Improving Artificial Intelligence Performance through Architectural and Procedural A-Priori

A case-study of optimising neural architectures for automotive applications

von der Fakultät für Elektrotechnik, Informationstechnik und Medientechnik
der Bergischen Universität Wuppertal genehmigte

Dissertation

zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften

von

M. Sc. Ido Freeman

aus

Düsseldorf

Wuppertal im Feb. 2023

Tag der Prüfung	09.12.2022
Hauptrerefent	Prof. Dr.-Ing. Anton Kummert
Korreferent	Prof. Dr.-Ing. Reinhard Möller

Abstract

Throughout this work, I will cover my humble contributions to the field of Machine Learning (ML) and Artificial Intelligence (AI) for Advanced Driver Assistance System (ADAS) applications. Over the course of three main chapters, three different types of a-priori are presented. Integrated into various neural architectures, I show how these a-priori, also world knowledge or priors, help improve performance in their respective fields.

First, Chapter 2 covers the topic of semantic segmentation and free space detection. It discusses the current state-of-the-art (SotA) and analyses the respective drawbacks. Understanding the reasons behind the limitations of the current art, a novel adaptive mask is proposed which improves both the runtime as well as the accuracy over the baseline model. The mask is efficiently generated on a per-sample basis and is used to direct the model towards the more challenging parts of the segmentation map, the edges.

In Chapter 3, the computational and mathematical foundations of the convolution function are discussed and analysed. Understanding the accompanying computational bottlenecks, an approximative convolution block is proposed, composed to solve the bottlenecks while reducing the computational effort.

Finally, Chapter 4 discusses the representation of predicted trajectories as polynomials. A new output formulation, which was developed for the use-case of motorway trajectory predictions. A field which demands a thorough understanding of vehicles' behaviours. With the direct prediction of a function, representing a movement in time and space, the model manages to better generalise and predict more realistic trajectories.

By the end of this work, I hope to have convinced the reader that there is more to Data Science and Machine Learning than executing a pre-trained model from GitHub. By truly understanding the task at hand, from the data through the model to the outputs, creative algorithms could be realised to bridge the gap between the SotA academic research and their applicability to every day consumer products.

Last but not least, a note regarding the first-person plural perspective in this work. This work covers my various research projects. It was generally implemented

by me around my ideas. Yet, no man is an island. My ideas were thoroughly discussed, debated, refined, extended and occasionally also completely revised by my colleagues, supervisors and friends before graduating to a project worth publishing.

I feel that writing the work in the first-person singular form would not do justice with one of the core concepts of the research community - the scientific exchange. The rest of this work is hence written in the first-person plural form, ‘we’. It is used as a way of recognising the time and effort of everyone who supported it along the way.

The generation of the cover photo of this print was assisted by OpenAI’s DALL-E2 generative neural network [1]. The final query for the presented cover picture was ‘Multiple cars on a motorway in photorealistic style from a car’s ego perspective. The ego car has its future driving trajectory drawn in-front of it in the form of a continuous orange line which starts at the ego vehicle and shows a planned lane change’. It was then manually edited and extended using DALL-E2’s editing tool to match my expectations and the format requirements.

Acknowledgements

Writing a dissertation while working a full-time job in parallel has proven to be a tremendous challenge. It is both physically as well as mentally demanding, to strive for a high-quality academic work while having to finish an eight-hours working day before sitting down to write. For this reason, this work cannot be complete without a special recognition of those who have dedicated a part of their extremely precious time to support me and my work. Support which had many faces. Be it professional, physical or mental. During but also prior to the period of writing, throughout the course of the research projects described in this work.

First and foremost, a special thank you to my supervisor, Prof. Anton Kummert. From day one you have always been a supporting column of optimism and support. You have provided me an academic freedom with the selection of projects, and at the same time you were always there to discuss and answer whatever questions I might have had. Your utter sense of reassurance has encouraged me to believe in myself, even and mostly through hard times.

I would also like to thank, in chronological order, Mirko Meuter and Christian Nunn who did a great job selling Aptiv and the team, when other options where on the table. I am so very happy you did. Jan Siegemund who can somehow immediately grasp the strangest of concepts and then improve it. Klaus Friedrichs who is not only one of the brightest minds I have ever worked with, but also by far the quickest thinker I know. Lutz Roese-Koerner who is such a good friend and mentor that his name literally appears in all acknowledgements sections from our group. Kun Zhao from whom I am still learning. Be it theoretical background or the astonishing drive to always fully understand every last little bit of information. Dennis Müller who could sell ice in the north pole and happily uses this trait to encourage you become a better version of yourself. Last but not least, Markus Bühren who is the most caring, quick learning and organised person I have ever had the pleasure of working with. I am still learning from you every day and awkwardly enough, I do not see it changing anytime soon.

For hours of discussions of all colours and kinds I owe a special thanks to, in alphabetical order, Antonia van Betteray, Frederik Hasecke, Lukas Hahn, Martin Alsfasser, Pascal Colling, Peet Cremer and Sönke Behrends. This dream team has not only taught me a lot but has also made me become a better version of myself. They challenged my creativity and compensated for my ignorance. At the same time, they also provided the occasional yet much needed break, allowing me to recharge and continue this marathon of a work.

For extensive and elaborate feedback and proofreading of this work, I thank Anna, Fabian, Frederik, Kun, Lukas, Lutz, Martin, Robert and Tobias.

Finally, the entire Machine Learning and Artificial Intelligence organisation, under the leadership of Christian Nunn. A wonderful lot of colleagues and friends which behaves and acts like a family.

For the aforementioned physical aspects, I would like to add Anke and Sebastian, who have always understood that the best way to take the mind off of work is to deprive it of oxygen and send it to the dark and cold bottom of random lakes. Annika, who is the most understanding, accepting and supportive person I know.

Last, my family, including my wonderful partner Lisa, who had to listen to hours on end of technical talk they could not have cared less about. They have done so just because they recognised its value for me and have demonstrated nothing but an utter support and patience.

You all are amazing people.

Contents

1 Introduction

Early in the last decade, the available computational capacity has reached an important milestone. For the first time, research institutes had enough computational power to develop the mostly theoretical and decades old fundamentals of Artificial Neural Network and Artificial Intelligence. A few of the most well established algorithms in the field have seen their foundations in theories, discussed by great minds like the father of modern computing himself, Alan Turing [2], and later by the Turing Award winners Bengio, Hinton and LeCun.

One of the most renounced modern works in Deep Learning (DL) is the 2012 work of Krizhevsky et al. [3] on the 2010 ImageNet Large Scale Visual Recognition Challenge [4]. The challenge consists of $1{,}000$ visual object classes, represented by ~ 1.2 million training images and $\sim 50{,}000$ validation samples. To date it is considered one of the more extensive datasets for image classification. It is still often used by researchers as a benchmark or as a baseline for supervised pre-training.

The aforementioned work, although novel and widely important, was mostly a re-implementation to modern scale of the Convolutional Neural Network (CNN) algorithm presented in [5]. However, arguably the main novelty was the distributed computing architecture. Equipped with two Graphics Processing Units (GPUs) of type GTX 580, it reached a sufficient computational capacity to properly train a CNN for image classification.

As a reference, while Krizhevsky et al. reported a training time of "five to six days" to reach a top-1 error rate of $37.5\ \%$ [3], the latest entry on the Sanford training benchmark page [6] claims a training time of $2{:}38$ hours to an error rate of $6.96\ \%$.

This is not to claim, that the model's size was unjustified. The ImageNet classification leader board shows a clear correlation between the number of parameters and the top-1 accuracy [7]. Furthermore trend line of models' sizes appears to not yet have reached saturation. The largest and best scoring models are also the among the newest. For reference, Figure 1.1 visualises a selected few of the language models published in and around 2022 by their number of parameters. One can clearly see to enormous extent of these networks.

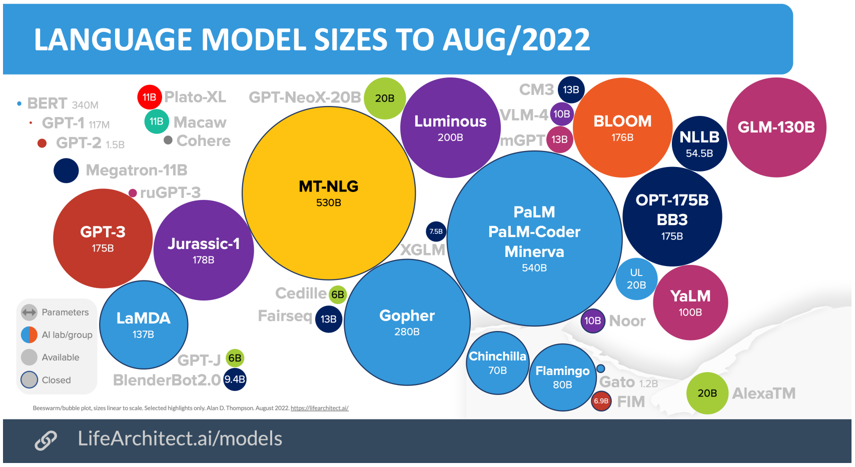


Fig. 1.1: A visualisation of the sizes of recent and selected language models. Notice how even large models as the renounced 'GPT-3' is dwarfed in size compared to the slightly more recent 'PaLM' model. From [8].

Nevertheless, the results presented a strong argument for Artificial Neural Networks (ANNs). Up to the 2010 challenge, the main focus in computer vision research was the manual engineering of visual feature extractors like edges and colour gradients. These were then given to a simple classifier like Support Vector Machines for the final assignment of a class [9]. Here, the disruptive nature of DL was demonstrated for the first time. An end-to-end system, without hand-crafted features which outperformed the second place by a staggering $\sim 8\,\%$. This had marked the beginning of the wide-scale adaption of ML for day to day applications, such as the in-vehicle active safety systems described in this work.

In the following years, the community witnessed an exponential growth in research and funding, see [10]. The parallel field of Natural Language Processing (NLP) was also quick to adapt with the development of the Word Embeddings [11] and projects like 'WaveNet' [12]. These projects were by themselves significant milestones. Word embeddings enabled, for the first time, the vector representation of tokens, i.e., words, by their semantic meaning instead of the conventional random 'one hot' vectors. WaveNet was the first to synthesize realistic sounding human voice in a Text to Speech (TTS) system.

With the advancements in Recurrent Neural Networkss (RNNs), in projects like the Long Short Term Memory (LSTM) networks [13] or the later Gated Recurrent Units (GRU) [14], it became possible to model conditionally dependent series, e.g., time, opening the door to applications in further fields like finance [15]. Notice the work by Kohzadi et al. from 1996 which also discussed the usage of ANN for a similar purpose and was most certainly ahead of its time.

Meanwhile, the number of challenges and applications increased drastically. From plain classification the focus shifted to localisation, complete scene understanding in the form of semantic segmentation and image captioning. Research in these fields was and still is supported by datasets like the infamous MS COCO [17], Cityscapes [18] and more.

The promise for new opportunities soon transitioned from academia to the industry. Early adopters were mainly start-ups, e.g., Tractable [19], DeepMind [20], etc. However, research affine technology corporations from the likes of Google have also followed, presenting renounced projects from the likes of 'GoogLeNet' [21].

The automotive industry is an additional early adopter. The nature of the industry puts it in a unique position in the field. In addition to using ML to increase revenue, it can also utilise it to save lives and increase equality. Better Mobility on Demand platforms are considered a key feature in a safer, greener and more accessible world.

According to the World Health Organisation (WHO), some ≈ 1.3 million road users have died in accidents in 2018 [22]. Even though the holy grail of the industry is a fully autonomous stack for mobility on demand, there are several intermediate levels of automation which are yet to be reached.

For example, an Automatic Emergency Breaking (AEB) system requires little more than a reliable mid-range classifier for an effective risk assessment. An additional example is driver monitoring systems. Such systems constantly evaluate the state of the driver and alert them upon disengagement or tiredness. They can detect events from texting while driving to impaired driving. The possible benefits are significant to the extend of quickly becoming a regulatory requirement by organisations like the Euro NCAP for current safety ratings [23].

From the plurality of open research topics in this fast-changing industry, this work focuses on advancing the capacities of ML for in-vehicle active safety systems. A diverse and exhaustive task which may not fully depict its true extent at first glimpse.

The main difficulties with production-ready, safety critical systems are the working environment and the performance requirements. Most current works report using high-end GPUs, often in a cluster configuration. These hardware architectures often

Advantages	Disadvantages
Research applicable to a wide variety of domains [25].	Reduced applicability; Only theoretical or cloud applications.
Little domain specific knowledge required.	Exclusion of institutes which cannot afford suitable clouds and clusters [26].
Good results lead to good publicity.	

Tab. 1.1: A tabular comparison of the main pros and cons of large models. One can see that, for many applications, larger is not necessarily better.

cost thousands of Euros and run in a thermal envelop of around 300 Watts [24], i.e., they are energy hungry and exhibit instabilities outside of their relatively limited thermal operating range.

These factors are often an advantage, as larger models tend to perform better, yet there are also disadvantages to consider. The comparison in Table 1.1 covers the main advantages and disadvantages of large, general purpose models.

Additionally, the costs play an important role, including such a hardware architecture in every vehicle would decrease competitiveness in a saturated market, which in turn reduces acceptance. High-energy systems which need to reliably operate in the coldest and warmest parts of the world also increase complexity and respective costs.

One common solution is to use dedicated embedded systems, like the Jetson product family from nVidia [27]. Embedded systems are designed to solve the aforementioned issues, while sacrificing a marginal portion of their computational power. There is, therefore, a strong demand for algorithms which do not only deliver SotA results, but do so with significant computational constraints.

The SotA performance is perhaps the most challenging requirement in the automotive industry. While it is common to read works reporting the likes of "93 % accuracy on dataset x", imagine these missing 7 % mean missing a child by an AEB system.

This work thus sets to mitigate the discrepancy between the academic SotA performance and its respective industrial requirements. It shifts the focus from a true-positive only paradigm to usability in automotive applications.

1.1 Outline

The different projects covered throughout this work are concentrated around the utilisation of world knowledge, i.e., a-priori. These priors allow to specialise a neural architecture to its task while removing excess capacity. The goal-optimised models are than able to better utilise their capacities for the most relevant aspects of their task. A utility which is often accompanied by improved learning performance and/or reduced computational requirements.

The structure of this work follows the just-in-time paradigm. Each chapter is encapsulated as a stand-alone unit. It includes the theoretical background, along with an overview of the relevant literature, an explanation of our contribution, its evaluation and, finally, the respective conclusions.

In Chapter 2, a novel adaptive masking algorithm is discussed. The systematic errors of semantic segmentation algorithms in the image domain are studied to establish that these mostly occur at the objects' edges. We move to propose an adaptive masking scheme, which based on the these edges is used in two ways to optimise performance.

First, runtime optimisation. We show how one could utilise such masks to real-time optimise the popular Conditional Random Field model for segmentation refinement. By coupling the masks with the CRF framework, we demonstrate a classification agreement of over $99\ \%$ with the baseline while reducing the runtime to $\sim 15\ \%$.

Second, classification optimisation. We utilise our adaptive masks to improve on the challenge of vanishing gradients. Since the common loss function for semantic segmentation equally accounts for all pixels, the more important edge pixels vanish under the extensive averaging. Using the edge masks, we re-focus the loss values on the 'harder' pixels, leading to improved generalisation performance.

In Chapter 3, we present adjustments to the vanilla convolution layer which drastically reduce the computational burden. A theoretical and empirical analysis of the exhaustive runtime intervals is provided. The lessons of this study are then used to justify a novel convolution block, dubbed 'EffNet'. The block increases efficiency by splitting the convolution operator to a series of axis-wise convolutions, each is optimised to be well streamlined on the given embedded hardware. We furthermore show that our proposed convolution block outperforms similar architectures like the commonly used 'MobileNet' and even its vanilla baseline.

Subsequentially, Chapter 4 discusses a novel method to exploit the temporal coherence of trajectories, improving the generalisation capacity of motion understanding models. A literature survey showed that trajectories are predicted as a series of mutually independent coordinates. We have furthermore established that such a prediction format encourages artefacts in the predictions and over-fitting on the predicted temporal offsets. These disadvantages are addressed by proposing a new ad-hoc output layer which predicts polynomial coefficients, composing a mapping of distance as a function of time. Since the polynomials offer an added flexibility, we accompany the layer with a complete suite of training optimisations. The framework is finally evaluated to demonstrate how the predicted trajectories are more natural as well as more accurate than the common art coordinates.

Finally, Chapter 5 recapitulates the main observations of this work with a concluding overview of the key contributions.

1.2 Contributions and Publications

The content of this work is based on the following publications

- **Ido Freeman, Jan Siegemund**
  "Device and a method for assigning labels of a plurality of predetermined classes to pixels of an image"
  US Patent US16143741, 2017
  Covered in Chapter 2

- **Ido Freeman, Pascal Colling**
  "Spatio-Focal Loss Adaptive Weighting of Semantic Segmentation Loss"
  European Patent Application EP21154378, 2021
  Covered in Chapter 2

- **Ido Freeman, Lutz Roese-Koerner, Anton Kummert**
  "Effnet: An efficient structure for convolutional neural networks"
  25th IEEE International Conference on Image Processing (ICIP), 2018
  Covered in Chapter 3

- **Ido Freeman, Kun Zhao, Anton Kummert**
  "Polynomial Trajectory Predictions for Improved Learning Performance"
  28th IEEE International Conference on Image Processing (ICIP), 2022
  Covered in Chapter 4

- **Dominic Spata, Arne Grumpe, Mirko Meuter, Ido Freeman**
  "Methods and Systems for Predicting a Trajectory of an Object"
  European Patent Application EP21186073, 2021
  Covered briefly as a derivative work in Chapter 4

- **Kun Zhao, Ido Freeman, Thomas Kurbiel**
  "Method and Computer System for Controlling the Movement of a Host Vehicle"
  UK Patent Application GB2203519.0, 2022
  Covered briefly as a derivative work in Chapter 4

- **Dominic Spata, Arne Grumpe, Ido Freeman**
  "Variance Estimation for Deeptracker"
  European Patent Application EP21188550, 2021
  Covered briefly as a derivative work in Chapter 4

Additionally, the following publications did not align with the silver lining of this work and were therefore not included

- **Alessandro Cennamo, Ido Freeman, Anton Kummert**
  "A Statistical Defense Approach for Detecting Adversarial Examples"
  International Conference on Pattern Recognition and Intelligent Systems, 2020
  I had the pleasure of supervising Alessandro in his research towards his Master's thesis. We revised ideas together, optimised the implementation and iterated on the algorithm to make it more robust and better performing.

- **Ido Freeman, Klaus Friedrichs**
  "Method for classifying a capture taken by a sensor"
  US Patent US11403498, 2022
  In this work, Klaus and I have developed a softmax-based temporal fusion algorithm to improve tracking results of spotlight classification on motorways at night. We used the system to address classification instability at ranges around 1 km where the common headlight is a patch of around 4×4 pixels in size. Our novel fusion algorithm encouraged the classifier to output large confidence values for certain classifications. Incorporated with the exponential nature of the softmax function, the fusion signal was made adaptive to the samples. It combined the benefits of a stable classification with the agility of quick reclassification. As soon as the target object enters the range of certain classification the classifier overrides the fusion and forces its output. The scheme has reduced our false positive rate from $\sim 5\ \%$ to $\sim 1\ \%$ on an internal dataset. From the ten patents covered in this work, this project is the only

one to have been successfully deployed in production. It is currently used to control the headlights of the latest generation of Volvo's lorries.

- **Ido Freeman, Kun Zhao**
  "Temporally Adaptive Attention for Trajectory Prediction"
  European Patent Application EP21150060, 2021
  In this work, I extended the aforementioned fusion algorithm to stabilise the attention in transformer networks. Adapting the domain, we showed that stable attention weights support trajectory stability in motorway scenarios. The model chooses its attention weights in the default case, yet under uncertainty, the temporal fusion becomes dominant and factors in the previously used weights. This implements the sparsity assumption, which states that consecutive frames would only contain a limited amount of new information. Hence the model can benefit from its previous predictions when the current ones are uncertain.

2 Adaptive Masking for Efficiency and Generalisation

Object classification is a term which traditionally follows a simple definition - given an image of an object, to what class of similar objects does it belong? Representations and examples for this definition are easily found in datasets from the likes of Modified National Institute of Standards and Technology (MNIST) [28], ImageNet [3] and Cifar10 [29]. Yet, ultimately, the main issue with applying object classification to the real world is the ill-posed task. Considering real-world scenes with a plurality of objects, which object should be classified? What about the other objects? Furthermore, in some cases, the added context is crucial for a correct classification. For example, is the snorkelling woman really snorkelling or is it merely a holiday photo hanging on the wall?

As a solution, a focus shift from object classification to object detection and semantic segmentation has taken place, with datasets like the Visual Object Classes (VOC) 2007 challenge [30] and Common Objects in Context (COCO) [17] representing some of the earliest examples. The once direct mapping of one input to one class, has evolved to one to many, thus also requiring substantially more computational power. This increased intensity poses a challenge for embedded systems, already pushed to their limits by simpler, traditional classifiers. Nevertheless, a challenge is nothing but an opportunity for novelty, some of which is presented in this chapter.

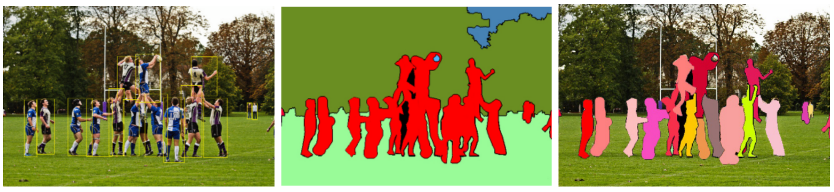
This chapter covers two projects, consisting of the same idea applied to two different scopes and purposes. At the core of our proposed algorithms is the recognition that for the common semantic segmentation algorithm, the most challenging areas in an image are along the objects' borders. We propose to improve the performance of such algorithms by artificially increasing the significance of these areas, using an adaptive, sample-wise edge mask. We first show how such an edge map could be utilised to significantly improve the runtime of the common Convolutional Neural Network-Conditional Random Field model by only applying the CRF to the interesting parts of the image. This work is covered in [31]. In [32], we later shifted the scope to generalisation and segmentation performance and show how such a dynamic mask could not only be beneficial on embedded systems, it could rather also assist reducing the size of a model without loosing performance.

The rest of the chapter is structured as follows. A joint related work section covers the topics relevant for both projects. It is then followed by the discussion and evaluation of the first method for efficient CRFs. Finally, the chapter is concluded with the second method for adaptive weighting of loss functions for semantic segmentation models.

2.1 Related Work

As hinted in the introduction for this chapter, semantic segmentation is the task of assigning one of a plurality of classes to each pixel in an input signal. The task resembles object detection, where each recognised object in the input is assigned a bounding box and a class. It is also regarded as the preceding generation of instance segmentation, where not only the given classes are assigned to each pixel, rather also an instance index.

For example, let us regard a large group of athletes on the field. In object detection, one can expect an overlapping set of bounding boxes, each represents a single person. In semantic segmentation, the group will be assigned a single, large blob of the class 'person'. Finally, instance segmentation adds unique pixel-wise assignment to represent each individual person. An example of all three cases is given in Figure 2.1.


(a) Object detection - each instance gets a bounding box and a class assignment

(b) Semantic segmentation - instances disregarded, only classes are assigned

(c) Instance segmentation - each instance is classified and segmented separately

Fig. 2.1: An example illustrating the differences between object detection, semantic segmentation and instance segmentation. From [33]

As the field addresses an extremely relevant task, it has seen a relatively large number of datasets over the years. Among the most renounced benchmarks is the Cityscapes dataset [18] which covers the open street domain, mostly of German cities. Similar, yet slightly more versatile datasets are the 2012 version of the VOC challenge [34] and the ADE20k [35] datasets. However, applications are not limited to day-to-day objects. Other interesting domains include biological segmentation of cells [36], [37], analysis of aerial images [38], [39] and even diagnostic of surgical endoscopy tools [40].

The large number of datasets provides a sandbox for experiments and developments with a persistent competition for the best model.

2.1.1 Object Detection

First, consider the trivial evolution of the one image one object paradigm, the one image multiple objects task. From an engineering point of view, bounding boxes are often preferred to pixel-wise segmentation as they deliver a very similar amount of information with only a few bytes of data. This is opposed to the more tedious pixel-wise information which often requires additional processing steps.

For example, for representing only the bounding box's two-dimensional location and class, one would need two anchor points, e.g., the upper left corner and the lower right corner of the box, and a class representation. Another advantage is that these bounding boxes could easily be extended to include further signals, which are not easily predicted on a pixel-wise resolution. These include, but not limited to, heading angle, pitch and even estimations of depth.

The first publication in the field is the Region CNN paper [41] in which the usage of a CNN classifier on pre-extracted bounding box candidates was proposed. The successor of this work, the Faster RCNN model [42], combined the region proposal and classification modules to allow for a quicker and better converging end to end training. It was then followed by Faster RCNN [43] which incrementally refines the pipeline to demonstrate even better detection rates at a lower runtime.

Finally, the YOLO models family [44], [45] and more. The YOLO strategy is to have a single model which processes the entire input at once and generates a bounding box estimation for all possible locations. It then uses a non-maximum suppression to eliminate the improbable boxes, resulting in only the most reliable predictions. Its simplicity has made it one of the more commonly used models and backbone architectures for object detection to date.

However, it is important to remember, that bounding boxes are not always the best solution. Classes like 'road', 'sky' or even 'building' are for most use cases pointless as bounding boxes.

2.1.2 Semantic Segmentation

Tasks which require more than bounding boxes are addressed by semantic segmentation. Before Deep Neural Networks (DNNs) became strong competitors in vision tasks, semantic segmentation was traditionally addressed by CRFs. First presented in the 2001 work of Lafferty et al., these are a form of a graphical model which models the interactions between nodes, in the visual case - pixels, superpixels (e.g., [47]) or regions, along weighted edges [46]. A more in depth discussion about CRFs and their properties is provided in Subsection 2.2.3. The extension to actual pixels was later proposed by Kraehenbuehl and Koltun which reformulated the inference process, making it efficient enough to run on a large number of nodes [48]. As a reference for the claim "efficient enough", Fig. 1 of [48] reports an inference time of 0.2 seconds of their approach while the reference SotA method of [49] took some 36 hours to deliver comparable segmentation results.

One of the earlier CNN based attempts at semantic segmentation was [50]. It was the first work to show that the fully connected layers at the top of a CNN architecture could be replaced by their convolution counterparts, delivering a low resolution pixel-wise classification map. At this point, the final results mostly suffered from the unlearned up-scaling, leading to errors along the transitions between objects. The work of Chen et al. then took the said CRF and partnered it with the aforementioned CNN classifier for the unary predictions [51]. It was later revised in [52], mostly by optimising the scope of the convolution layers. Finally, in [53] the CRF algorithm was realised as an RNN which allowed an end-to-end training along with a simpler implementation.

At roughly the same time, the 'U-Net' model was proposed [54]. It demonstrated an adaptation of the autoencoder framework [55, pp.499-523] such that a fully convolutional encoder-decoder architecture could be used in the domain of biological cell segmentation. The advantage of the encoder-decoder architecture is that it completely compresses the input signal, thus maximising the context. The result is then up-sampled by the decoder to reconstruct the higher resolution. For further details and references, we strongly recommend reading through the work of Arnab et al., which in their 2018 overview discussed the latest advancements in the field [56].

The latest SotA class of models for semantic segmentation evolves around hierarchical multi-scale attention architecture [57]. It generates segmentation maps at different scales along with a vector of weights to merge the prediction. The idea implements the divide and conquer strategy known as pyramid processing [58].

