing gradient. In contrast to the cross entropy loss, the least square loss has pronounced gradients nearly everywhere. Furthermore, the loss fosters the generation of data more closely to the real data, as samples classified correctly but located far from the decision boundary are penalized as well and therefore all samples are pulled towards the decision boundary which should pass through the real images. Let $\mathfrak{a} \in \mathbb{Z}$ denote the label the discriminator should assign to fake data and $\mathfrak{b} \in \mathbb{Z}$ the label for data stemming from the true data generating distribution. The loss for the LSGAN is then defined by

$$\min_D \mathcal{L}^{\text{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{X\sim\mu_{\text{data}}}[(D(X)-\mathfrak{b})^2] + \frac{1}{2}\mathbb{E}_{Z\sim\nu}[(D(G(Z))-\mathfrak{a})^2] \tag{3.36}$$

for the discriminator and

$$\min_G \mathcal{L}^{\text{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{Z\sim\nu}[(D(G(Z))-\mathfrak{c})^2] \tag{3.37}$$

for the generator, where $\mathfrak{c} \in \mathbb{Z}$ represents the label the generator intends for the discriminator when fed with generated data. The authors of [169] proved that the resulting GAN objective relates to the Pearson χ^2 divergence when $\mathfrak{b}-\mathfrak{c}=1$ and $\mathfrak{b}-\mathfrak{a}=2$. This can be realized by $\mathfrak{a}=-1, \mathfrak{b}=1, \mathfrak{c}=0$ and leads to the relation

$$\min_{D,G} \mathcal{L}^{\text{LSGAN}}(D,G) = \min_{D,G} \chi^2_{\text{Pearson}}(p_{\text{data}} + p_G \| 2p_G). \tag{3.38}$$

Not motivated by the Pearson χ^2 divergence but the 0-1 class encoding as introduced in Section 3.1.1 by setting $\mathfrak{b} = \mathfrak{c} = 1$ and $\mathfrak{a} = 0$ yields comparable results and is often used in practice (see Section 3.2.2). An overview over further GAN architecture and loss variants applied in computer vision and computer graphics was published by Wang et al. [290].

Besides approximating the data generating distribution μ_{data} of the random variable X describing the data generating process, the generator and discriminator can also be conditioned on additional information. Let $(\Upsilon, \mathcal{A}_\Upsilon, \mu_\Upsilon)$ be a probability measure space which models the additional information. As a consequence, a conditional data distribution $\mu_{\text{data}|Y=y}$ for a realization y of the random variable $Y \sim \mu_\Upsilon$ on Υ can be modelled by **conditional GANs** when data pairs $(\boldsymbol{x}, y)$ are available [177]. Data generation can for example be conditioned on label information such that $y \in \Upsilon = \{1, \dots, C\}$. With this approach, class-specific data can be generated. As a consequence, mode collapse is reduced as each class needs to be learned. Furthermore, when trained to learn the conditional distribution, GANs (more precisely the discriminator) can also serve as a classifier. The minimax game extends to

$$\min_{G_{\boldsymbol{\theta}}} \max_{D_{\boldsymbol{\vartheta}}} \mathbb{E}_{\substack{X \sim \mu_{\text{data}} \\ Y \sim \mu_\Upsilon}} [\log D_{\boldsymbol{\vartheta}}(X|Y)] + \mathbb{E}_{\substack{Z \sim \nu \\ Y \sim \mu_\Upsilon}} [\log(1 - D_{\boldsymbol{\vartheta}}(G_{\boldsymbol{\theta}}(Z|Y)))]. \tag{3.39}$$

The conditioning is realized in the parametric setup where the generator and the discriminator are given as neural networks by enlarging the input of the generator to $(\boldsymbol{z}, y)$ as well as the input of the discriminator by the information $(\boldsymbol{x}, y)$. Conditioning could also be done with respect to other information such as domain information, e.g., night, day, winter, summer.

3.1.3 Evaluation Metrics

Evaluation metrics of generative models need to cover three main aspects [184]:

- sample quality (point-wise alignment with the distribution)
- sample diversity (coverage of all modes of the distribution)
- generalization (coverage of the distribution outside the training data)

In contrast to vision tasks like classification or semantic segmentation, the evaluation of generative models has no standard metric. In addition, there exists no method which covers all three aspects simultaneously. As the range of application of generative models varies greatly, the measure of the model performance depends on the context of the application. Particularly models which perform well according to one metric can fail according to another [268]. Even though one can think of using distance measures like the KL-divergence or the Jensen-Shannon distance since the underlying task is to approximate a distribution, they can only serve as a metric for generative models with explicit

density approximation. For GANs which implicitly model the distribution such a measure is impractical. Furthermore, the true data generating distribution is unknown. For images, distance measures, e.g., the L_1 or L_2-norm on pixel-level are sometimes used to score the performance of image reconstruction, image denoising and in-painting. This has the drawback that pixel-level measures are sensitive to color, brightness and spatial shifts and therefore do not correspond to the perception of humans. As a consequence, another metric to measure the quality of generated data is to ask humans to distinguish between generated images and images from the distribution to approximate, i.e., asking humans to take the role of a discriminator [232]. A drawback of this method is that humans cannot restrict themselves from learning during the evaluation process. Thus, their evaluation score differs over time. As a consequence, Salimans et al. propose the Inception Score $\mathrm{IS}(G_{\hat{\boldsymbol{\theta}}}) \coloneqq \exp\left(\mathbb{E}_{p_{G_{\hat{\boldsymbol{\theta}}}}(\boldsymbol{z})} \mathfrak{d}_{\mathrm{KL}}\left(p(y|G_{\hat{\boldsymbol{\theta}}}(\boldsymbol{z})) \,||\, \int p(y|G_{\boldsymbol{\theta}}(\boldsymbol{z}))p_{G_{\hat{\boldsymbol{\theta}}}}(\boldsymbol{z})\right)\right)$ [232] based on the conditional probability $p(y|G_{\boldsymbol{\theta}}(\boldsymbol{z}))$ given by a ImageNet pre-trained inception model [264] fed with generated images. This always leads to the same score when repeating the evaluation but due to the dependence on the pre-trained model this measure is biased to the domain of real world and everyday objects and less suitable for synthetic or abstract image domains [18]. Further metrics are the Structural Similarity Index (SSIM) [289] the Learned Perceptual Image Patch Similarity (LPIPS) [318] and Fréchet Inception Distance (FID) [97]. When GANs or their output are used for a downstream task like classification, the performance of the GAN is measured by the metrics of the specific downstream task introduced in Sections 2.3.1.4 and 2.3.2.4. Each of the above-mentioned evaluation methods has its own advantages and disadvantages and none of them cover all the three desired aspects. We refer to [184] for more details. A detailed list of evaluation metrics is studied in [25].

3.2 Image-to-Image Translation

Besides generating images from an unknown distribution based on noise, where GANs have proven successful, another relevant task is to translate images from one representation to another. Mapping one image to another on pixel-level is referred to image-to-image (I2I) translation [114]. Applications of I2I translation approaches in computer graphics and computer vision range from image synthesis, image or video restoration, domain adaptation, image enhancement, to style transfer and aligning images with the help of image registration. A review on these applications is given by Pang et al. in [198]. In this thesis, we focus on its application to domain adaptation as it can be seen as a special case of domain adaptation on input level (cf. Section 2.4.4) where no downstream task network is considered, and the adaption is limited to the feature space distribution[25].

[25] To this end, we weaken the domain notation and omit the label space $\mathcal{Y}$.

Thereby, an image is transferred from a source domain $\mathcal{S} = (\mathcal{X}_S, \mu_S)$ to a visually different target domain $\mathcal{T} = (\mathcal{X}_T, \mu_T)$ by preserving the overall scene content but changing the style according to the new domain. Formally, the aim is to find a mapping

$$G : \mathcal{X}_S \to \mathcal{X}_T, \boldsymbol{x_S} \mapsto G(\boldsymbol{x_S}) \tag{3.40}$$

such that

$$G(\boldsymbol{x_S}) \in \mathcal{X}_T \text{ and } G(X_S) \sim \mu_T \quad \text{ for } X_S \sim \mu_S \tag{3.41}$$

and the overall characteristics of the scene are preserved.

3.2.1 Pix-to-Pix

Data from the two domains can be available as paired images, where the same scene is represented in each domain, denoted as supervised I2I, or in an unpaired manner, denoted as unsupervised I2I. Examples for paired data which are easy to collect are photos and their corresponding edge maps or grayscale images and their colored RGB counterparts. More manual effort is needed to get pairs of label masks and their corresponding photos. When image pairs from the source and target domain are given, variants of conditional GANs like pix2pix [114] or pix2pixHD [285] can be used to perform image-to-image translation [114, 184]. The objective translates into

$$\mathcal{L}^{\text{condGAN}}(G_{\boldsymbol{\theta}}, D_{\boldsymbol{\vartheta}}) = \mathbb{E}_{\substack{X \sim \mu_T \\ X_S \sim \mu_S}} [\log D_{\boldsymbol{\vartheta}}(X_T | X_S)] + \mathbb{E}_{\substack{Z \sim \nu \\ X_S \sim \mu_S}} [\log(1 - D_{\boldsymbol{\vartheta}}(G_{\boldsymbol{\theta}}(Z | X_S)))] \tag{3.42}$$

where $Z \sim \nu$ is a random noise vector, a realization of $X_S \sim \mu_S$ is an image to condition on, e.g., a label mask, and a realization of $X_T \sim \mu_T$ is the desired target image. An L_1 loss between the output and the ground truth is additionally added to the generator loss. This helps to preserve the input scene. The full training objective is given by

$$\min_{G_{\boldsymbol{\theta}}} \max_{D_{\boldsymbol{\vartheta}}} \mathcal{L}^{\text{condGAN}}(G_{\boldsymbol{\theta}}, D_{\boldsymbol{\vartheta}}) + \gamma \mathbb{E}_{X_T, X_S, Z}[\| X_T - G_{\boldsymbol{\theta}}(Z | X_S) \|_1]. \tag{3.43}$$

Nevertheless, paired data of real world scenes including movable objects or actors in different seasons or lighting conditions are hard or impossible to collect or to generate. As a consequence, so-called unsupervised methods were developed which, e.g., include specific losses to enable a translation based on unpaired data (see for example DiscoGAN [130] or DualGAN [309]). A prominent approach is CycleGAN where a cycle consistency loss leads to a consistent mapping of an image from one domain to the other [324]. We discuss this approach in more detail in the following section.

3.2.2 CycleGAN

As seen in Section 3.1.1, GANs can realize mappings $G : \mathcal{X}_S \to \mathcal{X}_T$ such that $G_*\mu_S = \mu_T$. Nevertheless, this mapping does not ensure to consistently map one image to another by preserving the overall scene. On the contrary, infinitely many mappings exist, for which $G_*\mu_S = \mu_T$ holds but no meaningful correlation between individual image pairs $(\boldsymbol{x_S}, G(\boldsymbol{x_S}))$ can be observed. Let X_S and X_T, denote random variables describing a sample realization drawn from the data generating distribution μ_S and μ_T, respectively. To model the context preservation, a cycle consistency between GANs was proposed in the publication of CycleGAN [324] which serves as the basis for the following section. For CycleGAN, additionally to the generator $G = G_{\mathcal{S}\to\mathcal{T}} : (\mathcal{X}_S, \mu_S) \to (\mathcal{X}_T, \mu_T)$, a second generator $G_{\mathcal{T}\to\mathcal{S}} : (\mathcal{X}_T, \mu_T) \to (\mathcal{X}_S, \mu_S)$ is introduced. The two mappings are coupled by the constraints

$$\begin{aligned} (G_{\mathcal{S}\to\mathcal{T}} \circ G_{\mathcal{T}\to\mathcal{S}})(\boldsymbol{x_T}) &= \boldsymbol{x_T} \\ (G_{\mathcal{T}\to\mathcal{S}} \circ G_{\mathcal{S}\to\mathcal{T}})(\boldsymbol{x_S}) &= \boldsymbol{x_S} \\ G_{\mathcal{S}\to\mathcal{T}}(\boldsymbol{x_S}) &= \boldsymbol{x_T} \\ G_{\mathcal{T}\to\mathcal{S}}(\boldsymbol{x_T}) &= \boldsymbol{x_S}, \end{aligned} \tag{3.44}$$

for all $\boldsymbol{x_S} \in \mathcal{X}_S$ and $\boldsymbol{x_T} \in \mathcal{X}_T$. Thus, the image-to-image translation by CycleGAN is realized by approximating two data generating measures. In contrast to vanilla GANs, where a noise measure is pushed forward, here the respective other domain with its data generating distribution serves as input space. The constraints in Equation (3.44) enforce the two mappings to be bijective and the inverse of each other. This is stimulated by the **cycle consistency loss**

$$\begin{aligned} \mathcal{L}^{\text{cyc}}(G_{\mathcal{T}\to\mathcal{S}}, G_{\mathcal{S}\to\mathcal{T}}) = \mathbb{E}_{X_T\sim\mu_T}\left[\left\| G_{\mathcal{S}\to\mathcal{T}}\Big(G_{\mathcal{T}\to\mathcal{S}}(X_T)\Big) - X_T \right\|_1\right] \\ + \ \mathbb{E}_{X_S\sim\mu_S}\left[\left\| G_{\mathcal{T}\to\mathcal{S}}\Big(G_{\mathcal{S}\to\mathcal{T}}(X_S)\Big) - X_S \right\|_1\right]. \end{aligned} \tag{3.45}$$

The first summand fosters the so-called forward cycle consistency and the second summand enforces backward cycle consistency. The forward cycle consistency is illustrated in Figure 3.3. Even though the L_1-norm seems too strict for image comparisons, the authors found no improvement by alternatively using an adversarial loss to encourage $(G_{\mathcal{S}\to\mathcal{T}} \circ G_{\mathcal{T}\to\mathcal{S}})(\boldsymbol{x_T}) = \boldsymbol{x_T}$. Nevertheless, the adversarial loss is used to learn the general pushforward measure which approximates the target and source domain data distribution, respectively, i.e., $G_{\mathcal{S}\to\mathcal{T}*}\mu_S \approx \mu_T$ and $G_{\mathcal{T}\to\mathcal{S}*}\mu_T \approx \mu_S$. Let $D_{\mathcal{S}} : \mathcal{X}_S \to (0,1)$ denote the discriminator on the domain $\mathcal{S}$ and $D_{\mathcal{T}} : \mathcal{X}_T \to (0,1)$ the discriminator distinguishing between data sampled from $G_{\mathcal{S}\to\mathcal{T}*}\mu_S$ and the real data distribution μ_T on domain $\mathcal{T}$. In contrast to the cross entropy loss in the vanilla GAN setup (cf. Equation (3.13)), the least square loss as proposed in LSGAN (cf. Equations (3.36) and (3.37)) with $\mathfrak{b} = \mathfrak{c} = 1$ and $\mathfrak{a} = 0$ is used in the CycleGAN implementation. With this we can formulate the

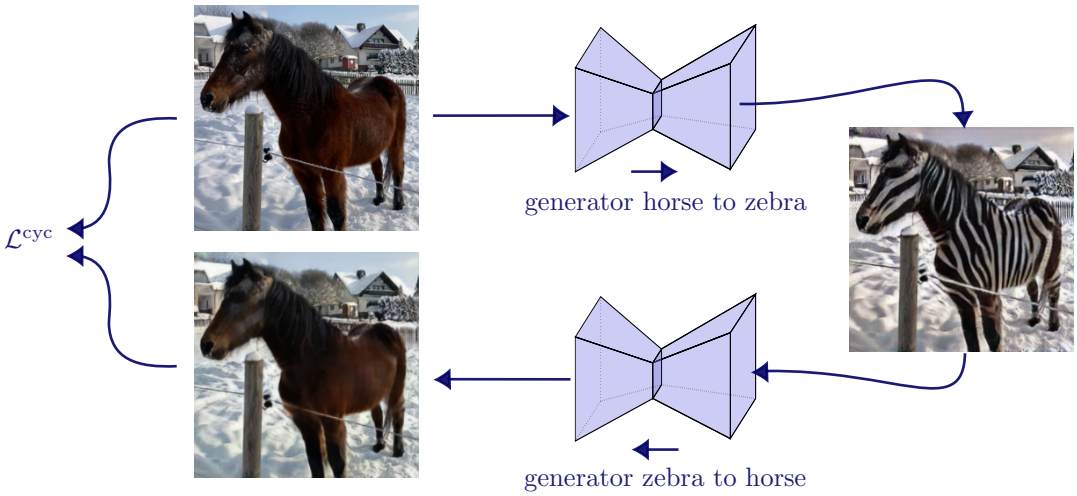


Figure 3.3: Schematic visualization of the forward cycle consistency loss $\mathcal{L}^{\text{cyc}}$. Zooming in is encouraged to view details between the original image and its reconstruction.

generator objectives

$$\mathcal{L}^{G_{\mathcal{S}\to\mathcal{T}}}(G_{\mathcal{S}\to\mathcal{T}}) = \mathbb{E}_{X_S\sim\mu_S}\left[\left(D_{\mathcal{T}}\Big(G_{\mathcal{S}\to\mathcal{T}}(X_S)\Big) - 1\right)^2\right] \tag{3.46}$$

and

$$\mathcal{L}^{G_{\mathcal{T}\to\mathcal{S}}}(G_{\mathcal{T}\to\mathcal{S}}) = \mathbb{E}_{X_T\sim\mu_T}\left[\left(D_{\mathcal{S}}\Big(G_{\mathcal{T}\to\mathcal{S}}(X_T)\Big) - 1\right)^2\right]. \tag{3.47}$$

The adversarial loss for the discriminators is given by

$$\mathcal{L}^{D_{\mathcal{S}}}(D_{\mathcal{S}}) = \mathbb{E}_{X_S\sim\mu_S}\left[\Big(D_{\mathcal{S}}(X_S) - 1\Big)^2\right] + \mathbb{E}_{X_T\sim\mu_T}\left[D_{\mathcal{S}}\Big(G_{\mathcal{T}\to\mathcal{S}}(X_T)\Big)^2\right] \tag{3.48}$$

and

$$\mathcal{L}^{D_{\mathcal{T}}}(D_{\mathcal{T}}) = \mathbb{E}_{X_T\sim\mu_T}\left[\Big(D_{\mathcal{T}}(X_T) - 1\Big)^2\right] + \mathbb{E}_{X_S\sim\mu_S}\left[D_{\mathcal{T}}\Big(G_{\mathcal{S}\to\mathcal{T}}(X_S)\Big)^2\right] \tag{3.49}$$

Lastly, an **identity preserving loss** is considered to hamper transformations of images which already belong to the target distribution. That means, it encourages the generators to act as identity mapping on samples that are already in the target domain, such that $G_{S\to T}(\boldsymbol{x_T}) = \boldsymbol{x_T}$ and $G_{T\to S}(\boldsymbol{x_S}) = \boldsymbol{x_S}$.

$$\begin{aligned}\mathcal{L}^{\text{id}}(G_{\mathcal{T}\to\mathcal{S}}, G_{\mathcal{S}\to\mathcal{T}}) = \mathbb{E}_{X_S\sim\mu_S}&\left[\Big\|G_{\mathcal{T}\to\mathcal{S}}(X_S) - X_S\Big\|_1\right] \\ + \mathbb{E}_{X_T\sim\mu_T}&\left[\Big\|G_{\mathcal{S}\to\mathcal{T}}(X_T) - X_T\Big\|_1\right].\end{aligned} \tag{3.50}$$

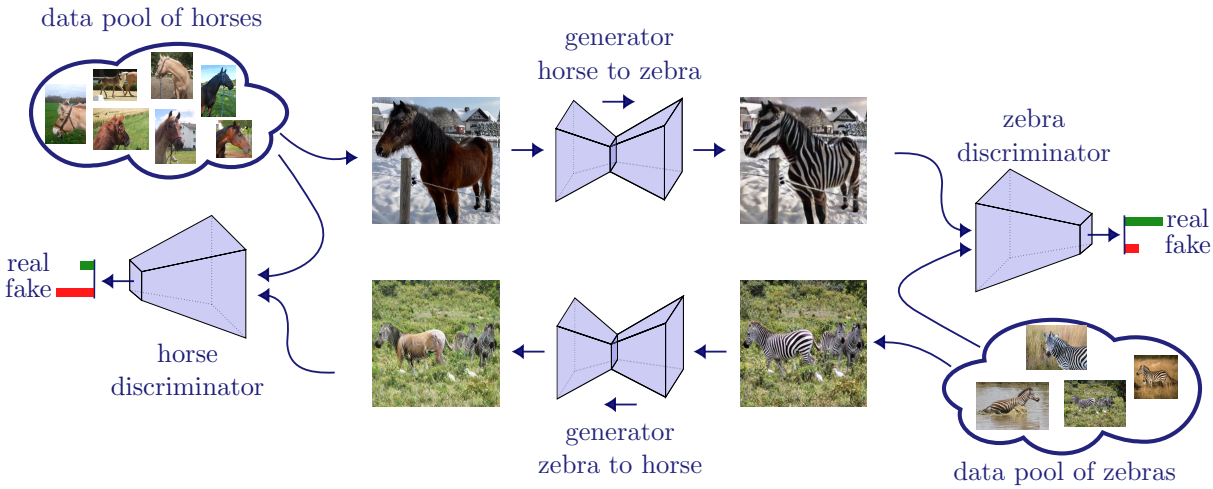


Figure 3.4: Schematic visualization of CycleGAN.

The overall training objective for the generative components in CycleGAN is a weighted sum of the losses in Equations (3.45), (3.46) and (3.50). Let $\gamma_{\text{cyc}}, \gamma_{\text{id}} \geq 0$ be scalars weighting the cycle consistency or the identity preserving loss, respectively. With this we define the objective

$$\mathcal{L}^{\text{Gen}} = \mathcal{L}^{G_{\mathcal{S}\to\mathcal{T}}} + \mathcal{L}^{G_{\mathcal{T}\to\mathcal{S}}} + \gamma_{\text{cyc}}\mathcal{L}^{\text{cyc}} + \gamma_{\text{cyc}}\gamma_{\text{id}}\mathcal{L}^{\text{id}}, \tag{3.51}$$

which is minimized with respect to $G_{\mathcal{S}\to\mathcal{T}}, G_{\mathcal{T}\to\mathcal{S}}, D_{\mathcal{S}}$ and $D_{\mathcal{T}}$. Additionally, the discriminator losses in the Equations (3.48) and (3.49) are minimized with respect to $D_{\mathcal{S}}$ and $D_{\mathcal{T}}$.

The implementation of the presented concept is realized with four neural networks, two serving as generators and two as discriminators. Their interaction is depicted in Figure 3.4. The generators are each realized by an FCN with 6 or 9 residual layers depending on the input size. It has an encoder-decoder structure where down- and upsampling operations are implemented with the help of strided and fractionally-strided convolutions. ReLUs are used as activation functions. For the discriminators a PatchGAN [114] is used. PatchGAN is a 5-layer CNN with LeakyReLU activation which operates with a receptive field of size 70×70 pixels. This leads to a feature map where each pixel represents the estimated probability that the 70×70 pixel seen from the input are generated or from the real dataset. The average over all patch predictions (i.e., the feature map) is taken to predict the source of the entire image. The reduced receptive field and the increased feedback due to a feature map prediction tend to reduce artifacts. As a consequence of the 'patch' discrimination, it is not enough for the generator to just specialize to a few image features, but it needs to produce valuable image features all over the image [251]. This discriminator architecture has several additional advantages. Firstly, it is independent of the input image size, making it applicable even if the input size varies between training and testing. Secondly, it requires fewer parameters to train

compared to a pixel-wise classification for the entire image.

As before, the loss is approximated by replacing the expected value with the help of the empirically computable arithmetic mean over the training datasets. The overall training concept is identical to that of GANs (see Section 3.1). That means the generators and discriminators are trained in an alternating manner. Additionally, some training stabilization techniques were applied. As proposed in [251], an image buffer, collecting generated images from the last 50 iterations, is used. Moreover, the learning rate is kept constant for the first half of the training and is linearly decreased thereafter. With this, CycleGAN learns a deterministic one-to-one mapping.

One limitation of the concept is that since no source of randomness is involved in the training process, except for the realization of the source and target dataset, CycleGAN cannot learn multi-modal outputs. Thus, an image is always mapped to the same image in the target domain without variation. This limitation was addressed by Almahairi et al. who published an augmented version of CycleGAN. They augment each domain with a latent representation of the missing information when translating between the two domains [5]. In their experiments, a simple noise augmentation was insufficient as the cycle consistency loss fosters the model to ignore the auxiliary variables [5,114]. Furthermore, image-to-image translation approaches relying on cycle consistency on pixel-level, including CycleGAN, are limited in object shape transformation [320]. According to Zhao et al., the constraint on pixel-level prevents the model to modify distinctly the shape of objects or remove large segments in the image. They propose to extend the input with a noise vector to achieve a multi-modal output and relax the constraints by replacing the cycle-consistency loss with an adversarial-consistency loss. This loss encourages generating similar but not identical images when images are transformed cyclically. Switching to a discriminator with dilated convolutions inspired by semantic segmentation is a different approach to improve shape deformation. Due to the dilated convolutions the discriminator has a broader view. Furthermore, the approach enables a per-pixel discrimination. Together with the enlarged field of view, this leads to a more context aware generator according to [76]. The noted limitations are not restrictive with respect to our research. On the contrary, the limited shape transformation enables us to use the same semantic segmentation masks for translated images in the target domain. Furthermore, a one-to-one mapping is sufficient for data generation for our domain adaptation approach, described in detail in Chapter 4.

Chapter 4

Semi-supervised Task Aware Image-to-Image Domain Adaptation

The content of the following chapter bases on the conference paper [185] and the accepted Springer Book section in the CCIS Series [186] which is in the formatting phase but unpublished at the time of publication of this thesis.

4.1 Introduction

For automatically understanding complex visual scenes from RGB images, semantic segmentation (Section 2.3.2) is a common but challenging task. The state-of-the-art results are achieved by deep neural networks [37, 161, 266]. These models need plenty of labeled images to generalize and react sensitive to domain shifts, i.e., situations when data during inference stems from a different source than the source of data during training. However, a manual label process on pixel-level detail is time and cost consuming and usually error-prone [47, 228]. To reduce the labeling cost, weakly- and semi-supervised methods were proposed (cf. Section 2.4.2). These methods use weak labels like bounding boxes for segmentation tasks or fewer labels as they can benefit from a pool of unlabeled data. In addition to that, active learning (Section 2.4.4.1) is used to limit the annotation cost [31, 46, 284]. Thereby only the most informative samples of an unlabeled pool are selected for manual annotation based on specific query strategies. However, they are often limited to scenarios captured in the real world and the annotation cost of weak labels still might be intractable [277]. On the other hand, in recent years simulations, especially of urban street scenes, were significantly improved [59, 296]. The advantage of synthetic data is that images generated by a computer simulation often come with labels for the semantic content for free. Training on synthetic data has the potential to build a well-performing network as plenty of data is available and diverse scenarios can

be generated which are rare or life-threatening in the real world. Furthermore, the creation of safety critical corner cases is unproblematic in the synthetic domain [139, 225]. However, neural networks do not generalize well to unseen domains [101]. Even if the model learns to generalize well on one domain (e.g., real world) it can fail completely on a different domain (e.g., synthetic) [296] or vice versa. The sensitivity of neural networks to the domain shift, makes this strategy problematic, but measures have been taken to favor the generalization throughout domains [101, 242].

In Section 2.4.4 we introduced the concept of domain adaptation (DA) which is used to mitigate the so-called domain shift [48] between one domain and another. In summary, DA aims to improve the model's performance on a target domain by transferring knowledge learned from a labeled source domain. It has become an active area of research in the context of deep learning [271] ranging from adaptation on feature level [277], adaptation on input level [26, 63, 100], self-training [171, 317], a combination thereof [129] to semi-supervised approaches [42]. In the field of DA, one distinguishes between the amount of labeled data available for the target domain. In unsupervised domain adaptation (UDA), no labels of the target domain are available, while semi-supervised domain adaptation (SSDA) grants access to a few labeled instances. These labels can either be available, or an oracle can be actively asked to reveal labels of specific instances, in which case we speak of active domain adaptation (cf. Section 2.4.4.1) [207]. Adapting on input level to the style of the target domain disregarding the downstream task at hand but preserving the overall scene is referred to image-to-image translation. For a formal introduction to I2I translation, we refer to Section 3.2. I2I translation using generative adversarial networks [79] is often used to change the graphical style of the source domain to that of the target domain [198]. Usually, synthetic training data is upgraded in this way and used for the training of networks that thereafter are deployed to the real domain [100].

In this work, however, we propose to revert this procedure and train a strong network in the synthetic domain serving as a more abstract representation. The hope is that such abstract representations are less sensitive to small perturbations. Besides, synthetic data is abundantly available and thus a classification or segmentation network can be very well-trained and tested. We refer to such networks as (downstream) task experts. Thereafter, we shift the out-of-domain input (real world) closer to the synthetic domain via a semi-supervised I2I approach based on CycleGAN (see Section 3.2.2) for mitigating the domain gap. In order to achieve that the image translation encodes the relevant features for the task expert, we employ some labeled data from the real domain (where we want to solve the downstream task) to train the CycleGAN (but not the task expert in the abstract domain) with a cross entropy loss based on the task expert's softmax output. In this way, we achieve task awareness of the I2I translation. Keeping the downstream task network fixed has multiple advantages: Firstly, the abstract representation is often less complex and therefore easier to train and requires potentially less network parameters to achieve outstanding performance compare to a real world setup. Secondly, the network is handled in a controllable environment so that

exhausted testing can be done to guarantee a certain in-domain model performance. To avoid a bias by pre-trained neural networks on data close to the target domain all our methods are based on training from scratch. We also extend the method to active domain adaptation, designing an active learning pipeline to find informative samples when selecting samples with ground truth for the semi-supervised component of our approach. Furthermore, we demonstrate experimentally that active learning can improve the performance of our method for classification as downstream task. Additionally, we compare our results to direct active learning on the real domain when only a limited amount of data can be queried. In the first part of the paper we show that a generator can be guided with task awareness to generate more meaningful inputs for semantic segmentation networks. Thereafter, we show that the choice of training data for a task aware generator can benefit from active learning queries. We thereby improve the data-driven intermediate representation which we establish due to the composite structure of our method without retraining the downstream task network. Thus, in contrast to many ADA methods, we query useful data points for the generator (instead of the downstream task neural network) to learn to generate inputs suitable for a synthetic expert even though the original image comes from e.g., the real domain.

Our main contributions are:

- we present a novel modular SSDA method for semantic segmentation guiding the generator of an I2I domain adaptation approach to a semantic segmentation task awareness. Thereby, our downstream task network does not need to be retrained.
- we demonstrate that our method is applicable to multiple complex domain adaptation tasks.
- we consider a pure domain separation in our analysis by using from-scratch-trained neural networks leading to a less biased domain gap.
- we extend our method to an ADA approach querying samples to train the generator of our SSDA method and show that this improves results in the classification setup.

Based on our knowledge this is the first time the generator of a GAN setup is guided with the help of a semantic segmentation network to focus on the downstream task and the input for a generator in a SSDA method is actively selected.

The remainder of this chapter is organized as follows: In Section 4.2 we review related approaches, particularly in the context of semi-supervised and active domain adaptation. It follows a detailed description of our new, modular SSDA method and its extension to an ADA method in Section 4.3. We evaluate our method on two complex domain adaptation tasks and three different datasets in Section 4.4, showing considerable improvements with only a few labeled data samples for the SSDA method and in the classification setup how an informative selection of labeled samples can further improve the results. Finally, we conclude and give an outlook to future work in Section 4.5.

4.2 Related Work

The following section, taken from [186], places our work in a broader context. In our work we focus on three main topics: DA with I2I, semi-supervised learning using GANs and active semi-supervised domain adaptation. Independently of DA, I2I has been used widely in computer vision tasks since its introduction by Hertzmann et al. [96] and is dominated by generative approaches like GANs [198]. Thereby the data generation distribution is modeled indirectly by a generator-discriminator interaction. When paired data between source and target domain is available, pix-to-pix approaches yield convincing results [114,285]. To overcome the pair-problem, Zhu et al. proposed CycleGAN [324] where scene alignment is achieved by the help of a cycle consistency loss and unpaired data can be used. In the context of I2I the latter case is called 'unsupervised I2I' and the former is categorized as 'supervised I2I'. A combination of these two approaches in a semi-supervised manner was proposed by Shukla et al. [252] using only a few paired data points to tackle image-to-label transformation in the context of semantic segmentation.

In contrast to this, taxonomy changes slightly when using I2I in the context of DA. Hereby the amount of available labels in the target domain is decisive as explained in Section 2.4.4. Throughout this chapter we use I2I in DA taxonomy. In addition to solely cover the data generation distribution of the target domain, a semantic consistency is pursued with I2I in DA approaches as they are combined with and measured by the performance of a downstream task. Therefore, CyCada [100], SUIT [155] and the work of Brehm et al. [26] incorporate the downstream task loss in an unsupervised manner to adapt the task network to the real domain. Taking advantage of the access to a few arbitrary but fixed labeled data points in the target domain multiple SSDA approaches for classification as downstream task were proposed [117, 131, 168, 231, 298].

Actively selecting data points to annotate is the purpose of AL. AL is an established method for reducing the annotation cost based on informed data selection strategies. A general overview over the concept and possible query strategies was given in Section 2.4.4.1 and can be found in [244]. AL queries for classification follow mainly two concepts: density or uncertainty quantification. While the first one aims at data diversity in feature space in sense of sample distance, uncertainty-based methods rely on the information that can be drawn from the prediction uncertainty of a probabilistic model (e.g., the one to train) [152, 297]. Established uncertainty-based methods include the probability margin [239], entropy and least confidence of a probabilistic model [284], dropout [67] and committee-based approaches (committee disagreement [245] or ensemble entropy [51]). Methods considering the data diversity as query indicator can be linked to clustering methods [189, 243], batch diversity [135] or can be combined with uncertainty methods [13]. An approach of combining GANs and active learning was published by Zhu and Bento [323], where they use GANs to actively generate training samples near to the models decision boundary which are then annotated and added to the training set of the model.

In recent years, first publications came up that describe the transfer of classical AL-strategies to domain adaptation. In contrast to SSDA where a random subset of target data is available with annotation, a limited amount of target data is actively selected for annotation in ADA to mitigate the domain gap by including these samples into the training dataset. Thus, the approaches take into account predictive uncertainty and targetness [301]. The approaches range from adversarial setups, where a domain adaptation network and a domain discriminator are combined to align the domains and query the most important data points based on entropy [261], to energy-based sampling [299]. Furthermore, calibration adaption to overcome potential overconfidence of a source model [301] combinations of transferred committee, uncertainty and domainness concepts [66] and uncertainty-weighted clustering to identify samples which are diverse in feature space and have a high model uncertainty [207] have been suggested.

Less consideration is given to SSDA for semantic segmentation. Semantic consistency is very important for pixel-wise classification. Therefore, Wang et al. [291] include a semantic-level adaption module which adapts the feature distribution on class level, additionally to feature alignment via adversarial training which adapts the domains on a global level. In [40] a two-step domain mixing model is introduced to extract domain-invariant representations. A dual-domain adaptation is proposed by Chen et al. [42] consisting of a cross-domain adaptation network and an intra-domain adaptation on the target domain realized by a student-teacher setup sharing weights between the two adaptation approaches.

Our SSDA method consists of a stage-wise training, a common approach in semi-supervised learning. The first stage serves as initialization for the model which is then adjusted to a specific task. In general this approach is applied to downstream tasks whereas we train the generator as well as the discriminator (as suggested by Grigoryev et al. [84]) in our model in two stages.

Gathering paired data like a digital twin for complex scenarios is a challenging task and an open research question [267]. Therefore, we restrict our I2I method to an unsupervised approach in contrast to [114,252] and [285]. Furthermore, pure I2I methods tend to neglect task specific information [271] which we tackle by incorporating task awareness in the generative component. Information loss compared to supervised trained methods is also encountered by UDA methods [42]. Thus, we developed an ADA approach and select actively meaningful annotated target samples to balance the information lack and the annotation cost.

Our SSDA method (method without AL) is closely related to the independently published work from Mabu et al. [168] in a medical context. Contrary to their work we consider domain gaps between real and abstract domains which vary not only in contrast and image intensity. In addition, our method handles more complex downstream tasks, like semantic segmentation, which leads to a wider range of applications of the method. Moreover, we include AL to select the labeled data in the target domain. Our active learning setup is related to the work of Su et al. [261] as we also incorporate

the downstream task in the adversarial training loss. In contrast to the ADA methods mentioned above we refrain from modifying the downstream task expert and analyze the data generation of the abstract representation. Thereby we query samples suitable for the generator and keep the downstream task network fixed what distinguishes our approach from the others. Moreover, we consider a pure domain separation by training from scratch. Furthermore, we consider the domain gap from real to synthetic/abstract. Thereby we can exploit that abstract representations are often less complex and more easy to collect with labels compared to real world scenarios and with that potentially require smaller networks to achieve excellent results. Furthermore, we can benefit from the advantages of a synthetic expert as training and testing of e.g., life-threatening scenes are possible. In the following we introduce our SSDA method in more detail.

4.3 Semi-supervised Task Aware I2I Translation

The method we developed is subdivided intro three steps, each of them is introduced in the following. Further, our method can be extended by actively sampling images with ground truth. A detailed explanation on how we incorporated active image sampling is given in Section 4.3.5. A summary of our method is depicted in Figure 4.1 and the active learning process is visualized in Figure 4.2.

4.3.1 Stage a) – Training the Task Expert

The aim of stage a) is to train an expert on a domain where labeled data is available or easy to get. Therefore, we take the assumption for our method that we have access to plenty of data samples with corresponding labels in a synthetic domain $\mathcal{S} = (\mathcal{X}_S, \mathcal{Y}, \mu_S)$. Given a training set $\mathcal{D}_\mathcal{S} \subseteq \mathcal{X}_S \times \mathcal{Y}$ in the synthetic domain, we train a neural network $f_{\boldsymbol{\theta}}$ in a supervised manner. To enhance readability, we exclude the parameter dependency from the notation in the following text. The considered computer vision task is flexible and could be, e.g., image classification or semantic segmentation, which we refer to as downstream task. When training f, it is common practice to start from an ImageNet [56] pre-trained backbone (cf. [126]) and train f to perform the given downstream task. In the present work, we refrain from this approach and train f from scratch, i.e., without making use of any pre-trained backbone weights. This ensures that the network trained on the synthetic domain has indeed never seen any data from any real world domain and is thus not already biased towards real world data. This enables us to do a pure domain gap analysis. In return, we have to accept lower overall performance in our experiments, in particular when evaluating with real world data later on. We stop training when f has reached a desired performance and then keep the network frozen in all subsequent steps, i.e., the network weights remain constant and are not trained anymore. Thus, f,

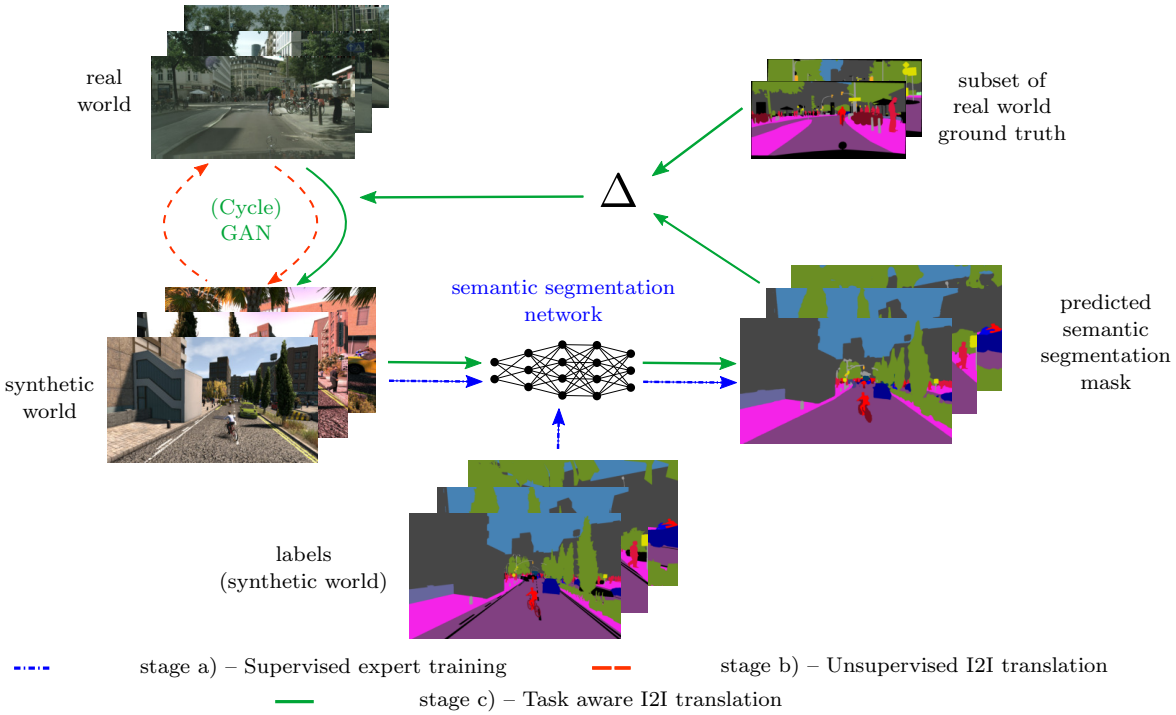


Figure 4.1: Training stages of our SSDA method. Stage a) – Training of a task model (e.g., semantic segmentation network) on the synthetic domain serving as domain expert. Stage b) – Training of a task agnostic CycleGAN based on unpaired data to bridge the domains between real and synthetic. Stage c) – Including task awareness by backpropagating the downstream task performance based on the task loss of the fixed weight, synthetic expert and a few (actively selected) annotated real world samples.

denoted by f_S if the training domain is not clear from context, can now be considered as a synthetic domain expert model.

4.3.2 Stage b) – Unsupervised Image-to-Image Translation

To mitigate the domain gap between the real domain $\mathcal{R} = (\mathcal{X}_R, \mathcal{Y}, \mu_R)$ and the synthetic domain $\mathcal{S} = (\mathcal{X}_S, \mathcal{Y}, \mu_S)$, we build on the established unpaired I2I translation method CycleGAN (see Section 3.2.2). When applying CycleGAN in our context, one of the CycleGAN generators maps from the synthetic to the real domain, termed $G_{\mathcal{S}\to\mathcal{R}}$, and the other one maps in the opposite direction, termed $G_{\mathcal{R}\to\mathcal{S}}$. These two generators are complemented by discriminators $D_{\mathcal{R}}$ and $D_{\mathcal{S}}$. The loss is composed of four additive terms $\mathcal{L}^{G_{\mathcal{R}\to\mathcal{S}}}, \mathcal{L}^{G_{\mathcal{S}\to\mathcal{R}}}, \mathcal{L}^{\text{cyc}}$ and $\mathcal{L}^{\text{identity}}$, described in detail in Section 3.2.2. For our application of CycleGAN to a real to synthetic I2I translation, we adopt the least-squares formulation (cf. Equations (3.36) and (3.37)) of the losses as proposed in the original paper by Zhu et al. [324] to stabilize training. We summarize the generator loss here, as it is important to understand the differences between stage b) and stage

c) in our method. Let $X_R \sim \mu_{R_X}$ denote the random variable describing the image generation process of the domain of real data and let μ_{R_X} be the corresponding marginal distribution of μ_R, then the loss function of the generator for the synthetic data is given by

$$\mathcal{L}^{G_{\mathcal{R}\to\mathcal{S}}} = \mathbb{E}_{X_R\sim\mu_{R_X}}\left[\left(D_{\mathcal{S}}\Big(G_{\mathcal{R}\to\mathcal{S}}(X_R)\Big) - 1\right)^2\right] \tag{4.1}$$

and analogously for $\mathcal{L}^{G_{\mathcal{S}\to\mathcal{R}}}$. The overall generator loss is then given by the weighted sum

$$\mathcal{L}^{\text{Gen}} = \mathcal{L}^{G_{\mathcal{R}\to\mathcal{S}}} + \mathcal{L}^{G_{\mathcal{S}\to\mathcal{R}}} + \gamma_{\text{cyc}}\mathcal{L}^{\text{cyc}} + \gamma_{\text{cyc}}\gamma_{\text{id}}\mathcal{L}^{\text{identity}}, \tag{4.2}$$

with real-valued parameters $\gamma_{\text{cyc}} > 0$ and $\gamma_{\text{id}} > 0$. Note that this I2I translation is task agnostic, i.e., it does not take the downstream task into account.

4.3.3 Stage c) – Downstream Task Awareness

To incorporate task awareness into the model we use the unsupervised I2I model from stage b) as initialization and extend the model training in a supervised manner with a few labeled data points from the real domain. Thereby, we guide only the generator $G_{\mathcal{R}\to\mathcal{S}}$, transferring real images ($\boldsymbol{x}^{\boldsymbol{R}} \in \mathcal{X}_R$) to synthetic/abstract images ($\boldsymbol{x}^{\boldsymbol{S}} \in \mathcal{X}_S$), to the downstream task while keeping the downstream task network f from stage a) fixed. For domain $\mathcal{R} = (\mathcal{X}_R, \mathcal{Y}, \mu_R)$, we define a labeled subset $\mathcal{T}_{\mathcal{R}} = \{(\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{R}}, \boldsymbol{y}_{\boldsymbol{i}}^{\boldsymbol{R}}) \in \mathcal{X}_R \times \mathcal{Y} \mid i = 1, \dots, N_0\} \subseteq \mathcal{X}_R \times \mathcal{Y}$ of size $N_0 \in \mathbb{N}$ which stems from the same distribution. The choice of labeled data points can be queried in different ways. Possible strategies are discussed in Section 4.3.5. We extract the information about the downstream task performance from the downstream task network loss. For semantic segmentation or classification we consider the (pixel-wise) cross entropy as defined in Equations (2.96) and (2.116) between the network prediction distribution $f(G_{\mathcal{R}\to\mathcal{S}}(X_R))$ and the ground truth distribution Y. Given f and the labeled data, the loss is calculated by inferring the generator first and then feeding its output to f yielding

$$f(\tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{\boldsymbol{S}}) \text{ with } \tilde{\boldsymbol{x}}_{\boldsymbol{i}}^{\boldsymbol{S}} = (G_{\mathcal{R}\to\mathcal{S}}(\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{R}})) \tag{4.3}$$

for $\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{R}} \in \mathcal{T}_{\mathcal{R}}$. We denoted the loss by $\mathcal{L}^{\text{task}}$. The task loss is added to the adversarial loss in Equation (4.1). To weigh the task specific and the generative aspect of the loss, we introduce a weighting factor $\alpha \in [0, 1]$. We use α for a linear combination of the two loss components to control the influence of one or the other loss during training. Setting $\alpha = 0$ recovers the original adversarial loss from Equation (4.1) and $\alpha = 1$ omits the generative component. With values of $\alpha \in (0, 1)$ the influence of the two loss components are controlled. Thereby, we define the extended generator loss $\tilde{\mathcal{L}}^{G_{\mathcal{R}\to\mathcal{S}}}$ with respect to the labeled dataset $\mathcal{T}_{\mathcal{R}}$ where each sample is a realization of $(X_R, Y) \sim \mu_R$ as

follows:

$$\begin{aligned}\tilde{\mathcal{L}}^{G_{\mathcal{R}\to\mathcal{S}}} = (1-\alpha)\ \underbrace{\mathbb{E}_{X_R\sim\mu_{R_X}}\left[\left(D_{\mathcal{S}}\Big(G_{\mathcal{R}\to\mathcal{S}}(X_R)\Big)-1\right)^2\right]}_{\text{adversarial loss as in Equation (4.1)}} \\ +\alpha\ \underbrace{\left(\mathcal{L}^{\text{CE}}_{\mathcal{T}_{\mathcal{R}}}\Big(f_{\mathcal{S}}(G_{\mathcal{R}\to\mathcal{S}}(X_R)),Y\Big)\right)}_{\text{task loss }\mathcal{L}^{\text{task}}}.\end{aligned} \tag{4.4}$$

Thus, the overall generator loss turns into:

$$\tilde{\mathcal{L}}^{\text{Gen}} = \tilde{\mathcal{L}}^{G_{\mathcal{R}\to\mathcal{S}}} + \mathcal{L}^{G_{\mathcal{S}\to\mathcal{R}}} + \gamma_{\text{cyc}}\mathcal{L}^{\text{cyc}} + \gamma_{\text{cyc}}\gamma_{\text{id}}\mathcal{L}^{\text{identity.}} \tag{4.5}$$

The discriminator losses remain unchanged. With the task loss of the fixed weight, synthetic expert information about the task performance is backpropagated to the CycleGAN, and we achieve task awareness.

4.3.4 Complete SSDA Method

Our final approach combines stage b) and c) subsequently leading to a modular semi-supervised learning approach for task aware I2I translation. The generator $G^{\text{aware}}_{\mathcal{R}\to\mathcal{S}}$ trained in that way is then used for the DA. Given an unlabeled real image $\boldsymbol{x}^{\boldsymbol{R}}$, we estimate its label via the prediction of the downstream task network f based on the GAN transformed input:

$$\underset{c\in\{1,\dots,C\}}{\text{argmax}}\left(f\left(G^{\text{aware}}_{\mathcal{R}\to\mathcal{S}}(\boldsymbol{x}^{\boldsymbol{R}})\right)_c\right). \tag{4.6}$$

Due to the modularity of our approach, the architecture of networks and therefore tasks can be modified as long as a task loss for the synthetic expert can be formulated. Furthermore, as a consequence of our modular approach, the intermediate representation obtained by $G_{\mathcal{R}\to\mathcal{S}}$ could also serve for further analyses as well as different tasks, e.g., as described in [198].

4.3.5 Sampling Strategies for the Labeled Subset $\mathcal{T}_{\mathcal{R}}$

As we have access to the complete unlabeled training dataset $\mathcal{D}_{\mathcal{R}} \subseteq \mathcal{X}_R$ from domain $\mathcal{R}$, active learning, introduced in Section 2.4.4.1, can help to draw informative samples from this pool. Our AL cycle is depicted in Figure 4.2. Firstly, the complete real dataset is inferred by our SSDA method established after stage b). Then, the data points expected to be most suitable for training are selected according to a given query strategy. An oracle (e.g., a human providing annotations or for analysis reasons revealing ground truth which is hold back) is then asked to annotate the selected data points and thereby a pool of annotated data arises. We use this labeled data for training the generator

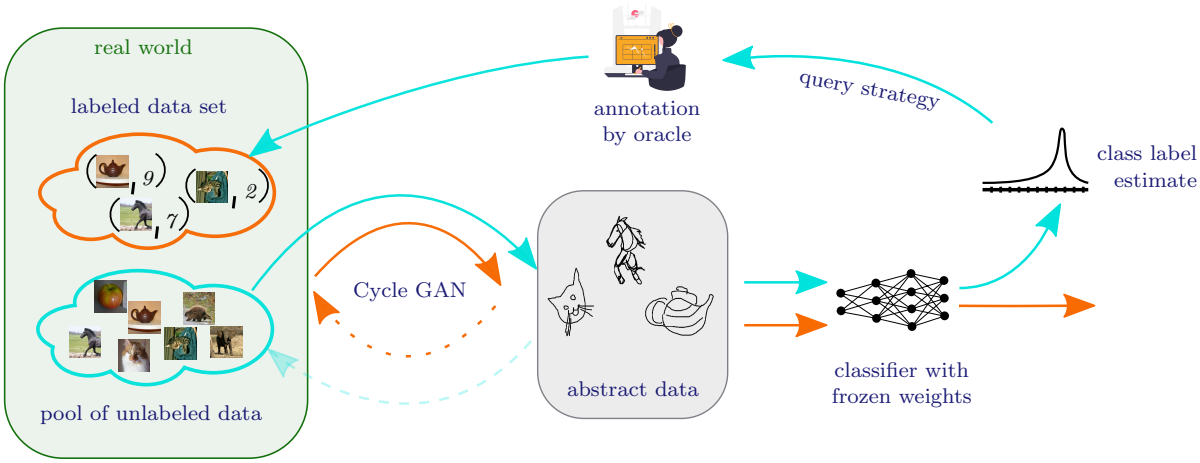


Figure 4.2: Concept of our active learning pipeline. The cyan arrows indicate the active learning query process while the orange colored arrows depict the iterative model training.

in stage c) as described in Section 4.3.3. After the SSDA model is trained for a given number of epochs or until convergence, the remaining unlabeled data pool is inferred by our newly trained SSDA model. New samples are queried for annotation based on the model's prediction uncertainty and thus the pool of labeled data is extended. One AL step, consisting of the data query and the training of the model on the new dataset, is called episode. In each episode, we query a predefined amount of data. To not bias the training towards early drawn samples, the model training in each episode starts with the same checkpoint achieved after stage b). The active learning cycle is repeated until a predefined budget of labeling is exhausted. The simplest query strategy we apply is random selection. This strategy is useful, if no information about the data can be extracted and serves as baseline. With the help of query strategies based on uncertainty measures, as introduced in Section 2.4.4.1 data points can be drawn more informatively. In the following we only consider query strategies for classification tasks. For a given input $\boldsymbol{x}^{\boldsymbol{R}}$, the uncertainty of the SSDA model is given by the uncertainty of the downstream task network f fed with GAN transferred data. For simplification of notation, we set $\boldsymbol{x} \coloneqq \boldsymbol{x}^{\boldsymbol{R}}$ in the following. The uncertainty is calculated based on the prediction distribution (softmax activation)

$$f(G_{\mathcal{R}\to\mathcal{S}}(\boldsymbol{x})) = \big(f(G_{\mathcal{R}\to\mathcal{S}}(\boldsymbol{x}))_1, \ldots, f(G_{\mathcal{R}\to\mathcal{S}}(\boldsymbol{x}))_C\big) \in (0,1)^C \tag{4.7}$$

or the predicted class

$$\hat{c} = \underset{c\in\{1,\ldots,C\}}{\operatorname{argmax}} f(G_{\mathcal{R}\to\mathcal{S}}(\boldsymbol{x}))_c, \tag{4.8}$$

where C is the number of different class labels. The simplest uncertainty-based sampling strategy we consider is to sample the data point where the model is the least confident in its prediction. For a multi classification problem we define the least confident sample

under our SSDA model as

$$\boldsymbol{x}^*_{\mathrm{LC}} = \underset{\boldsymbol{x}\in\mathcal{D}_{\mathcal{R}}}{\operatorname{argmax}} \left(1 - \max_{c\in\{1,\dots,C\}} \left(f(G_{\mathcal{R}\to\mathcal{S}}(\boldsymbol{x}))_c\right)\right). \tag{4.9}$$

To include more information about the data distribution, one can compare the first and the second most probable class label under the softmax distribution using their margin as uncertainty measure. The sample with the minimal margin is given by:

$$\boldsymbol{x}^*_{\mathrm{MARGIN}} = \underset{\boldsymbol{x}\in\mathcal{D}_{\mathcal{R}}}{\operatorname{argmin}} \left(f(G_{\mathcal{R}\to\mathcal{S}}(\boldsymbol{x}))_{\hat{c}} - \max_{c\in\{1,\dots,C\}\setminus\{\hat{c}\}} f(G_{\mathcal{R}\to\mathcal{S}}(\boldsymbol{x}))_c\right). \tag{4.10}$$

The third considered uncertainty sampling strategy bases on entropy [247]. We calculate the sample with the maximal entropy as follows:

$$\boldsymbol{x}^*_{\mathrm{ENT}} = \underset{\boldsymbol{x}\in\mathcal{D}_{\mathcal{R}}}{\operatorname{argmax}} \left(-\sum_{c=1}^{C} f(G_{\mathcal{R}\to\mathcal{S}}(\boldsymbol{x}))_c \log\left(f(G_{\mathcal{R}\to\mathcal{S}}(\boldsymbol{x}))_c\right)\right). \tag{4.11}$$

There is no overall rating which strategy should be used in general, as this may depend on the application, cf. [244].

4.4 Evaluation

Our general method is evaluated on two downstream tasks, semantic segmentation and classification. For semantic segmentation, the domain shift between real world images and synthetic images from Synthia [224] and the CARLA driving simulator [59] is considered. For classification, we employ the Sketchy dataset [235] that contains real world photos along with sketchy drawings with the same semantic content. Note that in both cases, the task is the segmentation or classification in the real (target) domain, which however is achieved by a goal oriented translation to more abstract synthetic or sketch representation. We therefore consider a 'reverse' domain adaptation from real to abstract. As evaluation metrics we use the well established mean Intersection over Union (mIoU) defined in Equation (2.119) for semantic segmentation and report accuracy as defined in Equation (2.109) for classification experiments.

4.4.1 Semantic Segmentation of Real and Simulated Street Scenes

We start with showing that our method is applicable to complex real world scenarios by considering DA with semantic segmentation as downstream task on real and simulated street scenes.

Figure 4.3: Example images from the Cityscapes dataset (first row) and their corresponding ground truth semantic segmentation masks (second row).

4.4.1.1 Datasets

For the semantic segmentation task we use the established dataset Cityscapes [47] for the real domain. The dataset contains 5,000 manually selected and fine annotated RGB images which were recorded in 27 different German and German neighboring cities and have a resolution of $2{,}048 \times 1{,}024$ pixels. The images were captured during daytime with good weather conditions ranging from dates during spring to fall. To ensure diversity in the dataset, frames with many dynamic objects, varying scene layout, and varying background were selected. The dataset is split in train, val and test data. For the latter, annotations are not publicly available as they serve for an independent benchmark test. The annotation is based on 19 class IDs covering a wide range of road users and typical scenery of street scenes. An example from the dataset is shown in Figure 4.3. For our experiments we use the 2,975 images of the train split as well as the 500 validation images where the dense pixel-level annotations are publicly available. It is common practice to consider the domain shift between Cityscapes (real domain) and the SYNTHIA-RAND-CITYSCAPES dataset (Synthia) [224] for the synthetic domain [242]. Synthia consists of 9,400 images with a resolution of $1{,}280 \times 760$, randomly recorded in a virtual town from multiple viewing angles. The camera position stays approximately constant for each chunk of 50 images but the scene, e.g., positions of pedestrians, cars etc., as well as the weather and lighting conditions vary on each image. The semantic segmentation includes 23 classes including most of the Cityscapes training classes. Examples for the different view points are visualized in Figure 4.4. To have coincided classes in both domains, we restrict the classes to the commonly used 16 for domain adaptation which are the Cityscapes training IDs except for *train*, *truck* and *terrain* [26]. As no fixed validation set is given, we leave out the last 1,400 images during training. Using the first 700 thereof for validation.

As a second setup we generated a dataset with the help of the open-source simulator CARLA [59] which allows for the extraction of a strongly controlled dataset to realize

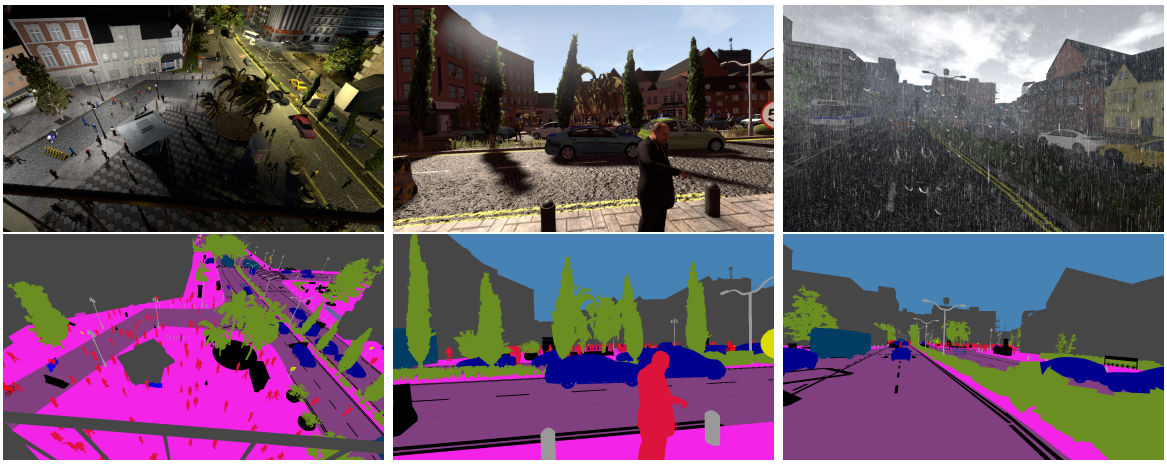


Figure 4.4: Example images of the different point of views and weather conditions sampled from the Synthia dataset (first row). The corresponding ground truth masks (second row) are colored according to the 16 classes that are considered for the domain adaptation.

our hypothesis of unlimited data in the synthetic domain. To showcase this we restrict our data to town 1 of CARLA with fixed environmental settings like weather, wind etc. We generated 3,900 images for training and 1,200 images for validation with a resolution of 1,920 × 1,080 by randomly spawning the ego vehicle on the map. Furthermore, we spawned each time a random number of road users for a diverse scenery. Similar to Synthia not all Cityscapes training classes exist in CARLA. In particular, there is no distinction between different vehicle and pedestrian types. To this end, we fuse them into a vehicle and a pedestrian metaclass. Therefore, we consider only 13 classes: *road*, *sidewalk*, *building*, *wall*, *fence*, *pole*, *traffic light*, *traffic sign*, *vegetation*, *terrain*, *sky*, *pedestrian*, *vehicles*. In contrast to manually labeling, the segmentation mask of CARLA exactly match the rendering and thereby are much finer as the Cityscapes labels. To adapt the more coarse labeling of a human annotator and to avoid an additional domain shift due to different labeling strategies, we employ the labeling coarsening procedure from [228] in a post-processing step to the semantic segmentation masks and the RGB images.

4.4.1.2 Semantic Segmentation Expert in the Synthetic Domain

For the semantic segmentation network f, we use a deeplabv3 with ResNet101 backbone (cf. Section 2.3.2) ranging under the top third of semantic segmentation models on Cityscapes with respect to the comparison of [175]. We train with Adam (cf. Section 2.2.4) with class weighting, polynomial learning rate and from scratch without pre-training to evaluate the domain gap accurately. To range the results, we trained and evaluated f once on Cityscapes to state the oracle performance of the from-scratch-trained network independently of our experiments. This led to a mIoU of 62.74% on the

validation set. This model is only used as reference, and therefore we refrained from hyperparameter tuning. For the experiments with Synthia as synthetic domain, we trained the same neural network f for 3 days with a batch size of 2 due to GPU memory capacity which led to 107 epochs of training on the training dataset. During training, we crop patches of size $1{,}024 \times 512$ and flip horizontally with a chance of 50%. On the in-domain validation set we achieve a mIoU of 64.83%. On our CARLA dataset, the neural network is trained over 200 epochs with random quadratic crops of size 512×512. The best mIoU achieved on the validation set during training is 91.89%. Benefitting from the simulation we constructed an extremely precise domain expert. As the image resolution of Synthia and CARLA images, differ from the resolution of Cityscapes, a resizing is necessary. Depending on the scaling and aspect ratio the network's prediction performance differs. We chose the scaling with the best performance, which we found for $1{,}024 \times 512$. For a fair comparison we let the GANs generate the same resolution.

4.4.1.3 Training Details for I2I Translation and Evaluation Methods

Our implementation bases on the code of CycleGAN [324], which we extended according to our method presented in Section 4.3. We train CycleGAN with the fixed resolution $1{,}024 \times 512$ in both domains over 175 epochs for the training stage b) (task agnostic training) and thereafter over 50 epochs of task aware training (stage c)) with a few Cityscapes labels available for Synthia and 160 epochs for CARLA. During the stage c) training, the pixel-wise cross entropy loss of the fixed weight domain expert is backpropagated to the CycleGAN. We scale the task loss $\mathcal{L}^{\text{task}}$ if needed once with a factor γ_{scale} in order to show similar ranges of variation during training compared to the adversarial generator loss $\mathcal{L}^{G_{\mathcal{R}\to\mathcal{S}}}$. This allows a better interpretability of the weighting factor α which defines the linear combination of the two losses. First experiments were done on a mixture of labeled and unlabeled data, but we experienced an unstable training when alternating between the corresponding loss functions $\tilde{\mathcal{L}}^{\text{Gen}}$ and $\mathcal{L}^{\text{Gen}}$. Splitting the generator training into two stages as described in Section 4.3, led to a more stable training and therefore better results.

As explained in Section 4.2, due to the pure domain separation we are considering, a direct comparison to other DA methods is barely meaningful. Hence, for evaluation we compare our approach with the same types of methods as done in [168]:

M1 The semantic segmentation expert on simulated data f_S is fed with generated data from pure CycleGAN training without task loss: $f_S(G^{\text{agnostic}}_{\mathcal{R}\to\mathcal{S}}(\mathcal{R}))$.

M2 The semantic segmentation expert on simulated data f_S is fed with images from Cityscapes (real world domain) without DA: $f_S(\mathcal{R})$.

M3 The segmentation network f_R is trained from scratch on Cityscapes (real world domain) with the same amount of data as used in the SSDA approach in stage c) for the direct comparison: $f_R(\mathcal{R})$

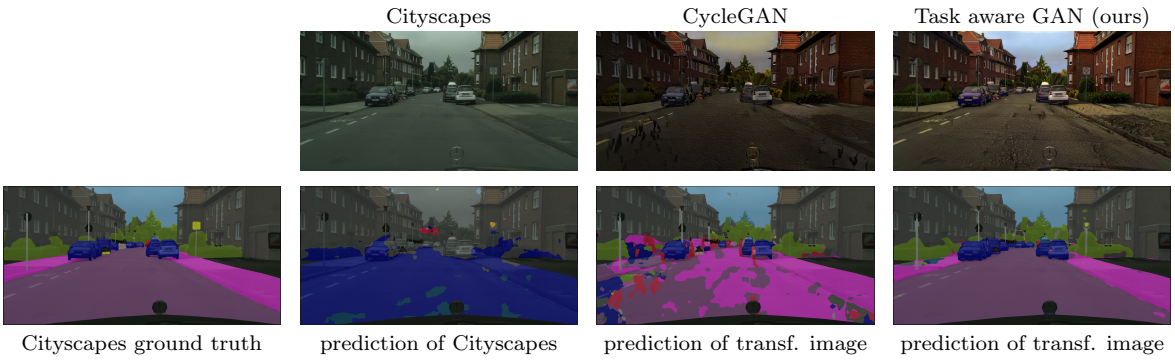


Figure 4.5: Comparison of prediction results of an untranslated Cityscapes image (left), task agnostic I2I (mid) and our approach (right) based on a semantic segmentation network trained on Synthia.

Table 4.1: Domain gap comparison of networks trained from scratch vs. ImageNet pre-trained [63] with Cityscapes as out-of-domain (ood) evaluation.

Synthia → Cityscapes (mIoU in %)			
method	ood	oracle	gap
ImageNet pre-trained	31.8	75.6	43.8
from scratch (ours)	9.9	62.7	52.8

When training the I2I module for DA, we report the best mIoU values obtained for f_S on the Cityscapes validation set. In the following we evaluate our SSDA approach with respect to the domain gap mitigation, the task loss weighting and the amount of ground truth samples for the Synthia setup. The results of the CARLA setups are summarized in the last paragraph.

4.4.1.4 Domain Gap Analysis

Exemplary images from the methods M2, M1 and our SSDA approach for $\alpha = 0.8$ as well as their segmentation by the task expert f_S are displayed in Figure 4.5. The column 'Cityscapes' in Figure 4.5 illustrates the low prediction performance of a synthetic domain expert when never having seen real-world images (M2). The network's performance drops to roughly 10% when real world images are used as input for the domain expert. Here we see a significant difference to results reported by other domain adaptation methods which use ImageNet pre-trained networks, e.g., [63]. The domain gap is summarized in Table 4.1 where we compare ImageNet pre-trained network performance to ours evaluated on Cityscapes. We state out-of-domain performance (i.e., trained on Synthia; second column), oracle performance (i.e., trained on the full Cityscapes training dataset; third column), and the domain gap between them, measured as difference in performance (last column). The results indicate that ImageNet pre-trained networks

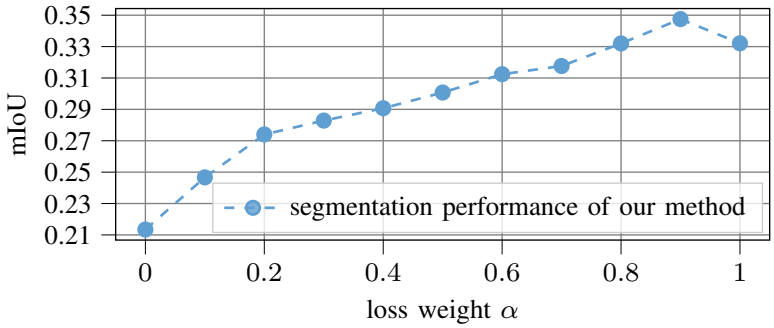


Figure 4.6: Influence of the task loss based on the Synthia experiment setup. The weighting represents a linear interpolation between the adversarial generator loss and the task loss (cf. Equation (4.4)) resulting in the original CycleGAN implementation for $\alpha = 0$ and the pixel-wise cross entropy loss for $\alpha = 1$.

have a much better scene understanding by having seen plentiful real world objects. If network and oracle are trained from scratch, the domain gap widens. Whether ImageNet pre-training is legitimate or not, can be seen as a matter of taste. However, due to the presented results, in this work we take the view that ImageNet pre-training introduces a certain bias in favor of the real domain and thus distorts a pure domain separation. We prefer to keep our experiment free from this effect.