By having separated model branches for the different scales, the variance in object properties is mitigated.

Semantic segmentation offers a good solution for tasks like free space detection, background extraction, etc. Yet it suffers from one significant drawback. Assigning each pixel a class, one misses the differentiation between instances of the same class. While being irrelevant in free space detection, being able to differently handle animate objects, e.g., pedestrians, is a desirable property.

2.1.3 Instance Segmentation

The next phase in evolution is segmenting an input image to only to its semantic classes, but rather also marking the individual instances of these classes. The most renounced extension to the Region CNN framework was the Mask-RCNN described in [59]. By extending the Faster RCNN model [42], He et al. did not only predict bounding boxes, they have also segmented the content of the bounding box. The model combines a fully convolutional head for the segmentation with a pixel-wise classification head for the bounding boxes.

The current state of the art is held the work of Ghiasi et al. which combined ResNet 50 with Mask-RCNN and extended it with a robust data augmentation scheme [60]. The main contribution of this work is the usage of the ground truth instance labels to randomly copy objects into different images. The augmented data adds a significant amount of virtual training samples which are required to train such large models.

2.2 Motivation

The algorithm proposed in this chapter was motivated by the disadvantages of its prior art. This section is opened with a very quick recapitulation of CNNs for classification and segmentation. We will see that, regardless of their impressive performance in vision tasks, this class of models embed an inherit design flaw, limiting their spatial scope. This important property of CNNs is then used to segue to CRFs. This class of models was developed with the sole purpose of accounting for spatial information dependencies, making them an ideal compliment to CNNs.

2.2.1 Convolutional Neural Networks

A convolutional neural network is the spatially adjusted version of the perceptron algorithm. Conceptualised in the 1943 publication of [61], it described a learnable, computational unit which mimics biological neurons. Some 14 years later, it was finally described in algorithmic terms in [62]. The core idea behind the perceptron is rather simple. Following the notation of [63, p. 227], it takes the following form

$$y = \sum_{j=0}^{J} \theta_j x_j + b. \tag{2.1}$$

Here, $\theta \in \mathbb{R}^J$ is the perceptron's trainable parameter which maps the input vector $x \in \mathbb{R}^J$, of dimension J to the output y. The bias term, b, was later added to extend the line equation for uncentred, i.e., unnormalised, data. The idea is that such a line could be used to divide data into class a, above the line, and class b, below the line. According to the hyperplane separation theorem, such a clear separation should exist when the dimensionality is high enough [64, pp.46-51].

The perceptron algorithm, along its generations, has proven a powerful classification algorithm, yet it exhibits a built in limitation when it comes to classifying natural images. While the perceptron is implemented with a trainable weight per input dimension, natural images are both large as well as translation invariant. This means that any given object could be seen anywhere in a given image, without changing its properties or its class. This matter was the main focus of LeCun et al. who have, in 1989 described a convolution based model for postcode recognition in letters [65]. They used small spatial kernels, often of size 3×3, which are swiped over the larger input image, thus applying the same weights to all positions of the input. By doing so, the number of parameters was reduced while also addressing the translation invariance issue.

This architecture design was heavily motivated by the receptive fields alignment in the mammalian visual cortex. As early as 1962, Hubel and Wiesel showed that the receptive fields in mammals implement a pipeline of increased complexity. Such a processing chain is illustrated in Figure 2.2. The computational aspects of convolution are discussed in further detail in Subsection 3.2.1.

Almost ten more year were required to get from optical digit recognition to the more general document recognition in [68]. This is mostly related to the slow rates of data transfer in the 90's as well as the limited computational capacity. The ability to efficiently process images in the wild arrived to market around 2012 with the

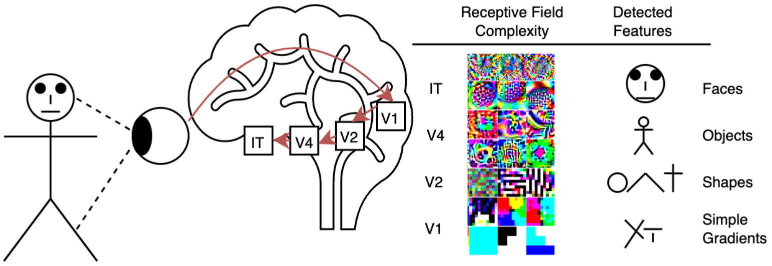


Fig. 2.2: An illustration of the processing pipeline in mammal brains following [66]. Notice the increasing complexity with each layer of processing, this motivated the design choices behind the commonly used CNN. The visualisation of the receptive fields is borrowed from [67].

publication of [3]. It initiated a revolution in image processing and brought the end of the so called AI Winter.

To further demonstrate the importance of computational power for the capacity of neural models, [69] plotted the, to-date, most popular models from the ImageNet challenge leaders board as a function of their Floating Point Operations (FLOPs). The near perfect correlation seen in the figure suggests an interesting observation. Many of the latest developments in classification networks might rely more on advancements in GPU technology, rather than some novel features or layers.

In order to increase the capacity and scope of CNNs, the said kernels are arranged in groups, called layers. These layers are then extended by two other operators which are applied on top of them. First, a non-linearity function allows neural networks to successfully approximate any arbitrary function, as explained and well visualised in [70, ch. 4]. The most common non-linear function to date is the so called Rectifying Linear Unit (ReLU) [71]. Second, a pooling, also subsampling, is used to artificially increase the scale on each layer of kernels. The pooling layer reduces the signal's size, mostly by picking the maximal valued element from a 2×2 neighbourhood. By halving the signal's size, along each of its spatial dimensions, the scope of the consecutive kernels is effectively increased. The benefits of pooling were suggested as early as 1995 in [5].

Evidentially, convolution layers have proven a strong non-linear feature extractor. However, the final step of classifying the said feature required an extension. The established modus operandi used to be a series of convolution layers, followed by a final perceptron layer for the final classification. This last layer, also called dense layer or fully connected layer, would then have the same number of perceptrons, i.e., neurons or units, as the number of possible classes. The most active neuron, i.e., the one with the largest value, represents the assigned class.

As the research community started demonstrating progress with object classification, the question about the applicability of classification arose. The underlying question asked in a classification framework is "which object is seen in this image?". This is clearly a highly distilled version of the actual, real-world question "what is seen this given scene?". I.e., the extensive scope of scene understanding is reduced to a single object in an isolated environment, in contrast to the natural world which exists in context.

As a side-note it is worth mentioning that works like [72], [73] have repeatedly shown how the background is in many cases just enough to support the classification, occasionally to the extend that the model "cheats" and uses solely the background for classification, see Figure 5 from [72], shown here in Figure 2.3.

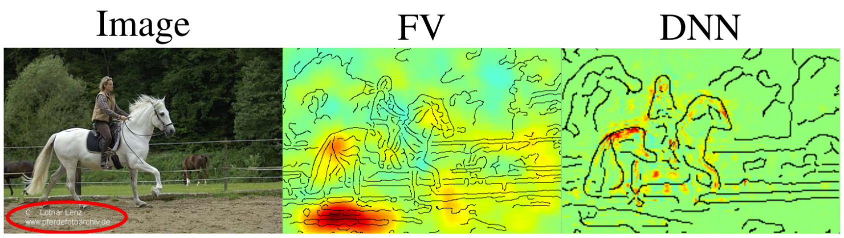


Fig. 2.3: A visualisation of the main attention areas for an image, correctly classified as a horse. Middle: a Fisher vector based classification [74]. Right: a deep neural network. Both classifiers have similar accuracy for the class horse. While the deep neural network focuses at features like the outline of the horse, the Fisher vectors solution learnt that all horse images in the dataset have a copyright text. Figure from [72].

Mutual Independence in CNNs

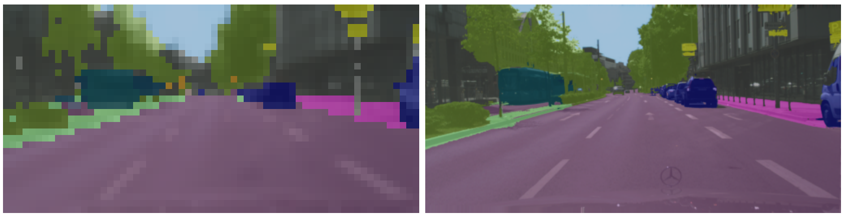
It wasn't until 2014 that a fully convolutional neural network demonstrated SotA results in a vision task [50]. By using a fully convolutional model, Long et al. resolved the constant input size limitation while producing a pixel-wise heat-map over the possible classes. The heat-map is then super-sampled to the final semantic map.

The fully convolutional setup showed impressive results on datasets as PASCAL VOC [30] and NYUD [75]. Yet it suffered from the limitations of scope, context and resolution.

Scope and Context Convolution filters are, practically by definition, significantly smaller than the input image they process. This poses a limitation on the amount of context they see in a patch as objects are often times larger than the filters. When a network is only based on subsampling, e.g., pooling or strided convolutions (see Subsection 3.3.2), it is bound to have its output in a smaller resolution than its inputs. In order to increase the context to the kernels, more subsampling could be used. This improves the information exchange between different patches at the cost of reducing the output's resolution. Notice that this issue is now regarded as solved with the common encoder-decoder architectures for vision, e.g., U-Net [54].

Output Resolution In the last paragraph, the trade-off between context and resolution was discussed. While the lower resolution output has a clear separation of classes, it demonstrates an unpleasantly low resolution. Nevertheless, although predicting at a higher resolution is visually more appealing, it exhibits awkward discontinuities and more erroneous pixels.

Figure 2.4 provides an example of the Cityscapes dataset [18] and the Fully Convolutional Network (FCN) model, described in [50]. One could see that the lower resolution prediction, 'FCN-32s' Figure 2.4a, is very pixelated yet generally accurate. At the same time, the prediction at the full resolution, 'FCN-8s' Figure 2.4b, contains more details. Unfortunately, these details come at the cost of additional compute, almost a factor of two comparing to Figure 2.4a. This additional compute was not widely available when it was first published, forcing the community to develop various tricks, i.e., incorporate different biases, for making the inference more efficient. One of these biases is the sparsity assumption, which is covered in the following section.



(a) Semantic segmentation at the smallest resolution of the model of [50]
(b) Semantic segmentation at the full input resolution

Fig. 2.4: A visualisation of semantic segmentation at different resolutions, coarse to fine. Notice how the trade-off goes from poor resolution on the left to more erroneous pixels on the right.

2.2.2 The Sparsity Assumption

Real-world images are commonly described by n rows and m columns of pixels (px), with both n and m being natural numbers, i.e., $m, n \in \mathbb{N}$. Common sensor sizes to-date range from around 10 megapixels to roughly 40 megapixels. This means that most modern images consist of $m \times n \approx 10^7$ px.

Without prior knowledge, the upper boundary for the number of different elements follows a simple combinatorics definition. Assuming a mutual independence of pixel content, an image with 10^7 pixels could theoretically show more objects than most modern compute architectures could process in real-time. That is, objects might be as little as 1 px in size and are allowed to appear more than once while not effecting each others probability of occurrence.

In order to make this practically infinite number of possibilities more manageable, the sparsity assumption in natural images introduces several assisting priors. First, objects are larger than a single pixel. Objects at the pixel scale are mostly irrelevant for most applications. Even when they do happen to appear in an image, e.g., an insect flying through the frame or a very distant object of interest, they do not contain enough visual cues for correct classification. Subsequently, this means that the actual amount of objects $k \in \mathbb{N}$ in an image is significantly smaller than its number of pixels, i.e., $k \ll m \times n$. Second, objects are mostly clustered into instances where each instance tends to exhibit unified characteristics. I.e., objects and instances are made of a very few materials with little variation in colours, viscosity, reflectively and other properties.

Figure 2.5 provides a general visualisation of this argument. It shows two images from the Cityscapes dataset [18] which were segmented into 20 superpixels using the SLIC superpixel algorithm [47]. The number of superpixels was chosen to match the number of classes in the dataset. These superpixels, also completely lacking understanding of the scenes and are only being based on the red, green and blue (RGB) values. Yet these intensities are already enough to demonstrate a very rough concept of semantics. This shows the power of the sparsity assumption, if neighbouring pixels have similar values, they probably belong to the same semantic class.

Understanding the limitations of CNNs and the classification framework, combined with the promising possibilities which emerge from the sparsity assumption, the CRF algorithm could now be explained in detail.

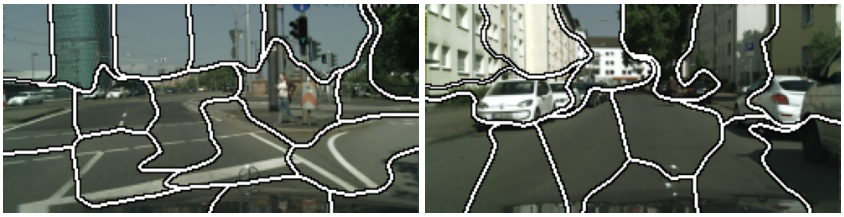


Fig. 2.5: Two randomly selected images from the Cityscapes dataset [18], segmented by the sparsity based SLIC superpixel algorithm [47]. The very rough semantic properties of the superpixels demonstrate the strength of the sparsity assumption in natural images.

2.2.3 Conditional Random Field (CRF)

CRFs [46] predate much of the hype around CNNs and Neural Networks (NNs) as a whole. They were originally described by Lafferty et al. in 2001 as a further development of the more established Hidden Markov Model [63, pp. 610-635].

While reliably classifying isolated objects is an all-but-trivial task, its applicability to the real-world is questionable. Real-world images provide a contextual frame in which multiple objects reside. For many applications, the focus on a single element is counterproductive and even life-threatening. Imagine an autonomous vehicle which can only focus on a single object at a time.

Utilising the prior knowledge of real-world objects, more efficient algorithms can be implemented. Such an algorithm is the CRF. It looks at an image as a connected graph, often fully connected [48], and filters unary, pixel-wise, classification results to refine the label assignment. Messages between nodes, e.g., pixels, superpixels, etc., are passed based on their spatial proximity and colour similarity. These two factors are the most common ones while the CRF framework does not dictate a specific logic to use. For example, my unpublished Master's thesis predicted the weights for the message passing using a CNN [76]. In hindsight, the algorithm resembles the more modern visual transformers architecture (ViT) [77].

The said Master's thesis is also used for the notation in the following equations.

Following the annotation of [48] to the work introduced in [46], a Conditional Random Field is defined as

$$p(y|x,\theta) \propto \exp\Big(\sum_{v\in\mathcal{V},k\in K} \psi_k(x_v,y_v) + \sum_{e\in\mathcal{E},l\in L} \varphi_l(x_{e0},x_{e1})\Big). \tag{2.2}$$

Aligned with the common graph annotation, $\mathcal{V}$ represents a set of nodes which are connected by edges $\mathcal{E}$. Each $v \in \mathcal{V}$ is processed by $k \in K$ functions while $l \in L$ functions are applied each $e \in \mathcal{E}$.

$\psi_k(\cdot)$ stands for the unary potential function k. Analogous is $\varphi_l(\cdot)$ which stands for the pairwise potential function l. Both of these terms are explained in the following subsections.

Unary Potentials

The unary potentials term represents the functions which operate on the node level, here - the very pixels. They are neither predefined nor limited by the framework as different domains have different node definitions and requirements. The number of functions, $|K|$, is an additional hyper-parameter of the model.

The formal definition for a unary function ψ is

$$\psi_k = \alpha_k f_k(\cdot). \tag{2.3}$$

Where α is a learned scalar which is mostly a real number, i.e., $\alpha \in \mathbb{R}$. The function $f(\cdot)$ is developed to give the unary potentials.

An example for such a function from a different domain is found in [46]. In this paper, the field of Natural Language Processing is considered and several unary feature functions are proposed. One of which is a binary function which returns True for an upper-cased token (word) and false otherwise. In this example, a node represents a word-token, a trend which precedes the currently common use of character tokens which are better suitable for dialects, spelling errors and slang [78].

In the vision domain, a unary function for semantic segmentation could be defined to take in an image and output pixel-wise class predictions. A very familiar type of function to satisfy this definition is, for example, a pixel-level CNN as the one proposed in [50].

Pairwise Potentials

The interaction between graph nodes happens on the basis of connections or, in graph jargon, edges.

The formal definition for a pairwise function is

$$\varphi_l = \beta_l g_l(\cdot). \tag{2.4}$$

Here as well, β is a trainable scalar.

The edges are freely defined to model a desired property and could be as simple as the Potts model [79]

$$g_l(y_i, y_j) = \begin{cases} 0 & \text{if } y_i \neq y_j \\ 1 & \text{otherwise} \end{cases} \tag{2.5}$$

which only allows communication between nodes which have the same predicted label y.

The bilateral filter was used in [80] for filtering the pairwise potentials. The filter smooths the signals of two pixels by evaluating two Gaussian kernels. One for the spatial coordinates and another for the intensities, i.e., the colour values. The closer the pixels are together and the closer their intensities values are, the stronger the signal smoothing becomes.

One of the most interesting functions here was proposed in [48]

$$g_l(y_i, y_j, X) = -l(p_i^{(l)}, p_j^{(l)})(y_i - y_j)^2. \tag{2.6}$$

With $p_i^{(l)}$ being the input vector for the function l at the index i. They then further defined l as a Gaussian kernel. The advantage is the differentiable form which allows for a fully connected graph to be trained using the back propagation algorithm [65]. This definition meant that for the first time, CRFs were used as fully connected graphs, as all reference indices i are set to be connected with all target indices j.

The Splat, Blur and Slice Algorithm

In 2011 an extension was proposed in [48] to significantly improve the computational complexity by using the splat, filter, slice algorithm of [81]. As suggested by its name, the algorithm comprises of three plus one steps, the splat, blue and slice steps, preceded by an initialisation step. The algorithm, which is discussed in details in the following paragraphs, is also depicted in Figure 2.6.

Initialisation This step, which does not officially count towards the runtime calculation, is based on the input signal and is thus required for each input once, regardless of the number of inference iterations. Following [81], the bilateral filter W for the image I at pixels i, j as defined in [82] is defined as

$$W(I) = \frac{1}{K_i} \exp\left[-\frac{|i-j|^2}{\sigma_s^2}\right] \exp\left[-\frac{|I_i - I_j|^2}{\sigma_I^2}\right]. \tag{2.7}$$

Here, K_i is a normalisation constant, I represents the pixel intensities. The elements in I are indexed by the spatial coordinates i and j. σ_s and σ_I are additional hyperparameters, representing the standard deviation for the spatial domain and the intensities domain, i.e., colour space, respectively. The bilateral filter helps us define a regular grid, which is in this case based on the RGB intensities along the spatial coordinates. Since the grid is spanned by constant n-dimensional increments, it was named permutodehral lattice. For initialising the lattice, one then follows the same definition which translates to

$$L_i = \left[\frac{r_i}{\sigma_s}, \frac{c_i}{\sigma_s}, \frac{I_{r_i}}{\sigma_I}, \frac{I_{g_i}}{\sigma_I}, \frac{I_{b_i}}{\sigma_I}\right]. \tag{2.8}$$

With L_i representing the five-dimensional lattice at the pixel index i. Its rows and columns coordinates are r_i and c_i, respectively. Its RGB intensities are I_{r_i}, I_{g_i} and I_{b_i}.

Notice the pixel-independent one-to-one mapping which corresponds to an $\mathcal{O}(N)$ complexity.

Splat Having the lattice, the next step is to assign all data points to their place on this regular grid. This step is called ‘splat’ and is demonstrated in Figure 2.6a. It is achieved by calculating the barycentric coordinates of each pixel with respect to (w.r.t) the lattice’s grid, practically projecting each pixel to its corresponding lattice vertices. This step is depicted using the arrows in the said figure. The weight

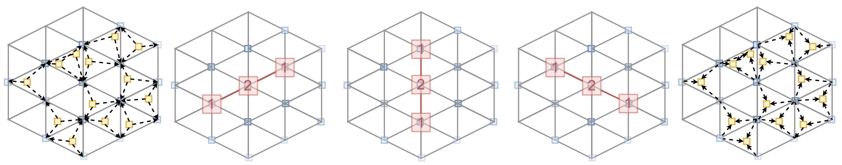

(a) Splat (b) Blur (c) Slice

Fig. 2.6: A visualisation of the Splat, Blur, Slice algorithm, based on [81]. First, project the data points, here in yellow, to the regular grid, the lattice, according to their barycentric coordinates. Then using axis-wise convolution kernels, here in red, blur the signals along each of the lattice's dimensions. Notice that the actual kernel should be normalised. Finally, collect the signals back to their original position.

of each arrow is proportional to its inverted length. The shorter the arrow is, the larger portion its assignment gets from the total data point. Since the barycentric projection should not change the data, its accumulated weight is one. The mapping from the image space and the lattice space is achieved by a bidirectional hash map which enables both the splatting and the slicing steps. This projection step is also done in a pixel-wise manner, meaning another $\mathcal{O}(N)$ steps.

Blur With the image fully projected to the permutohedral lattice, the message passing is achieved by a simple convolution with a normalised Gaussian blurring kernel $[\frac{1}{4}, \frac{1}{2}, \frac{1}{4}]$, depicted here in Figure 2.6b. The kernel is calculated separately along each of the axes, resulting in a similar computational benefit as discussed in Subsection 3.3.4. Put together, each vertex affects 3^d neighbouring vertices with d being the dimensionality of the lattice's grid, i.e., 5. As previously discussed, the convolution itself is computed in an $\mathcal{O}(N)$ complexity.

Slice The final step involves the collection of the filtered lattice information back to the image domain. It is done by accumulating the vertex values by their respective barycentric weights, saved in the hash map, into their original image space positions. This step is depicted in Figure 2.6c. This is a trivial pixel-wise operation which is also achieved in the linear complexity $\mathcal{O}(N)$.

The work discussed in [83] has put both model architectures covered in this chapter together to show SotA results in semantic segmentation using a CNN with an additional CRF for refining of its outputs. In 2015 an additional reformulation was proposed in [53] which implemented the CRF framework as an RNN, making it jointly trainable with the unary classifier in an end to end fashion.

An example for the performance of a CRF is shown in Figure 2.7. The figure shows the outputs of the unary classifier, i.e., the CNN, along three different CRF inference steps, after the third, sixth and twelfth iterations. From segmentation step to the other, one can see interesting developments. First and foremost, the improved resolution which results from the up-sampling. One can also see that the prior inference assumptions of the CRF allow it to defy the ground truth labels. While the official label of the road pixels visible through the bicycle's wheel is 'two wheeler', the CRF utilises proximity and colour information to properly assign these pixels the labels 'road'. Furthermore, the improved details of the tree on the upper right corner are also clearly visible.

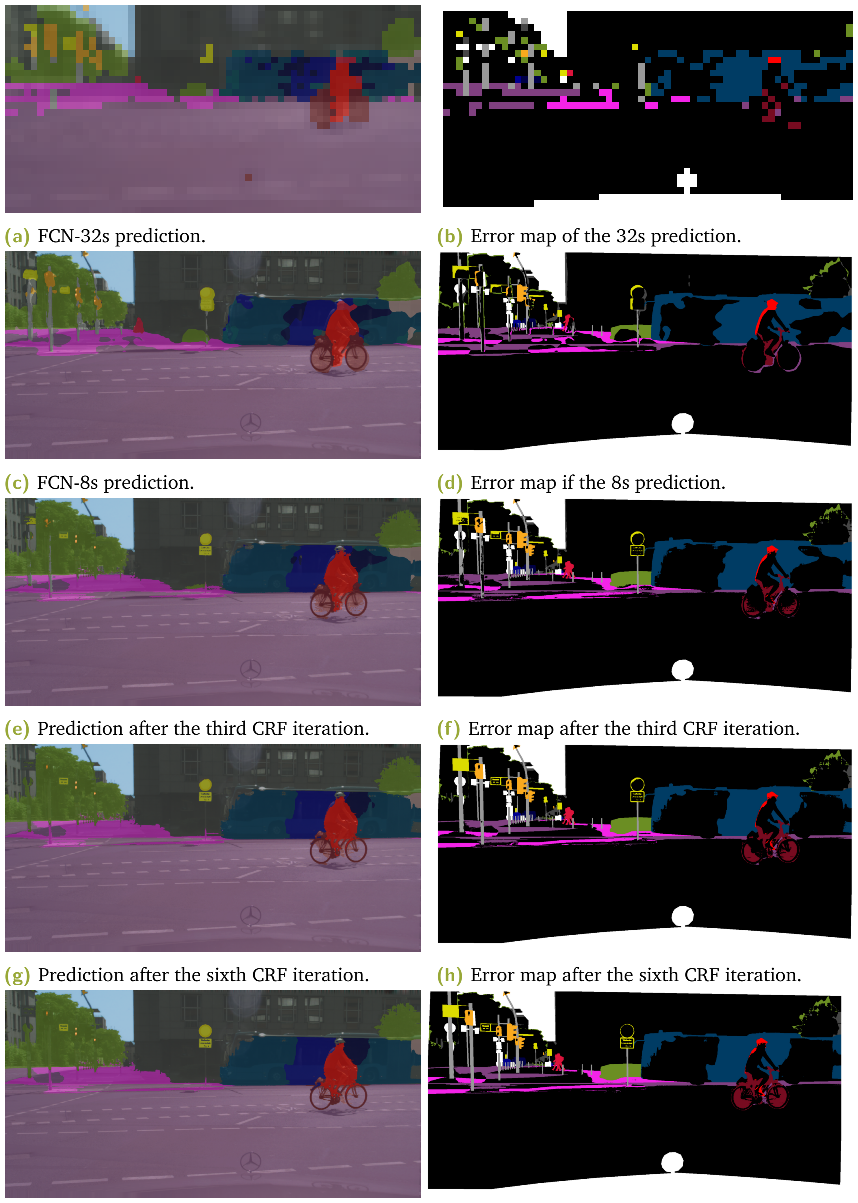
On the other hand, the traffic sign at the centre of the image and the traffic lights on the left present interesting edge cases. While the sign label is well propagated to the entire circle, the sign-conditioning text, just below the main sign, is assigned the class 'building' for its RGB information better matches the building behind it. The traffic lights are almost entirely assimilated in their background since they are completely surrounded by tress.

These symptoms show that while CRFs are well suitable for introducing structure to semantic segmentation maps, they also come with drawbacks. Indeed, the presented results might significantly improve by manually fine-tuning the different hyper parameters of the CRF, yet, as shown here, the benefits and the drawbacks are two sides of the same coin, making a global optimum infeasible.

The bottom line is that the benefits of CRFs outweigh their drawbacks. This was repeatedly demonstrated by multiple works from the likes of [56] and [48]. Yet when it comes to showing the results, most works present figures with significant colour differences between the different objects, thus practically cherry-picking their results.

2.3 Interactive Free Space Detection

The developments covered in the last section have lead to a system which does not only integrate CNNs and CRFs to a single, end-to-end trainable framework, but also reduces the running time complexity. However, executing such models on modern Central Processing Units (CPUs) and GPUs is significantly simpler than having them run on a small embedded hardware in interactive times and a low thermal envelope.



(a) FCN-32s prediction.
(b) Error map of the 32s prediction.
(c) FCN-8s prediction.
(d) Error map if the 8s prediction.
(e) Prediction after the third CRF iteration.
(f) Error map after the third CRF iteration.
(g) Prediction after the sixth CRF iteration.
(h) Error map after the sixth CRF iteration.
(i) Prediction after the twelfth CRF iteration.
(j) Error map after the twelfth CRF iteration.