4.4.1.5 Task Loss Influence

Based on our training-from-scratch setup, using task agnostic generated images (M1) improves already significantly the performance (11.44 percentage points (pp)). This can already be seen in the 'CycleGAN' column in Figure 4.5 and in Figure 4.6 for $\alpha = 0$. However, our approach (task aware GAN) can lead to a relative improvement of up to 24.85 pp when 5% of the ground truth data is available. To analyze the impact of the different loss components in the Synthia setup, we set the scaling parameter γ_{scale} empirically to 0.25, we fix the ground truth amount to 5% (148 labeled training images) and vary the weighting parameter α between 0 and 1. The corresponding results are shown in Figure 4.6. We see the positive impact of the task awareness in the growing mIoU values. Using a weighting of $\alpha = 0.9$ for $\mathcal{L}^{\text{task}}$, we achieve 34.75% mIoU which is a performance increase of 13.41 pp compared to M1 (task agnostic training).

4.4.1.6 Ground Truth Influence

Moreover, we analyze the capacity of the method based on the amount of ground truth data available. Therefore, we fix $\alpha = 0.8$ and vary the amount of ground truth data for the stage c) training. We randomly sample images from the Cityscapes training dataset for each percentage but fix the set of labeled data for the experiments with CARLA and

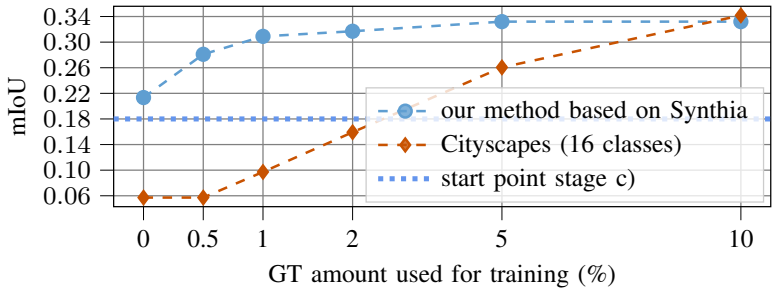


Figure 4.7: Performance comparison of our method based on Synthia setup with different amount of ground truth (blue) and a from scratch supervised training on Cityscapes with the same amount of data (orange).

'Cityscapes-only' training (M3) for the sake of comparison. The results are shown in Figure 4.7 (blue curve) where the x-axis denotes the amount of available ground truth data (GT). The dotted horizontal line is the mIoU achieved by f_S when exclusively feeding images generated by the task agnostic GAN after finishing stage b) training. For a fair comparison we trained the task agnostic GAN for another 125 epochs which results in a better mIoU of 21.34% which we use as result for 0% available ground truth data (also equaling $\alpha = 0.0$). For our experiment we compare 0.5% (14 images), 1% (29 images), 2% (59 images), 5% (148 images) and 10% (297 images) of ground truth data for the stage c) training. Triggering the task awareness with only 14 images already improves the network accuracy by 6.75 pp. The results show that training $G_{\mathcal{R}\to\mathcal{S}}$ with the task loss on negligible few ground truth data, improves the network's understanding of the scene without retraining the network itself.

Additionally, we compare our method with results of f_R (M3). Having no labeled data, a supervised method can barely learn anything. Therefore, we set the value to the same as for 0.5% available ground truth which most likely overestimates the performance. The results are visualized by the orange curve in Figure 4.7. As we can see, our method outperforms the supervised training up to an amount of roughly 10% of the Cityscapes ground truth data. Thereafter, direct supervised training is more efficient. Nevertheless, we saw that enhancing the I2I method with only 0.5% of ground truth data, or 14 frames respectively, already enhances the DA performance. The task expert network thus already profits significantly in the semantic interpretation of scenes, if a negligible amount of labeled ground truth data is available.

4.4.1.7 Evaluations Based on CARLA

Our findings for the CARLA experiments are in line with the observations of the Synthia domain adaptation. The numerical settings, however, have been slightly adapted by setting $\gamma_{\text{scale}} = 1$, as no pre-scaling of both loss contributions was needed as both varied in a similar range. We repeat the three experiments on our CARLA dataset. The

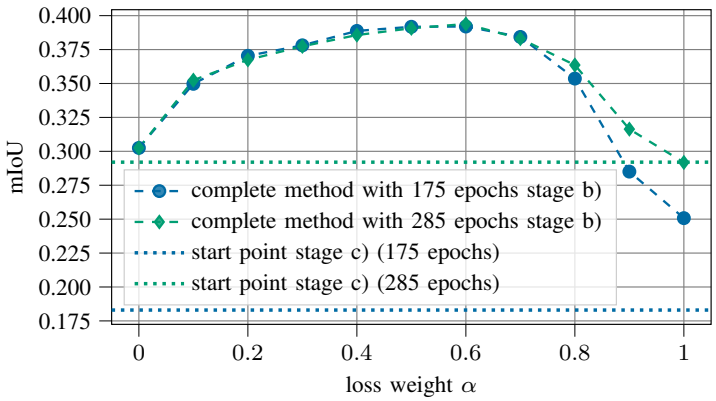


Figure 4.8: Influence of the task loss for the CARLA experiment setup. The weighting represents a linear interpolation between the adversarial generator loss and the task loss Equation (4.4). Results after stage c) based on a 175 epochs unsupervised GAN-training are shown in blue. The green graph shows the method's performance when stage b) is trained longer.

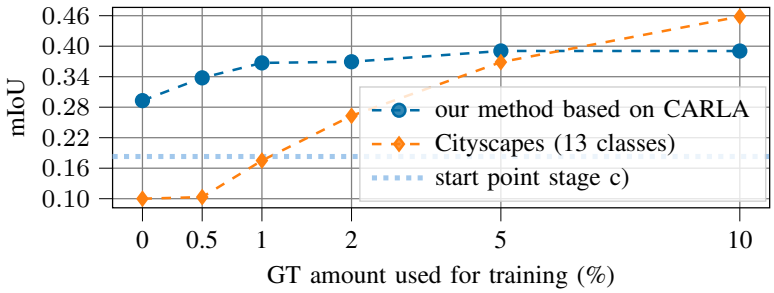


Figure 4.9: Performance comparison of our method based on CARLA setup with different amount of GT (blue) and of f_R which is trained from scratch in a supervised manner on Cityscapes with the same amount of data (orange).

results of varying α are visualized by the blue curve in Figure 4.8. Also, on the CARLA dataset our method, with 5% of Cityscapes ground truth available, shows a notable improvement over the CycleGAN baseline (M1) when choosing a balanced weighting between the adversarial and the task loss. The CycleGAN baseline corresponds to the value $\alpha = 0$, whereas peak performance is roughly 10% improved and is reached for $\alpha = 0.6$. These experiments confirm that the task awareness improves the performance, but the task loss should be used in addition and not as a stand-alone concept.

We also provide the ablation study on the amount of data available, in Figure 4.9 for a fixed value $\alpha = 0.4$ for the CARLA setup. The blue curve represents the best mIoU results achieved with our method and the orange curve shows the results of f_R given different amount of ground truth data. As before our method outperforms M1 as well as M3 when less than 5% ground truth data is available. Beyond 5% of ground truth availability, direct training is more efficient. The distinct improved semantic segmentation of the street scene can also be seen in the qualitative results shown in Figure 4.10

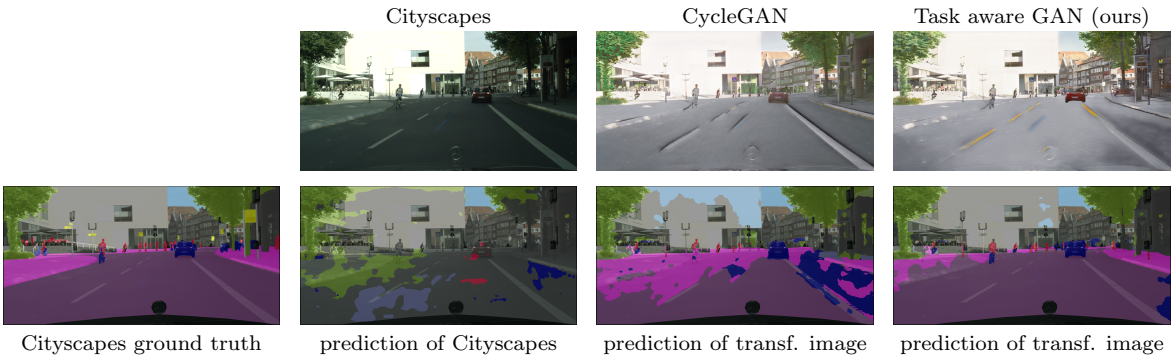


Figure 4.10: Comparison of prediction results of an untranslated Cityscapes image (left), task agnostic I2I transfer (mid) and our approach (right) based on a semantic segmentation network trained on our CARLA dataset.

(bottom row). As in the previous experiment, visual differences recognizable by humans of the generated images with CycleGAN (top row mid) and our method (top row right) are limited. Furthermore, we see again the low performance caused by the domain gap when feeding real images to our from-scratch-trained synthetic expert $f_{\mathcal{S}}$. On the untranslated images (M3), f_S yields an mIoU of 9%. Hence, the observed results achieved by our method demonstrate a significant reduction of the domain gap via generating more downstream task relevant visual features.

In a final experiment on semantic segmentation, we consider the effects of a prolonged stage b) training to find out whether a longer training further improves the results. We train in total 285 epochs in stage b) and show the results of the complete method based on a fixed amount of 5% ground truth data. The comparison is given in Figure 4.8 by the green curve. The experiments reveal that a moderate number of epochs for stage b) is already enough for a good initialization of stage c). Although we start the stage c) training with a higher mIoU (dotted lines) when trained with stage b) for more steps, the experiments show that we achieve nearly the same absolute mIoU values. This applies particularly to the range of α values where peak performance is attained.

After showing the potential of the method on a complex downstream task, we consider a more complex domain adaptation setup with a classifier as downstream task in the following. We thereby chose a setup where the domain differs significantly contrary to the experiments of [168] where the domains differ mainly in contrast and image intensity. Furthermore, we analyze the effectiveness of informative sampling for stage c) training.

4.4.2 Active SSDA on Real to Abstract Data

To evaluate a complex DA challenge, we chose a subset of the Sketchy dataset [235], where human drawn sketches serve as abstract representation of real world objects.

Figure 4.11: Examples of the Sketchy dataset. Top row: real photos. Bottom row: one of the corresponding sketches.

Exemplary data samples are shown in Figure 4.11. For each class exist photos (subset of ImageNet) of the named object and several sketches drawn by different humans under time pressure. Therefore, sketches can be "incorrect in some way" [235]. The complete data generation process is described in detail in [235]. In order to obtain a clean dataset for our experiments, sketches which could not be identified by their annotation were removed. Furthermore, we reduced the classification problem from 125 classes in the original dataset to the following 10 classes: *alarm clock*, *apple*, *cat*, *chair*, *cup*, *elephant*, *hedgehog*, *horse*, *shoe* and *teapot*. For an independent test set we split off 20 images per class of real images and 90 per class of sketch images remaining in a dataset with 700 real (photo) images and 4,633 sketch images for training. The imbalanced amount of data supports our hypothesis that abstract data is often easier to get.

4.4.2.1 Domain Gap Between Photos and Sketches

Firstly, we train from scratch a ResNet18 [92] with the help of the categorical cross entropy loss on sketch data, which serves as expert classifier f (stage a)). The comparably simple image structure of the sketches results in an in-domain test accuracy of 97.11% after 200 epochs of training. For evaluating stage b) and c) we split off randomly 70 photos (10%) of the training data uniformly over the classes. To incorporate active learning for the data selection in stage c), we extended the deep active learning toolkit for image classification [33], which bases on [181], with our task aware I2I model. We compare the classifier performances in-domain and under domain shift in Figure 4.12 (left). We observe a notable accuracy drop to 10% when evaluating the sketch expert on real images, revealing that the domain gap between gray scale sketches and RGB photos is significantly complex. In addition, from the in-domain accuracy comparison ($\approx 73\%$ vs $\approx 97\%$) we conclude that classifying real world images is more difficult (due to problem complexity or availability of data) than classifying sketches.

The use of images generated by a task agnostic I2I generator as classifier input already

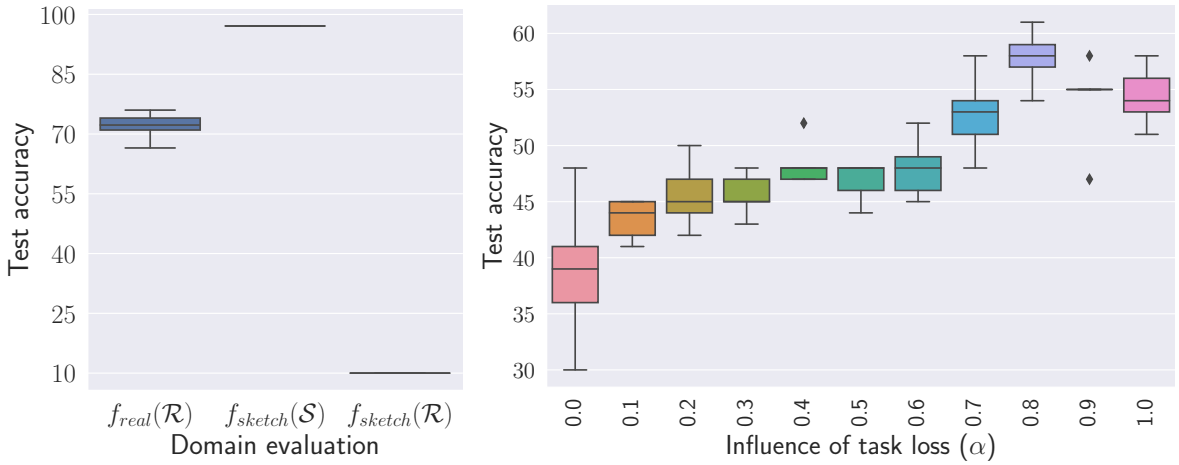


Figure 4.12: Left: Accuracy under domain shift: In-domain performance for real world data ($f_{real}(\mathcal{R})$), in-domain performance for sketch data ($f_{sketch}(\mathcal{S})$) and performance under domain shift ($f_{sketch}(\mathcal{R})$). Right: Accuracy of the sketch expert with varying influence of the task loss controlled by α showing the positive impact of our method. Setting $\alpha = 0$ equals CycleGAN-only-training and includes zero information about the downstream task. Setting $\alpha = 1$ omits the adversarial part of the loss and training is done with cross entropy loss only. Note the different scaling of the y-axes.

yields improved classifier predictions. In our experiments we set $\alpha = 0$ in Equation (4.4) to achieve a reasonable task agnostic generator model $G^{\text{agnostic}}_{\mathcal{R}\to\mathcal{S}}$ in an unsupervised manner. We observe that images generated that way facilitate distinctly the prediction by the sketch classifier f, leading to 38.8% accuracy on average (5 runs) and thus an average increase of 28.8 pp compared to feeding real world images. Continuing training and additionally incorporating task awareness with stage c) further improves the accuracy by up to 29.0 pp.

For the stage c) training we fixed the overall annotation budget to 75 data points (roughly 12% of the remaining real world training data). We selected a fixed subset of 75 labeled samples to perform the stage c) training. The set was created with the help of AL and the entropy query strategy. The dataset accumulatively queried during 14 episodes was then fixed and used for our ablation study of the task loss weighting. A detailed study on the query strategies is described in the next section. Due to randomness in the training, we report averaged results over 5 repeated trainings on the fixed dataset. We show the task loss influence with respect to the test accuracy evaluated by scanning α in steps of 0.1 in Figure 4.12 (right). Depending on the weighting α of the task loss, we see the positive influence of the task awareness in the increased performance of f when fed with the data generated from $G^{\text{aware}}_{\mathcal{R}\to\mathcal{S}}$ ranging from 43.4% accuracy for $\alpha = 0.1$ to 57.8% accuracy for $\alpha = 0.8$. The performance saturates or drops when only considering the task loss, i.e., omitting the adversarial part completely. The latter implies that the generator is not penalized by the discriminator when generating images outside the target domain

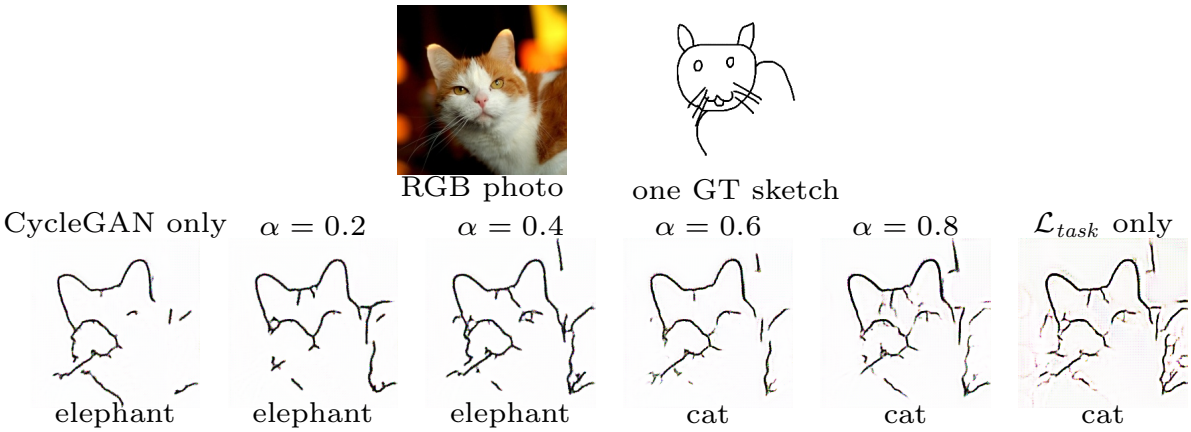


Figure 4.13: Images generated with our SSDA method with different weighting α of the task loss during training. Top row: RGB photo (input of $G_{\mathcal{R}\to\mathcal{S}}$) and an exemplary sketch. Bottom row: Generated sketches and their prediction by our synthetic expert.

as long as they are predicted correctly by f. Furthermore, the experiments reveal that our method has a regularizing effect, as the variance of performance is much higher for input images from generators trained with $\alpha = 0$ (task agnostic). In Figure 4.13 we show qualitative results of the generator outputs for different α-values during training and the corresponding classification results of our synthetic expert. More and more details seem to be added when the task loss influence rises. This supports the classification accuracy but leading to noisy images if the adversarial component of the loss is omitted (last image in the row).

4.4.2.2 Active Learning for Domain Adaptation

In the previous section, we have selected a specific split of the labeled data to perform the stage c) training. We investigate in the following, whether actively choosing images improves the generated data, in the sense of making it more suitable for the classifier to predict correctly. To this end, we compare four query strategies with the help of our extended deep active learning toolkit for image classification. Having a labeled dataset at hand, the oracle is replaced by revealing the given ground truth when queried. This is a common practice in AL to ensure comparable results and reduce label effort [261]. As stated in the previous section, we allow a budget of 75 real world images to query in total. They are queried in batches of 5 over 14 episodes. In each episode we initialize our model with a checkpoint obtained by stage b) training. The initial classifier performance, given generated images by this checkpoint, is 42.0% accuracy. Having a reasonable generator trained in an unsupervised manner and a completely trained sketch expert, we can query data points in the first run without the need of a random subset for a network initialization training, which is needed for classical AL setups. In each episode we train the so initialized model for 50 epochs as described in Section 4.3.5 stage c) on the growing

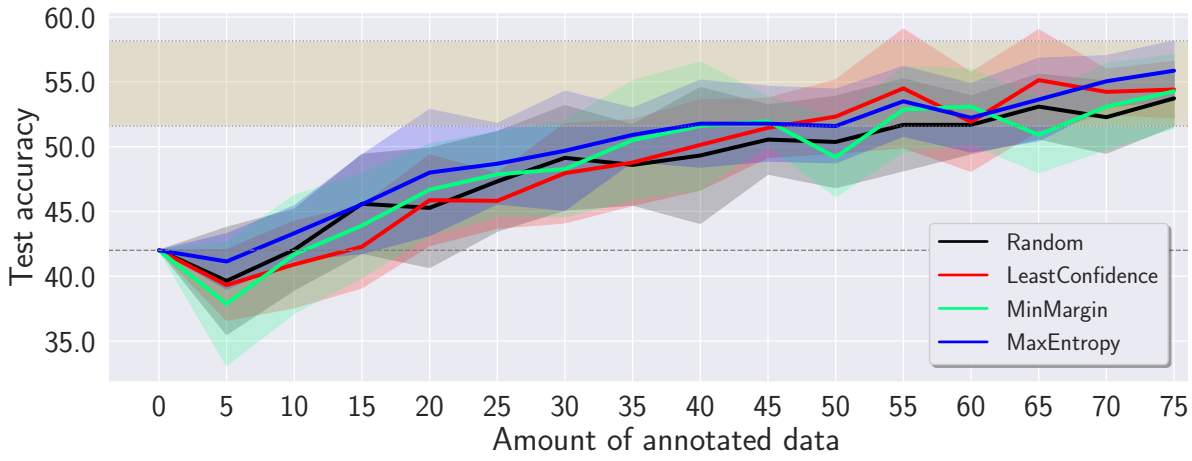


Figure 4.14: Active learning for SSDA data selection. Data selection for stage c) based on entropy leads to the best results. The lower dotted line depicts model performance at (unsupervised) initialization. The ocher colored band shows model performance range after 14 episodes actively training our SSDA method.

labeled dataset. We use $\alpha = 0.8$ for the weighting of the task loss in Equation (4.4), which gave the best results with a fixed set of labeled data (cf. Figure 4.12 (right)). To account for the randomness, we repeat our experiments 11 times and visualize mean and variance of the results. In the experiments we compare the four query strategies introduced in Section 4.3.5: *random*, *least confidence*, *maximal entropy* and *minimal margin* sampling. After a short warm-up phase (5 episodes; 25 annotated images) all query strategies exceed the performance of task agnostic generator training and show a performance increase when more labeled data is available. In Figure 4.14 we compare the results according to amount of annotated data and test accuracy of f when fed with the output of the generator $G^{\text{aware}}_{\mathcal{R}\to\mathcal{S}}$ that was trained on the actively chosen annotated data in stage c). We see that sampling based on entropy outperforms the random baseline nearly everywhere, leading to an averaged performance gain of 2.13 pp after 14 episodes.

Furthermore, we compare our results to direct active supervised training, i.e., classical AL. In contrast to our method, classical AL needs an initial training with some labeled data before being able to query. To ensure, nevertheless, comparable results we initialize the model with 10 data points which equals starting two episodes ($1/7$ episodes) later. We keep the query amount of 5 images per episode but continue actively sampling until the complete dataset is queried. Furthermore, in both setups (classifier training on real world data vs our SSDA method) we allow the model under consideration to converge. Hereby the classifier needs distinctly more epochs (200) compared to the generator training (50) which might be due to the number of parameters that need to be trained. Due to the increased computation cost, we repeat our experiments only with 6 (instead of 11) different seeds. The averaged results with variance are visualized in Figure 4.15.

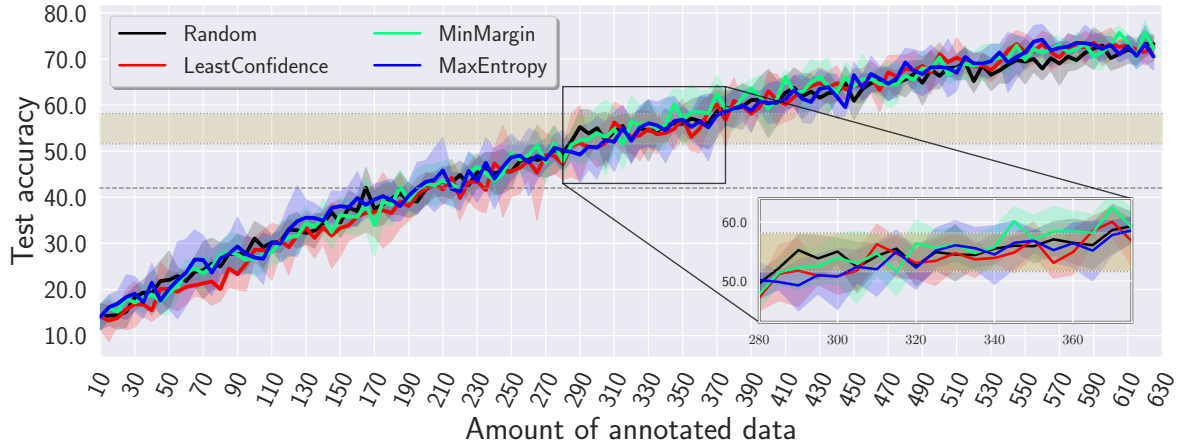


Figure 4.15: Active learning for from scratch training on real images. The lower dotted line and the ocher colored band show results of our ADA method (cf. Figure 4.14). The experiments show that our method requires distinctly fewer annotated data points to reach comparable classifier performance on the test set.

The horizontal ocher colored band visualizes the range of our SSDA method after 14 episodes (75 annotated data samples). The horizontal dotted line depicts our model performance at (unsupervised) stage c) initialization. The high fluctuation in the plot is mostly due to data augmentation, which we used during training to enlarge the limited training dataset. The results do not show a distinct performance gain for active learning strategies on the real dataset. This might be due to the minimal initialization set and the little heterogeneity of the data. The experiments reveal that our method needs considerably fewer annotated data. While the direct active supervised training on the real photos barely achieves 25.58% accuracy when 75 annotated data points are available, our approach reaches more than twice the performance. With 290 data samples (3.86 times more than our method) the direct supervised learning firstly reaches the minimal mean performance (achieved by random sampling) of our method and surpasses our method firstly (but temporarily) when trained on 345 chosen and annotated data points. Thus direct supervised training needs 4.6 times more data compared to our approach. We thereby greatly benefit from the huge labeled data pool available in the abstract (sketch) domain and the less complex vision task of sketch classification instead of real world photos.

4.5 Conclusion and Outlook

In this chapter, we presented a modular semi-supervised domain adaptation method based on CycleGAN where we guide the generator of the image-to-image approach towards downstream task awareness without retraining the downstream task network

itself. Additionally, we extended our method with an active learning pipeline for classification to sample the most informative data points for the downstream task guidance. In our experiments we showed that our method can be applied to complex downstream tasks like semantic segmentation yielding significant improvements compared to a pure I2I approach and from scratch training when a limited amount of ground truth data is available. Besides, we analyzed the impact of the task awareness and the amount of ground truth data used during training. In particular, for small amounts of annotated data, we significantly outperform supervised learning as well as the non-guided CycleGAN. For a fair and clean comparison, all our downstream task networks have been trained from scratch, i.e., without any real world data pre-training as, e.g., based on ImageNet. Additionally, we showed that we can achieve very strong models when considering abstract representations (like sketches or modifiable simulations). Furthermore, our experiments on a 'real to sketch' domain adaptation classification task demonstrates that our method can cope with large domain gaps. Here we have additionally shown that data selection in terms of active learning matters and further improves our method for a given annotation budget.

The results achieved by our active learning strategy in the classification task suggest that this extended method shows promise for improving results in semantic segmentation as well. Therefore, it is worth transferring our active learning component to semantic segmentation and elaborate additional query strategies. In addition, the method could be combined with self-training as these models need a good initialization to generate reasonable pseudo labels [171]. Nevertheless, when training the downstream task network completely from scratch, we have shown that the network performance is questionably low. Therefore, our method can be seen as complementary to the self-training approaches to ensure a reasonable prediction of the network in early stages. Furthermore, we are interested in elaborating more on the (intermediate) abstract representation. A follow-up question of our research addresses whether abstract intermediate representations can help to further improve our downstream task models, for example with respect to robustness. Additionally, it has potential to help us better understand which visual features are important for a downstream task network. Generating more informative images for a downstream task network might give insights into the network behavior and help generate datasets which are cut down to the most important aspects of the scene for a neural network which is not necessarily what a human would describe as meaningful. To this end, we introduce a method to disentangle image cues to get more insight into the behavior of semantic segmentation neural networks in the next chapter.

Chapter 5

Cooperation Is All You Need? A Study on the Ability of Neural Networks to Draw Information From Color, Texture and Shape

The work described in this chapter was assisted by Natalie Grabowsky. Under my supervision, she worked on the proof-of-concept of the method which was further developed by me. As a student research assistant, she supported the CARLA experiments of the elaborated method and generated the textureless world in CARLA. The anisotropically diffused dataset was created by Edgar Heinert.

5.1 Introduction

Convolutional neural networks take advantage of the fact that the majority of natural signals are hierarchical compositions. This means that lower-level features combine to create higher-level ones [144]. To recognize and distinguish objects, their shape and texture give complementary cues. Following the convention from Geirhos et al. [72], we call different concepts or stimuli in the data such as texture, shape or color '**cues**' to clearly distinguish them from features learned by a neural network. Several hypotheses about biases of CNNs were formulated and supported by experimental results. In 1980, Mitchell defines bias as "any basis for choosing one generalization over another, other than strict consistency with the observed training instances." This form of bias is the so-called inductive bias, introduced in Section 2.2.2, which describes additional assumptions made to the model based on e.g., information about the domain or the task such as choosing CNNs for image processing. Already at that time, the importance of understanding biases was pointed out by Mitchell: "Therefore, progress toward understanding learning mechanisms depends upon understanding the sources of, and justification for,

various biases" [178]. Nowadays, the term 'bias' is often associated with its definition from an ethical perspective. According to the definition in [194], bias is the "inclination or prejudice of a decision made by an AI system which is for or against one person or group, especially in a way considered to be unfair." This can also be framed as decisions made by a machine learning model that exhibit prejudice against individuals or groups of people based on protected attributes such as gender or race [64]. Investigations on the ethically motivated term emerged the research field of trustworthy AI. In this thesis, we use the term 'bias' in a more technical and general way. We refer to a **bias** in a dataset or a model if a certain aspect dominates another in the decision-making process. Early in the evolution of CNNs a shape bias was hypothesized stating that representations of CNN outputs seem to relate to human perceptual shape judgement [143]. Furthermore, it has been shown that kernels learned in early layers resemble edge detection kernels which leads to the hypothesis that CNNs learn local edge relations [141]. These relations are combined to more complex structures when the network gets deeper [141,144]. Even though this suggests that CNNs tend to base their prediction on shape information, this is only valid on a local perspective [16]. In general, they do not intrinsically display shape bias [103]. On the contrary, multiple studies were made on ImageNet trained CNNs, indicating that those have a bias towards texture [16, 27, 72]. In the pioneering paper of Geirhos et al. [72], an analysis based on cue-conflicting images was proposed. Therefore, a style transfer with a texture image showing e.g., an animal skin is applied to an ImageNet image. This leads to a newly texturized image with preserved shape and therefore conflicting cues. The method supports the texture bias hypothesis by revealing a distinct texture dominated prediction when shape-texture cue-conflicting images were given as input to CNNs trained on ImageNet [72]. However, a bias denotes that the other cues are not necessarily absent but sometimes more and sometimes less important to solving the task [44, 52, 111].

The mentioned investigations mainly focused on neural networks in the context of classification tasks and ImageNet pre-trained neural networks. This raises the questions of how the bias changes when the task changes and which cues in the data are the most relevant to solving the task. We investigate the different cue influences when changing the task to a task in which not only the type of object is relevant (as per classification which was introduced in Section 2.3.1) but also a localization within the image on pixel-level in form of semantic segmentation (Section 2.3.2) is demanded. Hermann et al. [95] showed in the classification context that the choice of data and the task has influence on the features and the cue biases which are learned by a CNN. We take advantage of this adaptive behavior to analyze how much information about the task can be extracted by CNNs restricted to limited image cues.

We present a method to disentangle best the different cues naturally arising in an image: shape, texture and color. As mentioned above network biases of classification neural networks are often studied based on cue-conflicting inputs. This approach is only transferable to semantic segmentation to a limited extend and cannot reveal how well neural networks can exploit when limited cues are present. Therefore, we propose a collection

of methods each extracting a single or combination of specific cues in an image, preserving the semantic segmentation task. This leads to datasets which contain only the desired number of cues, allowing for an in-depth behavioral analysis of neural networks. These datasets serve as the basis to train expert networks where each is biased towards a specific cue or cue combination. Due to the fact that the training was performed from scratch without the use of pre-trained models, we assure that the neural networks have no previous knowledge of cues other than the ones they are specifically trained on. This study aims to gain more insight into the cue awareness of neural networks. Particularly, we address the question of what neural networks can and cannot learn from a single or a limited number of cues. Our summarized contributions are:

- A method to generate a texture-only dataset for semantic segmentation tasks.
- A general setup to study disentangled cues (including color-only experiments) for datasets with segmentation masks.
- An in depth analysis of the influence of different cues (learned by up to 15 different experts) in the context of semantic segmentation.
- The first study on the differences of cue influence of transformers and convolutional neural networks on a semantic segmentation task.

After setting our approach into the context of the actual state-of-the-art, we present our setup and the cue extraction methods in Section 5.3 and Section 5.4. An in-depth analysis of the question "which cues in the data are the most relevant for solving a semantic segmentation task" is addressed in Section 5.5 for a real world and a simulated dataset. We further investigate the differences between convolutional neural networks and transformers in our study before we conclude and discuss our findings in the Section 5.6.

5.2 Related Work

First investigations on what CNNs learn from images date back to 2014 when Zeiler and Fergus used fractionally-strided convolutions (cf. Equation (2.24)) to visualize image regions which lead to high feature map activations in the classification context. We first give an overview of the different approaches to measure the bias, i.e., cue influence for the prediction. Our focus thereby is on cue influences in semantic segmentation, which is still an under-researched field and addressed by our method. Thereafter, we summarize the state-of-the-art insights about biases in semantic segmentation neural networks.

To analyze cue influences, datasets are prepared to either combine cues artificially to create cue conflicts or remove certain cues from the data. In contrast to classification, data preparation is more complex for semantic segmentation as multiple classes are present in the input which differ in their cues, such as shape, texture and color.

The stylization approach from Geirhos et al. [72] to construct cue-conflicting images is adapted by [111, 156, 269]. In [156], Li et al. propose to stylize images not with artificial textures but with a second image from the same dataset. For semantic segmentation, they extend their method to use a specific object rather than the full image as texture source. Therefore, they crop an object of a specific class from the background based on the ground truth segmentation mask. This serves as the source of the texture for a second image in the dataset which is texturized with the help of the style transfer algorithm AdaIn [106]. As the so generated image is entirely texturized, it is given a single-value segmentation mask holding the class ID of the texture source object. Theodoridis et al. propose a similar data preparation based on AdaIN for instance segmentation, i.e., pixel-wise segmentation of objects where each object is distinguished by a different object ID [269]. In contrast to Li et al., they additionally consider object-centric stylization. This leads to three stylized versions of the COCO [159] dataset. First, stylization of the entire image, second the stylization of just the object while preserving the background and, third, the stylization of the background while leaving the object untouched. Even though the objects and background are treated differently, the approach does not distinguish between different object classes and uses artificial textures from images taken from WikiArt.org [257]. Furthermore, Islam et al. exploit the stylization to construct images with the same concept, e.g., texture or shape, to evaluate the influence of texture and shape respectively on a per neuron level [111]. This is done by comparing the mutual information encoded by the neural network in the latent representation of two images sharing the same concept. Two images share the same concept, if they either share the same texture stylization (texture) or it is the same image but is differently texturized (shape). In [276], Tripathi et al. construct cue-conflicting images for classification tasks not by style transfer but by blending edge maps with puzzled images. Therefore, one image is processed by a standard edge detection algorithm, e.g., Canny edge detection [32], and the other image is partitioned in a regular 4×4 grid and shuffled like a sliding puzzle. These images are then blended. The blending is realized by a weighted linear combination of the images with a weighting scalar randomly drawn from a Beta distribution.

Besides preparing datasets with cue conflicts also datasets in which cues are removed can also help to shed light on neural network behavior. An approach to colorize images with respect to the class IDs is proposed by Kamann and Rother [125] to reduce the influence of texture. To improve robustness against common image corruptions, they randomly sample colors for each label ID and color the segments in the ground truth mask accordingly. The colored map is blended by an adaptable weight with the original image and used as data augmentation (cf. Section 2.4.1) during training. This method deliberately uses colors that do not occur in the image that is textured and is termed 'painting by numbers'. The approach in [276] to shuffle image patches to recreate images with distorted shapes has already been introduced in [27], was picked up in [52] by Dai et al. and used in the context of semantic segmentation for the first time in [319] by Zhang and Mazurowski. A common approach to remove all but the shape cue is to use

silhouettes, contour maps or edge maps [16]. In [52], Dai et al. propose to use mean shift filtering to reduce the texture while preserving the shape in an image. Conversely, they suggest edge blurring using a Gaussian filter for shape reduction. In addition, this work is one of the few works considering color as relevant cue by suggesting gray scaling to remove color. The influence of color, intensity and luminance for object boundary detection was studied in [172].

Based on these datasets it has already been shown in the literature that the bias between texture and shape in neural networks should be balanced [156] for classification settings. Experiments in [269] reveal that neural networks for instance segmentation do not only rely on texture - even though there is a certain texture bias - but also exploit shape information. Furthermore, it has been noted that neural networks with a reduced texture bias are less sensitive to image corruptions [125]. Image stylization and thereby a reduction of the texture reliability is also a common technique to facilitate domain adaptation. By pre-training on stylized images, Zaech et al. aim to reduce the texture bias in CNNs so that they adapt more easily to other domains [314]. Nevertheless, without external intervention, shape is the least prioritized feature learned by convolutional neural networks according to the experiments in [319] and requires a sufficiently large receptive field (cf. Section 2.1.2.3) compared to the object to recognize.

In contrast to convolutional neural networks, transformers have different inductive biases. Primarily, they have no inductive bias towards local spatial structure [279]. Due to their architecture based on self-attention (cf. Equation (2.104)), they have a global view on the input image compared to the local receptive field of convolutional neural networks (cf. Section 2.1.2.3). This allows them to learn relations between pixels that are far apart whereas convolutional neural networks benefit or suffer from their translation equivariance because their kernel shares weights for the entire image. Experiments in [71, 187, 276, 279] show that in classification tasks transformers reveal a bias towards shape. The content dependent receptive field of vision transformers (cf. Section 2.3.1.3) is said to be the reason for this shape affinity [187].

Although a number of approaches exists, they all suffer from different limitations. Firstly, all methods but particularly the edge or contour maps are subject to a quite broad domain gap since the image representation by a gray scale edge map differs distinctly from the RGB image. Moreover, style transfer is said to enforce shape bias. Despite the fact that the dependency on the original texture is reduced, an explicit shape recognition is not guaranteed when trained on this data [276]. It further introduces a data distribution shift and incorporates image corruptions in terms of noise [125]. In addition, most of the works are not trivially applicable to semantic segmentation. Albeit Li et al. proposed an adaptation of their method for semantic segmentation in [156], this adaption does not exceed a showcase. Similarly, the work of Theodoridis et al. is applied to instance segmentation and therefore a clear definition of background is given which is not necessarily the case for semantic segmentation. In addition, the approach of Islam et al. is difficult to transfer to semantic segmentation tasks as the question of being of the

'same concept' needs to be answered on a pixel level or at least a segment level. Despite addressing a semantic segmentation task, Zhang and Mazurowski study real world tasks with very few classes and objects [319]. Moreover, to the best of our knowledge, an analysis of shape and texture bias of transformers for semantic segmentation tasks has not yet been done.

In contrast to other methods, we study the influence of image native cues rather than studying a shape or texture bias of pre-trained neural networks. Besides getting insight into the cue influence of transformers and convolutional neural networks in semantic segmentation tasks, we aim to answer the question of what (still) can be learned when only a certain cue or a reduced number of cues are present in an image. Our method decomposes images into different and mostly orthogonal cues to measure their importance for semantic segmentation tasks. The work presented by Dai et al. is similar to our approach but is limited to only classification. They analyze the influence of combined or removed cues for ImageNet classification [52]. However, we use different extraction methods to take the difficulties of semantic segmentation into account. Besides, we perform a more detailed decomposition. We investigate different shape extraction methods and analyze additionally how much information can be drawn from the pure color, i.e., hue and saturation, or the grayscale value of one pixel. Instead, they investigate the influence of the topology by rearranging image patches. Splitting up and shuffling an image destroys its semantic content and cuts up segments. As a consequence, this approach does not preserve the semantic segmentation task which is why we refrain from using it. Instead, we propose a new method to extract texture from the dataset. The method is flexible enough to create arbitrary new segmentation tasks with texture from the original dataset since our texture dataset only consists of in-domain texture. This stands in contrast to most cue-conflicting approaches. Furthermore, we investigate the cue influence on complex datasets with at least 15 classes what sets us apart from other cue influence approaches in the field of semantic segmentation. Our study also allows for investigations on a per class and per pixel level. Hence, we can see if certain classes or pixels are more likely to be classified based on shape or texture.

5.3 Cue Decomposition

We investigate the question of which cues in an image encode the most accessible information for a neural network to solve a semantic segmentation task. To this end, we decompose each image of a semantic segmentation dataset $\mathcal{D} \subseteq \mathcal{X} \times \mathcal{Y}$ in its main cues: color (RGB = V + HS), texture (T) and shape (S). The resulting datasets contain only a reduced number of cues or a single cue.

In the following we describe all datasets in detail.

5.3.1 Color

For the color cue we can use the original dataset $\mathcal{D}$ but need to constrain the neural network so that it has only access to a single pixel value. By restricting it to the color value we prevent learning from pixel arrangements which form shapes or texture patterns. To this end, we create a neural network consisting of nothing but (1×1)-convolutions. This limits the receptive field of the neural network to single pixels (cf. Section 2.1.2.1). We further decompose the color into two parts: Its gray value and its hue and saturation amount. Besides the RGB (red, green, blue) channel representation, color can also be encoded in the HSV (hue, saturation, value) format [258]. The HSV color space is a non-linear transformation of the RGB space, often represented by a cone. Hue ranges between 0 and 359 degree, representing the color on a color circle. Saturation denotes how the color changes by tinting, i.e., by mixing the color with white. For example, reducing the saturation of red leads to pink. The so-called value affects the darkness of the color. We can extract the **pure color**[26] amount by switching to the HSV color space and discarding the value channel. The grayness of the image can be obtained by either averaging or taking the maximum over the RGB color channels. The latter is equal to taking the V channel of an image in HSV format. In the following we denote the full three channel RGB color with **color** or **RGB**. A complete removal of all color components is not feasible and therefore gray scaled images are said to be color free. When we explicitly consider the split between grayness and the hue-saturation pair we refer to the grayness as **gray** or **V**. The hue-saturation pair is termed **HS**.

5.3.2 Texture

The texture dataset is built in three main steps. In the first step, object patches are extracted based on the semantic segmentation mask. In a second step, mosaic images are generated with patches of only one class ID and in the last step voronoi diagrams are generated [274]. For the voronoi diagrams a random ID is sampled for each cell and filled with a cutout from a corresponding mosaic image for each cell. A scheme of the process is shown in Figure 5.1. In the following, the steps are described in more detail.

5.3.2.1 Patch Extraction

We start by collecting textures of objects of a specific class ID $c \in \{1, \ldots, C\}$. Thereby, 'object' is not limited to an instance or foreground object but denotes any segment of class c. To this end, we mask every but class c segments in the segmentation mask. This leads to a mask which masks out everything except pixels from an image with class c label when applied to the image as shown in the step 'class-wise segment masking'

[26] We refer to 'pure color' for hue and saturation for the sake of simplicity to distinguish from the gray component. However, in literature often only saturated pixels are denoted as pure color.

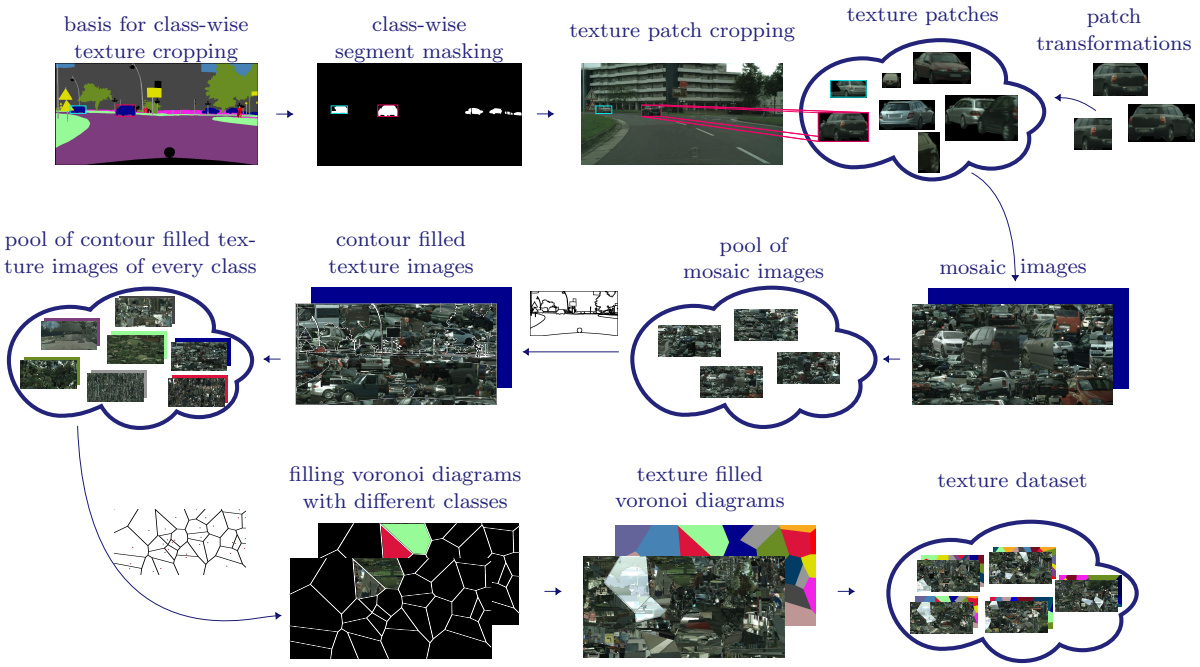


Figure 5.1: The diagram depicts our process of creating a texture dataset which only retains the texture cues from a given segmentation dataset. Zooming in is encouraged to view details.

in Figure 5.1. Next, the remaining segments are isolated by retrieving each contour with the help of the border following algorithm proposed by Suzuki et al. [262]. We discard segments which are below a certain threshold, as they will not contribute to meaningful texture patches. For all other isolated segments we cut out the corresponding image part along its enclosing bounding box based on the calculated contours. We denote the resulting image excerpts 'texture patches'. To enlarge the pool of texture patches, we upsample the patches according to a weighting factor with different image transformations as introduced in Section 2.4.1. A promising weighting to overcome class imbalance is to choose the weights according to the pixel counts per class over the entire dataset. For example, we add multiple differently transformed patches for classes which are underrepresented in terms of pixel count compared to classes that cover larger parts of the images. For the transformations, an invariance of the texture under these transformations should be assured.

5.3.2.2 Mosaic Images

After extracting all texture patches in the entire underlying dataset we randomly recompose them into mosaic images. Beginning always with the row that has the least amount of filled pixels and the first empty column. We then iteratively fill an image of the same size as the images in the original dataset. The silhouette of the texture patch which was filled in the mosaic is masked in a mask image to keep track of the empty and fill parts

of the image. Iterating this process creates an overlapping mosaic or patchwork image. We add noise to the insertion coordinates if the insertion position has not changed in two consecutive steps. This can happen as texture patches which define the insertion coordinates have varying contours but are cropped by their enclosing bounding box so that the silhouette does not fill up the entire box and empty pixels remain. To obtain differently sized patches in the mosaic, we randomly choose the biggest patch in the list of texture patches with a probability of $\mathfrak{p} = 0.1$ to be added next. This fill procedure is repeated until all texture patches are composed into mosaic images. When an image is completely filled, i.e., no empty pixels remain unmasked, a new empty image is filled. This leads to $N_c \in \mathbb{N}$ completely filled mosaic images which are paired with a constant label mask of class c.

To further reduce potentially preserved shape cues, we generate for each class ID a 'contour filled' texture dataset based on the original dataset. Therefore, we fill all original image segments with parts of different mosaic images of the same class c. This is done by masking everything but the specific segment in a randomly chosen mosaic image and combining these filled masks to a completely filled image. As a consequence, the object shapes in the original image are preserved but useless for prediction, as the texture and therefore also the class ID is identical inside and outside of any object. Furthermore, as for each class ID the same procedure is applied to generate contour filled texture mosaics, each object contour appears in any of these datasets. Applying this procedure to all classes $c \in \{1, \ldots, C\}$ leads to a pool of contour filled texture images where multiple images per class exist.

5.3.2.3 Voronoi Diagramms as Surrogate Task Data

In the last step, we build a semantic segmentation task based on the mosaic datasets. To this end, we create voronoi diagrams [274] where each cell represents a segment. The cells are filled with randomly chosen 'contour filled' texture image parts. We arrange the cell filling such that the resulting class IDs are balanced. That means that images from the pool are drawn randomly but uniformly with respect to the class ID. The corresponding segmentation label mask is created according to the class IDs of the filled texture per cell as seen in 'texture filled voronoi diagrams' in Figure 5.1. To generate an entire dataset, N voronoi diagrams are created and filled with texture.

The dataset created in that way is reduced to the cues texture and color. We refer to it as **texture RGB** or shortly **TRGB**. In Figure 5.2 we show two exemplary images based on the Cityscapes validation dataset. However, this method has challenges with long, thin objects, such as poles. Due to the thin structure of poles, it is difficult to capture their texture with our method without potentially introducing a new texture. The receptive field of standard neural networks is large enough to capture multiple poles next to each other which holds the potential that the texture expert learns a misleading pole texture. This is less of an issue for other classes as the segments vary more in size

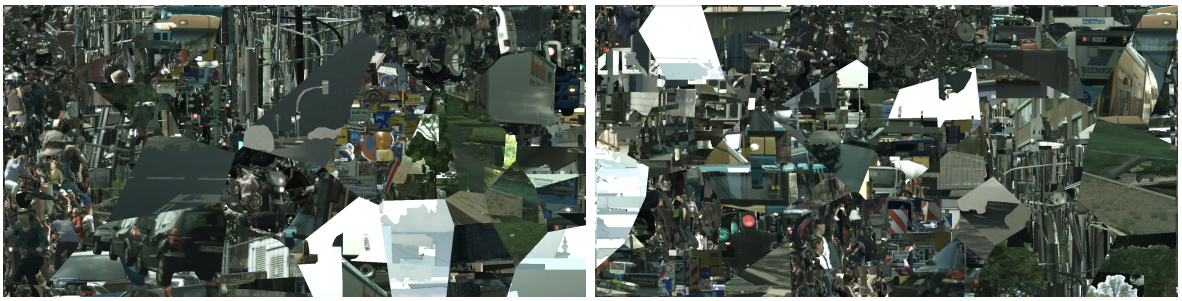


Figure 5.2: Two examples of our texture dataset.

and contour. To further remove the color cue, gray scaling is performed on the dataset, leading to the cues V and T referred to as **texture** or short **T**. Furthermore, we create a texture dataset with removed gray value but preserved hue and saturation values. We denote the dataset by **texture HS** or short **THS**. An overview of all cues datasets including the ones denoted here are listed in Table 5.1.

5.3.3 Shape

We consider three methods to extract the shape cue from images. This is anisotropic diffusion [93], holistically-nested edge detection [302] and, if the dataset is given by a modifiable simulation, texture removal. From a human visual perspective drawing the shape or contour of real world objects in a scene is fraught with uncertainty leading e.g., to a different granularity level of annotations [321]. We cover this ambiguity of shape perception with the three different shape cue extraction methods ranging from texture smoothing (anisotropic diffusion), edge detection (holistically-nested edge detection) and our approach to leave out the texture during the rendering process in a simulation.

5.3.3.1 Anisotropic Diffusion

In order to extract the shape cue, we need a method, which respects contour and shape defining edges but smooths the texture. A common approach to produce smooth images is applying a Gaussian blur filter. However, this smoothing is uniform with respect to all orientations, i.e., all image components are blurred in the same way. As a consequence, it fails to preserve the sharpness of edges. To preserve shape defining edges while smoothing local texture, we apply edge enhancing diffusion with smoothed orientation (EED) [93, 293] which is a special form of anisotropic diffusion [202]. Based on a partial differential equation (PDE), EED diffuses the image along edges but hampers the diffusion across them. This leads to a non-linear and space-variant image transformation where the image is blurred step-by-step in a diffusion process but the blurring filter is applied non-uniformly depending on the local spatial gradients.

We follow the notation of Schmaltz et al. [240] for formulating the underlying PDE. For the sake of simplicity we restrict the description to gray scale images[27]. The PDE with zero valued Neumann boundary conditions and the original image Φ_0 as initial value is given by

$$\begin{aligned} \partial_t \Phi &= \operatorname{div}\left(\bar{g}(\nabla\Phi_\sigma(\nabla\Phi_\sigma)^{\mathrm{T}})\nabla\Phi\right), \\ \Phi(y,x,0) &= \Phi_0(y,x) \end{aligned} \tag{5.1}$$

where $\Phi_\sigma := K_\sigma * \Phi$ is a smoothed version of the image Φ achieved by convolving it with a Gaussian kernel K_σ with standard deviation σ and size k, and the matrix function $\bar{g}$ is a so-called diffusivity function. It is chosen in such a way that the resulting diffusion tensor $\bar{g}(\nabla\Phi_\sigma(\nabla\Phi_\sigma)^{\mathrm{T}}) \in \mathbb{R}^{2\times 2}$ has one eigenvector which is parallel and the other one orthogonal to $\nabla\Phi_\sigma$. The corresponding eigenvalues are given by $\bar{g}(|\nabla\Phi_\sigma|^2)$ and 1. As proposed in [240] we use the Charbonnier diffusivity [34]

$$\bar{g}(s^2) = \frac{1}{\sqrt{1+\frac{s^2}{\gamma^2}}} \tag{5.2}$$

with some contrast parameter $\gamma > 0$ regulating the diffusion across edges. Charbonnier et al. studied conditions for edge preserving diffusivity functions in [34] and showed that this function type leads to a well posed problem and is much simpler than other candidates mentioned in the publication. The diffusion is implemented numerically by discretizing the diffusion process in space and time [294]. For each time interval a gradient descent step of the space-discretized quadratic energy model

$$E(\Phi) = \frac{1}{2}\int (\nabla\Phi)^{\mathrm{T}}\bar{g}(\nabla\Phi_\sigma(\nabla\Phi_\sigma)^{\mathrm{T}})\nabla\Phi \,\mathrm{d}x\,\mathrm{d}y \tag{5.3}$$

is computed. The space discretization is performed by the non-standard finite differences described by Weickert et al. [294]. To avoid circular singularities an additional orientation smoothing, originally applied in the context of coherence enhancing diffusion [292], is incorporated into the EED process as proposed in [93]. This procedure is iteratively repeated for each discrete time step t until a total amount of $T \in \mathbb{N}$ time steps is processed.

For multichannel images, e.g., color images, the edge enhancing diffusion is obtained by considering a joint diffusivity tensor leading to the PDE

$$\partial_t \Phi = \operatorname{div}\left(\bar{g}\left(\sum_{i=1}^{m} \nabla(\Phi_\sigma)_i(\nabla(\Phi_\sigma)_i)^{\mathrm{T}}\right)\nabla\Phi_i\right) \tag{5.4}$$

as proposed in [293].

[27] For better readability we use Φ instead of $\boldsymbol{x}$ to represent the input to easily distinguish between the spatial and temporal components.

Figure 5.3: Image diffused by EED (left) and original image (right). The texture is smoothed along the edges but only slightly across them.

The dataset generated by EED is referenced in the following as **SRGB** and has reduced texture, but color and shape are widely preserved. An example of a diffused image is shown in Figure 5.3. To further reduce the cues, we consider a gray scaled version of the dataset (**shape$_{\mathbf{EED}}$**) and a version where we only keep the pure color components of the diffused image by removing the V-channel in HSV format (**shape$_{\mathbf{EED}}$ HS**). The anisotropic diffusion allows us to create a color preserved shape dataset. Nevertheless, we need to accept that edges are less marked and even though the overall object shape is preserved, slight smoothing is observed across the edges. This leads to a minor pixel offset compared to the semantic segmentation mask.

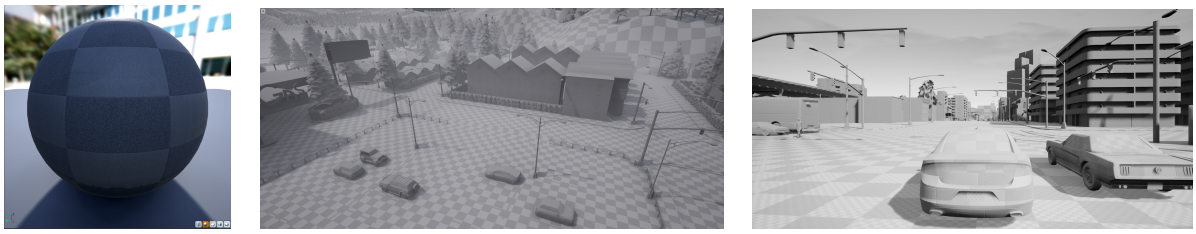
5.3.3.2 Holistically-Nested Edge Detection

Reducing everything but the object describing edges can be achieved by holistically-nested edge detection (HED) [302]. The following description is based on the original publication by Xie and Tu [302]. HED is an object contour detection method based on fully convolutional networks to learn a dense edge map through an end-to-end training. The fully convolutional approach is supplemented by a nested multiscale feature learning in which contours are predicted by side outputs of intermediate layers of different scales. This concept is inspired by deeply supervised networks [149]. These two core concepts framed the name components 'holistic' and 'nested'. The first one describes that an input image is translated entirely to an edge map and thought of as a whole by the fully convolutional approach. The second component ('nested') denotes the progressive refinement steps in which the edge map prediction is done, exploiting the natural multiscale feature representation in FCNs (cf. Section 2.1.1). In natural images different strength of shading and edges are present. Edges do not only define contours and thereby the shape of objects but also, e.g., the texture by thinner or lighter edges like fur of an animal. This ambiguity is tackled by learning a hierarchical representation of the edge maps by side outputs leading to a cascade of nested edge map predictors. Based on the down sampling structure in FCNs the receptive field enlarges in deeper layers while simultaneously reducing the size of the feature map by, e.g., pooling operations. Taking advantage of this multiscale feature representation, early layers side outputs

Figure 5.4: HED (left) and original image (right). The shape defining edges are preserved by HED whereas the finer texture defining edges are removed.

learn to predict edge maps with local information and therefore fine edges, whereas later layers side outputs learn more high level edge compositions. HED is realized by a single stream neural network, a trimmed VGG network [255], where after each of the 5 remaining stages, i.e., a block of a pooling layer (in all but the first block) and a couple of convolutional layers with ReLU activation, the layer activation is fed to a (1×1)-convolution and upsampled to the original image shape. This enables the comparison of the predictions per side output to the ground truth map. As a consequence, all side output layers can be understood as stand alone (nested) pixel-wise predictors at a certain scale sharing the same backbone parameters. This leads to a progressive refinement from local to global in terms of edge information and from fine to coarse in terms of predicted edge maps. These individual predictions are combined in a fusion layer. As a consequence, HED allows for coherent contributions of all layers due to an interpolation of the side outputs in the fusion layer. Training is done with the help of the sum of weighted cross entropy losses for each side output layer including the final fusion layer. The weight is set to mitigate the imbalance between edge and non-edge pixels. For inference the average over all side outputs and the final fusion layer is taken.

HED is a fast contour detection algorithm which allows for more transparency due to the intermediate representations enforced by deep supervision. The supervision of the hidden layers motivated in deeply-supervised nets [149] improves optimization and generalization as well as the transparency of the approach due to natural and intuitive outputs from the intermediate layers. It is worth noting that no explicit constraints are imposed on contextual information like neighbor pixel constraints to ensure connectivity of edges. Furthermore, the authors found that multiscale prediction is crucial for edge detection and a plain fully convolutional network is most likely not enough to solve the task properly as the exact pixel location is essential for edge detection. For shape cue extraction HED is a good choice as the object describing shapes are preserved while texture is removed [87]. In contrast to the original method we invert the colors to get black contours on white background. We postulate that the inverted colors might have a slightly smaller domain gap to real world images. An example of the so generated $\mathbf{shape_{HED}}$ dataset is visualized in Figure 5.4.



(a) basis texture (b) texture removed CARLA city (c) post-processed frame

Figure 5.5: Generation aspects of the textureless dataset based on a texture free world in CARLA.

5.3.3.3 Texture Removal

With the help of the open source simulator CARLA we are able to generate a world which is texture free. CARLA claims that online switching between textures is possible since version 0.9.14. However, this feature is limited to a few instances and therefore does not fulfill our requirements of a complete texture free world. As a consequence, we manipulate the property of each material instance by hand. By replacing all surfaces by volume objects they lose their original texture and are equipped with a basis texture. This texture is a gray checkerboard as visualized in Figure 5.5a. The sky is not a meshed object with material and therefore not rendered in the same way as the other objects. For that reason, we cannot manipulate its texture. By choosing clear noon as weather and lighting conditions, a minimal amount of clouds is generated. This leads to a nearly constant texture for the sky and is the best we could get from the simulation. The manipulated almost texture free world is used for recordings by an ego vehicle in autopilot mode equipped with a camera and a semantic segmentation sensor. The recording details are identically to those of the RGB CARLA base dataset which we introduce in Section 5.5.1. After recording frames in the texture free world, we post-process the images by gray scaling them to remove the color of the sky. In the following, we refer to this dataset as **textureless** respectively $\mathbf{shape}_{\mathbf{textureless}}$ or in short $\mathbf{S}_{\mathbf{rmv}}$. An example of a textureless city and a recorded and post-processed frame is shown in Figures 5.5b and 5.5c. As all objects are equipped with the same basis texture no information can be drawn from the texture cue and therefore only the shape cue is preserved in this dataset.

5.3.4 Cue Experts

Based on the cue decomposition, we train neural networks which we call 'cue experts' for each cue constellation of gray (V), pure color (HS), texture (T) and shape (S). We train 14 experts for real world data by including or excluding specific cues such that all cue combinations are covered. This includes an expert on the original dataset which consists of all cues as well as a randomly initialized neural network to simulate the

Table 5.1: Overview of cues. The included cues are V = gray component of the color, HS = hue and saturation component of the color, S = shape and T = texture.

short	long name	included cues	description
O	original / all cues	V + HS + S + T	all cues
OHS	original HS	HS + S + T	all but gray cues
OV	original gray	V + S + T	all but color cues
TRGB	texture RGB	V + HS + T	texture with color; shape removed
THS	texture HS	HS + T	texture with hue and saturation only
T	texture	V + T	texture only; shape and color removed
SRGB	$\text{shape}_{\text{EED}}$ RGB	V + HS + S	shape with color via smoothing texture by edge enhancing diffusion
SHS	$\text{shape}_{\text{EED}}$ HS	HS + S	shape with hue and saturation by edge enhancing diffusion
S	$\text{shape}_{\text{EED}}$	V + S	shape only by gray scaled edge enhancing diffusion results
S_{HED}	$\text{shape}_{\text{HED}}$	V + S	shape only via contour map by holistically-nested edge detection
S_{rmv}	$\text{shape}_{\text{textureless}}$	V + S	shape only via texture removal in the simulation process
RGB	color	V + HS	complete color component of an RGB image, pixel-wise
HS	HS	HS	hue and saturation of an RGB image, pixel-wise
V	gray	V	gray component of an RGB image, pixel-wise
no info	no information		no cues/data present

absence of all cues. For data which is generated by a computer simulation where access to the object materials is granted, we additionally train an expert on $\text{shape}_{\text{textureless}}$ data. In the following experiments, we analyze the cues and cue combinations listed in Table 5.1. As explained in Section 5.3.1, texture and shape cannot exist without any color component. Therefore, the texture and shape expert additionally contain the V cue. Since we consider up to three shape extraction methods, all typify the same cue combination S and V but are generated by three different methods. For more details see Section 5.3.3. By training up to 15 different experts, we can analyze the influence of the cues by comparing their capability of solving a segmentation task. The experts are evaluated with respect to their performance measured in mIoU (cf. Equation (2.119)). We further analyze the deviances between nested cue-experts in Section 5.5.10. A scheme of the method is depicted in Figure 5.6. A visual overview of the different cue datasets on Cityscapes is visualized in Figure 5.16.

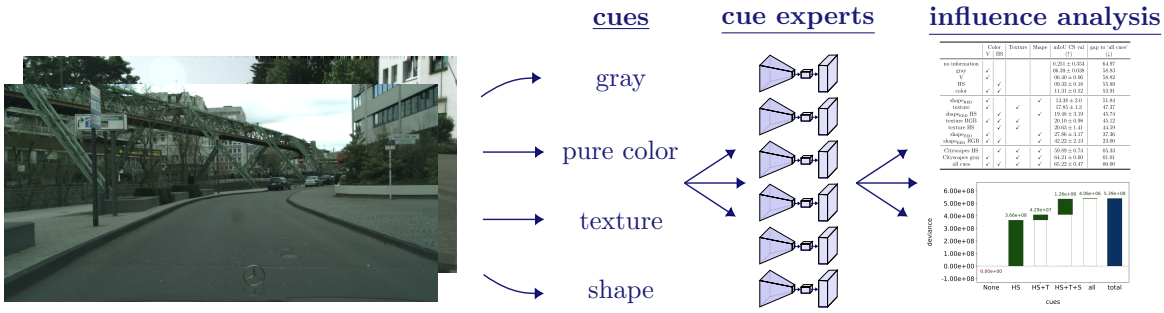


Figure 5.6: A scheme of our analysis method. First, cues are extracted from the dataset. Then neural networks are trained on each of the cue datasets. The resulting cue experts give insight into what neural networks can learn from certain cues.

5.4 Late Fusion

For semantic segmentation tasks predictions are made on pixel-level. To analyze the different cue influences on pixel-level we apply a late fusion approach to the cue experts. To achieve this, we start by training the experts on the individual cue datasets. Then, a slim semantic segmentation model is trained on top of the softmax outputs of the trained cue experts which serves as late fusion model. The objective of the fusion model is to predict, on a per-pixel basis, which expert is best suited to properly solve the semantic segmentation task. In more detail, the number of classes C_{exp} for the fusion model is set to the number of experts we would like to fuse. This ranges between two and three experts in our case. Thereby we reduce the redundancy in the cues to a minimum to be able to clearly distinguish between the source of influence and fit to memory limitations. Given the concatenated softmax tensors of the experts of an input image $\boldsymbol{x}$, the fusion network predicts a probability distribution over the experts in terms of a softmax output over C_{exp} classes. To achieve an end-to-end trainable neural network, we calculate the convex combination of the expert softmaxes weighted by the softmax output of the fusion network. Since a convex combination of a probability measure is also a probability measure, we can use this tensor to calculate the loss with respect to the segmentation mask corresponding to the input image $\boldsymbol{x}$. The softmax output of the fusion network can be understood as an attention on the different experts. It allows us to measure on pixel-level the influence of the different cue experts on the overall prediction.

5.5 Cue Influence Analysis

To analyze the influence of the different cues and their interplay in semantic segmentation tasks we consider two base datasets. We introduce the datasets in Section 5.5.1 and give details on our implementation and experimental setup in Section 5.5.2 and

Section 5.5.3. Thereafter, we report our analysis results on global, class and pixel level. The influences are measured with the help of a performance-based measure with respect to the mIoU of the different experts and the difference in deviance of nested experts.

5.5.1 Base Datasets

We evaluate the influences of the cues on two different automotive datasets. First we consider Cityscapes (CS), a dataset of urban street scenes with 19 different classes. A detailed description was given in Section 4.4.1.1. Additionally, we generated a dataset with the help of the open source simulator CARLA version 0.9.14 [59]. CARLA contains multiple enumerated maps of which we used the towns 1-5 and 7 for data generation. We chose these maps since they have the same level of detail. An ego vehicle records in total 5,000 RGB frames per city with corresponding semantic segmentation masks. Therefore, in each town we randomly spawn vehicles and pedestrians multiple times while recording every 50-th frame during the drive of the autopilot through the virtual city. The environmental settings like weather are set to 'clear noon'. The image resolution is set to the same as for Cityscapes, i.e., each frame consists of 2,048 $\times$ 1,024 pixels. We consider a reduced number of the CARLA classes to best coincide with the Cityscapes classes. This results in the 15 classes: *road, sidewalk, building, wall, pole, traffic lights, traffic sign, vegetation, terrain, sky, person, car, truck, bus, guard rail.* The classes 'bicycle', 'rider' and 'motorcycle' are excluded for technical reasons. They could not be included as these actors were problematic for the autopilot mode in CARLA. To ensure an independent train and test split we use town 1 and 5 for testing only, whereas 2,3,4 and 7 are used for training. This results in 20,000 training images and 10,000 test images.