Fig. 2.7: An example of the different performance of the segmentation stages. The left column shows the segmentation results. The right column shows a coloured error map where all colour pixels represent segmentation error, black pixels are correctly segmented and white are the ignore classes. From the lower resolution 'FCN-32s' layer through the higher resolution 'FCN-8s' layer to the 12 CRF iteration. Notice how the CRF adds structure, traffic sign in the middle, pedestrians and the bicycle's wheels. It also, sometimes, marginalises away desired classes like the traffic lights. Although the CRF actually assigns the correct class to the area between the bicycle's wheels, the ground truth annotations are not as precise, leading to an activated error map starting the third CRF iteration.

In this section, we propose a novel way to enable the application of a CNN-CRF model to runtime critical systems by significantly reducing the inference time of the CRF. The core idea is to strictly limit the scope of the CRF to the boundaries of its target objects. We show that these boundaries are more prone to errors than other inner-object regions of the image. By adaptively focusing the filtering on the areas around the edges, the algorithm reaches a segmentation agreement of $99.61\ \%$ with its naive baseline, while requiring only $21.37\ \%$ of its runtime, on average. The work was published in [31].

The rest of this section is structured as follows. We start with a quick overview of the problem and its definition. Then, the algorithm itself is described and its performance is demonstrated. Finally, we discuss the method's drawbacks.

2.3.1 Runtime Issues and Problem Definition

The confinement to interactive inference times has diverted the scope of many ADASs algorithms away from semantic segmentation towards the coarser object detection. Still, detecting objects which span over large portions of the image, while possibly also enclosing further objects, is not optimally tackled by bounding box detection. Example of such classes are tunnels, sky and road. Since the latter class is of significant relevance in the field of ADAS, it has its own task name, free space detection. Free space detection is a binary segmentation task which aims to assign each pixel in an input frame either the class drivable or not drivable. Man could often further refine the definition by only looking for the pixels of the class 'road'. It is an important signal for applications such as path planning, obstacle avoidance and more.

Yet, while the strategy for highly efficient object detection is often focusing on smaller Regions of Interest (ROIs) and working in patches, e.g., [84], a global task like semantic segmentation has to see the 'bigger picture', i.e., the entire frame or at the very least its majority.

Note that patch-wise processing, also known as tiling, is an established go-to solution for handling exhaustive algorithms. Even with modern computing capacities, this strategy is still found in models like the 'Vision Transformer' (ViT), described in [77].

Another simple solution for increasing efficiency is reducing the input size. As explained in Subsection 3.2.2, the axis-wise runtime complexity of convolution is $\mathcal{O}(N)$. I.e., the computational effort is linearly dependent on the input's size. Hence,

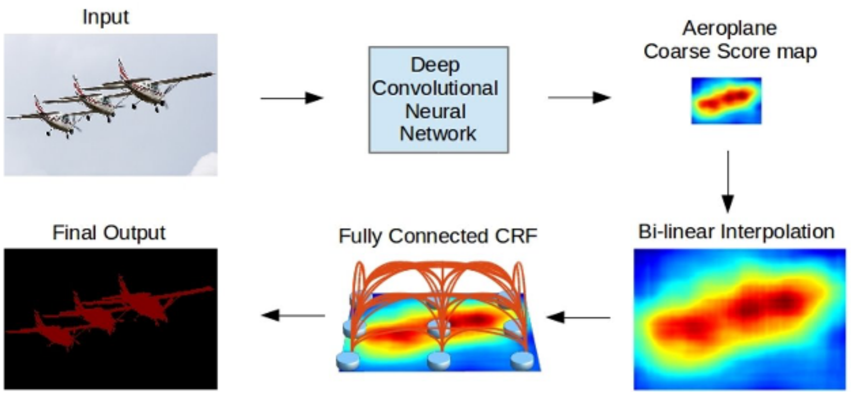


Fig. 2.8: The CNN-CRF model for semantic segmentation of [83].

subsampling the spatial axes by a factor of two, would reduce the computational effort by $75\ \%$. Unfortunately, the reduction would also result in a lower output resolution which is already down-sampled by the pooling operators of the model. A good example for such a lower resolution segmentation map is given in Figure 2.7a.

2.3.2 Efficient Inference in Fully Connected CRFs

Following the solution of Chen et al. in [83], depicted here in Figure 2.8, one could benefit from both worlds by reducing the input's size and then using a CRF to refine the lower resolution segmentation. Yet, the inference time of CRFs is still far from real-time. Even when following the splat, filter and slice scheme of [48], as described in subsubsection 2.2.3, which is multiple orders of magnitude more efficient than the prior art.

In Subsection 2.2.3 the splat, blur, slice algorithm was shown to run in linear time. Yet, it was also shown to consist of many different steps, each iterating over all pixels while evaluating tricky functions such as the exponential or the square root on varied values. The term 'tricky' is used here since these functions are hard to optimise, meaning that their evaluation takes a long time to compute, in spite of their linear complexity.

Evidently, Table 2.1 presents the profiling information of an SSE optimised CRF implementation on a 240×320 px input. While the runtime of ~ 74 ms might seem reasonable, it corresponds to 13.58 Frames per Second (FPS) for the rather small

	Time (ms)	Ratio
Initialisation	23.31	0.32
Splat	24.91	0.34
Blur	3.24	0.04
Slice	22.20	0.30
Total	73.66	1.00

Tab. 2.1: A function-wise runtime profiling of ten CRF iterations on a 240×320 px sized input. Notice that the implementation was optimised and evaluated on an Intel CPU which support SSE. Furthermore, the initialisation step is calculated only once per a given input, while the other steps are accumulated.

input size. Extrapolating to Cityscapes' 1024×2048 px input sizes [18] results in ~ 0.5 FPS.

2.3.3 Investigating Segmentation Errors

The improvement proposed to the aforementioned runtime complexity of the CRF model originates from a deeper reflection on the nature of the task at hand.

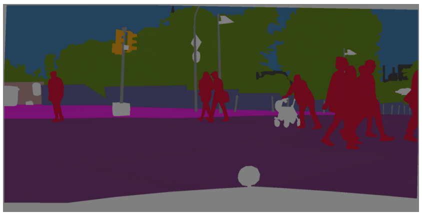
To better understand systematic weaknesses of a classification system, it is often helpful to visually inspect its inputs and outputs. We thus refer to the results of [51] and at Figure 2.7. The visually appealing segmentation maps, which some five years later still make for a common baseline, show a significant improvement in performance by the CRF. However, possibly even more important is the observation that most fail cases, which are improved on by the CRF, are located at the boundaries between objects.

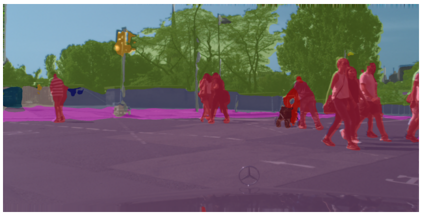
The sparsity assumption in natural images, as discussed in Subsection 2.2.2, states that an image normally consists of a proportionally small number of objects. This, in turn, implies that the number of border and edge pixels is significantly smaller than the number of pixels in the image. Accordingly, counting all the edge pixels of all objects in the Cityscapes validation set [18], the edge pixels make for $12.60\ \%$ of all pixels. An edge pixel is regarded as such if either the label to its right or the label above it are different than its own label.

Focusing on the 'road' class and on Cityscapes as a reference dataset [18], one could see how significant this assumption is. It is a large, sparse and contiguous blob which is often distinguishable by colour from the rest of the scene. These properties make it an especially favourable candidate for CRF refinement.

This typical behaviour is depicted in Figure 2.9 which shows the classification errors of an encoder-decoder CNN model, 'FCN8' in this case [50]. The segmentation errors mostly appear around the borders of the class blobs, shown as the fine lines surrounding the objects in Figure 2.9d. For clarity, these binary errors are coloured using the corresponding ground truth map.

(a) RGB


(b) Ground truth


(c) Prediction

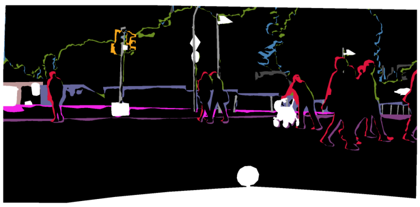

(d) Binary error map, coloured using the ground truth labels.

Fig. 2.9: A visualisation of the typical errors seen in semantic segmentation models. Notice how the mis-classifications, all not black pixels in (d), are concentrated around the borders while the inner parts are mostly well segmented. Visualisation created using the 'FCN8' model, trained on the Cityscapes dataset [18].

2.3.4 A Novel Adaptive Filtering Mask

Extending on the edge-sparsity, an adaptive filtering mask could be imagined to focus the CRF on the desired areas of an input signal. The algorithm, proposed in this section does so in the following manner. First, an off-the-shelf binary classifier is used to propose the unary potentials for the class Road. Then, based on the initial segmentation map, the blob boundaries are extracted and padded. This padding plays a crucial role in the accuracy of the algorithm, as it introduces context to the boundary pixels. Finally, a CRF is applied to the segmentation map to refine the unary results. Unlike the prior art though, the CRF is only applied at the padded edge pixels which leads to a significant reduction in processing time while exhibiting a near perfect agreement with the vanilla baseline.

Luckily, there are plenty of simple algorithms for reliable edge detection. For simplicity, the very well established Sobel Operator is used [85].

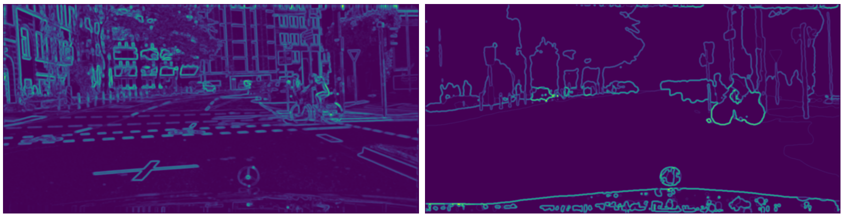
Edge Extraction

The Sobel operator defined two 3×3 convolutional kernels, G_x, G_y

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \tag{2.9}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{2.10}$$

Since convolution is one of the most wide-spread and extremely optimised algorithms in the field, algorithms which build on it automatically assume an advantageous high-ground.



(a) The edges extracted from the RGB input (b) The edges extracted from the segmentation map

Fig. 2.10: An example for the effects of the signal used for extracting the filtering edges. The edges from the RGB signal are too noisy and too detailed for filtering large and smooth classes like roads. For visual clarity, a high contrast colour map is used and both edge maps are normalised such that the brightest edge is normalised to 255. For the sake of print visibility, both images were convolved with a normalised 9×9 kernel to increase edge thickness.

The algorithm has two important hyper-parameters, the input to the edge detector and second the sensitivity threshold. As an input for the Sobel kernels, one could consider two options. First, the raw RGB input signal, containing valuable colour signals. Second, the unprocessed segmentation maps which exhibit a lower complexity and less ambiguity. Using the raw image or a low threshold would result in more edges while the segmentation output and a higher threshold value are less sensitive.

An example of this trade-off could be seen in Figure 2.10 which shows the non-dilated edge masks based both on the RGB input and on the output of the CNN classifier. One can see how the colour input (right) leads to a lot of redundant edges like around building windows or lane markers. On the other hand, by using the initial segmentation mask (left), the algorithm can't recover from missed segments which were overseen by the CNN.

Edge Processing

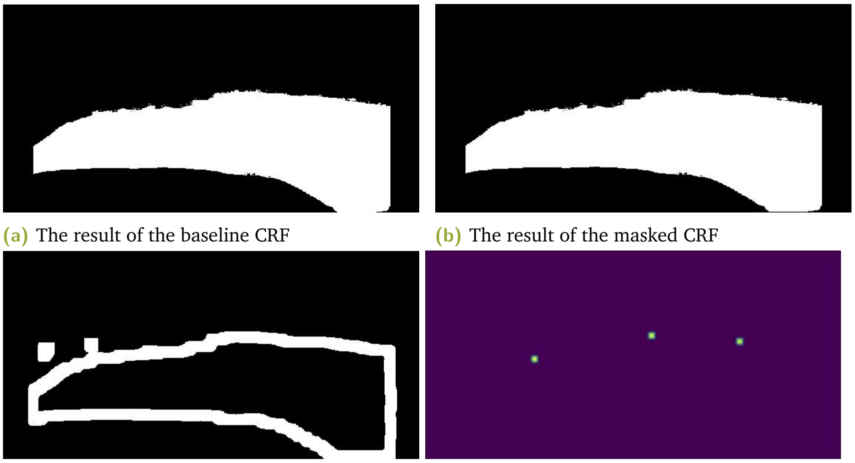
Additionally, edges are often merely a transition between classes, for which a class assignment is often ambiguous. The areas around the edges are therefore more relevant than in the edges themselves. To apply the CRF to the areas around the edges, the edges are dilated to ± 100 px around each edge pixel. This sort of selective edge padding is implemented by convolving the edge mask with a 100×100 kernel of ones, prior to binarising the signal. The exact size of the padding kernel is a hyper-parameter of the algorithm which varies according to target class and input size. The result is a binary map which is quickly and adaptively generated to include all filtering candidates according to this edge centred logic.

As a side note, while computing the edges on the segmentation map could be implemented via a pixel-wise series of 'not equals' comparisons, a Sobel kernel is easier to use and since the convolution operator is often times well optimised, it is also quicker to evaluate in practice.

For free space detection, the unfiltered segmentation maps have proven more suitable. This is due to the broad nature of roads which are generally coarse and cover large portions of the frame. For datasets and tasks with more fine-grained objects, calculating the edges based on the RGB image could prove beneficial.

Evaluation

The various tests on the Cityscapes dataset [18] show that our adaptive masks, focused only on the free-space blob edges, cover on average $17.75\ \%$ of the image, reporting a runtime of $21.37\ \%$ of that of the baseline while matching the baseline's performance for $99.61\ \%$ of the pixels. This practically promises the baseline's performance at a $\times 5$ less runtime and memory.



(a) The result of the baseline CRF

(b) The result of the masked CRF

(c) The padded filtering mask

(d) A high contrast view of the binary difference between both results. Purple stands for no difference and yellow means a difference between the results. The results were convolved with a 9×9 kernel to increase print visibility. The total number of non-zero pixels is five.

Fig. 2.11: An example of the results both from the full CRF and the masked CRF in the first row. White means road while black is the complementary assignment, i.e., not road. The second row shows the used mask and the binary difference between both results.

Method Drawbacks

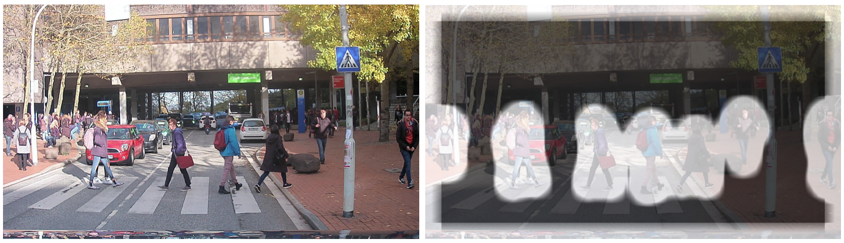
Despite an average significant runtime improvement, the frame-wise improvement might vary from frame to frame. The method itself might only introduce a negligible overhead, yet the content of the adaptive mask is directly dependent on the segmentation classifier used and the distribution of the target classes. When filtering multiple classes or even sparse, evenly distributed ones, the mask could be valid for a large portion of the frame, thus making this algorithm redundant.

Looking at the pedestrians class for example, there might be frames without any pedestrians or frames with a large number of them scattered around the scene. An example of such a scene is given in Figure 2.12 which shows a scene for which the binary filtering mask is active for $\sim 47\ \%$ of the pixels.

Nevertheless, even in such a case as the one depicted in Figure 2.12, our adaptive filtering cuts the runtime of the CRF by a half. That is, while a completely active filtering mask is theoretically possible, in practice it is not very plausible.

Finally, this example also demonstrates how the algorithm might have a large variance in its runtime from frame to frame. Since the resources allocation of an embedded System on a Chip (SoC) is a very delicate and well balanced topic, algorithms with such a large variance are less favourable than their constant time counterparts.

To address this issue, a method to benefit from the advantages of the adaptive masks combined with the constant runtime of a CNN-only solution is discussed in the following Section 2.4.



(a) An example of multiple scattered pedestrians in a frame. (b) The respective mask for the class 'pedestrians'.

Fig. 2.12: An example of a scene with a large number of pedestrians. Notice that in this specific case, a large portion of the mask, $\sim 47\ \%$, is active in the mask. Data from an internal, unpublished dataset.

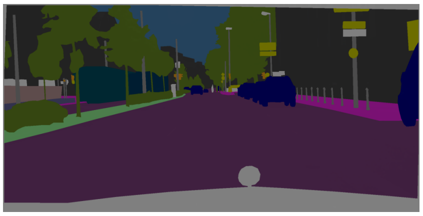
2.4 Adaptive Loss Weighting

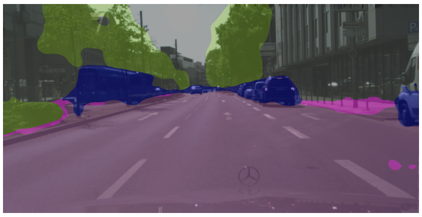
The work in the previous section was done in early 2017. The following years have brought a significant improvement in CNN based architectures which tipped the scale against the more complicated CRFs. The encoder-decoder architecture with skip connections, as originally described in [54], has gained further maturity. Unlike the low resolution outputs, these new architectures deliver crisp high resolution outputs. The 'DeepLab' model of [51] was succeeded by its fully convolutional third generation, proposed in [52]. 'Gated-Shape CNN', as discussed in [86], further improved the prediction quality by conditioning their model on the RGB edges. Additional works, like [87], have mostly benefited from the power and capacity of the latest GPU generations, proposing ever larger models.

Yet many of these works have continued showing a systematic weakness around the object edges [88]. Recognising this undesired property and based on the experience gain from the work described in Section 2.3, we proposed in [32] a simple technique to encourage better performance around object edges. We named the algorithm ‘Spatio-focal Loss’, hinting to an extension to the well known focal loss algorithm described in [89] to the spatial domain.

At the same time, the idea of edge based loss weighting was also discussed in [88]. A few months after our publication a very similar algorithm was published under the name of ‘Inverseform’ [90]. This class of optimisation algorithms for semantic segmentation addresses ways for shifting the focus of the loss function from the simple classes to the more complicated ones. In the context at hand, simple classes are regarded as classes which are vastly sparse and make for a large portion of the input. For example, looking at the Cityscapes validation set [18], the classes ‘sky’ and ‘road’ account for ≈ 45 % of the pixels. On the other hand, the classes ‘traffic light’ and ‘traffic sign’, which are extremely relevant for ADAS applications, make for 1.87 % of the pixels.

(a) RGB


(b) Ground truth


(c) Second epoch prediction

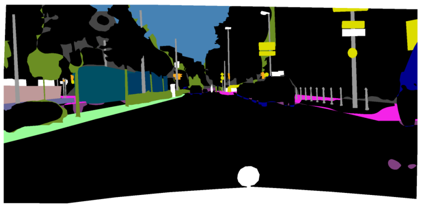

(d) Binary error map, coloured using the ground truth labels.

Fig. 2.13: An example of segmentation performance after the second epoch (corresponding to roughly 30 minutes of training) on the Cityscapes dataset [18]. Notice how the simple classes, e.g., building and road, are already taking shape while the smaller classes, e.g., traffic sign and pedestrians, have not yet converged.

Harder and Easier Segmentation Classes To better understand if all classes are equally hard to learn and predict, one could examine the early training epochs. Looking at the segmentation performance, the first evidence quickly become visible. As demonstrated in Figure 2.13, after only two epochs, one can see that those easy classes are already taking shape while other, possibly just as relevant classes,

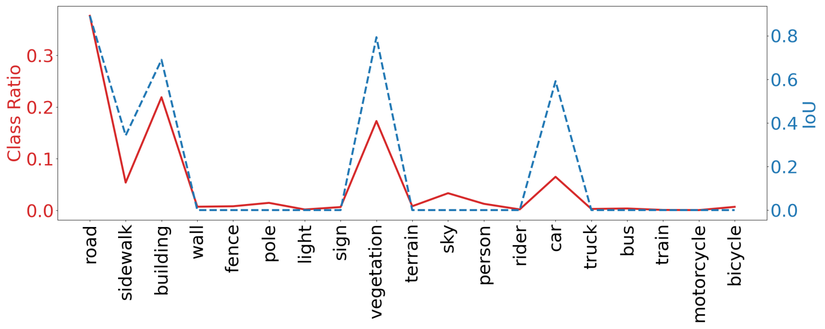


Fig. 2.14: The class-wise IoU after the second epoch, the dashed blue line, versus the normalised class ratio, the solid red line, in the validation set. The correlation is clearly visible.

are still ignored. The large variance in class Intersection over Union (IoU) [91] is also explained for the entire validation set in Figure 2.14. It shows the correlation between the IoU of the different classes and their respective portion of the dataset. This is again, after the second epoch. For reference, in our training environment, an epoch takes around 15 minutes while acceptable results, to our subjective judgement, are first seen after roughly 12 hours of training.

The rest of this section is dedicated to understanding our spatio-focal loss, which was published in [32]. It opens with an overview of the more common loss functions for semantic segmentation, discussing their advantages and disadvantages. Then, the spatio-focal loss algorithm is described and explained. Finally, a performance evaluation as well as an ablation study are provided to better asses the benefits of the loss algorithm.

2.4.1 Loss Functions for Semantic Segmentation

Traditionally, the loss functions used for semantic segmentation were adapted from single class classification. First and foremost, the softmax - cross entropy option is the most basic and yet still widely used function of choice.

Cross Entropy Entropy is often used under slightly more domain specific definitions in physics and chemistry. Though here, the definition used is rather the one from the field of information theory. It looks at entropy as the number of bits, on average, required to transmit the value of a random variable [63, p.49]. The mathematical definition, following [63, p.49], to accompany the verbal one is

$$H(X) = -\sum_{c=1}^{C} p(X_c) \log_2 p(X_c). \tag{2.11}$$

With $H(X)$ being the entropy for the signal X and $p(X_c)$ being the corresponding probability distribution for the class c out of C different possible classes.

Following the example from [92, pp.5-6], one can define a random variable of a uniform distribution over eight values. This would give the following entropy

$$H(X) = -\sum_{i=1}^{8} \frac{1}{8} \log_2 \frac{1}{8} = \log 8 = 3. \tag{2.12}$$

That is, three bits are required for describing the variable's value.

Now, looking at the non-uniform distribution of a random variable with eight signal values such that $X = \{a, b, c, d, e, f, g, h\}$ along with a respective probability function $p(X) = (\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64})$, one gets the following entropy

$$H(X) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{4} \log_2 \frac{1}{4} - \frac{1}{8} \log_2 \frac{1}{8} - \frac{1}{16} \log_2 \frac{1}{16} - \frac{4}{64} \log_2 \frac{1}{64} = 2. \tag{2.13}$$

Hence, on average, the non-uniform nature of $p(X)$ could be utilised to encode less data. Such a coding would be $\{0, 10, 110, 1110, 111100, 111101, 111110, 111111\}$. The coding of a signal according to this bit strings has the required benefit of not being ambiguous upon concatenation. Looking at the code 11001110, it could only be decoded to c, a, d.

Now, assuming a neural network $f(X)$ which should learn to approximate $p(X)$. As discussed in Figure 2.5, the actual dimensionality of $p(X)$ is often $\approx 10^7$ px times 8 bits per pixel in the vision domain. It is thus infeasible to actually get $p(X)$. One possible solution is to resort to modelling the approximating distribution $q(X)$. Having both distributions, cross entropy is used to evaluate how many additional bits are required to represent the same signal using $q(X)$

$$H(p, q) = \mathcal{L}_{CE} = -\sum_{c=1}^{C} p(X_c) \log_2 q(X_c). \tag{2.14}$$

Thus, by regarding the cross entropy as a loss function, denoted as $\mathcal{L}_{CE}$, and minimising it during training, one minimises the amount of noise or uncertainty introduced to the system by the approximation of $p(X)$. The standard notion for an arbitrary loss function is defined as $\mathcal{L}$. The interested reader is referred to [63, pp.55-58] to learn more about the relation between cross entropy and the KL divergence, or the relative entropy [93].

Finally, after understanding why cross entropy is a valuable loss function, one can look back at Figure 2.5 where some priors in the natural image domain were discussed. While cross entropy makes for a reliable loss function, it was developed for usage in general classification tasks which lack the spatial dimensions. It therefore evaluates the loss on a pixel-wise basis, ignoring not only the structural properties of the classes it is applied to, but also their distributions.

Focal Loss As seen in Figure 2.13, not all classes in a semantic segmentation task are equally easy to learn. The algorithm, presented in [89], extends the classical cross entropy to better account for the class imbalance problem. It does so by adding a weight factor to the loss function, giving it the following form

$$\mathcal{L}_{FL} = -\sum_{c=1}^{C} p(X_c)(1 - X_c)^{\gamma} \log_2 q(X_c). \tag{2.15}$$

The main difference from $\mathcal{L}_{CE}$ is the term $(1 - X_c)^{\gamma}$, whereas $\gamma > 0$ is a preset hyper-parameter. It practically means that very easy and certain classifications, i.e., pixels for which the classifier output goes towards 1, have their contribution to the final loss term discounted. At the same time, uncertain classifications receive an exponentially increased attention.

The focal loss was well accepted by the community, currently cited by $11{,}949$ different works, and is now a common extension to the classical cross entropy loss. Yet, its pixel-wise nature still fails to model the spatial information required for semantic segmentation.

Dice Loss The dice loss, as described in [94], is one of the earlier attempts at region-based loss modelling. It reformulates the Dice-Sørensen Coefficient (DSC) to act as a loss function, making it possible to train directly on the common evaluation metric. The coefficient is defined as

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|}. \tag{2.16}$$

Here, X corresponds to the prediction, while Y represents the ground truth labels.

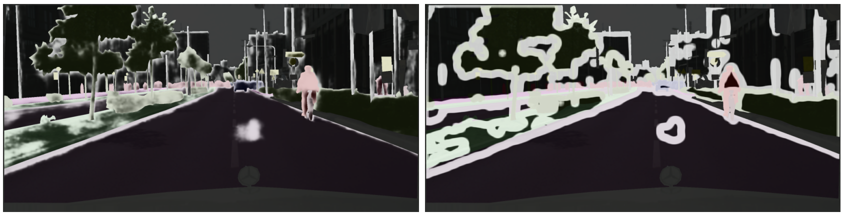
The corresponding loss function is then

$$\mathcal{L}_{DSC} = \frac{2\sum_{i=1}^{N} p(X_i)q(X_i)}{\sum_{i=1}^{N} p(X_i)^2 + \sum_{i=1}^{N} q(X_i)^2}. \tag{2.17}$$

With N representing the total dimensionality of the signal, i.e., the number of pixels.

Being the first loss function to account not only for the pixel-wise predictions, but rather also for their spatial structure, the dice loss suffers from one major issue. It only works for binary tasks, restricting its usability to specific use-cases like the aforementioned free-space detection.

Edge-Aware Loss In parallel to this work, a similar concept was published in [95], dubbed Edge-Aware Loss. Just as in this work, Zheng et al. realised that the classical cross entropy loss does not account for structure and context. At the same time, structural losses, like the dice loss, are binary and have limited applicability to multi-class segmentation tasks.


(a) Focal loss mask [89]

(b) Spatio-focal mask (ours)

Fig. 2.15: Different loss functions visualised as multiplicative masks on top of the cross entropy loss. The brighter areas represent larger weight values while the map is applied to the ground truth labels for the sake of visualisation. The corresponding cross entropy mask is a less intensive version of (a) while the nature of the edge-aware loss prevents a mask-like representation.