5.5.2 Implementation

In addition to the input data, Hermann et al. studied different sources of cue influences in [95]. These sources range from network architectures and data augmentation strategies to the training objective. To reduce the influence of non-data sources we fix all these components where possible during our study. Taking into account that we train all neural networks on moderately small semantic segmentation datasets from scratch to ensure only the desired cues are learned, we constrain ourselves to slim models. For all cue experts, except the color experts, we use deeplabv3 with ResNet18 backbone as CNN representative (cf. Sections 2.3.1.1 and 2.3.2.2) and a light weight SegFormer-B1 model as transformer model (cf. Section 2.3.2.3). With 15.9 million parameters for deeplabv3 with ResNet18 backbone and 13.7 million parameters for the SegFormer-B1 model, both models have a similar capacity with respect to the number of learnable parameters. For the fusion network we use the deeplabv3 model with an even smaller backbone. We reduce the ResNet18 from 4 to 2 blocks. This is to prevent overfitting due to the relatively

simple task. We adapt the deeplabv3 model provided by torchvision[28] to our need for reduced CNN backbones. For the transformer we make use of the implementation provided by the toolbox MMSegmentation [179]. As described in Section 5.3.1, we restrict the receptive field of the color expert to one pixel which we implemented using an FCN with composed (1×1)-convolutions. The backbone of the resulting color network for the Cityscapes experiments consists of two (1×1)-convolutions with 256 channels each followed by batch normalization and ReLU activations. For the CARLA experiments we use three (1×1)-convolutional layers. In both setups we add a customized FCN head in which all convolutions are replaced by (1×1)-convolutions. In detail, the head consists of a (1×1)-convolution with 64 channels with batch normalization and ReLU activation followed by a dropout layer with a probability of $\mathfrak{p} = 0.1$ for each weight to be dropped. Lastly, a (1×1)-convolution maps the features to the number of classes. We decided to create a shallower network for Cityscapes as the training dataset is smaller compared to the CARLA dataset. The objective for this is to prevent overfitting. To ensure that this does not restrain the color expert, we also trained the version with three layers for the RGB color expert over 5 runs. We observed a similar performance which only improves by 0.5% mIoU on average. Since all trainings are fraught with variances, we conclude that two layers are enough for the color expert on Cityscapes.

Except for the fusion network, all neural networks were trained from scratch on a single Quadro RTX 8000. We provide details on the fusion training at the end of this paragraph. We trained all but the transformers for 200 epochs with the Adam optimizer and poly-linear learning rate decay with an initial learning rate of 0.0005. For the transformers, we kept the default optimizer settings in MMSegmentation. As we trained the transformer from scratch we enlarged the number of epochs to 400. This translates in 170,000 iterations with a batch size of 7. For Cityscapes, we randomly crop patches of 512×512 pixels during training for data augmentation. For the CARLA dataset we use slightly smaller crop size of 256×256. We normalize the inputs using the mean and standard deviation of the pixel distribution of each specific dataset rather than the widely used ImageNet mean and standard deviation. The reason for this is the same as for training from scratch to mitigate the inductive bias from pre-trained models or normalization parameters. This ensures that training is free from external biases and that the neural network only learns features of the cue dataset. For the inference of the transformer, we follow the suggestion of the authors of SegFormer and apply a sliding window of the same size as the training cropped patches during test time. By inferring patches of size $(512, 512)$ at a stride of 384, the entire input image is processed.

For splitting the color further into its gray component and its hue and saturation values, we base our implementation on the color conversion implementation of Ma [166]. It translates Pytorch tensors to HSV. We extract the HS component by splitting the channels of the resulting tensor into a two-channel image (HS) and discard the V-channel.

[28] `https://github.com/pytorch/vision/blob/main/torchvision/models/segmentation/deeplabv3.py`

To remove the color we use the standard RGB to gray color conversion method from the Image module of Pillow[29].

For the anisotropic diffusion we used the empirically determined hyperparameters

- Contrast parameter $\gamma = 1/15$
- Gaussian kernel size $k = 5$ and standard deviation $\sigma = \sqrt{k}$
- number of steps $n = 8{,}192$.

These parameters lead to a significant texture smoothing for both the Cityscapes and CARLA dataset.

To extract object contours in images, we use the HED implementation from [87]. This implementation bases on [191], a reimplementation of the original HED using PyTorch. As only a few contour map datasets are publically available and training or fine-tuning low-level edge detection highly depends on hyperparameters according to [302], we use the pre-trained network provided by [191] for generating $\text{shape}_{\text{HED}}$ images.

The fusion network was trained from scratch on an A100 80GB to meet the large memory requirements of the concatenated softmax tensors. In contrast to the experts, the fusion converges much faster so that we trained it for only 75 epochs. The weights of the fusion network are initialized randomly according to a uniform distribution on $[-10^{-3}, 10^{-3}]$ to allow for equal weighting of the influence of the experts' softmax at training start. To further improve training stability, we normalize the softmax and use cropping as data augmentation for the fusion training.

5.5.3 Experimental Setup and Evaluation Metrics

We train 14 (15) experts each on one of the cue specific datasets listed in Table 5.1. We use the notation from Table 5.1 to refer to the datasets as well as to the experts. To analyze the influence of cues, we measure the performance of the experts in terms of mIoU (see Equation (2.119)). Therefore, we evaluate the experts on the Cityscapes validation dataset in the first experiment or on the CARLA test set in the second experiment respectively. To ensure input compatibility we evaluate one channel experts, i.e., all experts considering gray cues which equals (b), (e), (g), (h) in Figure 5.16, on gray scaled images and two-channel experts, i.e., all HS experts, cf. Figure 5.16 (c), (f), (i), (k), (l) on images transferred to HSV with a dropped V-channel. With this we evaluate how accurate the semantic segmentation task can be solved with reduced cues. Particularly, we study the gap between the oracle performance ($\text{mIoU}(\mathcal{D}_{\text{all}})$) and the experts trained on datasets with reduced cues ($\text{mIoU}(\mathcal{D}_{\text{cue}})$) using

$$\text{gap}_{\mathcal{D}_{\text{all}}}(\mathcal{D}_{\text{cue}}) = \text{mIoU}(\mathcal{D}_{\text{all}}) - \text{mIoU}(\mathcal{D}_{\text{cue}}). \tag{5.5}$$

[29] `https://pillow.readthedocs.io/en/stable/index.html`

Comparing the reduced model with the full model reveals the **importance** or **influence** of the removed cues for solving the original semantic segmentation task. The more the performance drops when removing a cue, the more important is the cue for solving the task. This is revealed by a large gap. For example, the importance of texture (T) can be analyzed by looking at the gap of the expert trained on the cues HS and S

$$\begin{aligned} \text{Importance(T)} = \text{Importance(all} \setminus \text{SHS)} &= \text{gap}_{\mathcal{D}_{\text{all}}}(\mathcal{D}_{\text{SHS}}) \\ &= \text{mIoU}(\mathcal{D}_{\text{all}}) - \text{mIoU}(\mathcal{D}_{\text{SHS}}). \end{aligned} \tag{5.6}$$

The performance gap also reveals how important the remaining cues are to solve the segmentation task. If the gap of a cue constellation is large, i.e., the gap between the remaining cues of the reduced model and the oracle model, the remaining cues are less important to solve the segmentation task. We can also analyze the importance of two orthogonal experts by comparing their gaps to the expert which includes both cues. For texture and shape we define

$$\begin{aligned} \text{gap}_{\mathcal{D}_{\text{OV}}}(\mathcal{D}_{\text{T}}, \mathcal{D}_{\text{S}}) &= \text{gap}_{\mathcal{D}_{\text{OV}}}(\mathcal{D}_{\text{T}}) - \text{gap}_{\mathcal{D}_{\text{OV}}}(\mathcal{D}_{\text{S}}) \\ &= \big(\text{mIoU}(\mathcal{D}_{\text{OV}}) - \text{mIoU}(\mathcal{D}_{\text{T}})\big) - \big(\text{mIoU}(\mathcal{D}_{\text{OV}}) - \text{mIoU}(\mathcal{D}_{\text{S}})\big) \end{aligned} \tag{5.7}$$

We trained each expert for the Cityscapes experiments with 5 different seeds and report the mean performance and its standard deviation by averaging over the 5 runs and calculating the empirical standard deviation with Bessel's correction. For CARLA due to the larger dataset and thus longer training times we use 3 different seeds for mean and standard deviation calculation to reduce computational effort. The resulting mIoU performances and the performance gaps according to Equation (5.5) for the convolutional neural network are shown in Table 5.2 for Cityscapes and in Table 5.3 for our experiments on the CARLA dataset. The results for the transformer model on Cityscapes are listed in Table 5.8. For a lower bound reference, we also compute for all network architectures the performance of a freshly initialized network which has not seen any cues or data for 5 seeds. We observe that a freshly initialized neural network does not exceed a performance of 1% mIoU on average for deeplabv3 with ResNet18 backbone and is below 1.5% mIoU for the FCN with (1×1)-convolutions. The SegFormer-B1 model for the Cityscapes setup achieves 0.328% mIoU on average when no information is given.

5.5.4 Cityscapes Experiments

The expert performances of the different cue constellations are summarized in Table 5.2. Firstly, our experiments confirm that the cue importance for a CNN follows the expected ordering of the form:

color related cues only < texture or shape cues < all but color cues < all cues.

Table 5.2: Cue influence in terms of mIoU performance drop on Cityscapes. Cue description follows the listing in Table 5.1 except that original is denoted by the dataset, i.e., Cityscapes.

	Color		Texture	Shape	mIoU CS val	$\text{gap}_{\mathcal{D}_{\text{all}}}$
	V	HS			(↑)	(↓)
no information					0.25 ± 0.35	64.97
gray	✓				6.39 ± 0.04	58.83
HS		✓			9.33 ± 0.18	55.89
color	✓	✓			11.31 ± 0.52	53.91
$\text{shape}_{\text{HED}}$	✓			✓	13.38 ± 2.00	51.84
texture	✓		✓		17.85 ± 1.30	47.37
$\text{shape}_{\text{EED}}$ HS		✓		✓	19.48 ± 3.19	45.74
texture RGB	✓	✓	✓		20.10 ± 0.98	45.12
texture HS		✓	✓		20.63 ± 1.41	44.59
$\text{shape}_{\text{EED}}$	✓			✓	27.86 ± 3.17	37.36
$\text{shape}_{\text{EED}}$ RGB	✓	✓		✓	42.22 ± 2.13	23.00
Cityscapes HS		✓	✓	✓	59.89 ± 0.74	5.33
Cityscapes gray	✓		✓	✓	64.21 ± 0.60	1.01
all cues	✓	✓	✓	✓	65.22 ± 0.47	0.00

Furthermore, our experiments show that neural networks can draw the most information from an image if all cues are present. This is hardly surprising, as the neural network can extract features from all cues and learn cue correlations. On the other hand, the pure color component (HS cue) does not have a significant influence if all other cues are present (cf. $\text{gap}_{\mathcal{D}_{\text{all}}}(\mathcal{D}_{\text{Cityscapes gray}})$). More interestingly, we observe that color cues alone enable a non-negligible gain in performance compared to an uninformed neural network. Although the performance of 11.31% mIoU is barely enough for a clear understanding of the scene, the classes *road*, *vegetation* and *sky* are well predicted and rarely confused as shown in Figure 5.7.

In the following, we have a closer look at the texture and shape cues and review their interference on the color cue and its components. The experiments reveal that when adding the gray cue to 'texture HS', which equals the 'texture RGB'-expert, the performance is identical within the variance range of the training. We observe that the gray shading is less important for texture than information encoded in the pure color component. On the other hand, when adding the hue and saturation to 'texture', the performance rises only by 2.25 pp. We conclude that color has no significant influence on texture cues in a real world German street scene segmentation. In contrast, we observe more differences in performance for the shape cue constellations. The results reveal that pure color seems to be less relevant for the shape expert when solving the Cityscapes segmentation task. Nevertheless, the combination of all color components

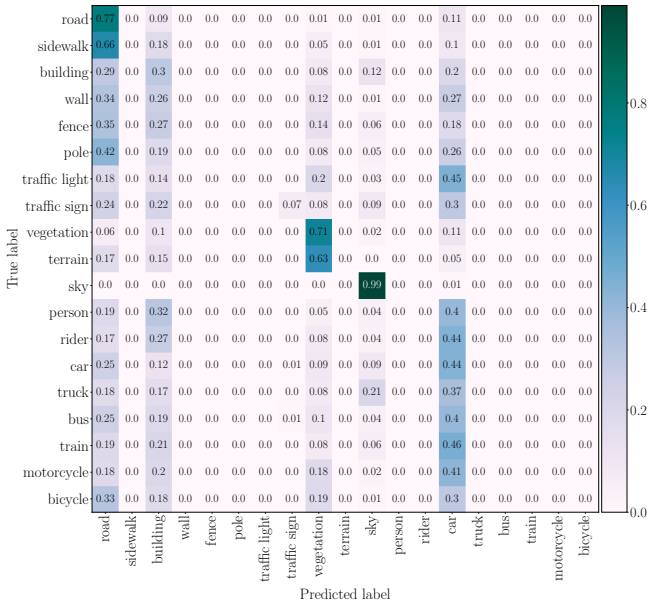


Figure 5.7: Confusion matrix of the RGB color expert. The true labels are displayed on the y-axis whereas the predictions from the color expert are plotted on the x-axis. The diagonal shows the percentage in $[0, 1]$ of the correctly predicted pixels. The class *sky* is nearly always correctly predicted. The *vegetation* class is mostly confused with the *terrain* class which are similar in color. The same can be observed for the classes *road* and *sidewalk*.

and the shape cue shows a distinct improvement in performance. This can be seen by looking at 'shape$_{\text{EED}}$ RGB' and comparing the relative performance drops when removing the gray component in contrast to removing the pure color component HS

$$\text{gap}_{\mathcal{D}_{\text{SRGB}}}(\mathcal{D}_{\text{SHS}}, \mathcal{D}_{\text{S}}) = \big(\text{mIoU}(\mathcal{D}_{\text{SRGB}}) - \text{mIoU}(\mathcal{D}_{\text{SHS}})\big) - \big(\text{mIoU}(\mathcal{D}_{\text{SRGB}}) - \text{mIoU}(\mathcal{D}_{\text{S}})\big).$$

We observe that the performance drops by 22.74 pp when the gray component is removed whereas performance drops only by 14.36 pp when removing the HS component. That the information encoded in the gray channel is more important than pure color for shape might be explained by the shape properties. Shape can be defined by edge compositions and those again are defined by large image gradients. Gradients occur when the intensity or color in an image changes directionally. It seems that a change in color has less influence than intensity changes in the Cityscapes setup. One reason for this observation might be that segment boundaries are more important than other edges. Therefore, color changes such as in traffic signs or colorful clothing might have been learned to be ignored. However, the combination of gray shading and a change in saturation and/or hue improves the shape experts' ability to correctly predict segments in Cityscapes. The finding that the composition of shape with all color components achieves a much higher performance suggests that there are correlations which cannot be learned by one of the standalone cue experts.

Table 5.3: Cue influence in terms of mIoU performance (drop) on the CARLA test dataset. The table is ordered ascending in terms of mIoU performance. The last column depicts if experts dropped positions in the table (↘) or raised positions (↗) compared to the ordering observed for the Cityscapes experts (cf. Table 5.2). Due to the ascending order, dropping in the table equals a better mIoU performance compared to other experts.

	Color		T	S	mIoU test	$\text{gap}_{\mathcal{D}_{\text{all}}}$	table position
	V	HS			(↑)	(↓)	change
no information					0.38 ± 0.44	75.27	→
gray	✓				6.01 ± 0.10	69.64	→
$\text{shape}_{\text{HED}}$	✓			✓	11.35 ± 0.65	64.30	↗
HS		✓			14.70 ± 0.18	60.95	→
color	✓	✓			15.60 ± 0.56	60.05	→
$\text{shape}_{\text{textureless}}$	✓			✓	26.96 ± 2.00	48.69	
$\text{shape}_{\text{EED}}$	✓			✓	37.10 ± 2.13	38.55	↗
$\text{shape}_{\text{EED}}$ HS		✓		✓	44.97 ± 0.93	30.68	→
texture	✓		✓		46.07 ± 3.34	29.58	↘
texture HS		✓	✓		52.38 ± 1.65	23.27	→
texture RGB	✓	✓	✓		56.51 ± 1.75	19.14	↘
$\text{shape}_{\text{EED}}$ RGB	✓	✓		✓	61.47 ± 1.26	14.18	→
CARLA HS		✓	✓	✓	71.02 ± 0.93	4.63	→
all cues	✓	✓	✓	✓	75.65 ± 1.80	0.0	↗
CARLA gray	✓		✓	✓	75.73 ± 1.07	−0.08	↘

5.5.5 CARLA Experiments

In addition to the Cityscapes experiments, we also applied our analysis method to the self-generated CARLA dataset introduced in Section 5.5.1. For CARLA, we see a slightly different ordering of the experts' performances and performance gaps which are presented in Table 5.3. Performance changes leading to a rearrangement in the expert's order are noted with ↘ and ↗ in the right-most column. The table shows that in contrast to the Cityscapes experiments the performances of the texture experts improve whereas two shape experts can draw less information from CARLA data to solve the segmentation task. One reason for this might be the limited variety of texture and shape that occur in a simulation. Due to the finite number of assets rendered in CARLA, the diversity of appearance of objects to be segmented is smaller compared to naturally appearing objects and scenes. Except for the combination with the full color cue, texture experts solve the segmentation task better in terms of mIoU than their shape counterparts with the same color cue component present (compare, the performance of e.g., $\text{shape}_{\text{EED}}$ and texture). There are two possible reasons for this: On the one hand we postulate that the shape of objects is somewhat more diverse as objects

Trees in the textureless world | Full rendered trees in CARLA

Figure 5.8: Comparison of textureless (left) and full (right) rendering in CARLA for tree structures. Removing the texture reveals that the shape of the leaves is encoded by the texture instead of shape of the objects itself.

are viewed at different distances and view angles, which could explain the gap between texture and shape experts. On the other hand texture experts tend to specialize in a few classes rather than being a general segmentation model. This hypothesis is discussed in more detail in Section 5.5.7. In addition, we see that the texture as well as the shape experts benefit from more and more color cue components. Particularly, $\text{shape}_{\text{EED}}$ RGB improves distinctly compared to $\text{shape}_{\text{EED}}$ where the pure color is removed. However, in line with the Cityscapes experiments the pure color components are less important to solve the segmentation task, if shape and texture are present as demonstrated in the last two rows in Table 5.3.

CARLA allows us to examine a completely 'textureless' world as we control the rendering process (cf. Section 5.3.3.3). Removing the texture completely enables us to compare the shape extraction methods for their suitability as generators for shape cue datasets. The textureless expert cannot learn from the texture in simulated images as all texture was replaced by a default layer. Even though it could be argued that there is a kind of texture on the objects, it is the same for all visual parts and therefore not discriminative. The textureless world demonstrates that texture can also have an influence on the shape of objects. An example is the rendering of leaves on trees. Leaf structures are generated with the help of transparent areas in the texture pattern. After removing the texture, only rectangular plates remain as the crown of the tree, which can hardly be perceived as realistic foliage. The examples in Figure 5.8 show that cues come with different peculiarities and interfere with each other. The consequence is that we encounter a more or less drastic domain shift when disentangling image cues.

5.5.6 Domain Shift Due to Cue Reduction

As experts are trained on a specific cue dataset but evaluated on the original dataset, our evaluation is subject to domain shift. The domain shift is differently pronounced

Table 5.4: Shape cue influence in terms of mIoU performance when evaluated in-domain, i.e., the validation dataset is transformed in the same way as the training dataset of the experts during evaluation. The table is sorted according to the performances on the original Cityscapes validation set (cf. Table 5.2). The best performance is highlighted in bold and the second best is underlined.

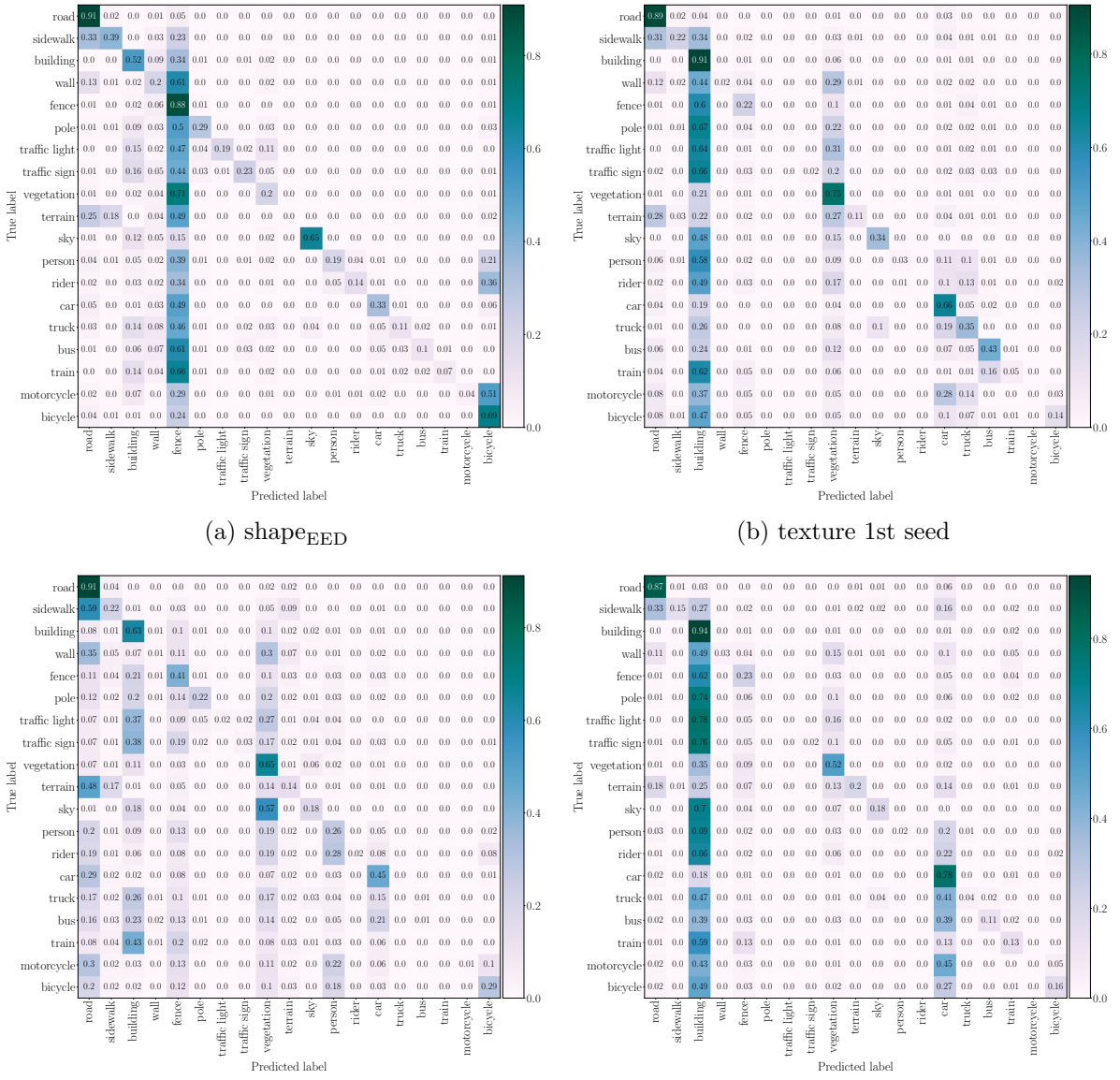
	mIoU Cityscapes val in-domain (↑)	mIoU CARLA test in-domain (↑)
$\text{shape}_{\text{HED}}$	**55.80** $\pm$ 0.59	<u>63.33</u> $\pm$ 1.11
$\text{shape}_{\text{EED}}$ HS	45.89 $\pm$ 0.51	61.67 $\pm$ 1.90
$\text{shape}_{\text{EED}}$	45.02 $\pm$ 0.51	60.73 $\pm$ 2.70
$\text{shape}_{\text{EED}}$ RGB	<u>48.47</u> $\pm$ 0.45	**65.93** $\pm$ 0.72
$\text{shape}_{\text{textureless}}$	-	62.03 $\pm$ 1.51

for different experts. However, this domain shift cannot be mitigated for the texture cue as we always need to at least relax the segment boundaries during training to avoid shape learning. Nevertheless, all shape cue extraction approaches can be understood as online methods. This means that we can apply them to a single input image without disturbing the semantic segments and therefore the actual semantic segmentation task. To allow for a fair comparison, the texture as well as the shape experts were evaluated on the base dataset and therefore faced the domain shift in the evaluation in Table 5.2. In the following, we review the shape cue for which we can surpass the domain gap. Therefore, we use the Cityscapes validation set as input for the shape experts but apply the corresponding shape cue extraction method to it before feeding the input to the expert. To distinguish the two performances, we call the performance based on the online evaluation process 'in-domain performance'. We compare the performances of $\text{shape}_{\text{EED}}$, $\text{shape}_{\text{HED}}$, $\text{shape}_{\text{EED}}$ HS and $\text{shape}_{\text{EED}}$ RGB in Table 5.4. The second column represents the in-domain performance results on the CARLA dataset where we additionally examine shape by omitting texture creation ($\text{shape}_{\text{textureless}}$) with the help of the renderer. The rows in Table 5.4 are sorted according to the Cityscapes validation set performances from Table 5.2 to show the change in performance. We highlight the best in-domain performance in bold and the second-best performance is underlined. Firstly, the table suggests that the experts are aligned with the textureless base model. However, the results reveal that $\text{shape}_{\text{HED}}$ outperforms $\text{shape}_{\text{EED}}$ when not facing a domain gap. We speculate that this is due to the different approaches to extract the shape cue from the image. HED and textureless are both used for texture removal whereas anisotropic diffusion aims to reduce texture by blurring. Texture removal seems to lead to more significant features for the semantic segmentation task. In order to achieve a similar performance with anisotropic diffusion, color is required as an additional cue. Particularly, $\text{shape}_{\text{HED}}$ outperforms $\text{shape}_{\text{EED}}$ RGB on Cityscapes and is nearly on par on CARLA despite the fact that an additional cue is present for $\text{shape}_{\text{EED}}$ RGB. This suggests that less discriminative information is encoded in texture smoothed patterns

than in edge maps. Another aspect could be that anisotropic diffusion even though it aims to blur along edges and not across them, edges are still slightly smoothed such that a pixel accurate prediction is more difficult. Nevertheless, as it suffers less from the domain shift and allows for shape-color cue combinations, it is a valuable shape expert.

5.5.7 Influence on Class Level

In contrast to other approaches our method allows for a class-wise investigation on what can be learned from a specific image cue on pixel-level. We already noted that color is enough to learn to distinguish segments of *road*, *vegetation* and *sky*. The classes *building* and *car* are also predicted by an expert only trained on the color cue but are more confused with other classes (cf. Figure 5.7). Furthermore, we analyze whether specific classes are learned better by shape or texture cues. Therefore, we compare the performances of comparable texture and shape experts class-wise in Table 5.5 and Table 5.6 for Cityscapes and in Table 5.7 for the CARLA dataset. Having the domain gap in mind, we select training runs, with most similar mIoU performance in order to avoid a general performance bias towards one of the experts. To further account for the bias in terms of pixel count per class, we sorted the results in descending order starting with the class *road* which on average covers most of the pixels in an image in the base datasets. The results show that the class *vegetation* and *terrain* seem to have texture discriminative features, whereas *pole* and *person* are discriminative for both shape models (EED and HED). It is noticeable that the runs of the experts do not perform equally well on the same classes which is explainable by the fact that the optimizer might end in different local minima when starting with different seeds. This can be seen for example when comparing the classes *terrain* and *fence* for the second and third seed. The performance of these two classes are flipped so that the two runs result in the same mean IoU performance. With this in mind we interpret our result in the way that the noted classes are more likely to be exploitable by e.g., shape cues and some by texture cues. For this study, shape is said to outperform the texture cues if both shape cue experts show at least 7 pp performance gain. As already seen in the in-domain performance comparison (cf. Table 5.4) the shape experts do not necessarily perform equally well on the task. For example for the class *vegetation*, we see a distinct gap between the two shape experts. The expert $\text{shape}_{\text{EED}}$ yields improvable results on this class whereas $\text{shape}_{\text{HED}}$ is nearly on par with at least one of the texture experts. This might rise from the fact that the latter ones specialize on a few classes. In general, we see a more diverse class prediction capability for the $\text{shape}_{\text{EED}}$ expert. All but this model specialize on a few classes. Figure 5.9 provides insight into the resulting false positive predictions or confused classes. We conclude that shape cues based on texture smoothing are representative for all classes in semantic segmentation tasks but not strong enough as a single cue. In contrast, models trained on texture cues focus on a few specific, texture dominated, objects.



(a) $\text{shape}_{\text{EED}}$

(b) texture 1st seed

(c) $\text{shape}_{\text{HED}}$

(d) texture 2nd seed

Figure 5.9: Confusion matrices of shape and texture experts.

Table 5.5: Class-wise comparison of shape and texture importance on the Cityscapes semantic segmentation task. The table is sorted in descending order according to the average number of pixel per class in an image. Expert runs with similar mIoU results are compared. Green (purple) highlights classes which are distinctly better (> 10 pp) predicted by shape cue (texture cue) experts. Lighter color highlights results of more than 7 pp performance difference.

class	$\text{shape}_{\text{EED}}$	texture 1st seed	$\text{shape}_{\text{HED}}$	texture 2nd seed	texture 3rd seed
road	84.44	82.97	72.72	81.26	78.89
building	48.98	55.21	53.34	53.28	52.75
vegetation	19.34	61.81	47.05	48.76	63.03
car	32.62	50.02	39.58	43.93	36.85
sidewalk	32.14	18.46	14.68	13.3	8.08
sky	63.63	32.59	12.07	15.11	21.68
pole	22.78	0.11	16.96	0.06	0.13
person	18.32	2.63	13.53	1.50	0.99
terrain	0.20	8.91	3.92	13.41	5.78
fence	2.34	11.13	6.21	5.86	13.96
wall	3.22	2.27	0.81	2.94	1.73
traffic sign	17.60	1.81	3.28	2.01	4.15
bicycle	24.92	13.32	23.85	14.55	21.68
truck	8.58	7.11	0.0	3.28	2.7
bus	9.46	20.26	1.04	9.28	5.0
train	6.56	2.28	0.24	1.52	0.55
traffic light	16.16	0.24	1.9	0.0	1.13
rider	10.22	0.0	1.58	0.0	0.04
motorcycle	3.82	0.0	0.55	0.0	0.76
mIoU	22.39	19.53	16.49	16.32	16.84

If we consider texture or shape where the hue and saturation cue is present but the gray component is missing, we get slightly different results as reported in Table 5.6. It is striking that the class *bicycle* which was better classified by shape experts in the absence of the pure color cue is notably better predicted by the texture expert when only the pure color components for texture and shape are present. The opposite class switching can be observed for the class *bus* which has more discriminative texture features for grayscale texture experts but shape cues are more descriptive when only the pure color components are present.

For the CARLA dataset it is more difficult to find pairs which differ in only one cue but whose mIoU performances are in a similar range. Texture RGB and $\text{shape}_{\text{EED}}$ RGB are the closest pair only differing in shape and texture but not in an additional cue as both have access to the complete color cue. We compare their performances in Table 5.7. Even

Table 5.6: Class-wise comparison of shape and texture importance with hue and saturation cue instead of gray component on the Cityscapes semantic segmentation task. Changes are denoted in the last column. The table is sorted in descending order according to the average number of pixel per class in an image. Rows highlighted in vibrant colors differ at least by 13.5 pp in IoU. Green corresponds to shape and purple to texture domination respectively. Lighter colors show classes with performance gaps between 9 and 13.5 pp.

class	IoU of $\text{shape}_{\text{EED}}$ HS	IoU of texture HS	expert switch
road	85.05	81.03	
building	47.92	61.66	
vegetation	27.53	64.94	=
car	42.57	52.33	
sidewalk	37.22	02.35	
sky	64.55	47.73	
pole	15.34	0.00	=
person	20.71	25.51	
terrain	07.76	20.09	
fence	03.58	15.69	
wall	02.87	2.96	
traffic sign	26.39	8.05	
bicycle	5.27	26.06	↷
truck	10.69	4.88	
bus	18.97	6.75	↶
train	01.68	3.80	
traffic light	10.03	0.79	
rider	00.11	0.20	
motorcycle	00.18	0.33	
mIoU	22.55	22.73	

though the overall texture performance is slightly less, *road* and *guardrail* are remarkably better predicted by the texture expert. As in the Cityscapes experiments, the classes *pole*[30] and *person* have more shape discriminative features. Against our expectations, the classes *terrain* and *vegetation* are better classified by the shape expert. When looking additionally at the confusion matrix which is shown in Figure 5.10 we notice that the classes *road* and *sidewalk* can be easily distinguished by a texture expert but are confused by the shape expert. On the contrary, the texture expert has difficulties distinguishing *terrain* and *vegetation*. They seem to have a badly distinguishable texture but quite discriminative shape. The visual inspection of the experts reveals that the shape expert has an acceptable boundary segmentation but fails on the semantic for particular classes. Examples are visualized in Figure 5.11. The texture expert fails at predicting accurate

[30]Keep in mind the problematic of generating the texture of the class *pole* (cf. Section 5.3.2).

Table 5.7: Class-wise comparison of shape and texture importance with color cue on the CARLA semantic segmentation task with 15 classes. The table is sorted in descending order according to the average number of pixel per class in an image. Rows highlighted in vibrant colors differ by at least 19.5 pp in IoU. We rose this threshold as the overall mIoU is higher for CARLA due to the reduced number of classes. Rows in green show classes which are distinctly better predicted by shape cue experts whereas purple rows depict classes where the texture experts dominate the performance. Lighter colors show classes with performance gaps between 15.5 and 19.5 pp.

class	IoU of $\text{shape}_{\text{EED}}$ RGB	IoU of texture RGB
road	70.02	89.80
sky	67.21	83.17
vegetation	90.19	71.50
building	65.82	75.38
sidewalk	61.66	71.84
car	75.00	65.40
pole	55.10	03.19
wall	27.21	44.13
terrain	81.09	38.36
guard rail	22.55	45.35
bus	77.60	84.64
truck	57.78	67.92
person	52.41	31.96
traffic lights	49.46	23.67
traffic sign	47.24	50.28
mIoU	60.02	56.44

segment boundaries (e.g., compare the predictions of the person in the last row or the vegetation in the middle image in Figure 5.11) but shows very strong prediction results on major classes like *sky* and *road*. In particular, the results show that the experts learn complementary classes from different cues. Our finding supports the common findings of pre-trained neural network analyses where a balanced or at least co-existence of shape and texture affinity is suggested to achieve performant CNNs. Moreover, we postulate that this CARLA experiment shows that a neural network distinctly biased to only one of the cues cannot solve the segmentation task properly.

5.5.8 Influence on the per Pixel Level

The visual inspection of the texture RGB and $\text{shape}_{\text{EED}}$ RGB on CARLA suggests that complementary features are learned by the different cue experts. Particularly, it seems that certain classes are easier to predict or less confused by the texture expert and some

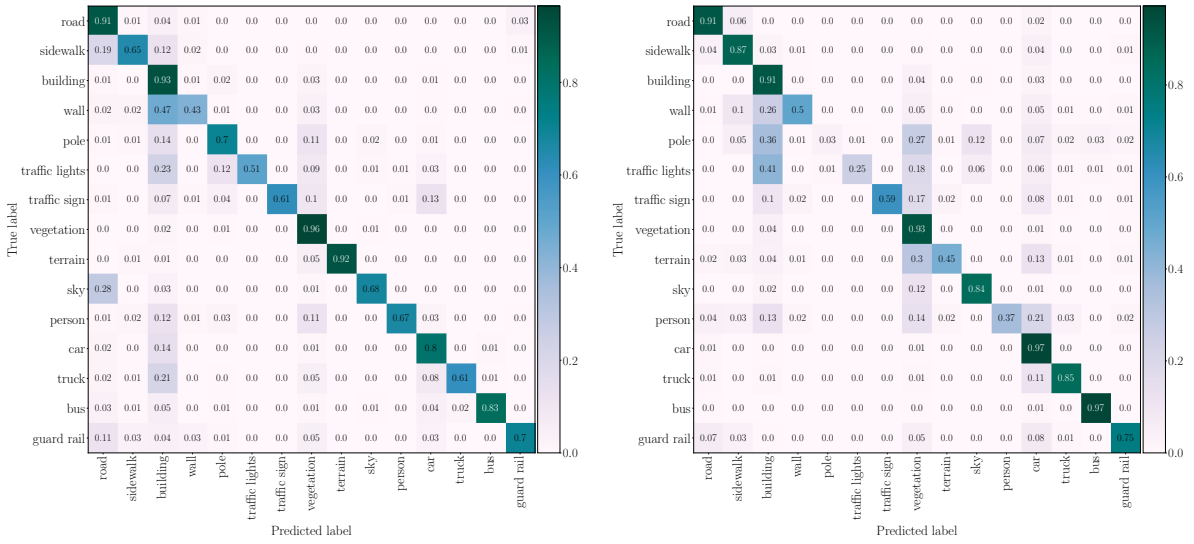
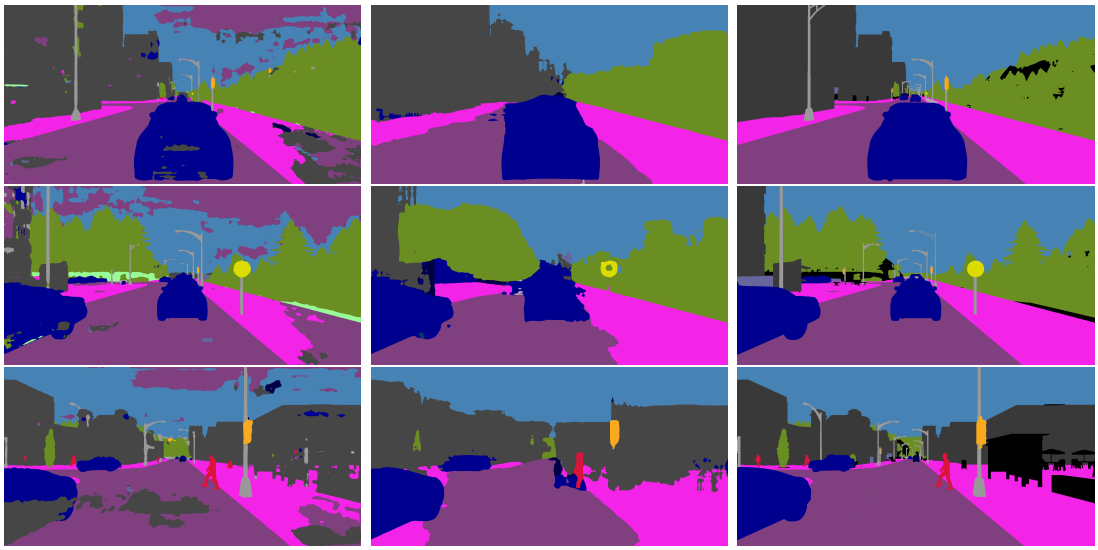


Figure 5.10: Confusion matrix of the two CARLA experts $\text{shape}_{\text{EED}}$ RGB (left) and texture RGB (right). *Road* and *sidewalk* is well distinguishable by texture but not by shape. On the contrary, less confusion is observed for *terrain* and *vegetation* by the shape expert compared to the texture expert.

others are better predicted by the shape model. To investigate the influence of the different cues on pixel level, we train a fusion model on these two experts as introduced in Section 5.4. Our results show that the fusion of two experts with complementary cues improves the overall scene understanding by a notable margin. The fusion model achieves a performance of 78.10% mIoU on the CARLA test set which is about 19 pp better than the test set performance of $\text{shape}_{\text{EED}}$ RGB and more than 22 pp better than relying only on the texture RGB cue (cf. Table 5.3). Thus, cooperation of complementary cues improves the performance. For the examples in Figure 5.11 we show the corresponding fusion prediction and the heatmap indicating the weighting of the softmax output in Figure 5.12. In Figure 5.13, we compare for a representative example side by side each expert prediction, the fusion heatmap, the fusion prediction as well as the input image and its corresponding ground truth. Our fusion results reveal that the cue importance may differ within a class. That means that classes are not inherently better predictable by a single cue expert. Instead, the cue importance for the prediction is dependent on the pixel position within the object as well as the distance of the object seems to have an influence on the cue importance. These phenomenons become obvious, e.g., at the borders of *vegetation* objects as it can be seen in Figure 5.13 when inspecting the heatmap for the shape expert. Dark green regions are shape dominated whereas light purple regions are dominated by the texture expert. The heatmap for the texture expert looks identically but with inverse color encoding. Furthermore, we see that the 'blobby' detection by the texture expert can be compensated by the shape expert. On the other hand in Figure 5.13 the prediction of the near car in the lower



shape$_{\text{EED}}$ RGB expert | texture RGB expert | CARLA ground truth

Figure 5.11: Comparison of the prediction of the two CARLA experts shape$_{\text{EED}}$ RGB (left) and texture RGB (mid). For reference reasons we display the ground truth in the third column (right). More clear predictions of the segment boundaries can be observed for the shape expert. The texture expert however can distinguish between the *sky* and *road* class without difficulty. It predicts more blob-like segments than the shape expert whose predictions are more granular.

left corner is improved by relying on the texture expert whereas the car further away was only correctly predicted by the shape expert what coherently was learned by the fusion model. On pixel-level we also get insight into the reliability of the predictions. Even though the street and the sky is predicted correctly by both models (cf. first row in Figure 5.13) the fusion (mid of second row) reveals that the texture expert seems to be more reliable on this class. This is less clear for the vegetation class, as there is a mixed weighting of the experts' predictions which is shown by the blue pixels in the heatmap.

5.5.9 Influence of the Architecture: Transformer Experiments

We additionally evaluate transformer models to study whether a different architecture shows significantly different cue affinity. Transformers are said to rely their predictions on shape cues due to the global view on the image [279]. In contrast to CNNs they do not have an inductive bias to local spatial structure and need to learn neighbor relations [279]. For our experiments we keep the neural network for the color components identical. We argue that this model simulates a transformer whose attention is limited to a single pixel and consequently the attention is suspended. To handle the restrictions of having a small dataset and the requirement of an unbiased (and therefore from scratch) training we use a Segformer model with B1 backbone (cf. Section 2.3.1.2

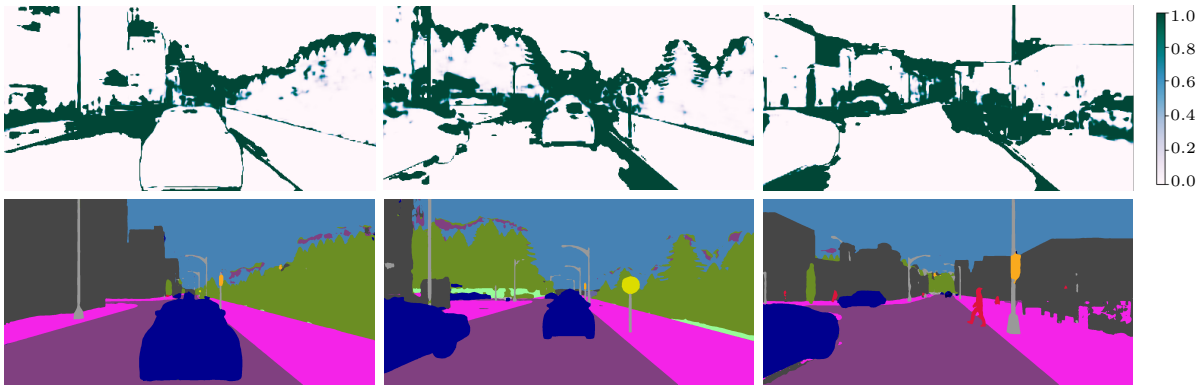
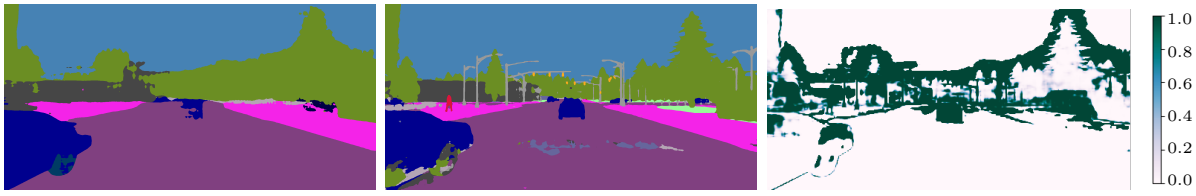


Figure 5.12: Fusion of the two experts texture RGB and shape$_{\text{EED}}$ RGB on CARLA (see Figure 5.11 for the single expert prediction). The heatmap visualizes the weighting of the convex combination of the experts' softmax values. In the two extremes, dark green colored pixels imply that the prediction of the fusion network is based solely on the shape expert softmax and light purple colored pixels are predicted based on the texture expert's softmax output.



prediction of texture expert | prediction of shape expert | heatmap for shape influence

input image | prediction of fusion model | ground truth

Figure 5.13: Comparison between the predictions of the fusion model and the two experts texture RGB and shape$_{\text{EED}}$ RGB on a representative CARLA test set image. The heatmap visualizes the weighting of the convex combination of the experts' softmax values. In the two extremes, dark green colored pixels imply that the prediction of the fusion model is based solely on the shape expert softmax and light purple colored pixels denotes the same for the texture expert.

Table 5.8: Cue influence on transformer models (Segformer with B1 backbone) solving a semantic segmentation task on Cityscapes.

	Color	T	S	mIoU Segformer CS val (↑)	$\text{gap}_{\mathcal{D}_{\text{all}}}$ (↓)	CNN mIoU
no information				0.33 ± 0.47	66.02	0.25 ± 0.35
$\text{shape}_{\text{HED}}$			✓	11.31 ± 1.95	55.05	13.38 ± 2.00
texture		✓		29.02 ± 0.31	37.33	17.85 ± 1.30
texture RGB	✓	✓		31.88 ± 0.38	34.47	20.10 ± 0.98
$\text{shape}_{\text{EED}}$			✓	39.01 ± 0.75	27.34	27.86 ± 3.17
$\text{shape}_{\text{EED}}$ RGB	✓		✓	50.48 ± 0.55	15.87	42.22 ± 2.13
Cityscapes gray		✓	✓	64.47 ± 0.21	1.88	64.21 ± 0.60
all cues	✓	✓	✓	66.35 ± 0.29	0.00	65.22 ± 0.47

and Section 5.5.2) which is of comparable size to the deeplabv3 with ResNet18 backbone in terms of parameters. Training this model on the dataset consisting of all cues (original Cityscapes) yields a comparable mIoU performance to the CNN analyzed before (cf. right-most column in Table 5.8). As a consequence, we can directly compare both architecture types. We observe in Table 5.8 that reducing the cues leads to the same order of cue influence for transformers. Nevertheless, the performance drop is not as pronounced as for the corresponding CNN models. This applies equally to the $\text{shape}_{\text{EED}}$ and texture versions. In comparison, the transformer is 8 to 10 pp better on the cue datasets than their CNN counterparts except for $\text{shape}_{\text{HED}}$ where they are on par. Transformers evaluated in context of classification are known to be more shape biased than convolutional neural networks [71, 279]. It seems that if the shape bias is also present in the context of semantic segmentation, it does not mean that transformers are worse at learning from texture. On the contrary, we observe that the performances of the cue experts based on the light-weighted Segformer-B1 model are ordered in the same way as for convolutional neural networks. This either means that the cues are equally well exploitable by the two architecture types or at least both models suffer similarly from the domain gaps which occur due to the cue reduction. The variance in the training runs of the transformer model is smaller, so that it is more difficult to find cue experts which are similar in the mean performance. In Table 5.9 we show the results on the class-wise comparison between $\text{shape}_{\text{EED}}$ and texture. On class level we observe minor dissimilarities. Classes like *terrain* and *bus* are again dominated by the texture expert. Additionally, the shape domination of the class *person* and *pole* seems independent of the architecture choice. However, we see a slight difference for the class *car* for which more discriminative information can be drawn from the shape cue when using transformer models. In addition, the transformer can extract equally well discriminative features from texture and shape for the class *vegetation* which was better predicted by texture in the context of CNNs. The latter might be explained by the better overall

Table 5.9: Class-wise comparison of shape and texture importance for transformers on the Cityscapes semantic segmentation task. The table is sorted in descending order according to the average number of pixel per class in an image. Classes which are distinctly better (> 10 pp) predicted by the shape cue expert are colored in green and those by the texture cue expert in purple respectively. Lighter color highlights results of more than 7 pp performance difference but less or equal to 10 *pp*. Changes compared to CNN results are noted in the last column.

class	IoU of $\text{shape}_{\text{EED}}$	IoU of texture	switch compared to CNN
road	92.32	85.42	
building	76.35	65.84	
vegetation	78.45	73.08	↩
car	83.10	63.81	↩
sidewalk	50.01	29.66	
sky	88.62	24.27	
pole	27.59	0.10	=
person	46.24	24.76	=
terrain	8.39	23.11	=
fence	17.77	14.57	
wall	9.61	18.11	
traffic sign	20.83	14.83	
bicycle	43.13	38.03	
truck	21.64	10.98	
bus	13.44	22.39	=
train	21.89	14.95	
traffic light	10.92	6.81	
rider	10.69	6.78	
motorcycle	9.96	9.55	
mIoU	38.38	28.79	

performance of the $\text{shape}_{\text{EED}}$ expert which catches up with the reduced diversity in class prediction of CNNs. As the CNN is *building* and *vegetation* loving, evidenced by the confusion matrix in Figure 5.9, it accepts performance limitations due to false positive predictions for correctly predicting a wide range of true positives. The similar performance we see when comparing transformers and convolutional neural networks might root in the limited amount of data the transformers are trained on. As shown in [71] transformer models seem to benefit from their architecture but also from the generally large amount of data they are trained on. We speculate that the data amount transformers are usually trained on are more likely the source of general performance boost rather than the ability to exploit a specific cue better compared to CNNs.

5.5.10 Likelihood Based Evaluation

In addition to mIoU based evaluations we also consider the likelihood ratios between nested models. This is known as a measure of goodness of fit in statistic. Following the definition in [81], let $\{p_{\boldsymbol{\theta}}\}_{\boldsymbol{\theta}\in\Theta}, \Theta \subseteq \mathbb{R}^{\tau}$ and $\{p_{\boldsymbol{\vartheta}}\}_{\boldsymbol{\vartheta}\in\Theta_{\tau_0}}, \Theta_{\tau_0} \subsetneq \Theta$ be two statistical models. If $\{p_{\boldsymbol{\vartheta}}\}_{\boldsymbol{\vartheta}\in\Theta_{\tau_0}}$ is defined on a subset of the parameter space of the other model it is said to be **nested**. Furthermore, let $\mathcal{D}$ be a dataset of size N where each sample was drawn identically and independently according to a model with parameter space Θ following the conditions of the maximum likelihood theory. Let $\hat{\boldsymbol{\vartheta}}$ denote the maximum likelihood estimator (cf. Equation (2.48)) of the model $\{p_{\boldsymbol{\vartheta}}\}_{\boldsymbol{\vartheta}\in\Theta_{\tau_0}}$ with restricted parameter space and analogous $\hat{\boldsymbol{\theta}}$ the maximum likelihood estimator for $\{p_{\boldsymbol{\theta}}\}_{\boldsymbol{\theta}\in\Theta}$. The deviance of the nested model $p_{\hat{\boldsymbol{\vartheta}}}$ is then defined by

$$\begin{aligned}\operatorname{dev}(p_{\hat{\boldsymbol{\vartheta}}}) &= 2\left(\texttt{NLL}(\mathcal{D}|\hat{\boldsymbol{\theta}}) - \texttt{NLL}(\mathcal{D}|\hat{\boldsymbol{\vartheta}})\right) \\ &= 2\left(\sum_{n=1}^{N}\log(p_{\hat{\boldsymbol{\theta}}}(\boldsymbol{y_n}|\boldsymbol{x_n})) - \sum_{n=1}^{N}\left(\log\left(p_{\hat{\boldsymbol{\vartheta}}}(\boldsymbol{y_n}|\boldsymbol{x_n})\right)\right)\right).\end{aligned} \tag{5.8}$$

In theory, the deviance is always greater or equal to zero, as the maximum of the likelihood of a parameter space is always greater or equal to the maximum of the likelihood taken with respect to a restricted model parameter space [81]. The deviance gives us insight into how well the model fits to the data. If the deviance is high the restricted parameter space is not enough to explain the data whereas when the deviance is zero or at least small, the reduced model already fits well.

We use this idea of model comparison to get insight into the importance of the nested image cues. In the following we explain how this idea transfers to our experiments and which limitations we are facing compared to the theory. For our analysis we consider models which are trained on datasets containing differently many cues. We can think of them as datasets with nested encoded information. In other words a dataset containing only the gray cue is nested in the texture dataset and texture data is nested in the dataset with all cues. Apart from that, we know that when more information is present we need a larger model to encode the information. Thus, we can also understand the nesting in terms of parameter spaces by the bypass of information encoding. Nevertheless, the property that the deviance is always non-negative is potentially violated in some cases in our analysis as the mentioned transfer step is not applied numerically, and therefore the nested models can be improper with respect to the parameter amount. The maximum likelihood can be easily calculated when training neural networks. Given two fully trained neural networks with learned parameters $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\vartheta}}$, we can calculate their negative log-likelihoods on a test set by computing their losses. Based on the statistical learning theory we can assume that the parameters are close to the maximum likelihood estimator when the model has been trained long enough (cf. Section 2.2.1).

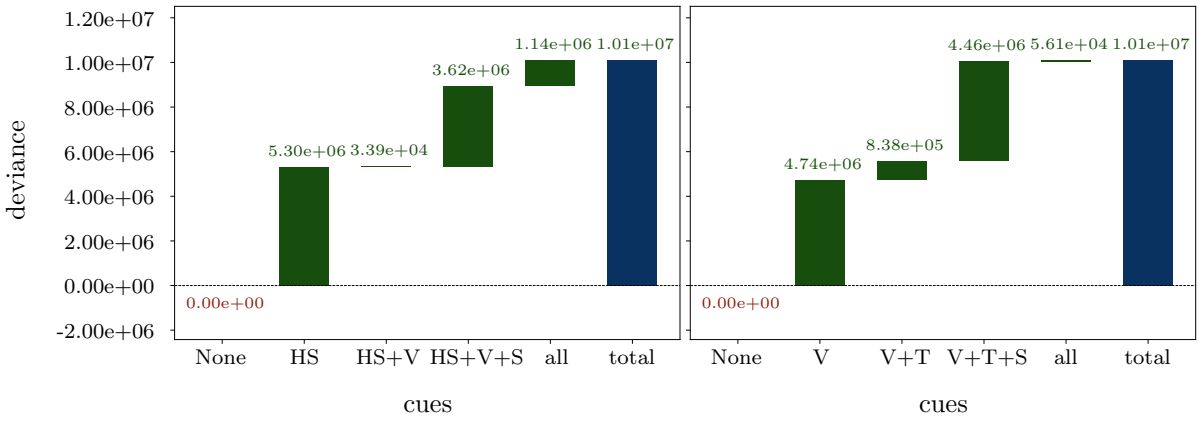


Figure 5.14: Waterfall chart of cue model deviances. Starting with one of the low-level color cue components we iteratively add the cue that best supports the model.

Therefore, we yield the deviance of two nested models by comparing their losses

$$\hat{\mathrm{dev}}(p_{\hat{\boldsymbol{\vartheta}}}) = 2\left(\mathcal{L}_{\mathcal{D}}(\hat{\boldsymbol{\theta}}) - \mathcal{L}_{\mathcal{D}'}(\hat{\boldsymbol{\vartheta}})\right). \tag{5.9}$$

The theoretical assumption that we consider the maximum likelihood estimator is not guaranteed in practice due to the heuristic approach of solving the optimization problem. That means, that during training the stochastic gradient descent may get stuck in a local minimum and potentially faces numerical instabilities (cf. Section 2.2.2). Nevertheless, having these inaccuracies in mind, this analysis gives us new insights into the influence of different cues. We start with the totally uninformed model (freshly initialized) and find the model or cue respectively which has the most significant deviance gain starting with one of the low-level color cue components V or HS. This model comparison is not free of ordering effects due to correlations between the cues. The results in the form of a waterfall chart are visualized in Figure 5.14. Additionally, we consider a class-wise averaged version of the deviance to take into account that classes appear differently often in terms of pixel count. Therefore, we calculate the loss with respect to each single class and average it over the number of classes. This leads to slightly different results for the cue influence selection when starting with the HS cue as shown in Figure 5.15. In line with [319], the results reveal that shape is a late prioritized cue. Nevertheless, shape contributes significantly as an additional cue to the texture regardless of whether the pure color component (HS), the gray component (V) or both (RGB) are present. The low deviance of the shape models traces back to a relatively high loss. In combination with our observations of the confusion matrix, we conclude that the shape expert has a certain understanding of a wide range of classes but is uncertain in its prediction. In contrast, the texture experts are certain in their limited classes leading to potentially false positives but a relatively lower loss value. We provide in Table 5.10 an overview of the different loss values of the cue experts along with their mIoU values. This table reveals that influence analyzed on pixel-level (likelihood analysis) is not on par with the

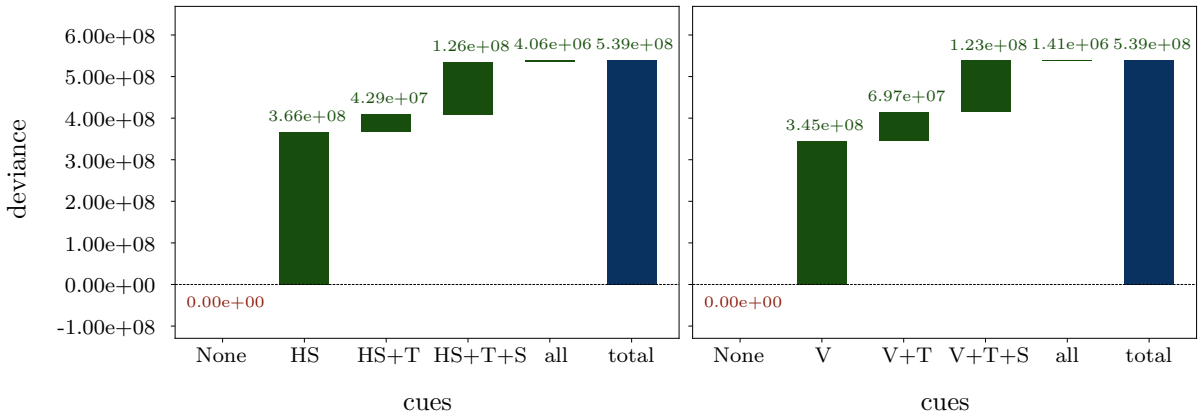


Figure 5.15: Waterfall chart of class-wise averaged cue model deviances. Starting with one of the low-level color cue components we iteratively add the cue that best supports the model.

mIoU observations. This might be explained by the deviations of theory and practice as explained earlier and the imbalance of pixel amount per class which is averaged out in the mIoU. However, this analysis allows for insights about, e.g., the uncertainty which comes with predictions.

5.6 Conclusion, Discussion and Outlook

We presented a method to disentangle cues, sometimes understood as concepts, which naturally exist in images of real-world scenes. Thereby, we introduced a new method to extract texture from a semantic segmentation dataset which allows us to generate new segmentation tasks based on this texture. Based on up to 15 cue experts ranging from color over texture to shape we investigated the question of what can be learned when only a reduced number of cues is present in an image. Among others, we analyzed the influence of the color cue on the solution of a semantic segmentation task which is rarely considered in the literature. We discovered that fundamental characteristics can be learned solely based on color information. Furthermore, our methods reveals that color is important for shape and texture experts to a different extent. Particularly for our shape expert based on EED, we saw a distinct improvement in terms of mIoU when the full color component is present whereas this was less or negligibly low for the texture expert. In general, by comparing a real world dataset and a simulated dataset, we found that datasets impact the cue influence on neural networks. Nevertheless, results from both datasets suggest that shape cues can be seen as the all-rounder for semantic segmentation tasks since they reveal information about most of the classes. In contrast, texture cues are useful for specific objects, depending on the other additional, present cues but do not lead to a satisfactory segmentation of the overall scene by themselves. However, shape as an individual cue is not enough to solve the problem even though contour maps are a promising representation to solve a semantic segmentation task

Table 5.10: Comparison of mIoU (a), loss in terms of negative log-likelihood (b) and class-wise averaged negative log-likelihood (c). All subtables are listed in ascending order according to the goodness of the metric value. Comparing the resulting order of the experts per table gives an insight on the impact of the class imbalance (cf. (b) and (c)) and the uncertainty which comes with the predictions of some experts (cf. (a) and (c)).

experts	mIoU (↑)
no info	0.251
V	06.39
HS	06.40
RGB	09.33
S_{HED}	11.31
T	13.38
SHS	17.85
TRGB	20.10
THS	20.63
S	27.86
SRGB	42.22
OHS	59.89
OV	64.21
all cues	65.22

(a) mIoU per expert

experts	NLL (↓)
no info	$5.39 \cdot 10^6$
S	$3.53 \cdot 10^6$
V	$3.02 \cdot 10^6$
SHS	$3.51 \cdot 10^6$
TRGB	$3.01 \cdot 10^6$
THS	$2.75 \cdot 10^6$
HS	$2.74 \cdot 10^6$
RGB	$2.72 \cdot 10^6$
S_{HED}	$2.65 \cdot 10^6$
T	$2.60 \cdot 10^6$
SRGB	$9.11 \cdot 10^5$
OHS	$4.00 \cdot 10^5$
OV	$3.69 \cdot 10^5$
all cues	$3.41 \cdot 10^5$

(b) negative log-likelihood

experts	avg. NLL (↓)
no info	$2.81 \cdot 10^8$
V	$1.08 \cdot 10^8$
S	$1.01 \cdot 10^8$
SHS	$9.95 \cdot 10^7$
HS	$9.73 \cdot 10^7$
RGB	$9.35 \cdot 10^7$
TRGB	$8.47 \cdot 10^7$
S_{HED}	$8.00 \cdot 10^7$
THS	$7.58 \cdot 10^7$
T	$7.29 \cdot 10^7$
SRGB	$2.73 \cdot 10^7$
OHS	$1.28 \cdot 10^7$
OV	$1.15 \cdot 10^7$
all cues	$1.08 \cdot 10^7$

(c) class-wise averaged NLL

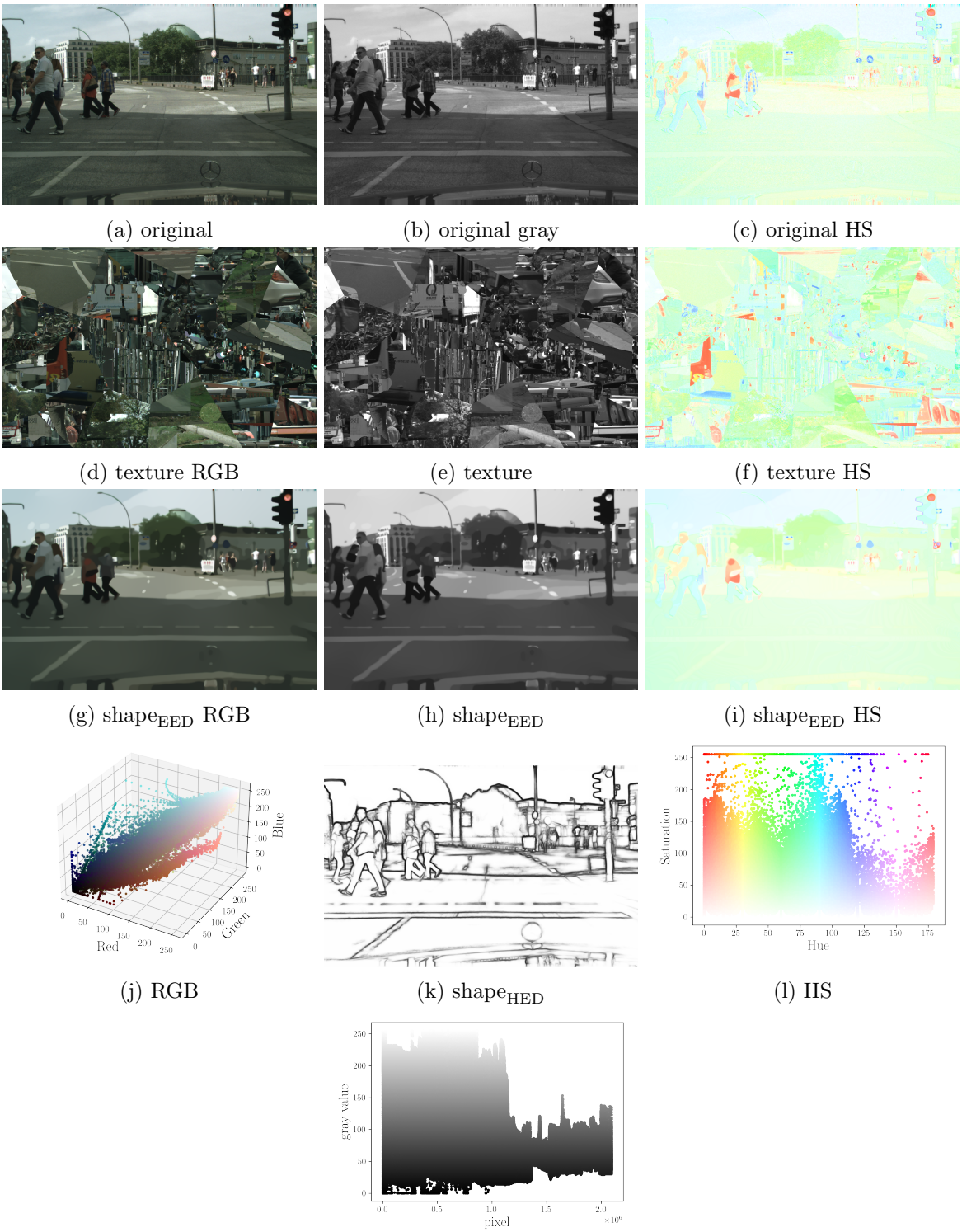
on a lower dimensional domain. Our CARLA experiments showed most clearly that shape and texture cues are beneficial to a different extent for some classes such that only the fusion of the two cues leads to a comprehensive segmentation of the scene. In addition, the importance of certain information changes depending on the position of the pixels in the object or of the object in the image. The results allow us to conclude that convolutional neural networks can draw more discriminative information from shape than from texture for real world semantic segmentation tasks. This does not contradict results from cue influence analyses for classification models. In drawing this conclusion, it should not be neglected that segmentation tasks demand to not only recognize objects but also their shapes and their location within an image. Unlike for classification tasks where contours are less relevant, the diversity of classes is better captured by a shape and color cue mixture than information encoded in texture. With respect to the architecture type, surprisingly, we see no difference in the cue affinity for transformer models compared to CNNs even though they are said to be more biased towards shape. This seems to be a useful property to solve segmentation tasks. This underlines the hypothesis that 'shape is not all you need'. However, 'cooperation is all you need' might be the more precise summary as it was shown that cues are largely complementary. Hereby, an early fusion is more successful than late fusion in real world scenarios such that the neural network can learn the correlations between the cues.

Nevertheless, late fusion was beneficial in the context of simulated data where the cue experts provided more complementary predictions.

The presented work gave insights into what can be learned by neural networks if certain cues are missing. In the future, we would like to investigate two aspects not yet covered with our study: Biases in already trained neural networks for semantic segmentation and more insight into what information is encoded in a dataset. The first aspect raises the question of whether neural networks trained on semantic segmentation tasks suffer from a certain bias. This can be analyzed by evaluating trained models on our different cue datasets. In addition, our cue extraction method allows for cue conflicts in semantic segmentation tasks. Our texture cue extraction method presented in Figure 5.1 can be used to fill arbitrary segments with a specific texture from mosaic images. By switching classes a conflict between shape and texture cues can be created. It can be analyzed if certain classes are more likely to be classified by shape or by texture.

The second aspect asks for the richness of information in a dataset. We would like to better understand the influence of different cues from an information theory point of view. With our experiments, we gave first insights into the model's behavior when certain cues are missing but, additionally, we would like to learn more about the different amount of information encoded in the different cue datasets and if this correlates to the ability of neural networks to extract this information.

Additionally, we see a connection between our approach and context-based explanations [205]. The cues 'color', 'texture' and 'shape' can equally be seen as high-level concepts perceived by humans. The expert fusion which was applied experimentally to a few cue constellations in the CARLA dataset can be understood as a proof-of-concept for a class-concept relation explanation [205] on a per pixel level. A more in-depth study of the concept explanation view seems to be promising for future research.



(a) original (b) original gray (c) original HS

(d) texture RGB (e) texture (f) texture HS

(g) $\text{shape}_{\text{EED}}$ RGB (h) $\text{shape}_{\text{EED}}$ (i) $\text{shape}_{\text{EED}}$ HS

(j) RGB (k) $\text{shape}_{\text{HED}}$ (l) HS

(m) gray

Figure 5.16: Cue decomposition of a Cityscapes image (original) into texture, shape, hue and saturation (HS) and gray components.

Chapter 6

Discussion and Outlook

In this dissertation, we investigated how deep convolutional neural networks for visual recognition tasks can benefit from data in an abstract domain, and studied from which data (cues) deep neural networks can draw the most information to accurately solve visual recognition tasks. In the following we summarize our main contributions and draw conclusions from our results. Thereafter, we close this thesis by outlining potential future research directions which complement the specific outlooks presented in Chapters 4 and 5.