Vaguely similar to the image restoration from a long and a short exposure, e.g., [96], the edge-aware loss combines a structural loss with a colour space loss to get the best of both worlds. The algorithm extracts the edges from both the output and the ground-truth segmentation maps. As the edges are a binary class, i.e., edge and not-edge, Zheng et al. then use the dice loss to structure their output. The full segmentation mask is processed using the cross entropy loss and both terms are merged to a final loss value.

A good reference to the amount of structure in edges is depicted in Figure 2.10, where the edges alone are enough to get a good understanding of the image's content.

2.4.2 Spatio-focal Loss: A New Edge-aware Loss Function for Semantic Segmentation

The algorithm itself is very similar to the one described in Subsection 2.3.4. It utilises a CNN based classifier to create a semantic segmentation map

$$f(X) = q(X). \tag{2.18}$$

With $q(X)$ being the segmentation map, $f(\cdot)$ the neural network and X an input image. The result is then passed through the Sobel operator [97] to extract all edges of all classes

$$e = \sqrt{(G_x \circledast q(X))^2 + (G_y \circledast q(X))^2}. \tag{2.19}$$

Here e represents the resulting edge map, G_x and G_y follow their definition in Equation 2.9 and Equation 2.10, respectively. $\circledast$ represents the convolution operator. Notice that two additional Sobel kernels are also used, which are rotated by ± 45 deg. For the sake of simplicity, Equation 2.19 only shows the main kernels. The extension is trivial. Furthermore, unlike the algorithm in Subsection 2.3.4, this algorithm supports multiple classes and the prediction is not reduced to a binary classification. The edges are than padded to the range of ± 40 px around each edge pixel using a 40×40 matrix of ones

$$e_{pad} = \begin{bmatrix} 1 & 1 & \dots \\ \vdots & \ddots & \\ 1 & & 1 \end{bmatrix} \circledast e. \tag{2.20}$$

The result is then normalised to the maximum value of 1

$$e_{pad} \leftarrow \frac{e_{pad}}{max(e_{pad})}. \tag{2.21}$$

Finally, the mask is used to weight the classical, pixel-wise cross entropy loss

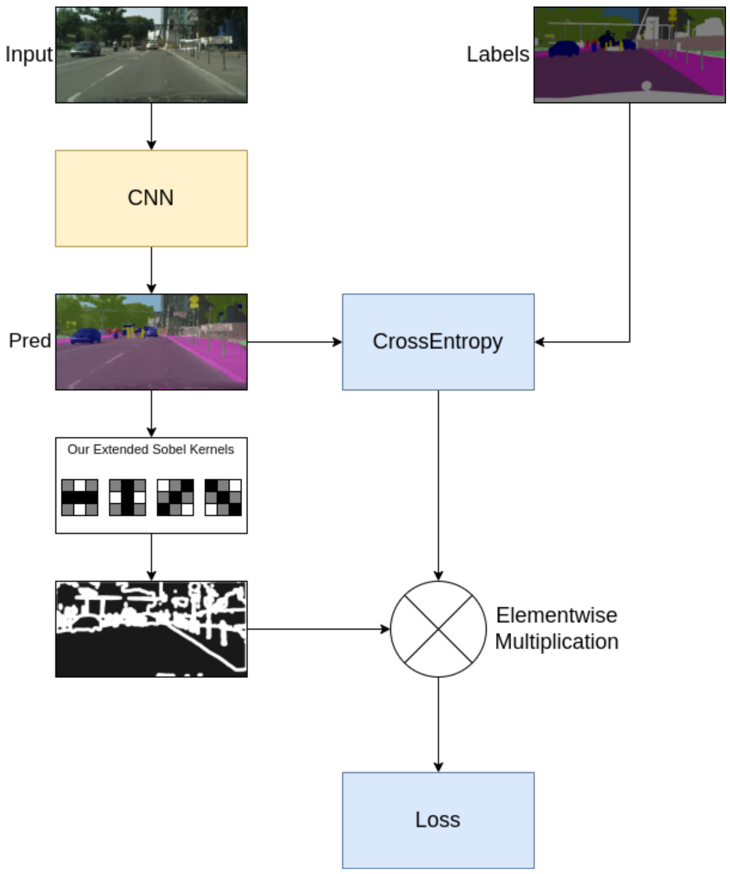


Fig. 2.16: The pipeline of the spatio-focal loss algorithm

$$\mathcal{L}_{SFL} = (0.99 e_{pad} + 0.01)(-\sum_{c=1}^{C} p(X_c) \log(q(X_c))). \tag{2.22}$$

First, e_{pad} is shifted to the range of $[0.01, 1.00]$ such that the simpler classes are still generating learning signals. The pixel-wise weight matrix is then multiplied with the cross entropy loss term $\mathcal{L}_{CE}$, where C is the total number of classes, $p(X_c)$ is the binary ground truth label for the respective class and $q(X_c)$ is the predicted value for the class c.

An example of the resulting mask is provided in Figure 2.15b which shows two masks applied to their respective ground truth maps for the sake of visualisation. As a reference Figure 2.15a shows the same sort of multiplicative mask using focal loss.

	Cross Entropy (naive baseline)	Focal Loss [89]	Edge-Aware Loss [95]	Spatio-Focal Loss (ours)
All Classes	$49.51\% \pm 0.62\%$	$50.32\% \pm 0.58\%$	$54.87\% \pm 0.66\%$	$\mathbf{57.23}\% \pm 0.72\%$
Road	96.48%	**96.72%**	96.58%	96.63%
Sidewalk	75.16%	**76.17%**	75.99%	75.93%
Building	87.00%	87.47%	**87.85%**	87.68%
Wall	25.69%	**29.86%**	29.49%	27.81%
Fence	37.58%	35.44%	35.58%	**35.66%**
Pole	47.09%	43.58%	**50.08%**	48.90%
Traffic Light	00.00%	00.00%	**48.85%**	39.18%
Traffic Sign	58.75%	57.90%	64.40%	**66.37%**
Vegetation	89.74%	89.98%	**90.34%**	90.04%
Terrain	50.94%	49.93%	**52.66%**	52.56%
Sky	90.98%	91.79%	**92.21%**	92.07%
Person	65.45%	64.93%	66.73%	**70.10%**
Rider	00.00%	00.00%	00.00%	**32.85%**
Car	87.89%	89.08%	89.05%	**89.32%**
Truck	00.00%	24.04%	**24.96%**	20.74%
Bus	40.72%	37.49%	**44.82%**	40.96%
Train	24.88%	19.88%	24.90%	**27.19%**
Motorcycle	00.00%	19.53%	00.00%	**24.62%**
Bicycle	62.47%	61.87%	64.97%	**67.90%**

Tab. 2.2: The class-wise IoU results of the same model, trained using cross entropy, focal loss, edge-aware loss and finally, our proposed spatio-focal loss. One can see the advantage of the structured loss functions, i.e., edge-aware and spatio-focal over the pixel-wise methods. The numbers are an average of three runs with the standard deviation stated for the overall IoU.

Notice how the focal loss weighting also tends to emphasise the edges, but it exhibits a hard cut at the edges. This means that the gradient signals are lacking context, a context which is crucial for robust learning, according to our hypothesis.

Furthermore, the edges themselves are merely the transition between classes. As such, they are prone to sensing uncertainty due to aliasing and do not make for the highest quality signal by themselves. The topic of aliasing is discussed in Subsection 3.3.2, however for now one can imagine that the extract transition from class 'A' to class 'B' is often ambiguous. Hence, focusing solely on the edges does not seem to be an ideal strategy.

The entire pipeline is depicted in Figure 2.16.

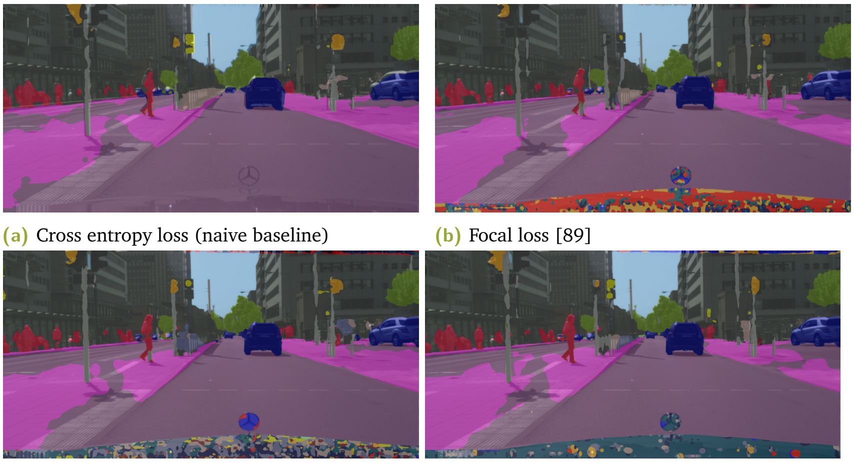
Evaluation

We evaluated our spatio-focal loss on the thoroughly discussed Cityspaces [18]. The architecture is a reduced version of the aforementioned FCN [50]. It was trained until no further significant improvement was recorded on the validation set for ten or more epochs, roughly 25 hours per model. The encoder's weights were pre-trained on the ImageNet dataset [98]. The model's hyper-parameters were fixed such that only the loss function varied throughout the experiments.

First, we examine the IoU values, both over all classes and for each individual class. These are presented in Table 2.2. One can see a clear advantage of the edge-based losses while our spatio-focal loss surpasses the second best method, the edge-aware loss, by a significant margin of $2.36\ \%$.

A better alignment is also witnessed in Figure 2.17. One can not only see the improved shape over focal loss Figure 2.17b, edge-aware loss Figure 2.17c and the baseline Figure 2.17a, but also the improved attention to details, seen best in the pedestrian class.

Finally, we removed the accompanying CRF and evaluated the spatio-focal loss with the larger DeepLab-V1 model [51]. On top of the mentioned pre-training time, each model takes around 7 days to train, fully occupying a high-end GPU. This has unfortunately limited our ability to freely experiment, yet Figure 2.18 visualises the results while a zoomed side-by-side comparison is presented in Figure 2.19. The figures show a significantly improved attention to details, achieved by a mere change to the loss function.

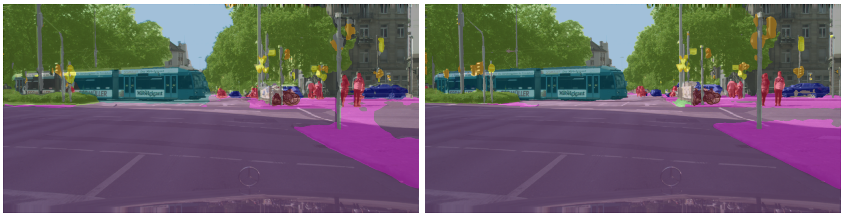


(a) Cross entropy loss (naive baseline) (b) Focal loss [89]

(c) Edge-aware loss [95] (d) Spatio-focal loss (ours)

Fig. 2.17: An example of the same frame predicted with the four different loss functions. The ego vehicle is ignored during training, resulting in random predictions during inference. One can see that the edge based approaches better match the shapes of the predicted elements, e.g., pole and pedestrian, while our spatio-focal loss generally reaches a slightly better performance.

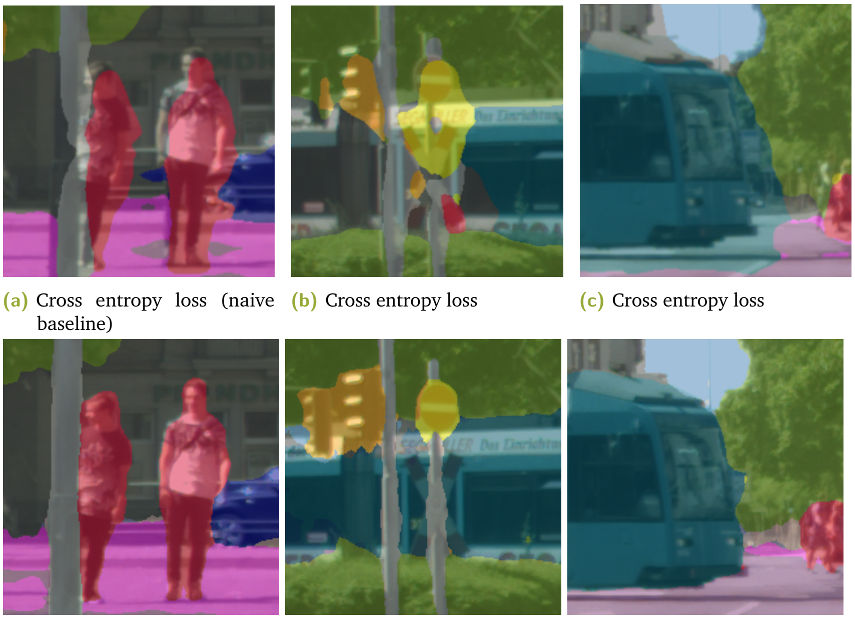
The models reach a final mean IoU of $75.05~\%$ for the cross entropy baseline and $76.14~\%$ for the spatio-focal model, showing that the improvement scales well to larger and more accurate models.



(a) Cross entropy loss (naive baseline) (b) Spatio-focal loss (ours)

Fig. 2.18: An example of the results from the Deeplab-V1 model [51]

Method Drawbacks The most obvious drawback of every method is its introduced hyper-parameters. In the case of the spatio-focal loss, this hyper-parameter is the padding size. Extensive padding reduces the attention to small details while no padding results in fine border lines, similar to the focal loss mask, as depicted in Figure 2.15.


(a) Cross entropy loss (naive baseline) (b) Cross entropy loss (c) Cross entropy loss

(d) Spatio-focal loss (ours) (e) Spatio-focal loss (f) Spatio-focal loss

Fig. 2.19: A zoomed side by side comparison of selected parts of Figure 2.18.

2.5 Conclusions

Throughout this chapter, we demonstrated how the most underlying property of natural images, their sparsity, could be used to improve the efficiency as well as the performance of semantic segmentation models. This sparsity could act as a prior by means as simple as an edge extraction.

Our adaptive masks could explicitly be incorporated into the model, as seen in Section 2.3. In this configuration, the edges help us understand what parts of an image are the likely candidates to profit from CRF filtering. The CRF is then only applied to these parts, thus resulting in a factor ~ 5 reduction in runtime, comparing to the baseline. A reduction which comes at hardly any cost, exhibiting a $99.61\ \%$ results agreement with the baseline.

Another embodiment would implicitly incorporate the edges into the model by guiding the loss function, as seen in Section 2.4. As most pixels are redundant, there is no actual justification for using them in training. On the contrary, they overshadow the higher quality signals and reduce their importance. By only learning to segment the wide areas around the edges, the quality of the gradients is artificially increased and their redundancy reduced. Doing so leads to an improved attention to details and representation of smaller classes.

Future Work The successful application of our method to natural images, also encourages its application to other domains. An example for such a domain which could possibly benefit from the utilisation of edges is Lidar point cloud segmentation [99], [100]. As Lidar outputs dense point clouds, it makes for a good candidate for a future extension.

It would furthermore be interesting to explore the focus on edges in conjunction with the future frame prediction, described in [101]. Such an extension of the loss function could increase the reliability of the predicted segmentation maps, possibly stretching further into the future.

3 Efficient Convolutional Neural Networks

As seen in the previous chapter, Convolutional Neural Networks make the current state-of-the-art in spatial signal understanding. It started with the more standardised visual scene parsing and models like AlexNet [3], ResNet [102], FCN [94] and many more. Once more data, more compute and cheaper sensors caught up with the availability of the common camera, CNNs also started presenting appealing results in additional domains like the PointNet for, among others, radar signals [103] or RangeNet++ for Lidar data [99].

Unfortunately, we now know that the impressive super-human performance of many of these works is at the very least partially attributed to the latest releases from nVidia [69]. In their work, Bianco et al. showed a very strong correlation between the reported accuracy of published models and their respective computational capacity. I.e., they implied that at least a part of the progress in performance in recent years should be attributed to improvements in GPUs rather than to actual improvement in neural architectures.

This correlation is, by itself, not enough to invalidate the hard work of the Computer Vision and Machine Learning community. Nevertheless it is enough to encourage a reconsideration of the current course of incremental developments. A reconsideration which would also consider Occam's Razor [104] and the high environmental impact of training such super-models [26].

Before making loud claims regarding the abandoned longing for simplicity in NN architectures, it is worth mentioning that recent works have raised concerns about the validity of Occam's Razor to NNs. Students and enthusiasts alike learn that large models are susceptible to over-fitting and that models should hence be as small as possible. Then, works as [105], [21] and [102], all from top tier researchers, shown the community that larger networks are not bad after all.

More recently, Belkin et al. studied the startling convergence of models which have more parameters than the dimensionality of their training data [106]. They concluded that for a not yet understood reason, the size of a model does not directly correlate with its actually learnt complexity at convergence. In fact, the larger the model is, the simpler the function it ends up learning.

Furthermore, many recent works have started presenting not only the final accuracy, but also their model's computational capacity [107], [69]. This shows for an additional step towards democratisation and environmental responsibility. First, by discounting the contribution of large models, attributing it to their mere size, researchers are discouraged form training such models to begin with. Second, accepting contributions which are maybe not the new SotA but are marginally more efficient than their predecessors, allows smaller research institutes without server farms and thousands of dollars training budget to publish as well. As a reference, Sharir et al. estimated the training costs of Google's T5 model [108] at 10 million US dollars [26].

Nevertheless, even if accounting for the appealing accuracies of large models, they are still not always applicable to all fields and use-cases. Perhaps the most obvious example is real-time, embedded applications. This class of algorithms is required to run at interactive inference frequencies on smaller hardware configurations, which are often also battery-powered. With the increasing demand for such applications, a novel class of architectures and algorithms has emerged. Here, the focus is shifted from squeezing every last bit of accuracy to getting models which can directly run on end devices, e.g., mobile devices, Digital Signal Processors (DSPs), etc.

This chapter covers our contribution to the field of efficient NN architectures in the form of the EffNet model, published in [109] and [110]. The background is discussed and the work is compared to the state-of-the-art in 2018. Finally the last section discusses the impact of this work as well as the development of the art since its publication.

3.1 Related Work

The field of efficient architectures mostly spans over three equally important branches. Hyper-parameter optimisation, pruning and, last but not least, alternative architectures. This work contributes to the latter branch in the form of an alternative, general purpose CNN block to replace the vanilla CNN layer.

3.1.1 Hyper-Parameter Optimisation

Although the question of architecture choices remains an active research field, there are several works which have significantly contributed to the community's knowledge. At the highest level, most tasks involving ML have a goal of the form

$$\text{argmin}_\theta(\mathcal{L}(y, f_m(x|\theta))). \tag{3.1}$$

This means, given the input x, find the set of model parameters θ, which minimises the loss $\mathcal{L}(\cdot)$ between the ground truth label y and the output of the model. The loss is an arbitrary function from the likes of l_2, 'negative log likelihood', the 'spatio-focal loss' as described in Subsection 2.4.2 or any other function of choice. As long as the function can be used to evaluate the difference between the outputs and the targets while having clear derivatives, it could be used as a loss function. Here, the model $f(\cdot|\theta)$ is parameterised by the trainable parameters θ, but also by the predetermined set of hyper-parameters m. The latter determine the amount of layers, number of filters, activation function, etc.

The task of hyper-parameter optimisation, which has recently taken on the more marketing-friendly name of 'Auto-ML' [111], focuses on a different type of optimisation, namely of the set m

$$\text{argmin}_m(\mathcal{L}(y, f_m(x|\theta))). \tag{3.2}$$

This task definition aims to find the optimal set of hyper-parameters for a given dataset. Such an optimisation is mostly achieved by one of three methods

1. Hyper-parameter optimisation; Here, the hyper parameters are addressed as a black box which spans a search space. By assessing the effects of the different parameters on the overall performance, an educated guess could be made about their desired values and their respective relevancy to the task. The most classical example for such a method is a grid search which samples the hyper parameters at regular intervals. For more information, see [111, pp.3-35].
2. Meta learning; A study of the compounding aspects of different models and their affect on the performance in different tasks. By collecting enough experiment data, e.g., from multiple publication, a good approximate could be made regarding a well suited set of hyper parameters for a given task. A more extensive overview in provided in [111, pp.35-63].

3. Neural architecture search; An alternative to the previous methods is to treat the hyper parameter optimisation as a regression task and try to learn a general model which predicts a matching architecture for a queried task. Here, too, a good overview could be found in [111, pp.63-77].

3.1.2 Pruning

Pruning is an interesting method, as it is the only common method to take a bottom up approach. Looking at the other methods covered in this section, like the hyper parameter optimisation or even the alternative architectures at the core of this chapter, one sees a top down approach. I.e., the attempt to create a new model which is just as accurate as the baseline, yet with a lower computational demand. Pruning, on the other hand, starts with a fully trained baseline and aims to remove as much redundancy as possible, until only the absolute necessary core remains.

The algorithmic pipeline is straight forward. After fully training a baseline model, it commences on Expectation Maximisation like pruning iterations [112]. While iterating to a predefined convergence criterion, the model is first pruned according to a given logic and then fine-tuned for the new architecture. The pruning itself mostly means querying for the neurons with the least contribution to the performance and removing them altogether. The fine-tuning step allows the remaining neurons to adjust to the removed capacity.

While the performance loss is immanent, one could often expect as much as a compression ratio of two along with a reduction of $\sim 2\%$ in the overall performance [113]. The strategy might seem to be the exact opposite to the aforementioned findings of Belkin et al. in [106] who showed the clear benefit to increasing the number of parameters. Yet, it is important to remember that the use case is completely different. Models are pruned knowing that some performance will be lost, but at the same time, it means that these pruned models become suitable to end devices.

A thorough review of the current state of pruning algorithms, including an elaborate meta study of no less than 81 different publications could be found in [114].

3.1.3 Quantisation

Quantisation is another familiar way to reduce the complexity of neural networks. Unlike the other methods, here the goal is to reduce the redundancy of the weights representations, mostly from 32 bit floating point to 8 bit integer. The idea builds

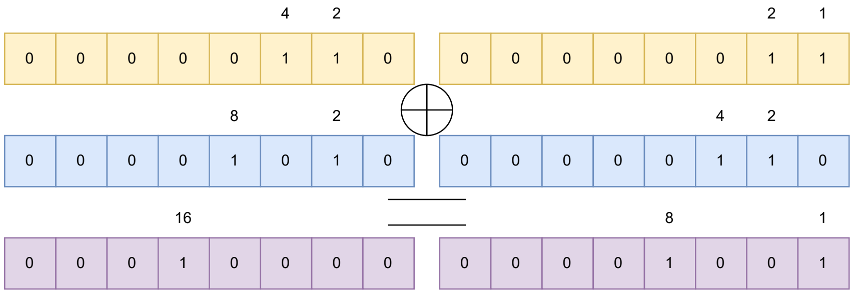


Fig. 3.1: An example showing how two pairs of 8 bit unsigned integers could be added in parallel without affecting each other. The left hand side evaluates $6 + 10 = 16$ while the right hand side calculates $3 + 6 = 9$. The numbers at the head of each integer hint the value of the bit they are over.

on the low level implementation of single word parallelisation in many different systems. Many given processors support 64 bit long words. These words could be either utilised for a single computation per cycle or, packing multiple integer variables, multiple computations in parallel. Looking at 8 bit integers, one could reach as much as eight computations per cycles by concatenating eight 8 bit variables into a single 64 bit word. The process of adding multiple variables in parallel is demonstrated in Figure 3.1. This process only leads to collisions on overflow, yet there are plenty of different methods to account for such a problem. The most common solution is returning longer words as a result. For example, each addition of two 8 bit numbers results in an 16 bit number.

This sort of parallelisation could be seen in two ways, either as reducing the computational burden for a given model, or, as shown by Jacob et al. in [115], increasing the throughput and therefore also the accuracy per a given latency. The second perspective is reasonable for embedded real-time application which normally have a maximum budget of latency per frame. A recent and beginner friendly survey of quantisation methods is provided in [113].

3.1.4 Alternative Architectures

Finally the branch of alternative architectures tackles the issue at its core. Here, by carefully analysing and understanding the root causes of the high computational demand, an alternative architecture is derived to optimise it altogether. These models are admittedly about as good as their baseline, yet with significantly less parameters and, in turn, also less FLOPs.

The work in this direction was initiated in [116] with an architecture which was dubbed ”MobileNet”. It was published in a technical report which has managed to set a new baseline for efficient architectures in the following years. It is currently at its third generation [117] with the original work is, according to Google Scholar, currently at over $6{,}000$ citations.

The key observation is that a substantial reduction in computational effort is achievable by a separation of the spatial convolution from the channels dimension. The notion was initially introduced in [118] and was dubbed depth-wise separable convolution. Rather than having all input channels going into the computation of each output channel, only a single input channel is used per output channel, see Figure 3.2a and Figure 3.2b. The intra-channel information flow is consecutively handled in a point-wise convolution layer. I.e., a layer over all channels with the spatial resolution of 1×1. The reformulation results is a computation reduction of

$$\frac{1}{Ch_{out}} + \frac{1}{h_k w_k} \tag{3.3}$$

where Ch_{out} represents the number of output channels and h_k, w_k represent the height and width of the spatial kernel, respectively.

The depth-wise convolution based architecture has demonstrated a similar accuracy on ImageNet as the ‘VGG16’ model [105] with only $\sim 4\ \%$ of its multiplications and additions.

The following development was the ‘ShuffleNet’ architecture [119]. Its main contribution was to group the point-wise convolutions, thus reducing their computational effort by a factor of the number of groups. The intra-group interactions were implemented by shuffling the elements along the channels dimension. This shuffling operation is depicted in Figure 3.2c.

Both of the aforementioned works have covered multiple architectural optimisations. These optimisations, along with the principles behind them, are covered in the following section. They are subsequently used to justify the design decisions in our proposed convolution block.

3.2 On Convolution and Computation

Convolution as a mathematical operation and Convolutional Neural Networks as a leading algorithm for neural architectures are so broadly used that it might seem

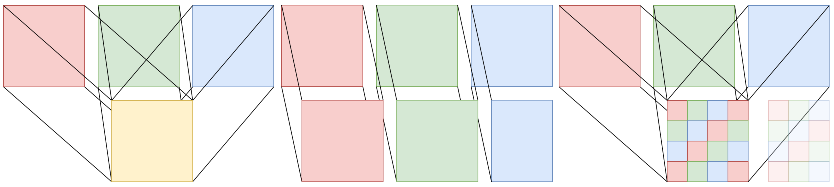

(a) Vanilla convolution (b) Depth-wise convolution (dw) (c) Channel shuffle

Fig. 3.2: A visualisation of the core elements of the works discussed in Subsection 3.1.4. The vanilla convolution (Figure 3.2a) processes all input channels for each output channel. The depth-wise convolution (Figure 3.2b) only applies a single kernel per input channel, resulting in a one-to-one mapping. The shuffle module (Figure 3.2c) shuffles the elements of all channels to allow for a consecutive grouped processing.

like many users are completely unaware of the fundamentals behind the term. It has now become a common knowledge that convolution is the application of a kernel to an input signal. Yet where does it come from, why has it grown so popular and what are its pros and cons?

3.2.1 Background

As the operator at hand is well established, it is hard to pinpoint its exact origin. Nevertheless, it seems to have been proposed as a derivation of Taylor's Theorem in [120]. The domain is continuous which means that the application, represented by $\circledast$, of the kernel g to the input I is an integral of the form

$$(I \circledast g)(t) := \int_{\tau=-\infty}^{\infty} g(\tau)I(t-\tau)d\tau. \tag{3.4}$$

While t is often a temporal offset, it might as well be any sort of a variational shift.

Notice that this original definition assumes a single-dimensional signal. However the expansion to n dimensions is rather straight forward and gets even simpler in the discrete domain.

Essentially, the convolution operator means the application of a known kernel of a manageable size to a larger, more extensive input signal. I.e., the same function is repeatedly evaluated on all possible positions on the input signal.

Additionally, it is worth mentioning that the challenge of integrating over the product of an arbitrary function with an arbitrary signal is often solved by transitioning to the Fourier domain. Following the transformation, a simple multiplication could be used to efficiently calculate the convolution. Yet here, even with efficient methods as the Fast Fourier Transform (FFT) [121], the bottleneck is not fully resolved. It is rather moved to the domain conversion and is, hence, not the ideal solution in terms of computational efficiency.

A significantly more efficient alternative to domain transformation is the discretization of both the input signal and the kernel. Since many types of sensors measure discrete signals by nature, this alternative quickly becomes favourable for many applications. The convolution of a discrete kernel, g, with a discrete input, I takes the following form

$$(I \circledast g)[t] = \sum_{m=-M}^{M} I[t-m]g[m]. \tag{3.5}$$

Here, g is a kernel of $2M+1$ elements with its centre at the index zero. The rest of the definition follows Equation 3.4.

In the field of ML where the kernel is optimised to the data, another computational optimisation could be made. As one can see in Equation 3.5, the kernel is actually applied in reversed order to the input. For two-dimensional inputs this means that the rows and columns are flipped before being applied to the input signal. Yet when the kernel is learnt, its order of application becomes arbitrary. For this reason, most common implementations nowadays actually ignore the original flipping in favour of more memory efficient cross correlation operator. Its definition is similar to convolution, yet is lacking the flipping of the rows and columns. This means that the kernels are only read from the memory once and remain in the much quicker cache throughout the computation.

For the sake of completeness please note that the kernel is not actually read in once, but rather $\left\lceil \frac{2M+1}{CL} \right\rceil$ times, with CL representing the cache-lines' length.

3.2.2 Convolution, a Computational Heavyweight

The next important recognition is that the four-dimensional computation, used by CNNs, is compute intensive. Such layers were only first discussed in [3], merely a decade ago. It was the first time in human history that enough computational power was available to meet the very high demand of such models. This is ignoring small scale, initial works like [5]. Subsection 3.2.1 covered the simple two-dimensional case of convolution. Yet, images often span over three dimensions or more, i.e., height, width, channels. Here, channels mostly mean the RGB information. This section covers the computational burden which accompanies such layers and takes the reader through a journey to visualise its intensity. The resulting computations, done on a toy example for a single vanilla convolution layer, are then used throughout the rest of this chapter as a reference illustrative baseline.

Building on top of the convolution operator, as described in Equation 3.5, a DNN typically consists of multiple layers of such kernels. Each layer is then followed by a non-linear function and often a pooling function before continuing to the next one.

Throughout the processing pipeline, the channels diverge ever further from their colour data and slowly transition into feature specific activations of different scales and complexities. For example, when classifying cats and dogs, the last layer would normally consist of two output channels while its height and width are processed down to one. Along the pipeline, one can expect kernels which look for distinct characteristics like pointy ears or tiger-like fur patterns [122]. These distinct features are combined to detect complex structural interactions from the individual input channels.

To enable the recognition of such inter-channel patterns, the convolution algorithm from Subsection 3.2.1 is extended to include the channel dimensions. Following the common notation, a 3×3 kernel means a kernel with the spatial span of three by three pixels. These are also referred to using h_k for the height dimension and w_k for the width dimension. A set of 3×3 kernels in a layer with 64 input channels, Ch_{in}, and 128 output channels, Ch_{out}, is represented by a tensor of shape $[h_k, w_k, Ch_{in}, Ch_{out}] = [3, 3, 64, 128]$. Subsequently, for the convolution of the input I with the kernel k, at the rows and columns pixel position $[r, c]$ and the output channel ch_{out}, the output O is defined as follows

$$O_{r,c,ch_{out}} = \sum_{ch_{in}=0}^{Ch_{in}-1} \sum_{i=-\lfloor \frac{h_k}{2} \rfloor}^{\lfloor \frac{h_k}{2} \rfloor} \sum_{j=-\lfloor \frac{w_k}{2} \rfloor}^{\lfloor \frac{w_k}{2} \rfloor} k_{i+\lfloor \frac{h_k}{2} \rfloor, j+\lfloor \frac{w_k}{2} \rfloor, ch_{in}, ch_{out}} I_{i+r, j+c, ch_{in}}. \tag{3.6}$$

The input channels are represented by Ch_{in} and i and j are used to index the rows and columns of the kernel, respectively. In this toy example we assume a convolution with a padded single pixel frame of zeros. This frame means that the input size is the same as the output size, thus simplifying both the annotation and the computations. The algorithmic term for such a convolution is `same` and is implemented in all major deep learning frameworks. An illustration of Equation 3.6 is provided in Figure 3.3

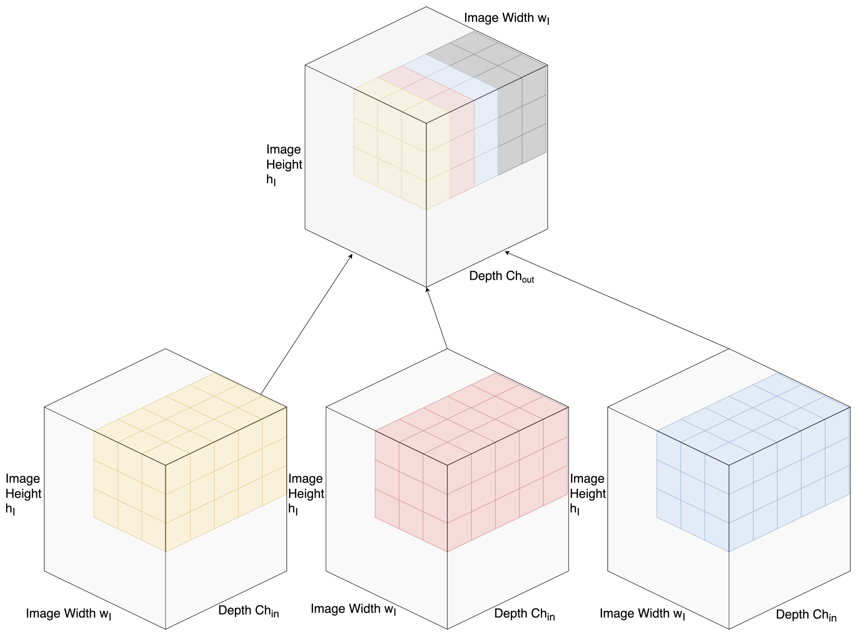


Fig. 3.3: An illustration of a convolution with three kernels and five input channels. Notice that the three kernels are applied to the same input tensor, depicted here three times for the sake of visual clarity.

where an input tensor is convolved by three different kernels to give a multi-channel output.

In terms of theoretical complexity, a summation over four dimensions is evaluated. These are the height, the width, the input channels and the output channels. Such a single iteration over the input data translates into the following Big $\mathcal{O}$ annotation

$$\mathcal{O}(h_I w_I Ch_{in} Ch_{out}). \tag{3.7}$$

With h_I and w_I representing the the input's height and width, respectively. The iterations over the kernel's size are a small negligible constant in the Big $\mathcal{O}$ annotation, since $h_k << h_I$ and $w_k << w_I$.

FLOPs

In terms of actual computations, the formulation in Equation 3.6 means a single multiplication, repeated $h_k \times w_k \times Ch_{in}$ times and a similar number of additions for each output channel and pixel. In our example, the FLOPs per pixel are then

$$\begin{aligned} \text{FLOPs}_{[r,c]} &= 2h_k w_k Ch_{in} Ch_{out} \\ &= 2 * 3 * 3 * 64 * 128 = 147{,}456. \end{aligned} \tag{3.8}$$

The leading factor of 2 is for the multiplications as well as the additions, which require the same amount of operations. Additionally, Equation 3.8 only accounts for a single output pixel of a single layer. Considering even the small input resolution of $256 \times 256 = 65{,}536$ px and that models like the renounced 'ResNet50' [102] consist of no less than 50 layers, it is clear how such models operate within the millions and billions ranges of FLOPs. A unified definition for the computation of FLOPs is hence

$$\text{FLOPs} = h_I w_I 2 h_k w_k Ch_{in} Ch_{out}. \tag{3.9}$$

For the single, toy example layer, a total of $65{,}536 * 147{,}456 = 9{,}663{,}676{,}416$ FLOPs is expected.

Memory Access

An additional, often ignored, aspect of computational effort is the memory access. Unfortunately, memory is scarce and expensive, meaning that data is not always at the right place at the right time. It rather needs to be loaded and made available for the computational kernels or cores to process. While data indexing is mostly done in some sort of iteration loops, mostly `for` loops, where the data is indexed one element at a time, in the background, the host machine is doing a lot more.

Since data access belongs to the most expensive operations in terms of execution time, computers utilise a hierarchical caching system to access their memory. Each call for loading data starts at the nearest preceding cache line address and loads an entire burst of data. This burst of data could sometimes have a variable size but it is limited by the memory bandwidth. E.g., even a call for a single 8 bit value often results in loading 64 bits of contiguous data around the target element.

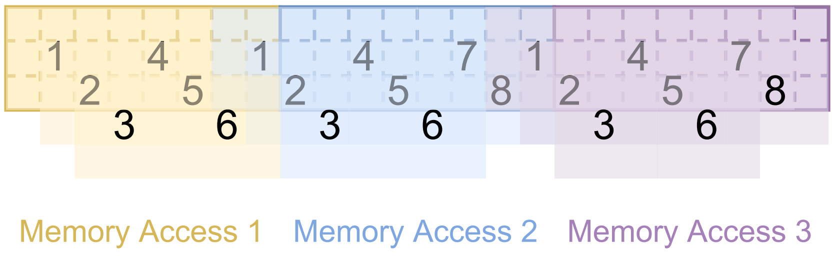


Fig. 3.4: A visualisation of three consecutive memory access calls along their respective convolution kernel placements. The numbers represent the kernel placement per loaded memory elements. Notice that all placements are aligned with the loaded rows, the vertical offset is for spacing and inseparability purposes only. One can see that the first memory read only fetches enough data for six convolution placements while consecutive data is combined with preceding elements to allow for eight placements.

This burst definition takes advantage of the natural sparsity assumption, assuming that if element i is currently required, the following iteration is likely to require element $i + 1$. In other words, related elements are contiguous in the memory.

Generally the bandwidth for accessing the memory might have a different lengths for different architectures. However given the embedded scope of this chapter and the assumption of data quantisation into an eight bit representation, there remains a small number of realistic options. Throughout this chapter, the common embedded bandwidth of 64 bits or eight pixels for each memory access is assumed [123].

Once a burst of elements is read, instead of simply passing it to the processor, it is loaded to the different levels of cache for more efficient fetching in consecutive iterations. As explained later in this section, the different cache levels have the advantages of significantly shorter read/write intervals.

Regarding this section's toy example, each cache-miss, i.e., indexing of an element which was not yet cached, triggers a memory access which retrieves eight pixels. A memory bandwidth of eight pixels allows for six kernel placements for the first memory access and eight placements for the consecutive fetches. For the sake for visual explanation, Figure 3.4 demonstrates the kernel placements for three three-row memory loads. The figure utilises colour coding and a fetch-wise placement enumeration to illustrate the computations associated with each memory access.

For the sake of simplicity, the initial six placements are disregarded and all memory loads are treated as if they were loading enough data for eight full placements. The number of memory accesses η for the vanilla convolution layer is thus

$$\eta = Ch_{in}h_k h_I \frac{w_I}{\rho}. \tag{3.10}$$

With ρ representing the placements per cache line. In terms of this section's toy example, this means $64 * 3 * 256 * 256/8 = 1{,}572{,}864$ data loading accesses to the memory. This order of magnitude validates the decision to ignore the effect of the first placement, as it is fully amortised by the large amount of fetches.

From Operations and Calls to Cycles and Clocks

Finally, in the computations and equations so far, there is a missing piece of information. When considering live applications, one often speaks in term of runtime on a target hardware. The units of choice are most commonly FPS. Yet the preceding computations of both the operations as well as the memory access calls were given in timeless units. For the conversion of these units to actual runtime, one needs to look at the cycles, or clocks per command.

Cycles and clocks are synonymous terms which are used to describe the time it takes the hardware to complete a given command. These numbers vary from architecture to architecture and are thus mostly ignored in favour of a simpler runtime analysis, using a profiler for example. Nevertheless, a general understanding of clocks and their meaning is essential for optimisation, as this crucial building block explains an otherwise widely misleading understanding of the computational burden.

The exact definition of clocks exceeds the scope of this work. It is enough to explain that the clocks belong to the main characteristics of a computational architecture. When speaking, for example, of a computational unit, running at 1.6 GHz, the reference is to the number of cycles this respective unit can execute in a second, i.e., $1.6 * 10^9$ cycles per second. Defining a real-time application as an application which runs at least at 24 FPS, one gets an average budget of ~ 0.042 seconds per frame, or $\sim 66.67 * 10^6$ cycles per frame. This number might sound large, yet in relation to the number of operations calculated earlier in this section, with $\sim 1.57 * 10^6$ memory reads and $\sim 663.7 * 10^6$ multiplications and additions, the problem of real-time application could be seen from a new perspective.

Furthermore, it is important to recognise that not all commands are equally demanding. Computational architectures have different types of memories which increase in size as well as in access latencies the further they are located from their computational unit. This work will not cover the deepest technical cavities of different hardware components. Yet for the sake of illustration, we refer to the access latencies from [124], also supported by the discussion in [125]. These vary between hardware types and models, yet the general relations and orders of magnitude are valid across the line.

1. Registry access latency 1 cycle.
2. L1 cache access latency 4 cycles.
3. L2 cache access latency 11 cycles.
4. L3 cache access latency 39 cycles.
5. Main memory access latency 107 cycles; The input data is initially loaded from here.
6. M.2 Solid State Drive (SSD) access latency ~ 400 cycles; This is an extremely rough estimation for an illustrative purpose only, as the SSD market exhibits immense variations of products, generations, technology and performance.

Furthermore, actual computations, e.g., additions and multiplications, are not only evaluated in a single cycle, they are rather also often highly optimised and executed in parallel over a few cycles. As visualised in Figure 3.1, such parallelisation is also common in more serial architectures like the common X86 CPU. Since it processes data in 64 bits, also known as 'word length', there are commands which fully utilise this capacity by multiplying eight 8 bit inputs at the same time, practically allowing for a factor eight parallelisation.

Having the cycle latencies for each of the aforementioned commands, one can finalise the computation and deduce the estimated runtime of the reference layer. For the memory access, some $1{,}572{,}864$ calls were estimated, each at 107 cycles, i.e., $168{,}296{,}448 \approx 168.3 * 10^6$ cycles. As for the actual computations, these were estimated at $9{,}663{,}676{,}416$ FLOPs, with a parallelisation factor of eight and a single clock cycle per batch of eight-elements, resulting in $1{,}207{,}959{,}552 \approx 1{,}208{*}10^6$ cycles. It should now be understandable, why the memory usage is also an important factor to consider when optimising for runtime.

3.3 Motivation

Prior to this work, the field of CNN architecture for computational sensitive applications consisted mostly of two popular architectures. These are 'MobileNet' and 'ShuffleNet', which were covered in Section 3.1 and considered promising for

	Accuracy	FLOPs	Ratio
Vanilla Baseline	79.30%	56,324,196	1.00
ShuffleNet Basline	72.05%	19,094,628	0.34
Vanilla Small	77.45%	18,263,908	0.32
ShuffleNet Small	71.28%	6,556,708	0.12

Tab. 3.1: Evaluating the necessity of an optimised architecture, here ShuffleNet [119]. First, the same vanilla baseline model, as evaluated throughout this chapter, 'Vanilla Baseline'. Then using the same hyper-parameters, same number of blocks, channels and the same pooling, a matching ShuffleNet architecture is trained, 'ShuffleNet Baseline'. The third model, 'Vanilla Small', is a version of 'Vanilla Baseline' with reduced channels to roughly match the computational cost of the 'ShuffleNet Baseline' model. Finally, 'ShuffleNet Small' uses the same hyper-parameters of 'Vanilla Small' but as a ShuffleNet architecture. The results support our hypothesis that the ShuffleNet architecture mainly reduces excess capacity.

applications with limited resources. Yet upon evaluation on different automotive benchmarks, we noticed a persistent decrease in accuracy, comparing to their vanilla convolution baseline. These comparisons are discussed in Section 3.5. In order to solve the observed decrease in accuracy, we first had to better understand its root causes. The various properties of the aforementioned architectures were dissected and analysed, delivering valuable insights for further advancements.

3.3.1 Bottleneck Blocks

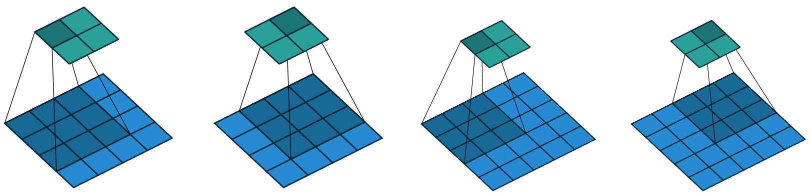
Initially introduced in [126], a reduction factor of eight was proposed, meaning the number of channels is compressed on block entry to one eighth of the block's output channels. The reduction is often done by a 1×1 convolution layer which only processes the channels dimension. For example, the `fire2` block from Table 1 of [126] consists of two layers, a squeeze layer and an expand layer. The squeeze layer gets an input with 96 channels. These are then 'squeezed' down to 16, before eventually inflated to 128 channels. This sort of block definition ignores the number of input channels and only considers the desired number of output channels. The justification is that by creating these artificial bottlenecks, the network is forced to use its available bandwidth more efficiently. ShuffleNet [119] revised the concept with the more conservative compression factor of four.

Looking at the architectures of these aforementioned works, one could notice that although they claim to aim at mobile hardware, they cover architectures larger than what was commonly supported back in 2016 and 2018, respectively, when they were first saw light. Choosing a large number of channels to begin with, just to than reduce it by a given bottleneck factor seems redundant. We therefore compared a bottleneck model to a smaller baseline. The results are presented in Table 3.1 and show that the bottleneck architecture is merely the reduction of sparsity which exists in the model due to its oversized architecture.

Another way to examine the effects of bottleneck architectures is by looking at the actual throughput after each layer. This is covered in Subsection 3.3.5 and in Table 3.5.

3.3.2 Strides and Pooling

In the second part of the design scrutiny, strided convolution was compared to pooling. While max pooling is a widely used practice in CNNs, strided convolutions are a less known and are therefore quickly recapitulated.



(a) First iteration, stride 1 (no stride) (b) Second iteration, stride 1 (no stride) (c) First iteration, stride 2 (d) Second iteration, stride 2

Fig. 3.5: Comparing vanilla convolution, (a) and (b), to strided convolution, (c) and (d). The blue squares represent the input signal, the darker blue is the 3×3 scope of the depicted iteration, the cyan squares represent the results. Notice how a larger input, in the strided case, results in the same output size as the not strided case. Created using the open source implementation of [127].

Strided convolution, first discussed in [128], aims to reduce the number of computations by coalescing the convolution operation with its consecutive dimensionality reduction layer. For example, consider the vanilla case of a swiping a 3×3 convolution kernel over a larger input image and then applying a 2×2 max-pooling kernel to the intermediate result. In the strided case, the convolution kernel is evaluated at every other placement, thus reducing the number of convolutions by skipping, for example, odd indices. Figure 3.5 is provided to better illustrate the process. It shows how the scope of the convolution kernel skips a position for the case of `stride`$= 2$. The mathematical adjustment to Equation 3.5 is merely the introduction of the index multiplier s, representing the desired stride

$$(I \circledast g)[t] = \sum_{m=-M}^{M} I[(ts) - m]g[m]. \tag{3.11}$$

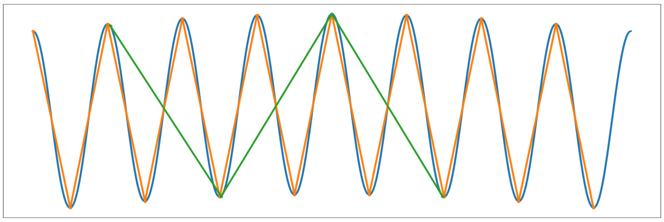
From the surface it appears that strides make for an elegant efficiency optimisation as they both reduce the number of FLOPs as well as obviate the need for a consequential pooling. Unfortunately, the reality is not as simple, mainly for two reasons.

First, for a technical reason. As discussed in Subsection 3.2.2, the computation itself only takes a single order of magnitude longer than the memory access. Depending on the specific hardware in use, the factor four reduction in operations might already divert the bottleneck to the memory bandwidth, thus blocking further improvements.

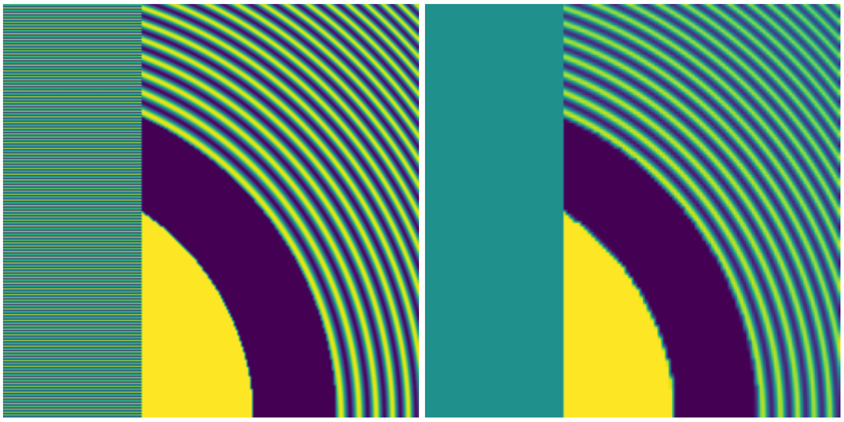
Additionally, systematically skipping kernel placements comes at the cost of spatial aliasing. Aliasing is a term from signal processing which refers to the inaccurate measurement of a signal due to insufficient sampling rates. The effect is more clearly explained using single-dimensional functions such as the one shown in Figure 3.6a. Trying to sample a wave function, one theoretically gets an infinite amount of possible waves between two sampling points. In order to more reliably sample a function, the sampling frequency needs to be increased. That is, more sampling points along the function are required. Luckily, there is a target sampling frequency which guarantees proper sampling of the target function. This frequency is called the Nyquist frequency [129] and dictates twice the highest frequency in the function for aliasing-free sampling.

In the image domain, high frequencies correspond to edges. Consider the example of scaling down an image of circular waves, like the one shown in Figure 3.6b. An awkward random selection of the yellow and the purple pixels only, i.e., the top and the bottom of the amplitude, would result in a yellow-purple stripe, like the one seen in the left part of Figure 3.6c. While the actual representation should include more of the green spectrum, as seen in Figure 3.6b. The pixel skipping, induced by larger stride values, is then equivalent to halving the sampling rate along each spatial dimension, making the result more susceptible to artefacts along the edges.

Using the different frequencies of a signal is often used for various types of processing. A low pass filter is defined as a kernel which removes higher frequencies such that only a smooth surface remains, making it useful for noise reduction. A high pass filter does exactly the opposite and is therefore often used for sharpening images by emphasising the edges. As a reference, Figure 3.7 shows the results of both filters on ten randomly selected traffic sign images.



(a) An example of a 40 Hz one-dimensional signal, in blue, sampled once at the Nyquist frequency of 80 Hz, in orange, and once at 27 Hz, in green. Notice how the orange line correctly matches the signal's amplitude while the green line approximates a completely different function.



(b) An high resolution image with high frequency data, i.e., edges.

(c) The linearly sub-sampled version of (b), showing clear aliasing artefacts both at the horizontal lines and at the higher frequency waves at the right corners.

Fig. 3.6: Examples for aliasing in one-dimensional signals, the stripes on the left of each figure, and two-dimensional signals represented by the circular wave pattern

To evaluate the effects of strided convolution on the performance in different conditions, we evaluated it in an isolated environment. We compared two versions of the same model, one using max pooling and the other using strided convolution with $s = 2$. To push the aliasing effect even further, we introduced a version of the input in which the edges were enhanced, i.e., a high-passed version, similar to the middle row of Figure 3.7. An additional final configuration had its activations passed through a high-pass filter before each convolution layer.

The results, presented in Table 3.2, show that while the max pooling version of the model proves rather robust to the different levels of edges, the strided version

	Max Pooling	Stride 2
Original	**96.70**%	**93.60**%
High Passed Input	96.47%	91.74%
All Layers High Passed	91.19%	81.74%

Tab. 3.2: The accuracy of the same network architecture, evaluated on German Traffic Sign Recognition Benchmark (GTSRB) [130] both using max pooling and strided convolution. One can see that the max pooling is only marginally affected by the high pass filter while the strided version suffers from as much as ~ 12 % accuracy when confronted with high-passed data.

is much more sensitive and quicker to lose accuracy. At the same time, even its baseline performance is worse than the max pooling version.

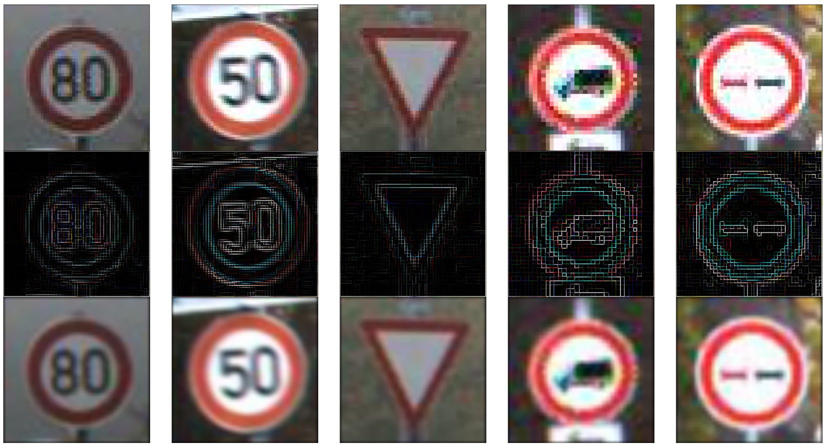


Fig. 3.7: A visualisation of randomly selected images from the GTSRB [130] dataset as:
Top: the original images.
Middle: the respective high-passed version.
Bottom: The respective low-passed version.

The in-depth analysis of the meaning of strides has not only encouraged the use of max-pooling, it has rather also motivated an architecture which optimises the data throughput. The goal is that the target hardware is neither idling while waiting for its data to be fetched from the memory, nor should it be waiting for the computations to finish. Ideally, a well streamlined architecture would utilise all of its hardware's parallel resources to the full, at any given time.

The issues with aliasing have also inspired others. In the recent [131], Ribeiro and Schön studied the ways neural networks handle aliasing. They drew the conclusion that models do not automatically allocate capacity to anti-aliasing. The work of Zhang proposed tackling aliasing in the neural activations [132]. While it did not

	Baseline	Baseline + Residual
Cifar10	**79.30%**	79.12%
SVHN	**91.86%**	91.13%
GTSRB	**95.35%**	92.93%

Tab. 3.3: Examining the effect of residual connections on small architectures. It can be seen how the residual variant is constantly trailing behind its baseline.

explicitly increase classification accuracies, it was extremely beneficial for handling augmented data. I.e., explicitly accounting for aliasing throughout the model has increased its robustness to various transformations.