Contributions and Conclusions Deep convolutional neural networks, which achieve outstanding performance on visual recognition tasks, need plenty of labeled data to learn and generalize well to data not seen during training. However, the labeling process is expensive and therefore labels are often scarce. Within the context of this thesis, we developed a modular semi-supervised image-to-image domain adaptation approach to take advantage from a simulation or an abstract domain where labels are easy to collect or generate (Chapter 4). Thereby, we train the downstream task neural network on the abstract domain and apply an unsupervised image-to-image translation method to translate real world data into the abstract domain. As this lacks task awareness, we guide the generator of the image-to-image translation from real to abstract with the help of a few labeled real world images. Once trained on the abstract domain the downstream task network is kept fixed and only serves for giving feedback how the data needs to be adapted to better suit for accurate predictions. By training on the abstract domain we benefit from a more controllable environment and less complex representation which facilitates training well performing neural networks. Besides, when the abstract domain is a simulation we can train and test for savety-critical scenarios. We evaluated our method on a complex downstream task (semantic segmentation of urban street scenes) and on a complex domain adaptation task (real to sketch for classification).

Both setups demonstrated that our method improves the network performance compared to from scratch training in the real world when only a small amount of labeled data is available. Furthermore, our method outperforms uninformed image-to-image translation by a significant margin. Our results revealed that the features in the image which improved the prediction of the neural network were not necessarily obvious to human perception. We also proposed an extended method for the classification setup, where we used active learning to sample those data points for labeling where the neural network is uncertain in its prediction. By guiding the generator with these specific data points, the performance was further improved. Although hybrid models with multiple complex components currently achieve the best performance, our method enables visual inspection of what a suitable input for the downstream task neural network looks like.

To understand more deeply the influence of the input data for a neural network, we developed a setup to analyze the influence of image cues in Chapter 5. In our research, we focused on the core image cues color, texture and shape. Therefore, we came up with a technique to extract the texture from semantic segmentation datasets and use it to create new segmentation tasks. This enables us to train semantic segmentation networks only focusing on texture. With the help of different methods which, e.g., create contour images or color reduction, we decompose the image in its core cues. By training on these datasets we achieved cue experts. To examine the influence of different cues, we measure the performance (drop) on the original image. We delved for indication of certain cue biases in our in-depth study. Our main findings are that shape can be seen as an all-rounder cue for semantic segmentation whereas texture is a specialist on a subset of classes. We conclude from our analysis that convolutional neural networks can draw more discriminative information with respect to all classes from shape than from texture for real world semantic segmentation tasks. Against intuition, our experiments reveal that fundamental characteristics can be learned solely through color information, although scene understanding is barely possible. Furthermore, we see limited differences between lean transformers and convolutional neural networks with respect to the cue influence. However, we see differences in the cue influence for different datasets. On data from a simulation we observe that shape and texture are more complementary than in our experiments on real world examples. Thereby, our investigation on pixel-level experimentally confirm the intuition that shape is more relevant at segment borders and texture information is helpful within a segment.

Future Research Directions In this thesis, we focused on the performance of deep neural networks on abstract and decomposed data. Therefore, our research can also be understood as a robustness analysis with respect to abstract data or data where individual cues were removed. The question that arises is how robust these neural networks are against other confounding factors such as image corruptions, adversarial attacks or out-of-distribution objects. Whether neural networks are more robust against these factors if they are biased towards one of our cue datasets can be analyzed in two ways. On the one hand, different cue experts can be evaluated on out-of-distribution datasets

and confronted with adversarial examples similar to the work in [269]. On the other hand, the cue datasets can be used for data augmentation during training. An analysis can show whether augmenting the original training data with all or certain cue datasets improves robustness for semantic segmentation tasks and how this relates to classification results for CNNs and transformers [276]. As the neural network can learn from all individual cues, it might be less susceptible if certain cues are missing in, e.g., domain shifted or adversarially disturbed images. A similar question can be addressed to abstract representations which we considered in Chapter 4. Due to our modular approach of transforming images from the real to the abstract domain, it is of particular interest to investigate if the translation by the guided image-to-image translation module mitigates a network's vulnerability to image corruptions and adversarial attacks. However, we hypothesize that robustness of out-of-distribution objects might not improve particularly if those objects are also outside the distribution of the generative model. A current highly emerging research field studies text-to-image diffusion models [204]. These models show high quality results for image generation based on a textual description [215, 222]. Furthermore, methods for image editing based on textual instructions have been proposed [28]. First publications address I2I translations based on text-to-image diffusion models without manual prompts [200]. It is worth investigating whether our methods can benefit from generating high-quality images for image translation in our SSDA method and generating specific shape or texture transformations for our cue influence study.

In addition, with respect to our method presented in Chapter 5, we aim to contribute beyond our research question by enabling other researchers to evaluate their segmentation dataset on its cue composition. This is of broad interest as it can uncover possible shortcut learnings or demonstrate potential task reductions if reduced cues are sufficient to solve the entire task. This offers the opportunity to train potentially more robust networks or to solve lighter tasks, making them feasible on leaner hardware.

List of Figures

List of Tables

List of Notations

We denote matrices with capital letters. Bold notation is used for vector-valued variables. We also use this notation for images since they can be represented by a vector when flattening the image. Estimators are denoted with $\hat{}$ e.g., $\hat{\mu}$. Optimal functions or models are denoted with a superscript *. The following abbreviations and notations listed alphabetically are used across all chapters:

	Abbreviations
ADA	Active domain adaptation
AI	Artificial intelligence
AL	Active learning
ASPP	Atrous spatial pyramid pooling
CNN	Convolutional neural networks
CRF	Conditional random field
CS	Cityscapes
DA	Domain adaptation
EED	Edge enhancing diffusion with smoothed orientation
FC	Fully connected
FCN	Fully convolutional network
FID	Fréchet inception distance
GAN	Generative adversarial network
GELU	Gaussian error linear unit
GPU	Graphics processing unit
GT	Ground truth
HED	Holistically-nested edge detection
HS	Hue saturation cue
HSV	Hue saturation value
i.i.d.	Identically and independently distributed
IoU	Intersection over union
I2I	Image-to-Image
LPIPS	Learned perceptual image patch similarity
LReLU	Leaky ReLU
LSGAN	Least square GAN

mIoU	Mean intersection over union
MLP	Multilayer perceptron
NN	Neural network
PDE	Partial differential equation
pp	Percentage points
PReLU	Parametric ReLU
ReLU	Rectified linear unit
RePU	Rectified power unit
RGB	Red green blue
S	Shape cue
SGD	Stochastic gradient descent
SDA	Superviced domain adaptation
SSDA	Semi-supervised domain adaptation
SSIM	Structural Similarity Index
T	Texture cue
UDA	Unsupervised domain adaptation
V	(Gray) value cue
ViT	Vision transformer

	Notation
$a : \mathbb{R} \to \mathbb{R}$	Activation function
b, B	Bias
$\hat{c}(\boldsymbol{x}\|\boldsymbol{\theta})$, $\hat{c}$	Predicted class given by $\operatorname{argmax}_{c\in\{1,\dots,C\}}(f_{\boldsymbol{\theta}}(\boldsymbol{x})_c)$
$d = m \times h \times w$	Input dimension
$f_{\boldsymbol{\theta}}$	Neural network with learnable parameters $\boldsymbol{\theta}$
h	Spatial height (number of rows) of an image
$\boldsymbol{h_l}$	Hidden state of layer l
k^{dil}	Kernel size of dilated kernel
k, (k_y, k_x)	Spatial kernel dimension; for quadratic kernels $k = k_x = k_y$
m	Channel dimension
m'	Output dimension
$p_{G_{\boldsymbol{\theta}}}$	Pushforward density function under $G_{\boldsymbol{\theta}}$
$p_{\boldsymbol{\theta}}(\cdot\|\cdot)$	Parametric conditional density function defined by a neural network
$p_{X,Y}$	density function of $P_{X,Y}$
$p_{Y\|X}$	density function of $P_{Y\|X}$
p_{data}	density function of μ_{data}
r	Dilatation rate
s, (s_x, s_y)	Stride in x and y dimension; for equal stride $s = s_x = s_y$
w	width (number of columns) of an image
$\boldsymbol{x} \in \mathcal{X}$	Input, e.g., an image
$y \in \mathcal{Y}$	Label

$\boldsymbol{z_l}$	Pre-activation vector of layer l
C	Number of classes
$D_{\boldsymbol{\vartheta}} : \mathcal{X} \to (0,1)$	Discriminator with learnable parameters $\boldsymbol{\vartheta}$
$D_{\mathcal{S}}$	Discriminator of domain $\mathcal{S}$
D_G^*	Optimal discriminator for s given generator G
$G_{\mathcal{S}\to\mathcal{T}}$	Generator from domain $\mathcal{S}$ to domain $\mathcal{T}$
$G_{\boldsymbol{\theta}} : \mathcal{Z} \to \mathcal{X}$	Generator with learnable parameters $\boldsymbol{\theta}$
$G_{\boldsymbol{\theta}*}\nu$	Pushforward measure of ν under $G_{\boldsymbol{\theta}}$
H^l	True layer mapping in a parametric layered function
K	Kernel
$K * \Phi$	Convolution of kernel K and a feature map Φ
L	Number of layers
$L(f_{\boldsymbol{\theta}}(\boldsymbol{x}), y)$	Loss of the neural network $f_{\boldsymbol{\theta}}$ in a single sample $(\boldsymbol{x}, y)$
$M \in \mathbb{R}^{p_x \times p_y}$	Subgrid of an image
N	Number of data samples in a dataset
$P_{X,Y}^{\otimes N}$	Product measure of i.i.d. random variables
$P_{X,Y}$	Joint probability measure on $\mathcal{X} \times \mathcal{Y}$
$P_{Y\|X}$	Conditional probability measure of Y given X
W, $\boldsymbol{w}$	Weight matrix, weight vector
$W^{\beta,\infty}([0,1]^d)$	Sobolev space
$X : (\Omega, \mathcal{A}, \mathbb{P}) \to \mathcal{X}$	$\mathcal{X}$-valued random variable/vector
$Y : (\Omega', \mathcal{A}', \mathbb{P}') \to \mathcal{Y}$	$\mathcal{Y}$-valued random variable/vector
$\mathcal{A}$	σ-algebra
$\mathcal{B} \subseteq \mathcal{D}$	(Mini-)batch of $N_{\mathcal{B}}$ samples, i.e., $\mathcal{B} = \{(\boldsymbol{x_n}, y_n)\}_{n=1}^{N_{\mathcal{B}}}$
$\mathcal{C}(\mathcal{K}, \mathbb{R}^C)$	Function space of continuous functions from $\mathcal{K}$ to $\mathbb{R}^C$
$\mathcal{D}$	Dataset
$\mathcal{D}_{\mathcal{S}}$	Dataset of domain $\mathcal{S}$
$\mathcal{F}$	Function space, e.g., $\mathcal{F} = \mathcal{C}([0,1]^d)$
$\mathcal{H}$	Hypothesis space
$\mathcal{I}$	Pixel index set $\{(i,j) \vert i = \{1,\dots,h\}, j = \{1,\dots,w\}\}$
$\mathcal{K}$	Compact set
$\mathcal{L}$	Loss
$\hat{\mathcal{L}}_N$, $\mathcal{L}_{\mathcal{D}}$	Empirical loss function based on the training dataset $\mathcal{D}$ with N samples
$\mathcal{M}_1(\mathbb{R}^d)$, $\mathcal{M}_1^+(\mathbb{R}^d)$	Space of (positive) probability measures over $\mathbb{R}^d$ with Borel-σ-algebra
$\mathcal{R} = (\mathcal{X}_R, \mathcal{Y}, \mu_R)$	Domain of real images
$\mathcal{S} = (\mathcal{X}_S, \mu_S)$	Domain with image space $\mathcal{X}_S$ and data generating distribution μ_S or as a triple
$\mathcal{S} = (\mathcal{X}_S, \mathcal{Y}_S, \mu_S)$	Domain with image space $\mathcal{X}_S$, label space $\mathcal{Y}_S$ and joint distribution μ_S on $\mathcal{X}_S \times \mathcal{Y}_S$
$\mathcal{T}_{\mathcal{R}}$	Labeled subset of the real domain

$\mathcal{T}_{\boldsymbol{\theta}}^{m_h,d_h,m_1}$	Transformer block
$\mathcal{X} \subseteq \mathbb{R}^d$	Input space, mostly $\mathcal{X} \subseteq [0,1]^{3\times h\times w}$
$\mathcal{Y}$	Label space
$\mathcal{Z}$	Latent space
δ_{mj}	Kronecker delta where $\delta_{mj} = 1$ if $m = j$ else $\delta_{mj} = 0$
ε	Error, e.g., model error ε_{model}
η	Learning rate
$\boldsymbol{\theta}$, $\boldsymbol{\vartheta}$	Learnable parameters
λ	Lebesgues measure
$\hat{\mu}$	Estimated data generating distribution of X
μ	Probability measure
μ_{data}	True data generating distribution of X
ν	Probability measure
ρ	Padding
$\phi : \mathbb{R}^{\lvert Y\rvert} \to \Xi \subseteq \mathbb{R}^{\lvert Y\rvert}$	(Statistical) head
$\varphi_{\boldsymbol{\theta}} : \mathcal{X} \to \mathbb{R}^{\lvert Y\rvert}$	Neural network with linear output layer
$\Theta \subseteq \mathbb{R}^{\tau}$, $\Theta_q \subseteq \mathbb{R}^{\tau_q}$	Parameter space
Φ	Feature map (or image if it is the network input)
$(\Omega, \mathcal{A}, \mathbb{P})$	Probability space with probability measure $\mathbb{P}$
$\mathfrak{d}(\cdot\vert\cdot), \mathfrak{D}(\cdot\vert\cdot)$	Divergence measures
$\mathfrak{p}$	Probability
$\mathfrak{B}$	Borel-σ-algebra
$\mathbb{1}$	matrix where all entries equal to 1
$\operatorname{diag}(x)$	Diagonal matrix A with $A_{i,i} = x$ and $A_{i,j} = 0$ for $i \neq j$. If $\boldsymbol{x}$ is a vector, $A_{i,i} = x_i$ and $A_{i,j} = 0$ for $i \neq j$
$\text{Attn}_{W_Q,W_K,W_V}(Z)$	Self-attention
$\text{MHSA}_{\boldsymbol{\theta}}(Z)$	Multi-head self-attention
$\texttt{NLL}(\mathcal{D}\vert\boldsymbol{\theta})$	Negative log-likelihood
$\partial_{\boldsymbol{x}}$	Vector, matrix or tensor whose entries are defined by the partial derivatives with respect to $\boldsymbol{x}$. Corresponds to the gradient for vectors and to the Jacobian matrix for matrices.

Bibliography

[1] *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR. 1983 Conf., Washington, D.C. Proceedings: Computer Vision and Pattern Recognition*, 1983.

[2] *How does DeepL work?* https://www.deepl.com/en/blog/how-does-deepl-work, Nov. 2021. Accessed: 17.11.2023.

[3] *Pause Giant AI Experiments: An Open Letter*, Mar. 2023.

[4] R. A. Adams and J. J. F. Fournier, *Sobolev Spaces*, Elsevier, June 2003.

[5] A. Almahairi, S. Rajeshwar, A. Sordoni, P. Bachman, and A. Courville, *Augmented CycleGAN: Learning Many-to-Many Mappings from Unpaired Data*, in Proceedings of the 35th International Conference on Machine Learning, PMLR, July 2018, pp. 195–204.

[6] M. Altalak, M. Ammad uddin, A. Alajmi, and A. Rizg, *Smart Agriculture Applications Using Deep Learning Technologies: A Survey*, Applied Sciences, 12 (2022), p. 5919.

[7] L. Alzubaidi, M. A. Fadhel, O. Al-Shamma, J. Zhang, J. Santamaría, Y. Duan, and S. R. Oleiwi, *Towards a Better Understanding of Transfer Learning for Medical Imaging: A Case Study*, Applied Sciences, 10 (2020), p. 4523.

[8] S. Amari, *A Theory of Adaptive Pattern Classifiers*, IEEE Transactions on Electronic Computers, EC-16 (1967), pp. 299–307.

[9] C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan, *An Introduction to MCMC for Machine Learning*, Machine Learning, 50 (2003), pp. 5–43.

[10] M. Arjovsky and L. Bottou, *Towards Principled Methods for Training Generative Adversarial Networks*, arXiv:1701.04862, (2017).

[11] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein Generative Adversarial Networks*, in Proceedings of the 34th International Conference on Machine Learning, PMLR, July 2017, pp. 214–223.

[12] H. Asatryan, H. Gottschalk, M. Lippert, and M. Rottmann, *A convenient infinite dimensional framework for generative adversarial learning*, Electronic Journal of Statistics, 17 (2023), pp. 391–428.

[13] J. T. Ash, C. Zhang, A. Krishnamurthy, J. Langford, and A. Agarwal, *Deep Batch Active Learning by Diverse, Uncertain Gradient Lower Bounds*, arXiv:1906.03671, (2020).

[14] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer Normalization*, arXiv:1607.06450, (2016).

[15] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, in 3rd International Conference on Learning Representations ICLR, Y. Bengio and Y. LeCun, eds., San Diego, CA, USA, May 2015.

[16] N. Baker, H. Lu, G. Erlikhman, and P. J. Kellman, *Deep convolutional networks do not classify based on global object shape*, PLOS Computational Biology, 14 (2018), p. e1006613.

[17] D. H. Ballard, G. E. Hinton, and T. J. Sejnowski, *Parallel visual computation*, Nature, 306 (1983), pp. 21–26.

[18] S. Barratt and R. Sharma, *A Note on the Inception Score*, in ICML 2018 Workshop on Theoretical Foundations and Applications of Deep Generative Models, June 2018.

[19] Z. Batmaz, A. Yurekli, A. Bilge, and C. Kaleli, *A review on deep learning for recommender systems: Challenges and remedies*, Artificial Intelligence Review, 52 (2019), pp. 1–37.

[20] S. Beery, G. Van Horn, and P. Perona, *Recognition in Terra Incognita*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 456–473.

[21] W. H. Beluch, T. Genewein, A. Nurnberger, and J. M. Kohler, *The Power of Ensembles for Active Learning in Image Classification*, in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, June 2018, pp. 9368–9377.

[22] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, *A theory of learning from different domains*, Machine Learning, 79 (2010), pp. 151–175.

[23] A. Birhane and V. U. Prabhu, *Large image datasets: A pyrrhic win for computer vision?*, in 2021 IEEE Winter Conference on Applications of Computer Vision (WACV), Jan. 2021, pp. 1536–1546.

[24] C. M. Bishop, *Pattern Recognition and Machine Learning*, vol. 2 of Information Science and Statistics, Springer, New York, NY, 2006.

[25] A. Borji, *Pros and cons of GAN evaluation measures*, Computer Vision and Image Understanding, 179 (2019), pp. 41–65.

[26] S. Brehm, S. Scherer, and R. Lienhart, *Semantically consistent image-to-image translation for unsupervised domain adaptation*, ICAART, 2 (2022), pp. 131–141.

[27] W. Brendel and M. Bethge, *Approximating CNNs with Bag-of-local-Features models works surprisingly well on ImageNet*, in International Conference on Learning Representations, Sept. 2018.

[28] T. Brooks, A. Holynski, and A. A. Efros, *InstructPix2Pix: Learning To Follow Image Editing Instructions*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 18392–18402.

[29] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, and A. Askell, *Language models are few-shot learners*, Advances in neural information processing systems, 33 (2020), pp. 1877–1901.

[30] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, *Albumentations: Fast and Flexible Image Augmentations*, Information, 11 (2020), p. 125.

[31] L. Cai, X. Xu, J. H. Liew, and C. S. Foo, *Revisiting Superpixels for Active Learning in Semantic Segmentation With Realistic Annotation Costs*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 10988–10997.

[32] J. Canny, *A Computational Approach to Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8 (1986), pp. 679–698.

[33] A. L. Chandra and V. N. Balasubramanian, *Deep Active Learning Toolkit for Image Classification in PyTorch*, https://github.com/acl21/deep-active-learning-pytorch, (2021).

[34] P. Charbonnier, L. Blanc-Feraud, G. Aubert, and M. Barlaud, *Deterministic edge-preserving regularization in computed imaging*, IEEE Transactions on Image Processing, 6 (1997), pp. 298–311.

[35] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. Yuille, *Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs*, in ICLR, Dec. 2015.

[36] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 40 (2018), pp. 834–848.

[37] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, *Rethinking Atrous Convolution for Semantic Image Segmentation*, arXiv:1706.05587, (2017).

[38] L.-C. Chen, Y. Zhu, G. Papandreou, H. Hui, M. D. Collins, and T. Liu, *DeepLab: Deep Labelling for Semantic Image Segmentation*. https://github.com/tensorflow/models/tree/master/research/deeplab. Accessed: 20.10.2023.

[39] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*, in Computer Vision – ECCV 2018, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, eds., Lecture Notes in Computer Science, Cham, 2018, Springer International Publishing, pp. 801–818.

[40] S. Chen, X. Jia, J. He, Y. Shi, and J. Liu, *Semi-supervised domain adaptation based on dual-level domain mixing for semantic segmentation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 11018–11027.

[41] Y. Chen, M. Mancini, X. Zhu, and Z. Akata, *Semi-supervised and unsupervised deep visual learning: A survey*, IEEE transactions on pattern analysis and machine intelligence, (2022).

[42] Y. Chen, X. Ouyang, K. Zhu, and G. Agam, *Semi-supervised Domain Adaptation for Semantic Segmentation*, arXiv:2110.10639, (2021).

[43] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar, *Masked-Attention Mask Transformer for Universal Image Segmentation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 1290–1299.

[44] H. Chung and K. H. Park, *Shape Prior is Not All You Need: Discovering Balance Between Texture and Shape Bias in CNN*, in Computer Vision – ACCV 2022, L. Wang, J. Gall, T.-J. Chin, I. Sato, and R. Chellappa, eds., Lecture Notes in Computer Science, Cham, 2023, Springer Nature Switzerland, pp. 491–506.

[45] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, *Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition*, Neural Computation, 22 (2010), pp. 3207–3220.

[46] P. Colling, L. Roese-Koerner, H. Gottschalk, and M. Rottmann, *Metabox+: A new region based active learning method for semantic segmentation using priority maps*, arXiv:2010.01884, (2020).

[47] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, *The Cityscapes Dataset for Semantic Urban Scene Understanding*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 3213–3223.

[48] G. Csurka, *Domain Adaptation for Visual Applications: A Comprehensive Survey*, arXiv:1702.05374 [cs], (2017).

[49] G. Csurka, R. Volpi, and B. Chidlovskii, *Unsupervised Domain Adaptation for Semantic Image Segmentation: A Comprehensive Survey*, arXiv:2112.03241, (2021).

[50] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of Control, Signals and Systems, 2 (1989), pp. 303–314.

[51] I. Dagan and S. P. Engelson, *Committee-Based Sampling For Training Probabilistic Classifiers*, in Machine Learning Proceedings 1995, A. Prieditis and S. Russell, eds., Morgan Kaufmann, San Francisco (CA), Jan. 1995, pp. 150–157.

[52] D. Dai, Y. Li, Y. Wang, H. Bao, and G. Wang, *Rethinking the image feature biases exhibited by deep convolutional neural network models in image recognition*, CAAI Transactions on Intelligence Technology, 7 (2022), pp. 721–731.

[53] J. Dai, K. He, and J. Sun, *Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation*, in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1635–1643.

[54] J. Dastin, *Insight - Amazon scraps secret AI recruiting tool that showed bias against women*, Reuters, (2018).

[55] DeepLearningAI, *A Chat with Andrew on MLOps: From Model-centric to Data-centric AI*, Mar. 2021.

[56] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *ImageNet: A Large-Scale Hierarchical Image Database*, in IEEE Conference on Computer Vision and Pattern Recognition, June 2009, pp. 248–255.

[57] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), J. Burstein, C. Doran, and T. Solorio, eds., Association for Computational Linguistics, 2019, pp. 4171–4186.

[58] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, in International Conference on Learning Representations, Oct. 2020.

[59] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, *CARLA: An Open Urban Driving Simulator*, in Conference on Robot Learning, PMLR, Oct. 2017, pp. 1–16.

[60] T. Duboudin, E. Dellandréa, C. Abgrall, G. Hénaff, and L. Chen, *Learning less generalizable patterns for better test-time adaptation*, in Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2023) - Volume 5: VISAPP, SciTePress / INSTICC, 2023, pp. 349–358.

[61] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization.*, Journal of machine learning research, 12 (2011).

[62] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, arXiv:1603.07285, (2018).

[63] A. Dundar, M.-Y. Liu, T.-C. Wang, J. Zedlewski, and J. Kautz, *Domain stylization: A strong, simple baseline for synthetic to real image domain adaptation*, arXiv:1807.09384, (2018).

[64] S. Fabbrizzi, S. Papadopoulos, E. Ntoutsi, and I. Kompatsiaris, *A survey on bias in visual datasets*, Computer Vision and Image Understanding, 223 (2022), p. 103552.

[65] R. Feng, D. Zhao, and Z.-J. Zha, *Understanding Noise Injection in GANs*, in Proceedings of the 38th International Conference on Machine Learning, PMLR, July 2021, pp. 3284–3293.

[66] B. Fu, Z. Cao, J. Wang, and M. Long, *Transferable Query Selection for Active Domain Adaptation*, 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), (2021), pp. 7268–7277.

[67] Y. Gal and Z. Ghahramani, *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*, in Proceedings of The 33rd International Conference on Machine Learning, PMLR, June 2016, pp. 1050–1059.

[68] Y. Gal, R. Islam, and Z. Ghahramani, *Deep bayesian active learning with image data*, in International Conference on Machine Learning, PMLR, 2017, pp. 1183–1192.

[69] C. Geiger and C. Kanzow, *Gradientenverfahren*, in Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben, C. Geiger and C. Kanzow, eds., Springer-Lehrbuch, Springer, Berlin, Heidelberg, 1999, pp. 67–81.

[70] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann, *Shortcut learning in deep neural networks*, Nature Machine Intelligence, 2 (2020), pp. 665–673.

[71] R. Geirhos, K. Narayanappa, B. Mitzkus, T. Thieringer, M. Bethge, F. A. Wichmann, and W. Brendel, *Partial success in closing the gap between human and machine vision*, in Advances in Neural Information Processing Systems, vol. 34, Curran Associates, Inc., 2021, pp. 23885–23899.

[72] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, *ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness*, in International Conference on Learning Representations, Sept. 2018.

[73] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, in 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Los Alamitos, CA, USA, June 2014, IEEE Computer Society, pp. 580–587.

[74] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Y. W. Teh and M. Titterington, eds., vol. 9 of Proceedings of Machine Learning Research, Chia Laguna Resort, Sardinia, Italy, Mar. 2010, PMLR, pp. 249–256.

[75] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, G. Gordon, D. Dunson, and M. Dudík, eds., vol. 15 of Proceedings of Machine Learning Research, Fort Lauderdale, FL, USA, Apr. 2011, PMLR, pp. 315–323.

[76] A. Gokaslan, V. Ramanujan, D. Ritchie, K. I. Kim, and J. Tompkin, *Improving Shape Deformation in Unsupervised Image-to-Image Translation*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 649–665.

[77] I. Goodfellow, *NIPS 2016 Tutorial: Generative Adversarial Networks*, arXiv:1701.00160, (2017).

[78] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[79] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative Adversarial Nets*, in Advances in Neural Information Processing Systems, vol. 27, Cambridge, MA, USA, 2014, Curran Associates, Inc.

[80] ——, *Generative adversarial networks*, Communications of the ACM, 63 (2020), pp. 139–144.

[81] H. Gottschalk, *Hochdimensionale Wahrscheinlichkeitstheorie und maschinelles Lernen.* 2020.

[82] ——, *Mathematical Foundations of Machine Learning.* 2023.

[83] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, Second Edition*, Society for Industrial and Applied Mathematics, Jan. 2008.

[84] T. Grigoryev, A. Voynov, and A. Babenko, *When, Why, and Which Pretrained GANs Are Useful?*, arXiv:2202.08937, (2022).

[85] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, *Conformer: Convolution-augmented Transformer for Speech Recognition*, arXiv:2005.08100, (2020).

[86] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, *Improved Training of Wasserstein GANs*, in Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., Mar. 2017.

[87] S. Harary, E. Schwartz, A. Arbelle, P. Staar, S. Abu-Hussein, E. Amrani, R. Herzig, A. Alfassy, R. Giryes, and H. Kuehne, *Unsupervised Domain Generalization by Learning a Bridge Across Domains*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, June 2022, pp. 5280–5290.

[88] J. He and J. Xu, *MgNet: A unified framework of multigrid and convolutional neural network*, Science China Mathematics, 62 (2019), pp. 1331–1354.

[89] K. He, R. Girshick, and P. Dollar, *Rethinking ImageNet Pre-Training*, in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Oct. 2019, pp. 4917–4926.

[90] K. He, X. Zhang, S. Ren, and J. Sun, *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*, in Computer Vision – ECCV 2014, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds., Lecture Notes in Computer Science, Cham, 2014, Springer International Publishing, pp. 346–361.

[91] ——, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034.

[92] ——, *Deep Residual Learning for Image Recognition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[93] E. Heinert, M. Rottmann, K. Maag, and K. Kahl, *Reducing texture bias of deep neural networks via edge enhancing diffusion*, arXiv:2402.09530, (2024).

[94] D. Hendrycks and K. Gimpel, *Gaussian Error Linear Units (GELUs)*, arXiv:1606.08415, (2023).

[95] K. Hermann, T. Chen, and S. Kornblith, *The Origins and Prevalence of Texture Bias in Convolutional Neural Networks*, in Advances in Neural Information Processing Systems, vol. 33, Curran Associates, Inc., 2020, pp. 19000–19015.

[96] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, *Image analogies*, in Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, 2001, pp. 327–340.

[97] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*, in Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017.

[98] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley, *Boltzmann Machines: Constraint Satisfaction Networks That Learn*, Technical Report TR-CMU-CS-84-119, Carnegie-Mellon University, Department of Computer Science, Pittsburgh, PA, 1984.

[99] S. Hochreiter, *Untersuchungen Zu Dynamischen Neuronalen Netzen*, diplom thesis, Institut for Computer Science, Technical University of Munich, Apr. 1991.

[100] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell, *CyCADA: Cycle-Consistent Adversarial Domain Adaptation*, in International Conference on Machine Learning, PMLR, July 2018, pp. 1989–1998.

[101] J. Hoffman, D. Wang, F. Yu, and T. Darrell, *FCNs in the Wild: Pixel-level Adversarial and Constraint-based Adaptation*, arXiv:1612.02649, (2016).

[102] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural Networks, 2 (1989), pp. 359–366.

[103] H. Hosseini, B. Xiao, M. Jaiswal, and R. Poovendran, *Assessing Shape Bias Property of Convolutional Neural Networks*, arXiv:1803.07739, (2018).

[104] P. V. C. Hough, *Method and means for recognizing complex patterns*, Dec. 1962.

[105] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, arXiv:1704.04861, (2017).

[106] X. Huang and S. Belongie, *Arbitrary Style Transfer in Real-Time With Adaptive Instance Normalization*, in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1501–1510.

[107] E. Hüllermeier and W. Waegeman, *Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods*, Machine Learning, 110 (2021), pp. 457–506.

[108] R. Hussain and S. Zeadally, *Autonomous Cars: Research Results, Issues, and Future Challenges*, IEEE Communications Surveys & Tutorials, 21 (2019), pp. 1275–1313.

[109] S. Ioffe, *Batch renormalization: Towards reducing minibatch dependence in batch-normalized models*, Advances in neural information processing systems, 30 (2017).

[110] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, in International Conference on Machine Learning, pmlr, 2015, pp. 448–456.

[111] M. A. Islam, M. Kowal, P. Esser, S. Jia, B. Ommer, K. G. Derpanis, and N. Bruce, *Shape or Texture: Understanding Discriminative Features in CNNs*, in International Conference on Learning Representations, Jan. 2021.

[112] M. Z. Islam, M. M. Islam, and A. Asraf, *A combined deep CNN-LSTM network for the detection of novel coronavirus (COVID-19) using X-ray images*, Informatics in Medicine Unlocked, 20 (2020), p. 100412.

[113] P. Isola, S. Jegelka, T. Wang, and A. Curtis, *6.S898 Deep Learning Fall 2022*. 2022.

[114] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, *Image-to-Image Translation with Conditional Adversarial Networks*, in 2017 IEEE Conference on Computer Vision and Pattern Recognition, July 2017, pp. 5967–5976.

[115] P. Jaccard, *The Distribution of the Flora in the Alpine Zone*, New Phytologist, 11 (1912), pp. 37–50.

[116] B. Jähne, *Digital Image Processing*, Springer-Verlag, Berlin/Heidelberg, 6 ed., 2005.

[117] P. Jiang, A. Wu, Y. Han, Y. Shao, M. Qi, and B. Li, *Bidirectional Adversarial Training for Semi-Supervised Domain Adaptation*, in Twenty-Ninth International Joint Conference on Artificial Intelligence, vol. 1, July 2020, pp. 934–940.

[118] R. Jiao, Y. Zhang, L. Ding, B. Xue, J. Zhang, R. Cai, and C. Jin, *Learning with limited annotations: A survey on deep semi-supervised learning for medical image segmentation*, Computers in Biology and Medicine, 169 (2024), p. 107840.

[119] L. Jing and Y. Tian, *Self-supervised visual feature learning with deep neural networks: A survey*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 43 (2019), pp. 4037–4058.

[120] K. Johnson, *How Wrongful Arrests Based on AI Derailed 3 Men's Lives*, Wired, (2022).

[121] M. Joneidi, S. Vahidian, A. Esmaeili, W. Wang, N. Rahnavard, B. Lin, and M. Shah, *Select to Better Learn: Fast and Accurate Deep Learning Using Data Selection From Nonlinear Manifolds*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 7819–7829.

[122] M. Jordà, P. Valero-Lara, and A. J. Peña, *Performance Evaluation of cuDNN Convolution Algorithms on NVIDIA Volta GPUs*, IEEE Access, 7 (2019), pp. 70461–70473.

[123] Jülich Supercomputing Centre, *JUWELS: Modular Tier-0/1 Supercomputer at the Jülich Supercomputing Centre*, Journal of large-scale research facilities, 5 (2019).

[124] T. Kalb, *Principles of Catastrophic Forgetting for Continual Semantic Segmentation in Automated Driving*, PhD thesis, Karlsruhe Institut of Technology, 2024.

[125] C. Kamann and C. Rother, *Increasing the Robustness of Semantic Segmentation Models with Painting-by-Numbers*, in Computer Vision – ECCV 2020, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds., Lecture Notes in Computer Science, Cham, 2020, Springer International Publishing, pp. 369–387.

[126] G. Kang, Y. Wei, Y. Yang, Y. Zhuang, and A. Hauptmann, *Pixel-Level Cycle Association: A New Perspective for Domain Adaptive Semantic Segmentation*, in Advances in Neural Information Processing Systems, vol. 33, Curran Associates, Inc., 2020, pp. 3569–3580.