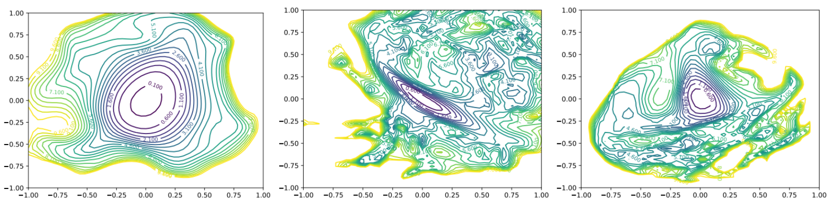
3.3.3 Residual Connections

Initially described by He et al. in [102], ‘residual connections’, also known as ‘skip connections’, have established their place as a simple design trick for stabilising the training of large networks. Nevertheless it was the very same work which showed that this technique is best used with very large and very deep models. The experiments, presented in Table 3.3, confirm this observation. They demonstrate a persistent drop in accuracies across all attempts to include skip connections in the final block architecture.

Unlike the nowadays more common encoder-decoder architecture, the simpler classifier design used here can’t aggregate the data from different resolutions. We have, therefore, experimented with connections similar to the bypass connections as described in [126].

Furthermore, slim models do not require as much stabilisation and regularisation measurements as larger models. This argument was the conclusion in [133], where the loss landscape of many common architectures was studied and visualised. The results, shown in Figure 3.8, clearly show the correlation between the complexity of a model and its depth. It thus seems that the theoretical justification for such connections is also lacking.

Finally, we examine the skip connections from an information flow perspective. There are two common ways to implement skip connections, either by concatenating the ‘skipped’ data to the main signal along the channels dimensions, or using an element-wise addition to fuse the signals. Both methods inflate the amount of data in the signal. As discussed in Subsection 3.3.1, slim models are sensitive to data



(a) ResNet-20 (b) ResNet-56 (c) RestNet-110

Fig. 3.8: A visualisation of the loss spaces of the ResNet model [102] at three different depths, given by number in the model's name. One can see how the complexity of the loss space increases with the size of the model. The more complex the optimisation terrain is, the likelier it becomes that a gradient based optimisation scheme will converge to a local rather the global minimum, harming the model's performance. Taken from [133]

compression and evidently also for inflation which requires a larger capacity to process properly.

3.3.4 Separable Convolutions

Based on the separability properties of convolution [134, p.101] and following the proposal of Szegedy et al. in [135], the notion of spatially separable convolutions is revisited. To do so, we refer to the toy example for this chapter. The goal is to process an RGB input of size $[h_I, w_I, Ch_{in}] = 256 \times 256 \times 3$ using a convolution layer with 128 output channels, i.e., a kernel of size $[h_k, w_k, Ch_{in}, Ch_{out}] = 3 \times 3 \times 3 \times 128$. Hardware and algorithmic assumptions follow Subsection 3.2.2.

The processing requirements of axis-wise convolutions are a strong argument for the significance of separable convolutions. The separable version of the two-dimensional kernels, discussed throughout this section, is two sets of single-dimensional kernels, 3×1 and 1×3. Such an ensemble has a single limitation comparing to the full scale kernels, it cannot simulate asymmetrical kernel functions. Nevertheless, it comes with two alleged benefits.

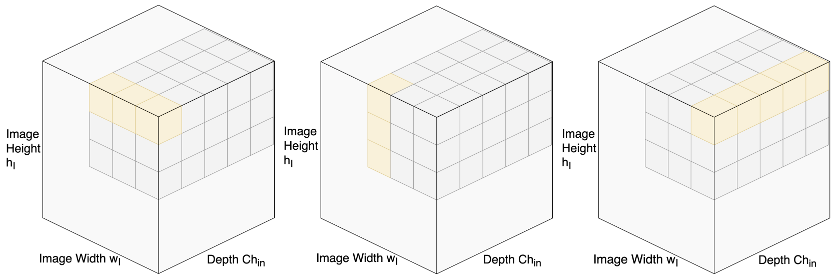
First, as shown in [107], the depth of a neural network is at least as important as its breadth or its input resolution. As the single-dimensional kernels also mean less parameters, the added depth and non-linearity functions act as counter measures to balance out the loss in potential. We have however put this to test by training ShuffleNet [119] once with a ReLU after each convolution layer and once with a single ReLU at the end of each block. Unfortunately, the results, presented in

	ShuffleNet single ReLU	ShuffleNet full ReLU
Cifar10 [29]	71.58%	**72.68**%
SVHN [136]	**83.73**%	82.25%
GTSRB [130]	88.23%	**88.91**%

Tab. 3.4: Comparing the performance of ShuffleNet [119] in two different configurations. Once with a ReLU after each convolution layer and once with a single ReLU at the end of each block. Results are inconclusive.

Table 3.4, are inconclusive, showing no clear preference on any of the benchmark datasets.

Second, by having a one-dimensional pooling operation after each layer, one can half the amount of computation in the consecutive layer. This means that the initial 3×1 convolution is followed by a 2×1 max pooling layer. Then the complementary 1×3 convolution along with its respective 1×2 max pooling follow and complete the approximation. Unlike strided convolution, this does not only reduce the number of FLOPs, but rather also the number of memory accesses. We visualise the different single-dimensional convolution configurations in Figure 3.9.



(a) 1×3 convolution (b) 3×1 convolution (c) 1×1 convolution

Fig. 3.9: A visualisation of the different one-dimensional convolution kernels.

Furthermore, separable convolutions often follow the proposal of Howard et al. of using their so called depth-wise separable convolutions [116]. These refer to a configuration in which each input channel is processed by a single spatial kernel, while there is no flow of information between these kernels, recall Figure 3.2b. Meaning the definition of spatial convolution for two-dimensional signals is linearly extended to the multi-channel domain at a lower computational burden. Turning back to our toy example with 64 input channels, the naive implementation would use 128 spatial kernels on each input channel. It therefore amounts to $64 * 128 = 8{,}192$ spatial kernels. In contrast to the vanilla convolution layer, the basic depth-wise convolution layer has a single kernel per input channel. This means a reduction factor of $\times 128$, or $\sim 66.67\ \%$, in the amount of computations.

The definition of such convolutions comes with a hyper-parameter called depth multiplier (κ, dm). It allows to assign multiple kernels to each input channel, thus countering the drastic reduction to a desired extent. Setting the depth multiplier to 2 would mean that the total number of kernels is increased to 128, with two kernels per input channel. The total number of kernels is thus κCh_{in}, while the premise is that $\kappa << Ch_{out}$.

In computational terms, the influence is also significant. Each kernel placement costs 3 multiplications and 3 additions, repeated along all of the 64 input channels and resulting in 384 operations per placement. In the separable case, the number of placements does not change. For the example layer this means 256×256 placements, resulting in a total of $25{,}165{,}824$ operations and $3{,}145{,}728$ clock cycles. This significant reduction is often overshadowed by a consecutive 1×1 convolution, which increases the computational effort. Yet, this is not always the case and different architectures propose different solutions.

3.3.5 Data Compression

Although FLOPs are the most commonly compared aspect of efficiency, they are certainly not the sole factor. Trying to better understand the influence of algorithmic adjustments to the model's performance, we have also taken the road less travelled, studying further factors. We learnt that the compression rates of the activations throughout a network highly correlate with its performance. The activations are important for two reasons. For one, they carry the information from the input throughout the model to the output. Consider a full scale model, optimised for the Cifar 10 dataset [29] whereas all but ten activations are set to zero after the first layer. It may be intuitive to understand that such a model could not be very successful in object classification. The same logic works for any sort of data compression, regardless of through bottlenecks or 'drop out' [137], each compression is bound to reach the point from which information is permanently and destructively lost.

At the same time, activation compression could prove beneficial as it virtually increases the kernels' size while reducing the computational load. Revisiting the toy example for this chapter, having 32 input channels to convolve, instead of 64 channels, would have the same effect on the computational effort as halving the number of output channels. As for compression along the spatial dimensions,

Vanilla Baseline		MobileNet [116]		ShuffleNet [119]		EffNet (this work)	
Layer	Data Size	Layer	Data Size	Layer	Data Size	Layer	Data Size
$3 \times 3 \times 64$, mp	$16,384$	$3 \times 3 \times 64$, mp	$16,384$	$3 \times 3 \times 64$, mp	$16,384$	$1 \times 1 \times 32$	$32,768$
						1×3 dw, 1dmp	$16,384$
						3×1 dw, 1dmp	$16,384$
						$1 \times 1 \times 64$	$16,384$
$3 \times 3 \times 128$, mp	$8,192$	3×3 dw, s	$4,096$	$1 \times 1 \times 32$ gc4	$8,192$	$1 \times 1 \times 64$	$16,384$
		$1 \times 1 \times 128$	$8,192$	3×3 dw, s	$2,048$	1×3 dw, 1dmp	$8,192$
				$1 \times 1 \times 128$ gc4	$8,192$	3×1 dw, 1dmp	$8,192$
						$1 \times 1 \times 128$	$8,192$
$3 \times 3 \times 256$, mp	$4,096$	3×3 dw, s	$2,048$	$1 \times 1 \times 64$ gc4	$4,096$	$1 \times 1 \times 128$	$8,192$
		$1 \times 1 \times 256$	$4,096$	3×3 dw, s	$1,024$	1×3 dw, 1dmp	$4,096$
				$1 \times 1 \times 256$ gc4	$4,096$	3×1 dw, 1dmp	$4,096$
						$1 \times 1 \times 256$	$4,096$
FC	10	FC	10	FC	10	FC	10

Tab. 3.5: A comparison of the different data compressions throughout different models. s stands for strided convolution, mp stands for max pooling, dw means depth-wise convolution, gc4 follows the definition for group convolutions as presented in [119] and the 1d prefix means that the following term is only applied to a single dimension. One can see the aggressive compression rates as described in [116], but also the persistent compression-expansion in both other works. The numbers marked in purple highlight a compression factor larger than two. Table adapted from [109].

keeping the kernels at 3×3 while reducing the size of their input would allow them to cover more contextual information.

Table 3.5 was compiled to compare the above considerations of data compression, side by side. The table shows how the design proposals from [116] and [119] aggressively utilise the bottleneck principal, as described in Subsection 3.3.1. The issue with such large compression rates is a twofold.

First, embedded models are small to begin with, ideally on the verge of under-fitting, by reducing the data throughput, one risks breaching pass the lower bound of lossless compression.

Second, the usefulness of the data expansion, which follows the compression, is debatable. Could the lost information be recovered? If so, why is the expansion necessary? As neural networks are high-dimensional, non-linear functions which adjust to their given conditions, it is not trivial to find a direct answer. Nevertheless the results, presented in Section 3.5 suggest that the practice is not beneficial.

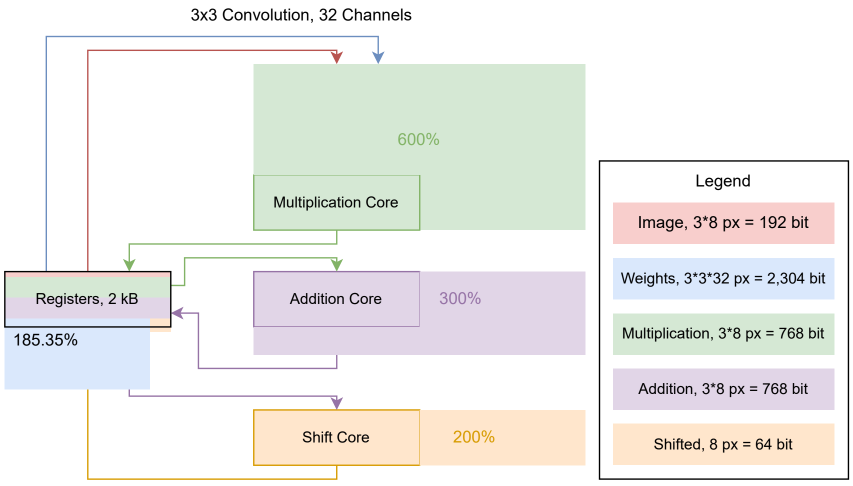


Fig. 3.10: A visualisation of the calculated memory and computational load of Texas Instruments' TDA3 board [123] for a 3×3 vanilla convolution layer with 32 input channels. The visualisation unifies both loads by looking at the status for a **single clock cycle**. The legend also shows the storage requirements of each module while the colour-coded boxes for the registers assume the capacity of 2 kB as a 100% and are normalised to size.

3.3.6 Processing Bottlenecks

Finally, the best type of optimisation is also the least flexible one. Only by profiling the target hardware and its processing load, one could make the final tweaks to get the best possible hardware - software synergy. Directly measuring the load across the architecture is not always feasible. Not all types of hardware support the right tooling for the task. We have therefore compared the numbers throughout this section with the capacity declared by the manufacturer. Specifically, the target hardware was the TDA3 board from Texas Instruments [123].

Figure 3.10 depicts the hardware pipeline with the load representations both for the registers as well as for the different compute cores of the board. To reliably visualise both memory utilisation and the core load, the data is presented as a snapshot of a single clock cycle. That is, how much is expected to be processed by the algorithm versus the actual capacity of the respective part. The reference TDA3 board [123] from Texas Instruments has multiple dedicated cores for computations such as the depicted multiplication, addition and shift.

Each core has a maximum capacity and a maximum throughput. These are represented as the size of their respective graph node. E.g., the framed part of the 'Addition Core' node represents the core at a $100\ \%$ load. The framed part of the 'Registers' node represents the entire 2 kB of memory.

The borderless rectangles are used to visualise the actual load on the core/memory when evaluating a vanilla 3×3 convolution kernel on an arbitrary input. It can be understood that the operation is compute bound, i.e., the load on the compute cores is responsible for the bottleneck. The main bottleneck is at the multiplication core, which has six time more to calculate than its capacity. The addition and shift cores are also overloaded. Moreover, since the registers are also full, meaning they rely intensively on the L1 cache to support the disproportional storage demand of $\sim 185\ \%$. As discussed in Subsection 3.2.2, the L1 fetches take four times longer than a register fetch, another bottleneck is thus created at the memory level.

For example, per kernel placement, 3×3 multiplications are required. As seen in Figure 3.4, there are eight placements for each three rows loaded from the memory. The hardware level instructions only support the multiplication of eight elements, twice, each cycle. These eight elements are split into 2×4 whereas the two stands for the rows and the four is the amount of elements multiplied. Since the kernel is only three elements wide, the last element is a stub, multiplied with a zero. Applied twice, once per multiplication core, this results in 16 multiplications per cycles including the stub and 12 multiplications per cycle of relevant data. At eight loaded pixels from the required three rows, one gets $8 * 9 = 72$ multiplications. At 12 multiplications a cycle, the final load of $600\ \%$ is calculated.

In this section, several different aspects of architecture optimisation were discussed. Opening with the bottleneck blocks, their incompatibility to the domain of smaller network was demonstrated. The strided convolution was shown to drastically reduce the computational demand, but also suppress accuracies by encouraging aliasing. The evaluation of residual connections matched the expectation and observation of their inventors. They are only beneficial for large and very deep models. As this is not the case with models for embedded architectures, they too were deemed unsuitable. Finally separable convolutions and data intra-model compression rate were examined and showed promising results, if properly handled. In the next section, these lessons are used to realise a novel efficient convolutional block.

3.4 Introducing Our EffNet Block

Following the thorough scrutiny of the varied design aspects, we came to propose a novel convolution block, designed specifically for small and efficient networks, hence its name, 'EffNet'. Each block consists of a total of four convolution layers, each is followed by a leaky ReLU [138].

The first layer is a depth-wise, single-dimensional spatial convolution, accompanied by a one-dimensional max pooling operator along the same dimension. Throughout this chapter, this layer is regarded as a 1×3 convolution, however depending on the specific target hardware, swapping the order of both spatial convolution layers might prove beneficial. For example, if the hardware at hand has three parallel cache lines or a command to merge, multiply and add all eight placements in a single instruction.

The third layer compliments the previous one by convolving the second spatial axis with a 3×1 kernel.

A final weighted pooling layer [139] with the kernel size 2×1 is applied and used to expand the channels, often by a factor of two. The decision to use weighted pooling was empirical.

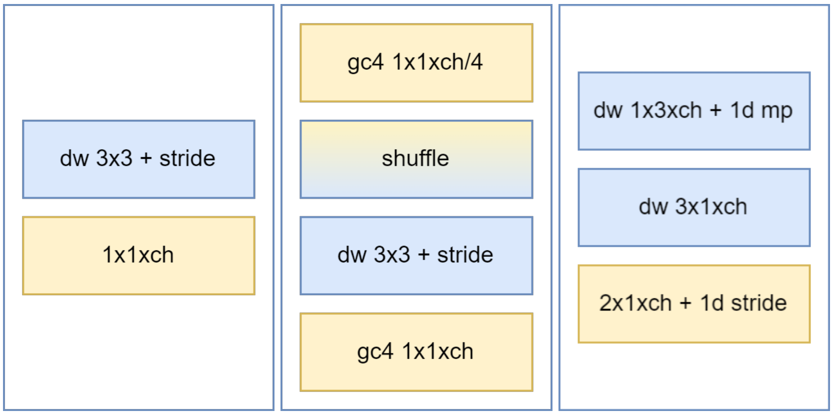
Finally, please notice that all depth-wise spatial kernels are used with a depth multiplier factor of two ($\kappa = 2$).

For the sake of clarity, a visualisation of the block, along with the MobileNet block [116] and ShuffleNet block [119] is presented in Figure 3.11.

All of the aforementioned blocks are flexible enough to be compounded into a full size network structure. The resulting models are then evaluated on several benchmarks and compared in the following section.

3.4.1 Multi-aspect Optimisation

Following the elaborate study of the different efficiency strategies, this section discusses the design of the EffNet block from the same perspective. We show how the block design significantly improves on many of the issues discussed in Section 3.3.



(a) A MobileNet Block [116] (b) A ShuffleNet block [119] (c) An EffNet block [109] (ours)

Fig. 3.11: A comparison of the MobileNet and the ShuffleNet architectures with the EffNet architecture, presented in this chapter. `dw` means depth-wise, `mp` means max pooling, `gc` means group convolutions and `ch` represents the number of channels. The blue nodes represent spatial operations while the yellow nodes represent point-wise operations. A gradient is used for the shuffling operation, since it is a point-wise operation which is supporting the consecutive spatial operation. Based on Fig. 1 of [109].

Replacing the First Layer Both MobileNet and ShuffleNet use vanilla convolution for their first layer. The aim of this work was to propose a unified architecture for an entire model. As a compromise, we proposed to extend the first EffNet block in the model with a preceding point-wise convolution. Here, the number of channels is determined by the so called 'expansion rate' (e_{rate}) which is defined as

$$Ch_{out} = \left\lfloor \frac{Ch_{in} e_{rate}}{2} \right\rfloor. \tag{3.12}$$

This hyper parameter does not have a significant meaning for the model, as it is mostly kept at $e_{rate} = 2$, yet it allows for a direct comparability with MobileNetV2 [140] and is therefore included in the system.

Bottleneck Blocks and Data Compression Having established that bottleneck architectures implement an inherit design flow, the EffNet block implements the classical paradigm of reduction by a factor of two. It means that the throughput is decreased by a factor of two at each reduction layer. In the case of two-dimensional strided convolution and pooling, this is achieved by increasing the number of channels to account for the loss in spatial resolution. By doing so the model enjoys enough throughput for parsing and processing the input signals, before further reducing them in a following layer. The effect of this design decision to the actual compression factors is shown in Table 3.5.

Strides and Pooling The results shown in Table 3.2 make a strong argument against strided convolution. We have also experimented with a depth-wise version of weighted pooling [139]. While the theoretical advantage is the performance gain at a very low computational cost, the experiment results were indecisive, leading us to keep the classical max pooling algorithm for the first sub-sampling layer. For the second subsampling weighted pooling is used. Weighted pooling is, in this case, a 2×1 convolution layer applied with a stride of 2. Such a layer merges the single-dimensional max pooling with its following point-wise convolution and is thus computed at an extremely low computational cost.

Residual Connections Models for interactive execution on embedded systems are notably shallower than the current SotA. This observation alone is enough to nullify the main alleged advantage of residual connections. We conclude the discussion with the understanding that residual connections, although popular, are counterproductive for designing a slim and computationally efficient model. We thus refrain from using them throughout the EffNet architecture.

Separable Convolutions One of the more significant adjustments to the proposed architecture, comparing to other models, is the separation of convolution dimensions. On top of the clear reduction in computations, it also allows us to effectively sub-sample an entire dimension, before processing the next one. By doing so, the input resolution is halved along the column dimension, leading to 256×128 placements. These translate to $12{,}582{,}912$ operations and $1{,}572{,}864$ clock cycles.

The time spent accessing the memory is also simplified. The row-wise layer follows the same computation as the the vanilla layer. The sole difference is that it is not accessing three rows for each placement, thus resulting in $Ch_{in}h_I w_I/8 = 524{,}288$ calls. The column-wise layer implements a different logic. While requiring three parallel cache lines, it benefits from its position after the row-wise pooling layer, resulting in less columns to consume. The computation requires only $3Ch_{in}h_I w_I/2/8 = 786{,}432$ calls.

The last layer has no spatial resolution, i.e., a so called point-wise convolution [141] with 1×1 kernels. It rather spans across the channels, combines and increases them from the 64 input channels up to the 128 output channels. A kernel placement costs $[h_k w_k Ch_{in} Ch_{out}] = 1 * 1 * 64 * 128 = 8{,}192$ multiplications and the same amount of additions. These $16{,}384$ operations are evaluated $[h_I w_I] = 256 * 256 = 65{,}536$ times, amounting to a total of $1{,}073{,}741{,}824$ operations or, following the factor eight parallelisation assumption, $134{,}217{,}728$ cycles.

The memory access is rather straight forward and amounts to $[Ch_{in} h_I w_I / \upsilon] = 64 * 256 * 256/8 = 524{,}288$ calls and $56{,}098{,}816$ cycles, where υ represents the memory bandwidth.

Finally, adding the operations from all three axis-wise layers, one gets $524{,}288 + 786{,}432 + 524{,}288 = 1{,}835{,}008$ memory reading calls, resulting in $196{,}345{,}856$ cycles. The computational side costs $25{,}165{,}824 + 12{,}582{,}912 + 1{,}073{,}741{,}824 = 1{,}111{,}490{,}560$ operations and $138{,}936{,}320$ cycles.

Looking at the corresponding vanilla convolution layer for reference, it exhibits $1{,}207{,}959{,}552$ operation cycles and $168{,}296{,}448$ memory access cycles. This means that for a minor increase in memory access time to $\sim 116\ \%$, a large portion of the bottleneck is resolved and an entire order of magnitude less runtime is achieved, $11.50\ \%$ from the baseline operations.

This aggressive reduction in complexity comes almost without a cost to the model's accuracy, as shown in Section 3.5.

Processing Bottlenecks Finally, we calculate the same sort of loads as done for Figure 3.10 to evaluate an EffNet convolution layer. The results are shown in Figure 3.12. Immediately, one can see that the computational bottleneck was drastically reduced to a third of its vanilla size. I.e., the multiplications take two cycles instead of six. The registers overflow was also unclogged, allowing for a more efficient memory loading time which is as much as four times quicker comparing to the vanilla case.

Unfortunately, the reduction also comes at the cost of additional layers which further increase the runtime. Nevertheless, dedicated profiling of the layers on the target hardware show a net reduction of cycles to $\sim 52\ \%$, for the entire EffNet block. As a reference, a similar profiling of a matching MobileNetV1 layer resulted in $\sim 76\ \%$ of the baseline's number of cycles.

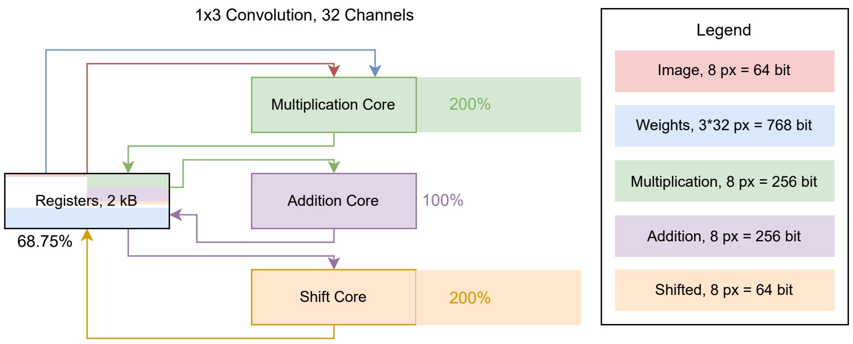


Fig. 3.12: A visualisation of the calculated memory and computational load of Texas Instrument's TDA3 board [123] for a 1×3 depth-wise convolution layer with 32 input channels. The visualisation unifies both loads by looking at the status for a **single clock cycle**. The legend also shows the storage requirements of each module while the colour-coded boxes for the registers assume the capacity of 2 kB as a $100\ \%$ and are normalised to size.

3.5 Evaluation

For the evaluation and comparison datasets were sought, which match the target use-case of automotive applications. A narrow domain, with a limited number of classes, sorted by a small model which should present interactive runtimes on a minimal embedded hardware. This definition significantly reduces the number of datasets to choose from, as the most common vision datasets, like ImageNet [98], usually take pride of terabytes of images and thousands of classes. The three well represented benchmarks which were finally selected are

- Cifar10 [29] - published back in 2009, this dataset is often referred to as one of the main enablers of the deep learning renaissance of the last decade. The data itself is a good match to the scope of this project. It exhibits ten classes, each represented by $6{,}000$ images of size 32×32 pixels. Another positive trait is the predefined split into a train and a test set, which means that all users report their numbers on the same samples, allowing for a reliable comparison with other works.

- Street View House Numbers [136], another early days dataset, doing exactly as suggested by its name. It presents number, gathered from Google's Streetview data and divided into single digit-wise labels. Similar to Cifar10, the input size is 32×32 pixels and the $\sim 90{,}000$ images are presorted into training and testing data.

- German Traffic Sign Recognition Benchmark [130] - a slightly more extensive dataset with larger images of variable sizes between 30×30 pixels and 60×60 pixels. It covers 40 classes of various traffic signs collected on German roads and extracted to only include patches of the traffic signs themselves.

The rest of this section follows [109] and is divided into two parts. First we cover the initial comparison to [116] and [119]. Less than 48 hours prior to the initial publication of our work, the MobileNetV2 model was published [140]. For a better reference, we extended our work by an additional set of experiments. Since the experiment configuration is not the same, these two sets of experiments are kept apart in this work as well.

During all experiments, in both sections, each configuration was trained and evaluated five times to cancel out the effects of randomness. Furthermore, neither data augmentation nor pre-training on additional data was used. The baseline model was composed with a size, for which it was empirically established that an execution in real-time is feasible. The layers of this baseline model were then replaced in favour of the alternative convolution blocks. The implementation was done in Python [142] and Tensorflow [143] and trained using the Adam optimiser [144] with a learning rate of $\lambda = 0.001$ and $\beta_1 = 0.75$.

3.5.1 Initial Comparison

Throughout the experiments in this section and since the 'EffNet' architecture is significantly less compute intensive than its baseline, an additional experiment was done to evaluate the accuracies of an EffNet model inflated to roughly the same amount of FLOPs as its baseline. These results are provided along the other models and are dubbed 'EffNet large'.

Cifar10

Table 3.6 shows the different experiments on this dataset. First and foremost, our EffNet model demonstrates SotA results while requiring factor ~ 7 less computations than the baseline. Nevertheless, since both other models require roughly half the computations of our model, the question arise regarding the reason for the improved accuracy. We therefore trained a larger version of both models, which roughly matches the amount of computations required by our model. The results show that while their performance did improve by a small margin, they still clearly remain behind the EffNet accuracy. Furthermore, the accuracy of 'EffNet large' also surpasses the baseline accuracy.

Model Name	Mean Accuracy	Mil. FLOPs	Factor
Baseline	82.78%	80.3	1.00
EffNet large (ours)	**85.02%**	**79.8**	**0.99**
MobileNet [116]	77.48%	5.8	0.07
ShuffleNet [119]	77.30%	**4.7**	**0.06**
EffNet (ours)	**80.20%**	11.4	0.14
MobileNet large [116]	78.18%	11.6	0.14
ShuffleNet large [119]	77.90%	11.1	0.14

Tab. 3.6: A model comparison on the Cifar10 dataset. Data from [109]

Street View House Numbers

The results on SVHN, shown in Table 3.7, confirm those of the previous experiment. One trend is clearly visible though, while the reference models for the Cifar10 experiment required less FLOPs than the EffNet model, here a different trend is shown. The EffNet model delivers higher accuracy while being less compute intensive at the same time. The reason for this difference is the portion of the first layer in the total number of FLOPs. Since both MobileNet and ShuffleNet use vanilla convolution as a first layer, the reduction in FLOPs for this layer is the most significant. Furthermore, the 'EffNet large' model surpasses the baseline while requiring less computations.

Model Name	Mean Accuracy	kFLOPs	Factor
Baseline	91.08%	3,563.5	1.00
EffNet large (ours)	**91.12%**	**3,530.7**	**0.99**
MobileNet [116]	85.64%	773.4	0.22
ShuffleNet [119]	82.73%	733.1	0.21
EffNet (ours)	**88.51%**	**517.6**	**0.14**

Tab. 3.7: A model comparison on the SVHN dataset. Data from [109].

German Traffic Sign Recognition Benchmark

Finally, the GTSRB experiment shows an even stronger trend in Table 3.8, with the EffNet model being both the most efficient and the most accurate among the optimised architectures. Also the 'EffNet large' model surpasses the baseline accuracy while only requiring ~ 93 % of its computations.

Model Name	Mean Accuracy	kFLOPs	Factor
Baseline	94.48%	2,326.5	1.00
EffNet large (ours)	**94.82%**	**2,171.9**	**0.93**
MobileNet [116]	88.15%	533.0	0.23
ShuffleNet [119]	88.99%	540.7	0.23
EffNet (ours)	**91.79%**	**344.1**	**0.15**

Tab. 3.8: A model comparison on the GTSRB dataset. Data from [109].

3.5.2 Comparison with MobileNetV2

In the second batch of experiments, we compared the proposed model to MobileNetV2 as proposed in [140] which saw light parallel to our original publication. The following tables show a comparison of the different expansion rates, which are the main hyper-parameter proposed by Sandler et al. Each model was evaluated on three different expansion rate configurations: 2, 4 and 6. The results, which are shown in Table 3.9, Table 3.10 and Table 3.11 show that MobileNetV2 favourably compares to our model in most configurations by around $1\,\%-3\,\%$ FLOPs comparing to the baseline. Yet the EffNet model consistently draws the higher accuracy, often by a larger margin of up to 5 %, as seen in Table 3.9.

Ex. Rate	Model Name	Mean Acc.	Mil. FLOPs	Fact.
	Baseline	82.78%	80.3	1.00
6	EffNet	**83.20%**	44.1	0.55
	MobileNetV2	79.10%	**42.0**	**0.52**
4	EffNet	**82.45%**	31.1	0.39
	MobileNetV2	78.91%	**29.2**	**0.36**
2	EffNet	**81.67%**	18.1	0.22
	MobileNetV2	76.47%	**16.4**	**0.20**

Tab. 3.9: A comparison of MobileNetV2 [140] and our EffNet [109] on the Cifar10 dataset for various expansion rates. Data from [109].

Ex. Rate	Model Name	Mean Acc.	kFLOPs	Fact.
	Baseline	91.08%	3,563.5	1.00
6	EffNet	**87.80%**	2,254.8	0.63
	MobileNetV2	87.16%	**2,130.4**	**0.60**
4	EffNet	**87.49%**	1,729.5	0.49
	MobileNetV2	86.93%	**1,646.6**	**0.46**
2	EffNet	**87.30%**	1,204.2	0.34
	MobileNetV2	86.71%	**1,162.8**	**0.33**

Tab. 3.10: A comparison of MobileNetV2 [140] and our EffNet [109] on the SVHN dataset for various expansion rates. Data from [109].

Ex. Rate	Model Name	Mean Acc.	kFLOPs	Fact.
	Baseline	94.48%	2,326, 5	1.00
6	EffNet	**93.74%**	1,208.3	0.51
	MobileNetV2	92.82%	**1,159.2**	**0.50**
4	EffNet	**92.30%**	956.4	0.41
	MobileNetV2	91.56%	**934.9**	**0.40**
2	EffNet	90.40%	**704.5**	**0.30**
	MobileNetV2	**90.74%**	710.7	0.31

Tab. 3.11: A comparison of MobileNetV2 [140] and our EffNet [109] on the GTSRB dataset for various expansion rates. Data from [109].

3.6 Conclusions

Throughout this chapter, the different design aspects were discussed, which are commonly used to reduce the computational burden of convolutional neural networks. We showed that there are many possible tricks which make networks more computationally efficient, yet these often bring along new challenges. From isolated tests and experiments, we showed that from data compression, through memory efficiency and to the very type of convolution, each of these factors is responsible for a small portion of the solution, but also of the problem. Optimising only the number of FLOPs, as seen in many other works, does not seem to reliably solve the problem, but rather shift it to an unobserved aspect.

Our novel solution was proposed in the form of the ‘EffNet’ block. It tackles the optimisation task from several different angles at the same time. While it does not always have the upper hand in terms of FLOPs, its overall performance is superior to its competitors.

The field of efficient architectures has been enjoying positive sentiment. Our original publication, [109], followed MobileNetV1 [116] and ShuffleNet [119], and was published in parallel to MobileNetV2 [140]. It was furthermore cited over 80 times according to Google Scholar, as of early May, 2022.

Additional high-impact publications have also followed, among which are MobileNetV3 [117] and ShuffleNetV2 [145]. While these publications might sound interesting for further evaluation, a deeper look into their proposed architectures shows that the field is starting to converge. The differences between publications are decreasing to the point in which the choice of architecture becomes redundant.

Finally, recent works have established that depth-wise convolution suffers from accuracy degradation when quantised to 8 bit [146], [147]. While the minimisation of quantisation impact is an active field of research, we found that methods like merging layers significantly reduce the severity of the matter. The advantages of layer-merging were also discussed in [148] as a possible solution.

4 Polynomial Predictions as a Strong Regulariser

4.1 Introduction

This thesis is mostly concentrated around various favourable constraints and biases for NN architectures. Following the adaptive masking of Chapter 2 and the efficient CNN blocks of Chapter 3, this chapter covers an additional form of a-priori biasing. Developed for the field of trajectory prediction in motorway scenarios, we propose a novel polynomial output layer which is dedicated to predicting continuous movement over time. This structural bias, published in [149], is shown to increase both generalisation and robustness of the trained models.

From AEB in the widespread level 1 Advanced Driver Assistance System to the complete automation of level 5, the scale of features might increase, yet many of them extensively rely on motion understanding [150]. Consider an Automatic Emergency Breaking system. Early anticipation of dangerous situations and slowing down in advance would reduce abrupt breaking. Such a system would not only increase the comfort for the passengers, but would rather also save lives.

At the top of the automation hierarchy, the fully automated level 5, it is hard to imagine a system without robust motion understanding capabilities. These make for a vital part of planning and reacting. They, therefore, are an absolute necessity for every fully autonomous decision-making. Luckily, the reasoning behind motion understanding is not complicated to explain.

As humans, at a driver's capacity but also as pedestrians, the first thing we do when deciding on an action, e.g., a lane change, is to look around and read the road. People mostly do that using their 'System 1' brains [151], which is fully capable of handling clear low risk cases. Upon seeing another agent which might pose a risk, the 'System 2' brains is seamlessly activated to observe the situation with increased attention.

Notable Work Amongst the first works in the field was the 1995 publication of [152]. In their work, Helbing and Molnar focused on modelling pedestrians moving in a crowd. The relevancy of each surrounding agent for the ego motion was described by a set of forces which shape the trajectory planning. In spite of the lacking datasets for evaluation, the work has created a baseline and a new field. Addressing trajectories as the result of social forces is still a common practice and has seen support in publications like [153], [154], [155].

Other works have looked at the mechanics of the movement itself, before attending to the interactions between agents. Since kinematics is a well studied field in physics, the gained knowledge could be utilised to improve models. Works like [156] and [157] have recently introduced different approaches for integrating kinematic knowledge into neural networks.

Finally, an additional class of works is focusing on the uncertain nature of predicting the future. With free choice as an agency, there always remains an unknown factor in possible trajectories. Accounting for this plurality of possible futures, works like [158] and [159] proposed novel multimodal prediction frameworks.

Datasets Following [152], there emerged a requirement for data to test and evaluate on. Here one can find the Collective Activity dataset [160], the ETH dataset [153] and the Zara/UCY dataset [161]. All published with accompanying annotated camera recordings of pedestrians in outdoor scenes. Such datasets mostly use the footage from high fidelity security cameras, often of students walking around the institute which collected the data.

It was only at a later phase that the field branched again to cover vehicles and motorway scenarios. One of the earlier datasets here was the NGSIM dataset [162]. It was published in 2006 and has super-scaled the methods previously used for pedestrians. The US Department of Transportation has strategically placed traffic control cameras along, mainly, two motorway segments in California, each covering roughly 500 metres of road. They then analysed the traffic over four 15 minute periods during the day, tracked the passing vehicles and normalised the data to a global coordinates system.

For many years, this has been the dataset of choice for development and evaluation with currently around $4,500$ works referring to it. Yet, the progress of recent years in ML and model accuracies has reached the edges of what could be achieved with this dataset. Modern models require large amount of data along with a large temporal and geographical variance. Furthermore, in the meta study reported in

[163] inconsistencies in the data were shown, in the form of random noise and unrealistic interactions. These include, for example, two cases of phantom collisions, sudden and unrealistic acceleration, etc.

In 2018, the HighD dataset was published [164]. The institute behind the dataset was licensed to deploy drones over motorways for data acquisition. This allowed them to successfully record and annotate some 16.5 hours of data with a total driven distance of $\sim 45{,}000$ km. The choice of drones is well motivated as well. They are more flexible than fixed monitoring cameras while not biasing the road as much as an attention catching data-collection vehicle. Additionally, such a data-collection vehicle can only collect the data around it, missing possible long-term interactions.

While the acceptance of the dataset is bound by its limiting licensing terms, it is to date the most extensive dataset of vehicle movement understanding in the motorway domain. Its size and quality have enabled interesting works like the adaptation of Graph Neural Networks (GCNs) to the field in [165], reinforcement learning in [166] and attention pooling in [167].

The problem of licensing still remains an issue in the field. Since many efforts are supported by the industry, data is often acquired for a specific purpose and is never published. Yet, be it in the vision domain, radar processing or even a multi-sensor environment, trajectories have a wildly important property which was generally overlooked in neural architectures, their spatio-temporal continuity. Experiencing the world at any scale above the subatomic level, a movement from A to B must traverse a continuous trajectory.

Temporally Consistent Trajectories Many of the current SotA models predict an independent set of spatial coordinates. By using a vanilla fully connected layer to predict future coordinates, the inherent spatio-temporal dependency becomes weakly adherent. Meaning, it is only indirectly learnt as a correlation during training.

As repeatedly seen throughout this work, the right kind of bias tends to favourably affect performance. Furthermore, an architectural constraint which is confined to the rules of physics poses neither a limitation, nor a disadvantage. Notice algorithms, like the one proposed in [168], which explicitly provide a map of the drivable area to discourage invalid predictions, see Figure 4.1. By limiting the model to the subset of physically plausible predictions, we implement Ockham's razor [104] by removing degrees of freedom which are beyond necessity.

In this chapter, we propose a replacement for the common art coordinates prediction in favour of the marginally more complicated, yet more physically sound polynomial trajectory prediction. The polynomial prediction layer acts as an output layer of an arbitrary NN architecture, predicting the axis-wise polynomial coefficients. These represent a function, mapping the temporal axis to one of the spatial axes, the longitudinal axis, x, or the lateral axis, y.

Since the conclusion of the original project in late 2019, published in [149], the proposed algorithm has enjoyed a certain popularity within our research group. It became a part of and extended in

1. [169] which adjusted the polynomial framework for tracking.

2. [170] which extended the said tracker to support our proposed variance prediction capacities.

3. [171] which used the polynomial framework to predict the acceleration of surrounding vehicles.

The focus of this chapter will be on [149], as it is my main first-author contribution to the topic. Nevertheless, the additional spawned projects are also mentioned for they deliver valuable conclusions and were a pleasure to contribute to.

Please also notice that the trajectories in this chapter are drawn as a function of their position from the starting point while the evaluation are given in Average Displacement Error, representing the actual error, comparing to the ground truth signal.

4.2 Related Work

This work builds on the findings of a relatively large number of works from several different fields. This section covers the main developments from those various fields. We look at early works in trajectory prediction, more recent ML based models and algorithms dedicated for vehicle trajectory prediction. We then branch to variance estimation, multi-modal prediction and their relation to each other. We continue with an overview of models which implement different levels of kinematic constraints and finally conclude with the main works to address the intersection of polynomials and neural networks.

4.2.1 Pedestrian Trajectory Prediction

The social forces model As discussed in the previous section, trajectory prediction predates the current ML resurgence. Early work in the field is often attributed to [152], published as early as 1995. Helbing and Molnar defined a set of attractive and repulsive forces as a physical system to describe pedestrians' movement in a crowd. An example for such an attractive force is the acceleration to walking velocity, i.e., pedestrians are naturally drawn towards their comfortable walking speed. An example for a repulsive force is the field around other pedestrians, meaning that pedestrians would generally prefer to avoid collisions.

In order to properly work, the proposed simulation model needed a large crowd, resulting in scenarios which resemble fluid dynamics. One could also argue that the large amount of pedestrians reduces the search space to a small number of plausible trajectories. But nevertheless, the resulting simulations appear empirically feasible.

Over the years, the concept of a handcrafted framework for trajectory prediction was advocated by works such as, for example, [154], [172] and [173].

The former looked at the use case of detecting emergencies in security footage. By estimating the social forces between agents in a video, they looked for local regions of abnormally large repulsive forces. These are then further observed as candidates for panic situations, mass-escaping, etc.

Later, Leal-Taixé et al. proposed a graphical model for the interactions between agents [172]. With each agent being a node in a graph, the intrinsic forces, e.g., acceleration to walking speed, were modelled by recursive potentials, similar to the logic discussed in subsubsection 2.2.3. The extrinsic forces, also named interactive forces, were modelled by the edges connecting the nodes. Such a graph also enables the building of spontaneous clusters which are referred to as 'collective activities'.

Finally, Choi and Savarese studied the grouping habits of pedestrians, recognising that collision repulsive forces are irrelevant in close groups, walking together [173]. Thus, when multiple agents travel in proximity to each other over time and at a similar velocity, they assume a group-definition and the collision avoidance term is nullified.

ML based solutions One of the first works to address trajectory estimation using NNs is [174] with the proposed Social LSTM model. Here, for each agent an instance of the same NN is created with the past trajectory as an input and the future trajectory as the ground-truth output. For the interactions between the different pedestrians, each agent instance is also provided the states of its neighbouring agents. The challenge of a variable number of surrounding agents is solved by assigning the agents to a grid and adding the activations of all agents in the same grid. The process was dubbed 'social pooling' and leads to a fixed-size vector for all agents at each step. The work has also shown that a blind LSTM, i.e., an LSTM without context, is less performant than classical methods. Yet, once the social aspect is added, it outperformed the references on almost all benchmarks.

The concept of social pooling was further developed in [175], which became one of the more impactful papers in the field. It considers an LSTM based prediction model which is learnt under the Generative Adverserial Network (GAN) framework [176] for all agents in parallel. Here, the handcrafted series of forces is replaced by a pooling mechanism which is designed to encode an interaction state for each pedestrian, allowing the neural network to learn the relevant interactions during training. The weight of each state is supported by the euclidean distance between each agent and all other agents.

Finally, the ability to quickly scan through marginally important inputs and focus on the more acute ones is not limited to humans. Following the 2017 introduction of attention modules in neural networks [177], Sadeghian et al. covered the adaptation of the technique to trajectory prediction [178]. They proposed a dual-attention network, with one social attention which resembles the model proposed in [175] and an additional environmental attention. Given the fact that pedestrians are confined to the walkable parts of their environment, it is easy to imagine how accounting to this environment would be beneficial. Sadeghian et al. have rectified their scenes into a map-like grid which was then processes by a convolution module [178]. Utilising this additional piece of the puzzle, their model demonstrated a marginal leap in performance on all benchmarks.

4.2.2 Vehicle Trajectory Prediction

Motivated by automotive applications and industrial resources, some of the research was also diverted to prediction of vehicle trajectories. This domain enjoys the benefits of a reduced dimensionality and degrees of freedom, as pedestrian movement is not as regular as that of moving vehicles. At the same time, products which require motorway trajectory anticipation are already in production while urban scenarios are still under active research.

A few of the more interesting works here are the aforementioned [165], [166] and [167]. First, in [165] the different types of graph convolution were evaluated, both on NGSIM as well as on the HighD dataset. The authors concluded that the best performance for the task is achieved using a Graph Attention Network (GAT) with residual connections and local connectivity, as opposed to a fully connected graph.

Another interesting finding was that the results on the HighD dataset were not significantly affected by the model's architecture. This was also true with the simple fully connected model, which was trained as a reference. A further analysis of the data supported the hypothesis that the trajectories in the dataset are not highly dependent on interactions. Supporting the hypothesis is also the Constant Velocity Model which presented a displacement error of 2.66 metres at five seconds. Meaning that the simple "nothing changes" assumption performs better on HighD than the best NGSIM model.

The work of Krasowski et al. described a pseudo reinforcement learning environment on top of the HighD dataset [166]. Instead of enforcing the measured location of the target agent, the model was allowed to drive as it deemed fit while a series of reward terms defined its final learn-signal. First and foremost, the simulated agent should have reached the position of the observed agent. Then, much like the attractive and repulsive forces, a set of positive and negative reward terms was defined. For example, a crash results in a strong negative reward, reaching the target position means a strong positive reward, etc.

The logical introduction of attention models to the field could be found in [167]. In their model, Messaoud et al. have described an agent-wise LSTM, similar to the one seen in [174]. Here, instead of an arbitrary pooling mechanism, the model was trained to assign attention weights to all neighbouring agents [177].

A completely different strategy for consuming road data could be found in [179]. Here, instead of the sequential representation of movements, a binary mask was created for each semantic class. These semantic classes were from the likes of lane markers, vehicles, current ego agent, etc. Each mask makes for a channel in a two-dimensional spatial image and all masks are concatenated along the channels dimension. The multi-channel input signal is then processed by a CNN and passed onwards to an RNN in the form of a state vector.

4.2.3 Variance Estimation

The concept of predicting the parameters of a Gaussian distribution was first published in [180] which coined the term Mixture Density Networks. However the tech report seems to have been published almost 20 years ahead of its time. In 2013 the field saw its first high impact work in [181]. The scope was to convert text into handwriting using an LSTM. Since pencil strokes are not discrete though, there had to be a naturally appearing randomness in the results.

The solution came in the form of a mixture of M bivariate Gaussians. Rather than predicting the spatial coordinates of each point in the sequence, Graves predicted the mean and standard deviation corresponding to a point. One would then sample from the Gaussians to get the drawing position. This sort of sampling from a distribution bridged the gap between discrete predictions and the desired natural randomness.

The concept of predicting the mean and variance of data points was since revisited in various works from the likes of [174], [178], [182] and many others.

4.2.4 Multi-modality

Multi-modality addresses the uncertainty aspect in a broader sense than the variance prediction. At a given point in time, an agent can take any number of actions, leading to multiple possible outcomes. For example, one could accelerate or break, overtake or keep following. Such decisions are motivated by acute reactions, e.g., emergency braking, as well as long term planning, e.g., following the planned route to destination. Models are regarded as multi-modal if they allow for different, plausible predictions given the same input. Additionally, variance estimation and multi-modality predictions are complimentary methods. Meaning that each predicted modality can include its own estimated variance.

Traditionally, there have been a few main model designs which support multi-model predictions. First, multi-headed networks. I.e., networks with multiple output layers, each predicting a different modality. One of the earlier representations of this class of algorithms is brought in [158]. It proposed multiple predictions for each input signal. The final loss was dominated by the most reliable prediction w.r.t the ground truth label. Interestingly enough, Rupprecht et al. presented appealing results on various tasks from object classification to pose estimation. Yet, they did not assess their performance on trajectory prediction.

The extension to behaviour understanding was presented in [159]. This work of Cui et al. used the same concept on their unpublished data and showed visually appealing results. The work has also coined the term ‘min of k’ which has been used ever since to describe the procedure of predicting k times on the same input while only using the best prediction for the training signal.

An alternative to the ‘min of k’ algorithm was proposed in [155]. Instead of using the ground truth labels during training to pick the best modality, the labels were used as an input to the model, conditioning it on a desired modality. The main advantage of this method is that one can query the model for certain plausible futures. Example queries could be imagined as "what would a lane change to the left look like?" or "simulate a braking scenario". This ability to examine specific modalities is helpful in various applications. E.g., an Adaptive Cruise Control (ACC) system which needs to ensure a sufficient braking distance at any given time. Nevertheless, it also limits the types of learned modalities to a predefined set of classes.

Another class of models utilises generative models, which by design include a stochastic term. By doing so, the need for multiple prediction heads is made obsolete. In GANs [176] the variation comes from the sampled noise while the randomness in Varational Auto Encoders (VAEs) [183] originates from the random sampling of the predicted mean and standard deviation. In terms of losses and target modalities, both architectures inherently support both of the aforementioned strategies.

Finally, consider the class of occupancy grid maps. Here, instead of an agent-wise trajectory prediction, the models are trained to predict a heat map of the different agents on a grid map of an arbitrary resolution. The grid map is designed to correspond to the temporal dimension and cover future discrete offsets in time. By using a positional distribution, the model can predict multiple modalities simply by dividing the probability along several grid cells. Thus the multi-modality is made an inherent property of the model. This class of works is represented by works like [184], [185] and also our group’s [186] and [168]. An example for such a grid map is provided in Figure 4.1.

4.2.5 Kinematic Constraints

While focusing on the best representation of surrounding context, addressing the nature of the outputs was left uncharted. The vast majority of works in the field discuss a novel input-crunching mechanism, followed by a state-of-the-art backbone

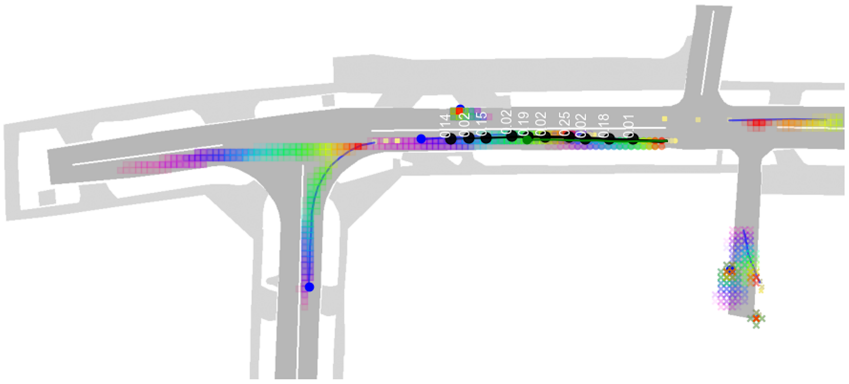


Fig. 4.1: An example scene, created using the algorithm of [168]. The map of the drivable area is used to discourage invalid predictions. The agents are represented in blue while the trajectory predictions are shown in black.

architecture which outputs a set of two-dimensional coordinates, representing the predicted future trajectory of the target object. There exists however an additional, significantly smaller strain of works. These works assume the premise that predicting coordinates is redundant as it diverts model capacity to re-learning the well studied rules of physics. By incorporating this prior knowledge into the model, the redundancy is reduced to the variable parts of the movement.

At first glimpse, developing a sound kinematic model of a car might sound infeasible. One would not just have to consider the trivial variables like heading angle and acceleration, but also rather factors like the wheel base, power curves of the motor, wheel slip, etc. These factors are well considered and explained in [187].

Luckily, a study was presented in [188], which covered the performance difference of different levels of model complexity. It was established that while the full kinematic models do perform better than the simple bicycle model, the increase in performance is often irrelevant for common driving manoeuvres.

The work of Cui et al. has integrated the bicycle model into a neural network [156]. They did so by predicting the future velocity, heading angle, longitudinal acceleration and the steering angle, on top of the future Cartesian coordinates.

A different approach was taken in [157], where the kinematic model was used for encoding the input measurements, rather than the predictions.

The above-referenced works demonstrate the potential of a simple kinematic model for trajectory prediction. Nevertheless, they all resolve to temporally discrete predictions for naturally continuous systems. The work of [189] presented the planning of offline flight trajectory, for airborne drones. It utilises the continuous and smooth nature of three-dimensional curves to create offline trajectory plans of a varying level of aggressiveness.

4.2.6 Polynomial Predictions

In this chapter, we aim at combining both strategies to reach a continuous and kinematically motivated model. We propose a novel formulation for an ad-hoc, architecture-independent output layer which directly outputs polynomials. The concept is a simplified online variation of [189] for a kinematic system for trajectory prediction. Nevertheless, unlike the aforementioned works, the polynomials are a relaxed kinematic constraint. Instead of explicitly developing loss signals for the velocity and acceleration, these are represented as derivatives of the position, see discussion in Subsection 4.4.1.

While being the first to predict polynomial trajectories, the concept of predicting coefficients was discussed before. Amongst the first works in the field, one can find [190]. It covered a series of experiments for predicting sparse polynomials from an experimental perspective. The idea was to explore and better understand the nature of neural networks and their convergence, rather than to propose an application or a solution to a problem. The prediction of ego agent movement polynomials was presented in [191]. By better modelling the movement of the camera, Pérez-Rúa et al. were able to improve several video related tasks, like video stabilisation and optical flow. Finally, the usage of polynomials as an inter-layer activation function was discussed in [192]. The work claimed that such an activation function improves the expressibility of the model, leading to visually appealing results on a wide variety of visual generative tasks.

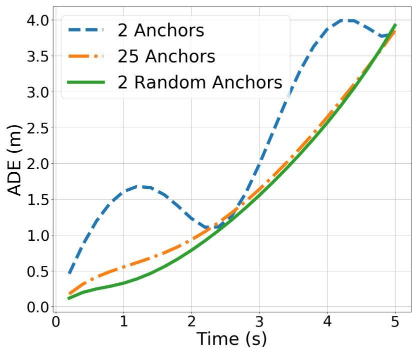
4.3 Motivation

Our main concern with the prior art covered in Subsection 4.2.2 regarded the over-fitting potential of the predicted trajectories. Traditionally, the common art architecture involves a high-dimensional model, trained to predict fixed temporal offsets as a set of Cartesian coordinates. While the problem of over-fitting is well documented and addressed in the literature [55, pp.108-113], the idea of over-fitting the output is broadly unattended. This hypothesis was tested using a simple experiment. Two models with an identical backbone and two different output layers were trained

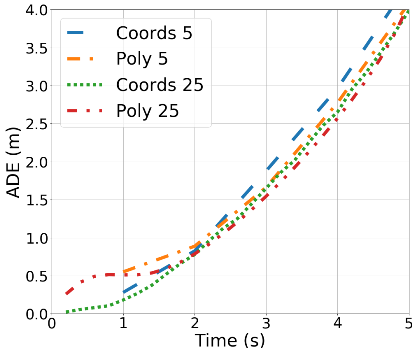
- The first model was trained on two fixed offsets (anchors), one at 2.5 seconds and another at 5 seconds.
- The second model was trained on fixed offsets at 0.2 second steps up to 5 seconds.