[127] T. Karras, S. Laine, and T. Aila, *A Style-Based Generator Architecture for Generative Adversarial Networks*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 4401–4410.

[128] A. Khoreva, R. Benenson, J. Hosang, M. Hein, and B. Schiele, *Simple Does It: Weakly Supervised Instance and Semantic Segmentation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 876–885.

[129] M. Kim and H. Byun, *Learning Texture Invariant Representation for Domain Adaptation of Semantic Segmentation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 12975–12984.

[130] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, *Learning to Discover Cross-Domain Relations with Generative Adversarial Networks*, in Proceedings of the 34th International Conference on Machine Learning, PMLR, July 2017, pp. 1857–1865.

[131] T. Kim and C. Kim, *Attract, Perturb, and Explore: Learning a Feature Alignment Network for Semi-supervised Domain Adaptation*, in European Conference on Computer Vision – ECCV 2020, Cham, 2020, Springer International Publishing, pp. 591–607.

[132] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, in 3rd International Conference on Learning Representations, Y. Bengio and Y. LeCun, eds., San Diego, CA, USA, 2015.

[133] D. P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*, arXiv:1312.6114, (2022).

[134] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollar, and R. Girshick, *Segment anything*, in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Oct. 2023, pp. 4015–4026.

[135] A. Kirsch, J. van Amersfoort, and Y. Gal, *BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning*, in Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019.

[136] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, *Big Transfer (BiT): General Visual Representation Learning*, in Computer Vision – ECCV 2020, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds., Lecture Notes in Computer Science, Cham, 2020, Springer International Publishing, pp. 491–507.

[137] W. M. Kouw and M. Loog, *An introduction to domain adaptation and transfer learning*, arXiv:1812.11806, (2019).

[138] K. Kowol, S. Bracke, and H. Gottschalk, *survAIval: Survival Analysis with the Eyes of AI*, in Computer-Human Interaction Research and Applications, A. Holzinger, H. P. da Silva, J. Vanderdonckt, and L. Constantine, eds., Communications in Computer and Information Science, Cham, 2023, Springer Nature Switzerland, pp. 153–170.

[139] ——, *A-Eye: Driving with the Eyes of AI for Corner Case Generation*, in 6th International Conference on Computer-Human Interaction Research and Applications, Jan. 2024, pp. 41–48.

[140] P. Krähenbühl and V. Koltun, *Efficient inference in fully connected crfs with gaussian edge potentials*, Advances in neural information processing systems, 24 (2011).

[141] N. Kriegeskorte, *Deep Neural Networks: A New Framework for Modeling Biological Vision and Brain Information Processing*, Annual Review of Vision Science, 1 (2015), pp. 417–446.

[142] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, in Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc., 2012.

[143] J. Kubilius, S. Bracci, and H. P. O. de Beeck, *Deep Neural Networks as a Computational Model for Human Shape Sensitivity*, PLOS Computational Biology, 12 (2016), p. e1004896.

[144] Y. LeCun, Y. Bengio, and G. Hinton, *Deep Learning*, Nature, 521 (2015), pp. 436–44.

[145] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation, 1 (1989), pp. 541–551.

[146] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[147] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*, in Neural Networks: Tricks of the Trade: Second Edition, G. Montavon, G. B. Orr, and K.-R. Müller, eds., Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 9–48.

[148] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4681–4690.

[149] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, *Deeply-Supervised Nets*, in Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, PMLR, Feb. 2015, pp. 562–570.

[150] D.-H. Lee, *Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks*, in Workshop on Challenges in Representation Learning, ICML, vol. 3, Atlanta, 2013, p. 896.

[151] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, *Multilayer feedforward networks with a nonpolynomial activation function can approximate any function*, Neural Networks, 6 (1993), pp. 861–867.

[152] D. D. Lewis and J. Catlett, *Heterogeneous Uncertainty Sampling for Supervised Learning*, in Machine Learning Proceedings 1994, W. W. Cohen and H. Hirsh, eds., Morgan Kaufmann, San Francisco (CA), Jan. 1994, pp. 148–156.

[153] F.-F. Li, Y. Li, and R. Gao, *CS231n Convolutional Neural Networks for Visual Recognition*. 2023.

[154] M. Li and Z.-H. Zhou, *Improve Computer-Aided Diagnosis With Machine Learning Techniques Using Undiagnosed Samples*, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 37 (2007), pp. 1088–1098.

[155] R. Li, W. Cao, Q. Jiao, S. Wu, and H.-S. Wong, *Simplified Unsupervised Image Translation for Semantic Segmentation Adaptation*, Pattern Recognition, 105 (2020), p. 107343.

[156] Y. Li, Q. Yu, M. Tan, J. Mei, P. Tang, W. Shen, A. Yuille, and C. Xie, *Shape-Texture Debiased Neural Network Training*, in International Conference on Learning Representations, Oct. 2020.

[157] M. Lin, Q. Chen, and S. Yan, *Network In Network*, arXiv:1312.4400, (2014).

[158] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, *Focal Loss for Dense Object Detection*, in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2980–2988.

[159] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, *Microsoft COCO: Common Objects in Context*, in Computer Vision – ECCV 2014, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds., vol. 8693, Springer International Publishing, Cham, 2014, pp. 740–755.

[160] S. Linnainmaa, *Taylor expansion of the accumulated rounding error*, BIT Numerical Mathematics, 16 (1976), pp. 146–160.

[161] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, *Swin transformer: Hierarchical vision transformer using shifted windows*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 10012–10022.

[162] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, *A ConvNet for the 2020s*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 11976–11986.

[163] J. Long, E. Shelhamer, and T. Darrell, *Fully Convolutional Networks for Semantic Segmentation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431–3440.

[164] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization*, in International Conference on Learning Representations, Sept. 2018.

[165] T. Luong, H. Pham, and C. D. Manning, *Effective Approaches to Attention-based Neural Machine Translation*, in Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, L. Màrquez, C. Callison-Burch, and J. Su, eds., Lisbon, Portugal, Sept. 2015, Association for Computational Linguistics, pp. 1412–1421.

[166] L. Ma, *Pure Pytorch implementation of RGB to HSV/HSL conversion*, Dec. 2023.

[167] A. L. Maas, A. Y. Hannun, and A. Y. Ng, *Rectifier nonlinearities improve neural network acoustic models*, in ICM Workshop on Deep Learning for Audio, Speech, and Language Processing., vol. 30, Atlanta, GA, 2013, p. 3.

[168] S. Mabu, M. Miyake, T. Kuremoto, and S. Kido, *Semi-supervised CycleGAN for domain transformation of chest CT images and its application to opacity classification of diffuse lung diseases*, International Journal of Computer Assisted Radiology and Surgery, 16 (2021), pp. 1925–1935.

[169] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. Paul Smolley, *Least Squares Generative Adversarial Networks*, in Proceedings of the IEEE International Conference on Computer Vision, Oct. 2017, pp. 2794–2802.

[170] W. S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics, 5 (1943), pp. 115–133.

[171] K. Mei, C. Zhu, J. Zou, and S. Zhang, *Instance Adaptive Self-Training for Unsupervised Domain Adaptation*, arXiv:2008.12197, (2020).

[172] D. A. Mély, J. Kim, M. McGill, Y. Guo, and T. Serre, *A systematic comparison between visual cues for boundary detection*, Vision Research, 120 (2016), pp. 93–107.

[173] J. B. Merrill and F. Siddiqui, *17 fatalities, 736 crashes: The shocking toll of Tesla's Autopilot*, Washington Post, (2023).

[174] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, *Unrolled Generative Adversarial Networks*, in International Conference on Learning Representations, Nov. 2016.

[175] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, *Image Segmentation Using Deep Learning: A Survey*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2021).

[176] D. Mindlin, M. Schilling, and P. Cimiano, *ABC-GAN: Spatially Constrained Counterfactual Generation for Image Classification Explanations*, in Explainable Artificial Intelligence, L. Longo, ed., Communications in Computer and Information Science, Cham, 2023, Springer Nature Switzerland, pp. 260–282.

[177] M. Mirza and S. Osindero, *Conditional Generative Adversarial Nets*, arXiv:1411.1784, (2014).

[178] T. M. Mitchell, *The Need for Biases in Learning Generalizations*, Tech. Rep. CBM-TR-117, Computer Science Department, New Brunswick, NJ 08904, May 1980.

[179] MMSegmentation Contributors, *OpenMMLab semantic segmentation toolbox and benchmark*, July 2020.

[180] S. Mohamed and B. Lakshminarayanan, *Learning in Implicit Generative Models*, arXiv:1610.03483, (2017).

[181] P. Munjal, N. Hayat, M. Hayat, J. Sourati, and S. Khan, *Towards Robust and Reproducible Active Learning Using Neural Networks*, arxiv:2002.09564, (2020).

[182] Z. Murez, S. Kolouri, D. Kriegman, R. Ramamoorthi, and K. Kim, *Image to Image Translation for Domain Adaptation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, June 2018, pp. 4500–4509.

[183] K. P. Murphy, *Probabilistic Machine Learning: An Introduction*, MIT Press, 2022.

[184] ——, *Probabilistic Machine Learning: Advanced Topics*, MIT Press, 2023.

[185] A. Mütze, M. Rottmann, and H. Gottschalk, *Semi-Supervised Domain Adaptation with CycleGAN Guided by Downstream Task Awareness*, in Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2023) - Volume 5: VISAPP, SciTePress, 2023, pp. 80–90.

[186] ——, *Semi-supervised Task Aware Image-to-Image Translation*, in International Joint Conference on Computer Vision, Imaging and Computer Graphics, vol. 2103 of Communications in Computer and Information Science (CCIS), Springer Nature Switzerland, Cham, to appear.

[187] M. M. Naseer, K. Ranasinghe, S. H. Khan, M. Hayat, F. Shahbaz Khan, and M.-H. Yang, *Intriguing properties of vision transformers*, in Advances in Neural Information Processing Systems, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds., vol. 34, Curran Associates, Inc., 2021, pp. 23296–23308.

[188] H. Navidan, P. F. Moshiri, M. Nabati, R. Shahbazian, S. A. Ghorashi, V. Shah-Mansouri, and D. Windridge, *Generative Adversarial Networks (GANs) in networking: A comprehensive survey & evaluation*, Computer Networks, 194 (2021), p. 108149.

[189] H. T. Nguyen and A. Smeulders, *Active learning using pre-clustering*, in Proceedings of the Twenty-First International Conference on Machine Learning, 2004, p. 79.

[190] T. T. Nguyen, Q. V. H. Nguyen, D. T. Nguyen, D. T. Nguyen, T. Huynh-The, S. Nahavandi, T. T. Nguyen, Q.-V. Pham, and C. M. Nguyen, *Deep learning for deepfakes creation and detection: A survey*, Computer Vision and Image Understanding, 223 (2022), p. 103525.

[191] S. Niklaus, *A Reimplementation of {HED} Using {PyTorch}*, 2018.

[192] C. G. Northcutt, A. Athalye, and J. Mueller, *Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks*, in Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1), June 2021.

[193] S. Nowozin, B. Cseke, and R. Tomioka, *F-GAN: Training Generative Neural Samplers using Variational Divergence Minimization*, in Advances in Neural Information Processing Systems, vol. 29, Curran Associates, Inc., 2016.

[194] E. Ntoutsi, P. Fafalios, U. Gadiraju, V. Iosifidis, W. Nejdl, M.-E. Vidal, S. Ruggieri, F. Turini, S. Papadopoulos, E. Krasanakis, I. Kompatsiaris, K. Kinder-Kurlanda, C. Wagner, F. Karimi, M. Fernandez, H. Alani, B. Berendt, T. Kruegel, C. Heinze, K. Broelemann, G. Kasneci, T. Tiropanis, and S. Staab, *Bias in data-driven artificial intelligence systems—An introductory survey*, WIREs Data Mining and Knowledge Discovery, 10 (2020), p. e1356.

[195] P. Oberdiek, G. Fink, and M. Rottmann, *UQGAN: A unified model for uncertainty quantification of deep classifiers trained via conditional GANs*, in Advances in Neural Information Processing Systems, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds., vol. 35, Curran Associates, Inc., 2022, pp. 21371–21385.

[196] OpenAI, *GPT-4 Technical Report*, arXiv:2303.08774, (2023).

[197] S. J. Pan and Q. Yang, *A Survey on Transfer Learning*, IEEE Transactions on Knowledge and Data Engineering, 22 (2010), pp. 1345–1359.

[198] Y. Pang, J. Lin, T. Qin, and Z. Chen, *Image-to-Image Translation: Methods and Applications*, IEEE Transactions on Multimedia, (2021).

[199] J.-H. Park, Y.-S. Kim, H. Seo, and Y.-J. Cho, *Analysis of Training Deep Learning Models for PCB Defect Detection*, Sensors, 23 (2023), p. 2766.

[200] G. Parmar, K. Kumar Singh, R. Zhang, Y. Li, J. Lu, and J.-Y. Zhu, *Zero-shot Image-to-Image Translation*, in ACM SIGGRAPH 2023 Conference Proceedings, SIGGRAPH '23, New York, NY, USA, July 2023, Association for Computing Machinery, pp. 1–11.

[201] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[202] P. Perona and J. Malik, *Scale-space and edge detection using anisotropic diffusion*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 12 (1990), pp. 629–639.

[203] J. C. Peterson, R. M. Battleday, T. L. Griffiths, and O. Russakovsky, *Human uncertainty makes classification more robust*, in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Los Alamitos, CA, USA, Nov. 2019, IEEE Computer Society, pp. 9617–9626.

[204] R. Po, W. Yifan, V. Golyanik, K. Aberman, J. T. Barron, A. H. Bermano, E. R. Chan, T. Dekel, A. Holynski, A. Kanazawa, C. K. Liu, L. Liu, B. Mildenhall, M. Niessner, B. Ommer, C. Theobalt, P. Wonka, and G. Wetzstein, *State of the Art on Diffusion Models for Visual Computing*, arXiv:2310.07204, (2023).

[205] E. Poeta, G. Ciravegna, E. Pastor, T. Cerquitelli, and E. Baralis, *Concept-based Explainable Artificial Intelligence: A Survey*, arXiv:2312.12936, (2023).

[206] B. T. Polyak, *Some methods of speeding up the convergence of iteration methods*, Ussr computational mathematics and mathematical physics, 4 (1964), pp. 1–17.

[207] V. Prabhu, A. Chandrasekaran, K. Saenko, and J. Hoffman, *Active Domain Adaptation via Clustering Uncertainty-Weighted Embeddings*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 8505–8514.

[208] S. J. Prince, *Understanding Deep Learning*, MIT Press, 2023.

[209] A. Radford, L. Metz, and S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, arXiv:1511.06434, (2016).

[210] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, *Improving language understanding by generative pre-training*, (2018).

[211] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, *Language models are unsupervised multitask learners*, OpenAI blog, 1 (2019), p. 9.

[212] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, *Exploring the limits of transfer learning with a unified text-to-text transformer*, The Journal of Machine Learning Research, 21 (2020), pp. 140:5485–140:5551.

[213] M. Raghu, C. Zhang, J. Kleinberg, and S. Bengio, *Transfusion: Understanding Transfer Learning for Medical Imaging*, in Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019.

[214] P. Rai, A. Saha, H. Daumé, and S. Venkatasubramanian, *Domain Adaptation meets Active Learning*, in Proceedings of the NAACL HLT 2010 Workshop on Active Learning for Natural Language Processing, B. Settles, K. Small, and K. Tomanek, eds., Los Angeles, California, June 2010, Association for Computational Linguistics, pp. 27–32.

[215] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, *Hierarchical Text-Conditional Image Generation with CLIP Latents*, arXiv:2204.06125, (2022).

[216] L. J. Ratliff, S. A. Burden, and S. S. Sastry, *Characterization and computation of local Nash equilibria in continuous games*, in 2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton), IEEE, Oct. 2013, pp. 917–924.

[217] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré, *Data Programming: Creating Large Training Sets, Quickly*, in Advances in Neural Information Processing Systems, vol. 29, Curran Associates, Inc., 2016.

[218] S. Ren, K. He, R. Girshick, and J. Sun, *Faster R-CNN: Towards real-time object detection with region proposal networks*, in Advances in Neural Information Processing Systems, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds., vol. 28, Curran Associates, Inc., 2015.

[219] D. J. Rezende, S. Mohamed, and D. Wierstra, *Stochastic Backpropagation and Approximate Inference in Deep Generative Models*, in Proceedings of the 31st International Conference on Machine Learning, PMLR, June 2014, pp. 1278–1286.

[220] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, *Playing for Data: Ground Truth from Computer Games*, in European Conference on Computer Vision (ECCV), B. Leibe, J. Matas, N. Sebe, and M. Welling, eds., vol. 9906 of LNCS, Springer International Publishing, 2016, pp. 102–118.

[221] L. Roberts, *Machine Perception of Three-Dimensional Solids*, Garland Publishing, New York, NY, USA, 1963.

[222] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-Resolution Image Synthesis With Latent Diffusion Models*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10684–10695.

[223] O. Ronneberger, P. Fischer, and T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, in Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds., vol. 9351, Springer International Publishing, Cham, 2015, pp. 234–241.

[224] G. Ros, L. Sellart, J. Materzynska, D. Vázquez, and A. M. López, *The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes*, in The IEEE Conference on Computer Vision and Pattern Recognition, June 2016, pp. 3234–3243.

[225] K. Rösch, F. Heidecker, J. Truetsch, K. Kowol, C. Schicktanz, M. Bieshaare, B. Sick, and C. Stiller, *Space, Time, and Interaction: A Taxonomy of Corner Cases in Trajectory Datasets for Automated Driving*, in 2022 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2022, pp. 86–93.

[226] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain.*, Psychological Review, 65 (1958), pp. 386–408.

[227] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, vol. 55, Spartan books Washington, DC, 1962.

[228] M. Rottmann and M. Reese, *Automated Detection of Label Errors in Semantic Segmentation Datasets via Deep Learning and Uncertainty Quantification*, in 2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), Jan. 2023, pp. 3213–3222.

[229] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature, 323 (1986), pp. 533–536.

[230] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision, 115 (2015), pp. 211–252.

[231] K. Saito, D. Kim, S. Sclaroff, T. Darrell, and K. Saenko, *Semi-supervised domain adaptation via minimax entropy*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 8050–8058.

[232] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen, *Improved techniques for training GANs*, in Advances in Neural Information Processing Systems, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, eds., vol. 29, Curran Associates, Inc., 2016.

[233] N. Sambasivan, S. Kapania, H. Highfill, D. Akrong, P. Paritosh, and L. M. Aroyo, *"Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI*, Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, (2021), pp. 1–15.

[234] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[235] P. Sangkloy, N. Burnell, C. Ham, and J. Hays, *The sketchy database: Learning to retrieve badly drawn bunnies*, ACM Transactions on Graphics, 35 (2016), pp. 119:1–119:12.

[236] I. H. Sarker, *Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions*, SN Computer Science, 2 (2021), p. 420.

[237] A. Sauer and A. Geiger, *Counterfactual generative networks*, in International Conference on Learning Representations, 2021.

[238] A. Sauer, K. Schwarz, and A. Geiger, *StyleGAN-XL: Scaling StyleGAN to Large Diverse Datasets*, in ACM SIGGRAPH 2022 Conference Proceedings, SIGGRAPH '22, New York, NY, USA, July 2022, Association for Computing Machinery, pp. 1–10.

[239] T. Scheffer, C. Decomain, and S. Wrobel, *Active Hidden Markov Models for Information Extraction*, in Advances in Intelligent Data Analysis, F. Hoffmann, D. J. Hand, N. Adams, D. Fisher, and G. Guimaraes, eds., Lecture Notes in Computer Science, Berlin, Heidelberg, 2001, Springer, pp. 309–318.

[240] C. Schmaltz, J. Weickert, and A. Bruhn, *Beating the Quality of JPEG 2000 with Anisotropic Diffusion*, in Pattern Recognition, J. Denzler, G. Notni, and H. Süße, eds., Lecture Notes in Computer Science, Berlin, Heidelberg, 2009, Springer, pp. 452–461.

[241] M. Schwonberg, F. E. Bouazati, N. M. Schmidt, and H. Gottschalk, *Augmentation-based Domain Generalization for Semantic Segmentation*, in 2023 IEEE Intelligent Vehicles Symposium (IV), June 2023, pp. 1–8.

[242] M. Schwonberg, J. Niemeijer, J.-A. Termöhlen, J. P. schäfer, N. M. Schmidt, H. Gottschalk, and T. Fingscheidt, *Survey on Unsupervised Domain Adaptation for Semantic Segmentation for Visual Perception in Automated Driving*, IEEE Access, 11 (2023), pp. 54296–54336.

[243] O. Sener and S. Savarese, *Active Learning for Convolutional Neural Networks: A Core-Set Approach*, in International Conference on Learning Representations, Feb. 2018.

[244] B. Settles, *Active Learning Literature Survey*, Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[245] H. S. Seung, M. Opper, and H. Sompolinsky, *Query by committee*, in Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92, New York, NY, USA, July 1992, Association for Computing Machinery, pp. 287–294.

[246] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, Cambridge, 2014.

[247] C. E. Shannon, *A mathematical theory of communication*, The Bell System Technical Journal, 27 (1948), pp. 379–423.

[248] L. G. Shapiro and G. C. Stockman, *Computer Vision*, Prentice Hall, 2001.

[249] G. Shen, Y. Jiao, Y. Lin, and J. Huang, *Differentiable Neural Networks with RePU Activation: With Applications to Score Estimation and Isotonic Regression*, Apr. 2023.

[250] I. Shin, D.-J. Kim, J. W. Cho, S. Woo, K. Park, and I. S. Kweon, *LabOR: Labeling Only if Required for Domain Adaptive Semantic Segmentation*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 8588–8598.

[251] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, *Learning From Simulated and Unsupervised Images Through Adversarial Training*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 2107–2116.

[252] S. Shukla, L. Van Gool, and R. Timofte, *Extremely Weak Supervised Image-to-Image Translation for Semantic Segmentation*, in 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), Oct. 2019, pp. 3368–3377.

[253] J. Sietsma and R. J. F. Dow, *Creating artificial neural networks that generalize*, Neural Networks, 4 (1991), pp. 67–79.

[254] T. Simonite, *When It Comes to Gorillas, Google Photos Remains Blind*, Wired, (2018).

[255] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, in International Conference on Learning Representations, 2015.

[256] A. Singh, *CLDA: Contrastive Learning for Semi-Supervised Domain Adaptation*, in Advances in Neural Information Processing Systems, Nov. 2021.

[257] W. K. small yellow duck, *Painter by numbers*, 2016.

[258] A. R. Smith, *Color gamut transform pairs*, ACM SIGGRAPH Computer Graphics, 12 (1978), pp. 12–19.

[259] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research, 15 (2014), pp. 1929–1958.

[260] R. Strudel, R. Garcia, I. Laptev, and C. Schmid, *Segmenter: Transformer for Semantic Segmentation*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 7262–7272.

[261] J.-C. Su, Y.-H. Tsai, K. Sohn, B. Liu, S. Maji, and M. Chandraker, *Active Adversarial Domain Adaptation*, in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2020, pp. 739–748.

[262] S. Suzuki and K. be, *Topological structural analysis of digitized binary images by border following*, Computer Vision, Graphics, and Image Processing, 30 (1985), pp. 32–46.

[263] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015, pp. 1–9.

[264] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the Inception Architecture for Computer Vision*, in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016, pp. 2818–2826.

[265] M. Tan and Q. Le, *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*, in Proceedings of the 36th International Conference on Machine Learning, PMLR, May 2019, pp. 6105–6114.

[266] A. Tao, K. Sapra, and B. Catanzaro, *Hierarchical multi-scale attention for semantic segmentation*, arXiv:2005.10821, (2020).

[267] F. Tao, B. Xiao, Q. Qi, J. Cheng, and P. Ji, *Digital twin modeling*, Journal of Manufacturing Systems, 64 (2022), pp. 372–389.

[268] L. Theis, A. van den Oord, and M. Bethge, *A note on the evaluation of generative models*, arXiv:1511.01844, (2016).

[269] J. Theodoridis, J. Hofmann, J. Maucher, and A. Schilling, *Trapped in Texture Bias? A Large Scale Comparison of Deep Instance Segmentation*, in Computer Vision – ECCV 2022, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, eds., Lecture Notes in Computer Science, Cham, 2022, Springer Nature Switzerland, pp. 609–627.

[270] T. Tieleman, G. Hinton, et al., *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural networks for machine learning, 4 (2012), pp. 26–31.

[271] M. Toldo, A. Maracani, U. Michieli, and P. Zanuttigh, *Unsupervised Domain Adaptation in Semantic Segmentation: A Review*, Technologies, 8 (2020), p. 35.

[272] M. Toldo, U. Michieli, G. Agresti, and P. Zanuttigh, *Unsupervised domain adaptation for mobile semantic segmentation based on cycle consistency and feature alignment*, Image and Vision Computing, 95 (2020), p. 103889.

[273] J. M. Tomczak, *Deep Generative Modeling*, Springer International Publishing, Cham, 2022.

[274] S. Torquato, *Cell and Random-Field Models*, in Random Heterogeneous Materials: Microstructure and Macroscopic Properties, Interdisciplinary Applied Mathematics, Springer, New York, NY, 2002, pp. 188–209.

[275] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, *Training data-efficient image transformers & distillation through attention*, in Proceedings of the 38th International Conference on Machine Learning, PMLR, July 2021, pp. 10347–10357.

[276] A. Tripathi, R. Singh, A. Chakraborty, and P. Shenoy, *Edges to Shapes to Concepts: Adversarial Augmentation for Robust Vision*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 24470–24479.

[277] Y.-H. Tsai, W.-C. Hung, S. Schulter, K. Sohn, M.-H. Yang, and M. Chandraker, *Learning to Adapt Structured Output Space for Semantic Segmentation*, in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, June 2018, pp. 7472–7481.

[278] Y.-H. Tsai, K. Sohn, S. Schulter, and M. Chandraker, *Domain adaptation for structured output via discriminative patch representations*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1456–1465.

[279] S. Tuli, I. Dasgupta, E. Grant, and T. L. Griffiths, *Are Convolutional Neural Networks or Transformers more like human vision?*, arXiv:2105.07197, (2021).

[280] J. E. van Engelen and H. H. Hoos, *A survey on semi-supervised learning*, Machine Learning, 109 (2020), pp. 373–440.

[281] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., vol. 30, Curran Associates, Inc., 2017.

[282] A. Veit, M. J. Wilber, and S. Belongie, *Residual Networks Behave Like Ensembles of Relatively Shallow Networks*, in Advances in Neural Information Processing Systems, vol. 29, Curran Associates, Inc., 2016.

[283] T.-H. Vu, H. Jain, M. Bucher, M. Cord, and P. Perez, *ADVENT: Adversarial Entropy Minimization for Domain Adaptation in Semantic Segmentation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[284] D. Wang and Y. Shang, *A new active labeling method for deep learning*, in 2014 International Joint Conference on Neural Networks (IJCNN), July 2014, pp. 112–119.

[285] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, *High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 8798–8807.

[286] W. Wang, J. Dai, Z. Chen, Z. Huang, Z. Li, X. Zhu, X. Hu, T. Lu, L. Lu, H. Li, X. Wang, and Y. Qiao, *InternImage: Exploring Large-Scale Vision Foundation Models With Deformable Convolutions*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 14408–14419.

[287] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, *Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction Without Convolutions*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 568–578.

[288] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. Change Loy, *ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks*, in Proceedings of the European Conference on Computer Vision (ECCV) Workshops, 2018, pp. 0–0.

[289] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, *Image quality assessment: From error visibility to structural similarity*, IEEE Transactions on Image Processing, 13 (2004), pp. 600–612.

[290] Z. Wang, Q. She, and T. E. Ward, *Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy*, ACM Computing Surveys, 54 (2021), pp. 1–38.

[291] Z. Wang, Y. Wei, R. Feris, J. Xiong, W.-M. Hwu, T. S. Huang, and H. Shi, *Alleviating Semantic-Level Shift: A Semi-Supervised Domain Adaptation Method for Semantic Segmentation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2020, pp. 936–937.

[292] J. Weickert, *Coherence-enhancing diffusion of colour images1Extended version of a presentation at the Seventh National Symposium on Pattern Recognition and Image Analysis (Barcelona, April 21–25, 1997).1*, Image and Vision Computing, 17 (1999), pp. 201–212.

[293] J. Weickert and M. Welk, *Tensor Field Interpolation with PDEs*, in Visualization and Processing of Tensor Fields, J. Weickert and H. Hagen, eds., Mathematics and Visualization, Springer, Berlin, Heidelberg, 2006, pp. 315–325.

[294] J. Weickert, M. Welk, and M. Wickert, *L2-Stable Nonstandard Finite Differences for Anisotropic Diffusion*, in Scale Space and Variational Methods in Computer Vision, A. Kuijper, K. Bredies, T. Pock, and H. Bischof, eds., Lecture Notes in Computer Science, Berlin, Heidelberg, 2013, Springer, pp. 380–391.

[295] B. Wilson, J. Hoffman, and J. Morgenstern, *Predictive Inequity in Object Detection*, arXiv:1902.11097, (2019).

[296] M. Wrenninge and J. Unger, *Synscapes: A Photorealistic Synthetic Dataset for Street Scene Parsing*, arXiv:1810.08705, (2018).

[297] T.-H. Wu, Y.-S. Liou, S.-J. Yuan, H.-Y. Lee, T.-I. Chen, K.-C. Huang, and W. H. Hsu, *$\mathrm {D^2ADA}$: Dynamic Density-Aware Active Domain Adaptation for Semantic Segmentation*, in Computer Vision – ECCV 2022, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, eds., Lecture Notes in Computer Science, Cham, 2022, Springer Nature Switzerland, pp. 449–467.

[298] Z. Wu, X. Han, Y.-L. Lin, M. G. Uzunbas, T. Goldstein, S. N. Lim, and L. S. Davis, *Dcan: Dual channel-wise alignment networks for unsupervised scene adaptation*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 518–534.

[299] B. Xie, L. Yuan, S. Li, C. H. Liu, X. Cheng, and G. Wang, *Active Learning for Domain Adaptation: An Energy-Based Approach*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, June 2022, pp. 8708–8716.

[300] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, *SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers*, in Advances in Neural Information Processing Systems, Nov. 2021.

[301] M. Xie, S. Li, R. Zhang, and C. H. Liu, *Dirichlet-based Uncertainty Calibration for Active Domain Adaptation*, in The Eleventh International Conference on Learning Representations, 2023.

[302] S. Xie and Z. Tu, *Holistically-nested edge detection*, in Proceedings of IEEE International Conference on Computer Vision, 2015.

[303] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, and J. Zhu, *Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges*, in Natural Language Processing and Chinese Computing, J. Tang, M.-Y. Kan, D. Zhao, S. Li, and H. Zan, eds., Lecture Notes in Computer Science, Cham, 2019, Springer International Publishing, pp. 563–574.

[304] J. Xu, *A deep learning approach to building an intelligent video surveillance system*, Multimedia Tools and Applications, 80 (2021), pp. 5495–5515.

[305] Y. Yang and S. Soatto, *Fda: Fourier domain adaptation for semantic segmentation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 4085–4095.

[306] H. Yao, X. Hu, and X. Li, *Enhancing pseudo label quality for semi-supervised domain-generalized medical image segmentation*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, 2022, pp. 3099–3107.

[307] D. Yarotsky, *Error bounds for approximations with deep ReLU networks*, Neural Networks, 94 (2017), pp. 103–114.

[308] ——, *Universal Approximations of Invariant Maps by Neural Networks*, Constructive Approximation, 55 (2022), pp. 407–474.

[309] Z. Yi, H. Zhang, P. Tan, and M. Gong, *DualGAN: Unsupervised Dual Learning for Image-to-Image Translation*, in 2017 IEEE International Conference on Computer Vision (ICCV), IEEE Computer Society, Oct. 2017, pp. 2868–2876.

[310] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, *Generative image inpainting with contextual attention*, arXiv:1801.07892, (2018).

[311] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z.-H. Jiang, F. E. Tay, J. Feng, and S. Yan, *Tokens-to-token ViT: Training vision transformers from scratch on ImageNet*, in Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Oct. 2021, pp. 558–567.

[312] L. Yuan, Q. Hou, Z. Jiang, J. Feng, and S. Yan, *VOLO: Vision Outlooker for Visual Recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 45 (2023), pp. 6575–6586.

[313] C. Yun, S. Bhojanapalli, A. S. Rawat, S. Reddi, and S. Kumar, *Are Transformers universal approximators of sequence-to-sequence functions?*, in International Conference on Learning Representations, Sept. 2019.

[314] J.-N. Zaech, D. Dai, M. Hahner, and L. V. Gool, *Texture Underfitting for Domain Adaptation*, in 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Oct. 2019, pp. 547–552.

[315] J. R. Zech, M. A. Badgeley, M. Liu, A. B. Costa, J. J. Titano, and E. K. Oermann, *Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study*, PLOS Medicine, 15 (2018), p. e1002683.

[316] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, Cambridge University Press, 2023.

[317] P. Zhang, B. Zhang, T. Zhang, D. Chen, Y. Wang, and F. Wen, *Prototypical pseudo label denoising and target structure learning for domain adaptive semantic segmentation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 12414–12424.

[318] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, *The unreasonable effectiveness of deep features as a perceptual metric*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 586–595.

[319] Y. Zhang and M. A. Mazurowski, *Convolutional neural networks rarely learn shape for semantic segmentation*, Pattern Recognition, 146 (2024), p. 110018.

[320] Y. Zhao, R. Wu, and H. Dong, *Unpaired Image-to-Image Translation Using Adversarial Consistency Loss*, in Computer Vision – ECCV 2020, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds., Lecture Notes in Computer Science, Cham, 2020, Springer International Publishing, pp. 800–815.

[321] C. Zhou, Y. Huang, M. Pu, Q. Guan, L. Huang, and H. Ling, *The Treasure Beneath Multiple Annotations: An Uncertainty-Aware Edge Detector*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 15507–15517.

[322] J. Zhou, C. Wei, H. Wang, W. Shen, C. Xie, A. Yuille, and T. Kong, *Image BERT Pre-training with Online Tokenizer*, in International Conference on Learning Representations, Oct. 2021.

[323] J.-J. Zhu and J. Bento, *Generative Adversarial Active Learning*, arXiv:1702.07956, (2017).

[324] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, in IEEE International Conference on Computer Vision (ICCV), 2017.

[325] Y. Zou, Z. Yu, B. V. K. V. Kumar, and J. Wang, *Unsupervised Domain Adaptation for Semantic Segmentation via Class-Balanced Self-Training*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 289–305.

[326] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, *Object Detection in 20 Years: A Survey*, Proceedings of the IEEE, 111 (2023), pp. 257–276.