Both models were trained and evaluated on the NGSIM dataset [162] until convergence, using the same train/test split and the same hyper-parameters like learning rate, optimiser, etc.

The models were then evaluated in a 0.1 seconds resolution up to 5 seconds into the future. Figure 4.2a confirmed our intuition. It shows that for the same model, with only the randomness of the predicted offsets as a difference, the fixed offsets variant, '2 Anchors', clearly over-fits its target offsets. Its prediction has a relatively small displacement error on its fixed offsets and significantly worse performance between the anchors. While this experiment also proposes a solution, namely training on more offsets in '25 Anchors', it also raises the question of possible further improvement.



(a) The effects of the random anchoring scheme.



(b) Test set performance of the different configurations

Fig. 4.2: **Left**: evaluating the random anchoring scheme. Even with as little as 2 anchors the polynomial accuracies exceed the 25 fixed anchors. Notice that the 2 anchors model slightly over-fits the 2.5 second offset and scores there better than the corresponding 25 anchors model.
Right: the results with 5 and 25 anchors per trajectory. For a given number of anchors, the polynomial prediction models outperform the classical coordinates. From [149].

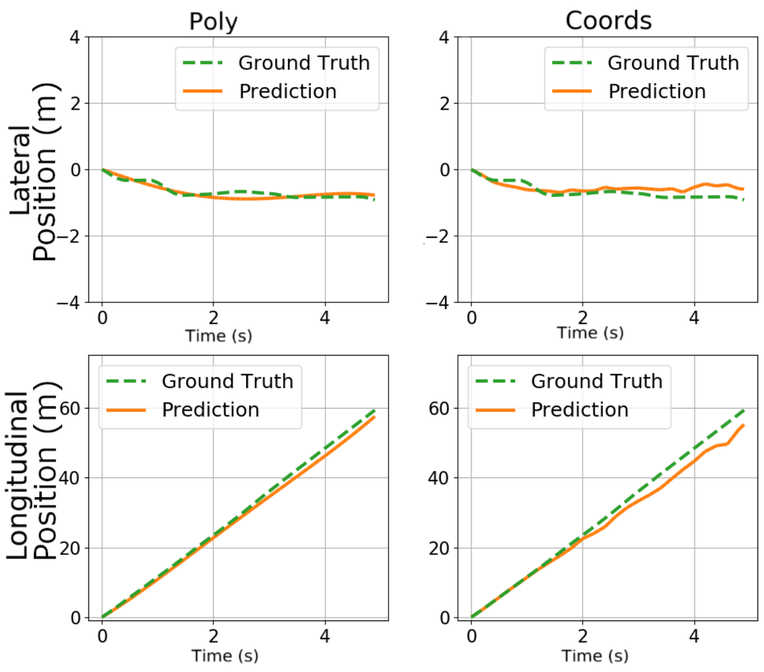


Fig. 4.3: Axis-wise visualisation of a random prediction from the test set.
Right: coordinates prediction.
Left: polynomial prediction.
The top and bottom rows show the lateral and longitudinal axes, respectively. Notice how the polynomial prediction results in smoother and less jagged trajectories. From [149].

We furthermore looked at the trajectories as predicted by common architectures. Figure 4.3 shows the axis-wise predictions of our model in the Cartesian configuration. One can clearly see jittery trajectories, which, comparing to the given ground truth trajectories, do not appear very natural. We attribute the observation to the tendency of neural networks to demonstrate a limited regression resolution. This tendency was established as early as 1998 in [193]. It stated that although the commonly used bandwidth of 32 bit allows for a high prediction resolution, it is often unused by the network.

The matter of regression precision is easily explained by looking at the weight of the loss signals. Consider the toy example of regression to the value of 1. While the 32 bit floating point value range supports a scale as low as 10^{-38}, a value differentiation on such a small scale has a negligible effect on the final loss value. The effect is further magnified when considering that modern neural networks incorporate millions of neurons. The magnitude and direction of the final loss signal is thus an averaged compromise over all units. The vanishing gradients towards the target value is depicted in Figure 4.4a. Also notice the similarity of using the cross entropy loss function on an entire image while there is only a handful of relevant pixels as discussed in Subsection 2.4.1.

With the results from the experiments in this sections, we set to explore the effects of polynomial trajectories on neural networks.

4.4 A Novel Output Layer for Continuous Functions

The concept of polynomial trajectories is simple. Instead of predicting an array of coordinates, it is possible to directly predict the polynomial coefficients, corresponding to an agent's respective trajectory. Training such a polynomial model requires an entire framework which makes for a significant part of this work's contribution.

Formally, an arbitrary neural network, $f_{baseline}(\cdot)$, which comprises an arbitrary number of arbitrary layers. For the temporally coherent nature of the trajectory sequences, we resort to the recurrent GRU layers. However, by only adjusting the output layer, this work becomes agnostic to its backbone architecture. Thus, other types of layers, e.g., fully connected or convolutional, are equally supported.

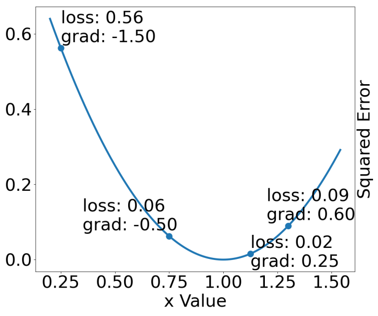
4.4.1 Layer Definition

One common way to predict trajectories is

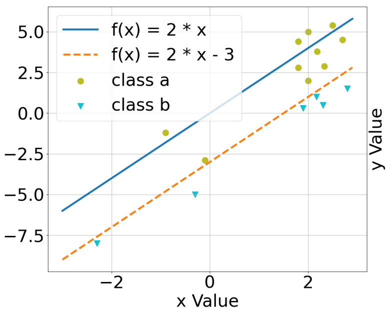
$$f_{baseline}(\cdot) = [x_1, y_1, x_2, y_2, \ldots, x_n, y_n]. \tag{4.1}$$

A set of n Cartesian coordinates which represent the course of a target agent's future movement, see Figure 4.5a. The exact temporal offsets are hence a hyper-parameter of the model and are fixed by design. For example, consider a model with pre-selected offsets of $10, 20, 30$ and 40 frames. The model, $f_{baseline}(\cdot)$, will predict four coordinates. Assuming the common frame-rate of 10 FPS, the points will represent ${}^{10}/{}_{10} = 1, {}^{20}/{}_{10} = 2, 3$ and 4 seconds into the future, respectively.

Even in the kinematically justified class of models, the scheme is not significantly different. For example, in [156] the acceleration, steering and velocity are predicted at fixed offsets in the future.



(a) Precision illustration



(b) Bias illustration

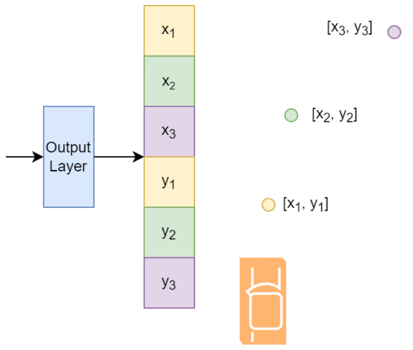
Fig. 4.4: Left: A visualisation of the loss of a single parameter regression to 1.0. Used is the common Squared Error loss, also known as (aka) l_2 loss. The marked values are the loss corresponding to the input values $[0.25, 1.3, 0.75, 1.13]$. Notice how the three values closest to 1.0 result in very small gradient values. These values only have a minimal effect when considered as a part of a model with millions of parameters.
Right: An illustration of the effect of the bias coefficient on an arbitrary function. Notice how the bias enables the predicted function to neatly separate class a from class b.

Notice that the coordinate system is mostly target-centric. I.e., it progresses with the target agent and changes at each time step such that the origin is always the current position of the predicted agent. While requiring additional post-processing to project all predictions back to the ego's coordinate system, it correctly teaches the model that the trajectory is not a function of distance to the ego vehicle. Additionally, by simulating each agent as the target agent, one can drastically increase the size of the dataset.

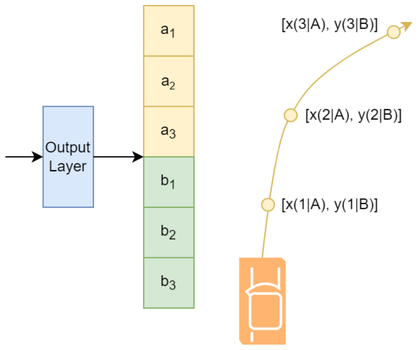
In this work, we propose to adjust the output to predict

$$\begin{aligned} f_{ours}(\cdot) &= \{A, B\} \\ &= \{[a_1, \dots, a_{d_x}], [b_1, \dots, b_{d_y}]\}. \end{aligned} \tag{4.2}$$

The omission of a_0 and b_0 in discussed in Subsection 4.4.2. Here, f_{ours} references the model presented in this chapter, d_x and d_y represent the degrees of the polynomials for the x axis and the y axis, respectively, see Figure 4.5b. The sets $A \in \mathbb{R}^{d_x}$ and $B \in \mathbb{R}^{d_y}$ are the predicted coefficients which parameterise two polynomial functions, one for each spatial axis. Combined, these polynomials represent the course of the trajectory of a given agent, as a function of time t, given throughout the rest of this



(a) Cartesian coordinates prediction



(b) Polynomial prediction

Fig. 4.5: **Left**: An illustration of the common prediction of Cartesian coordinates **Right**: Our polynomial prediction layer. Notice that the depicted points along the polynomial trajectory are arbitrary while the Cartesian offsets are fixed.

chapter in seconds. The polynomial for the x axis, parameterised by a_j is defined as

$$x(t|A) = \sum_{j=1}^{d_x} a_j t^j. \tag{4.3}$$

The definition of $y(t|B)$ is analogous.

The linear combination of coefficients with a temporal offset to derive a position is, similar to [156], kinematically motivated.

- The zeroth degree is referred to as the bias dimension and is thoroughly discussed in the next section.
- The first degree stands for a variable of the position over time, i.e., it represents the velocity.
- The second degree corresponds to metre over squared second, i.e., the acceleration.
- The third degree is called jerk.
- The fourth degree is called snap.

Finally, combining the aforementioned considerations means that our model outputs the parameters of two functions, one for each spatial axis. These functions jointly represent the future trajectory of a given target agent as a function of time. This is illustrated in Figure 4.5b.

In the derivative work, [171], we established that the spatial domain is not the only option for polynomial predictions. The work, which I supported with discussions and ideas, extended the scope to predicting the development of the acceleration and steering angle curves over time.

4.4.2 Bias

Under the polynomial definition, the zeroth coefficient, e.g., a_0 or b_0, assumes the role of the scalar. The machine learning term for this zeroth element is 'bias' [63, p.227]. A term which earned its name by being an input-independent offset to the decision line, represented by each perceptron. Figure 4.4b illustrates the effect of the bias term on the classification line. However, this very offset poses a contradiction in the case of trajectory prediction. Recalling that our coordinate system is set with its origin at the target agent, see Subsection 4.4.1, an offsetting parameter would inherently contracting this property.

4.4.3 Our Proposed Polynomial Training Scheme

The benefits of the predicted coefficients are a twofold. First, the predicted trajectories are smooth and more realistic. Second, it opens the hatch for a powerful training scheme. This training scheme is the focus in the rest of this section.

It is opened with the random anchoring scheme which is the most significant module in our framework. Then the natural loss weighting is covered, a label selection logic which focuses the training on the more challenging parts of the trajectories. Sanity forcing backwards propagate the prediction to the input signal, acting as an additional regulariser. Finally, coefficients regularisation and scaling are used to stabilise the training and accelerate convergence.

Random Anchor Selection To address the over-fitting issue recognised in Section 4.3, one could look at increasing the number of anchor (label) points. However, this would assign the same importance to the trivial offsets, directly following the current position, as to the more difficult ones, further out in the future. This follows the same easy versus hard samples, discussed in Section 2.4. Additionally, evaluating all frame-wise offsets is redundant and computationally tedious.

We tackle these unfortunate properties by introducing a novel random anchoring approach. For each training iteration, a target temporal offset is drawn at random and used to fetch the corresponding label. The process is roughly inspired by stochastic sampling [194], a sort of temporal Monte Carlo sampling. The following paragraphs explain the scheme in detail.

In order to properly train a model, one needs to reliably direct it towards the desired target function. Here, this function is the future trajectory of the target agent. The learning is achieved by repeatedly sampling positions from the observed trajectory, using the randomly drawn offsets, and adjusting the model's weights accordingly. With classical training algorithms, i.e., training with fixed offsets, only the same fixed points along the trajectory are sampled. In contrast, using our random anchoring scheme promises to eventually sample the entire length of the target function.

Consequently, we noticed that a single offset per sample is not enough for optimal training and have thus introduced additional anchors, T in total. First, the maximal offset o_T is drawn from the uniform range $\mathcal{U}(o_{min} \in \mathbb{N}, o_{max} \in \mathbb{N})$. Whereas o_{min} and o_{max} are hyper-parameters of our training scheme. Their role is merely to define the range of the uniform random sampling. Notice that these hyper-parameters are strongly dependent on the dataset's frame rate and are further discussed in the following part of Section 4.4.3.

Using the drawn offset, the rest of the offsets are computed such that

$$[o_1, o_2, \dots, o_T] = \left[\lceil o_T \frac{1}{T} \rceil, \lceil o_T \frac{2}{T} \rceil, \dots, \lceil o_T \frac{T}{T} \rceil\right]. \tag{4.4}$$

The labels for trajectory prediction tasks are derived from the fact that an object is tracked over time. Since the observed position of a vehicle over the course of the track is a part of the collected data, the measurements from further along the track could be regarded as the labels for training.

The drawn offsets $[o_1, o_2, \dots, o_T]$ are then used as indices. For example, consider the illustrated track in Figure 4.6, observed over the course of 120 frames. The entire observed track, $S_{0:120}$, is represented by the blue trajectory line. We define an additional hyper-parameter, j_{ref}, which sets the division of the entire sequence into an input and an output part. The fiftieth frame, $j_{ref} = 50$ (S_{50}), is called the reference frame as it references a scenario in which it is the current frame. The first 51 frames ($S_{0:51}$), shown in orange, are used as the input.

Considering the rest of the sequence, i.e., $S_{50:120}$, the indices $[25, 50]$ are drawn according to our random anchoring scheme. These mean that the labels, L, for the current training iteration are taken from $S_{25+j_{ref}} = S_{75}$ and $S_{50+j_{ref}} = S_{100}$, shown in purple. Possibly confusing is the fact the $S_{j_{ref}}$ appears both in the input sequence as well as the output sequence. This follows from the design decision to consider all unseen indices, while the smallest possible value of o_1 is 1.

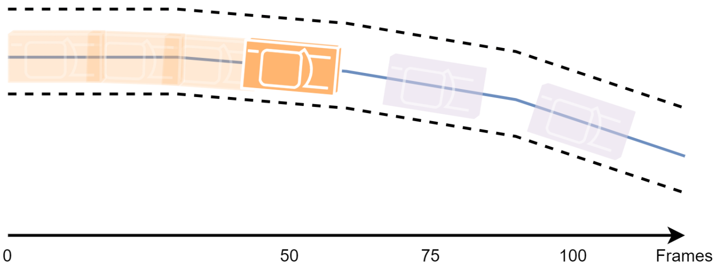


Fig. 4.6: An illustration of the input and output selection from a track. The target agent is currently at frame $i_{ref} = 50$. The current frame is concatenated with its preceding 50 frames to create an input sequence of length 51 frames. Frame $o_T = 50$ was drawn at random, meaning that the first label is taken from frame index $j_{ref} + o_T = 50 + 50 = 100$. For $T = 2$, there is an additional intermediate label, to further support the learning signal. This second label is taken from index $j_{ref} + o_T{}^1/_T = 50 + 50 * {}^1/_2 = 75$. Since the entire track was recorded and due to causality, the 'future' position of the agent is already known and could therefore be used as a label.

Having both the random offsets, as well as their respective spatial positions, the training algorithm could be finalised. The first step is to convert the frame indices to temporal offsets. The conversion requires the frame rate which is 10 FPS for the NGSIM dataset. Knowing this the conversion means computing the frame index by the frame rate, e.g., ${}^{o_T}/_{FPS}$. For this section's toy example, the prediction points ${}^{25}/_{10} = 2.5$ and ${}^{50}/_{10} = 5$ seconds from the current time step are derived. These values are then used to compose the temporal vector, e.g., $[5^1, 5^2, 5^3]$, which is used to evaluate the polynomials for the x axis coordinates, X, and the y axis coordinates, Y, following Equation 4.3. We can now use the regression loss of choice to evaluate the loss and calculate the gradients. For example, consider the Euclidean loss, also known as the Mean Squared Error (MSE) [55, p.106]

$$\mathcal{L}_{MSE}(\{X, Y\}, L) = \frac{\sum_{j=1}^{T}(L_{j,x} - X_j)^2 + (L_{j,y} - Y_j)^2}{T}. \tag{4.5}$$

Here, $L_{j,x}$ refers to the x part of the label j. The entire algorithm is listed in Algorithm 1 and depicted in Figure 4.7.

Natural Loss Weighting The additional similarity to Monte Carlo sampling comes from our range selection. Monte Carlo sampling is often used to approximate an unknown distribution. It does so by generating more samples which are likely to

Algorithm 1 Our proposed polynomial training scheme with random anchoring

```
Require: $S$                                                        ▷ Observed track
Require: $T$                                                        ▷ Number of labels
    $A, B \leftarrow f_{ours}(S[0:51])$          ▷ Predict poly. coefficients following Equation 4.2
    $o_T \leftarrow random\_int(50)$
    $O \leftarrow [\lceil o_T \frac{1}{T} \rceil, \lceil o_T \frac{2}{T} \rceil, \dots, \lceil o_T \frac{T}{T} \rceil]$          ▷ Anchor indices following Equation 4.4
    $L \leftarrow S[O]$                          ▷ Query the sequence at the offsets ‘O‘ for the labels
    $t_x \leftarrow \begin{bmatrix} (o_1/10)^1 & \dots & (o_1/10)^{d_x} \\ & \vdots & \\ (o_T/10)^1 & \dots & (o_T/10)^{d_x} \end{bmatrix}$          ▷ Create the temporal vectors
    $t_y \leftarrow \begin{bmatrix} (o_1/10)^1 & \dots & (o_1/10)^{d_y} \\ & \vdots & \\ (o_T/10)^1 & \dots & (o_T/10)^{d_y} \end{bmatrix}$
    $X \leftarrow x(t_x|A)$                                         ▷ Predict for both axes
    $Y \leftarrow y(t_y|B)$
    $\mathcal{L} \leftarrow \mathcal{L}_{MSE}(\{X, Y\}, L)$          ▷ Compute the loss, following Equation 4.5
```

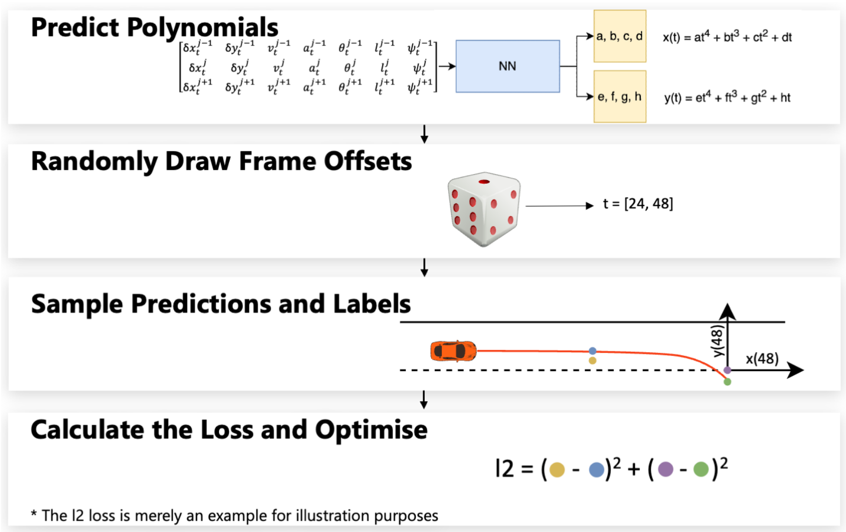
reside within the distribution than out of it, based on the evaluation of previously drawn samples. Considering trajectories, it means that a sampling scheme is preferred which favours temporal offsets with a larger error. Yet, unlike the classical use case for Monte Carlo and as could be seen in Figure 4.2, the displacement error of trajectories grows along the temporal axis. This matches the common expectation. The larger the prediction offset is, the less linear, and thus more complicated, the extrapolation becomes. Considering a car on the motorway, predicting its position 0.5 seconds into the future could often be trivially achieved by a linear extrapolation. In fact, in [174] a simple linear extrapolation was used as a reference baseline for the evaluation. The mean final displacement error at 4.8 seconds was merely $\sim 50\,\%$ worse than the SotA.

To decrease the exposure to easier samples, the sampling range is defined as

$$o_T = \mathcal{U}(\lceil 0.75M \rceil, \lceil 1.25M \rceil). \tag{4.6}$$

With M being the maximal predicted offset. E.g., For a prediction task of 50 frames into the future, an offset between 38 and 63 frames is uniformly selected.

The advantage for this sampling scheme is that the loss is focused on the more relevant areas of the target function. Similar to Subsection 2.4.2, this resembles the weighing scheme of [89]. We recognise the harder samples at the model-design level and inherently steer the learning process towards them.



Predict Polynomials
NN
a, b, c, d
e, f, g, h
x(t) = at^4 + bt^3 + ct^2 + dt
y(t) = et^4 + ft^3 + gt^2 + ht
Randomly Draw Frame Offsets
t = [24, 48]
Sample Predictions and Labels
y(48)
x(48)
Calculate the Loss and Optimise
l2 = (● - ●)^2 + (● - ●)^2
* The l2 loss is merely an example for illustration purposes

Fig. 4.7: A visualisation of the main aspects of our polynomial training pipeline. First, an input matrix with the ego as well as neighbouring agents is processed by the neural network to output axis-wise, polynomial trajectories. Then, random frame offsets are drawn, here 24 and 48. The polynomials are then sampled at those offsets, resulting in Cartesian coordinates. The coordinates are then evaluated against a measured ground-truth and an arbitrary loss function provide the learn signal to the network. Best viewed in colour. The input variables follow the definition in Subsection 4.5.1.

Sanity Forcing In the subsequent publication of Spata et al., an extension was considered to the anchoring scheme [170]. This extension, which considers the domain of object tracking, recognises that the polynomial loss also supports the beneficial property of sanity forcing. Instead of evaluating the predicted polynomials on the future trajectory only, one could go further and sample negative temporal offsets. That is, making sure that for $t < 0$ the respective observation, i.e., input signal, is recovered.

The benefits of such a learning signal are threefold.

- First, the amount of training data is artificially increased.
- Second, the model is directly trained to extract the input trajectory. In architectures like the one proposed in [179] this is especially important, since the observed trajectory is only implicitly given in the form of a binary image.
- Finally, the model is directly encouraged to learn that the future trajectory is merely the not yet observed part of the already seen trajectory. I.e., continuity and temporal consistency are explicitly enforced.

Equation 4.4 is thus revised to its final form of

$$[o_1, o_2, \ldots, o_{2T}] = \left[\lceil o_T \frac{-T}{T} \rceil, \ldots, \lceil o_T \frac{-1}{T} \rceil, \lceil o_T \frac{1}{T} \rceil, \ldots, \lceil o_T \frac{T}{T} \rceil \right]. \tag{4.7}$$

Coefficients Scaling and Regularisation Throughout the experiments leading to this work's proposed algorithm, we have developed two numerical tricks to normalise the learning signal and direct the network in the desired direction.

The first trick exploits the nature of motorway traffic to generally head forwards. A property which is used to direct the network towards convergence at the first few epochs. The signal is an l_2 regulariser of an increasing weight to the predicted coefficients. I.e., the larger the degree of the coefficient, the stronger its regularisation factor becomes. For example, the regularisation for the x axis, r_x is represented by

$$r_x = \sum_{j=1}^{d_x} 2^{j-1} |a_j|. \tag{4.8}$$

With r_x assuming the regularisation value which is then added to the final loss term. The scaler 2 was selected by manual hyper parameter optimisation. Such a regulariser leads to a strong learning signal towards linear trajectories which are a reliable initial hypothesis in motion understanding.

Second, we follow the work of Bernardo et al. It established that common gradient based learning is not well suited for predicting different orders of magnitude [193]. As the scale of the variables increases with their degree, we expect the predicted coefficients to decrease accordingly. This property is incorporated into the model in a similar manner to the regularisation term. An exponential scaling factor is added to the model's outputs, extending the definition of Equation 4.3.

$$x(t|A) = \sum_{j=1}^{d_x} a_j 10^{-j} t^j. \tag{4.9}$$

By upscaling the larger degree coefficients, the network is encouraged to keep them small, leaning towards linearity. Since the higher coefficients have a significant effect on the curvature, a minor change in value suffices for representing common motorway manoeuvres, e.g., lane change. This corresponds to the precision aspect in the coordinates model, discussed in Section 4.3. Here as well, the scalers were manually determined during development such that the predicted coefficients are normalised to the range $[-10, 10]$.

As a concrete example, one can regard a lane change event four seconds in the future. This translates to a predicted lateral offset of three metres in four seconds. Assuming this translation is only represented by the highest degree coefficient, one gets $y(4|B) = 3 = 4^4 b_4$. Solving for b_4 results in $b_4 = 0.01172$. Assuming the same manoeuvre is strictly linear leads to $y(4|B) = 3 = 4^1 b_1$ and $b_1 = 0.75$. Since both coefficients are subjected to the same loss value, they should have the same scale. They are thus scaled by a factor of 10 for b_1 and $1{,}000$ for b_4. As the trajectory is composed of four degrees of coefficients, the actual values are even smaller, resulting in an additional order of magnitude to the scalers.

4.4.4 Novel Polynomial Variance Estimation

In addition to the adjustments to the actual trajectories, we also discuss an extension to the variance prediction framework. Variance estimation acts as an indicator for the quality of the prediction. It is useful for downstream applications which can use it to understand the reliability of the prediction in real-time. So far, the common variance estimation framework only regarded the prediction of fixed coordinates. Here, we cover the extension of the framework to support our polynomial representation.

Adherently, please notice that this section refers to variance prediction while actually predicting the standard deviation (σ). The reason is anchored in the terminology coined in [181]. Since the mapping between the values is constant, the terms are abused and used interchangeably.

Variance for Coordinates

In Subsection 4.2.3, the work of Graves was mentioned as the first incorporation of a distribution predicting framework to the very architecture of a neural network [181]. It proposed a network for generation of hand writings and since pen strokes are constantly varying, a factor of randomness in the prediction was required.

We consider a slightly simplified version of the mathematical derivations proposed in [181]. In the following the parts which are irrelevant to this work are omitted and the notation is unified with the one introduced in preceding sections.

The sought randomness is achieved by predicting a series of bivariate Gaussians which are combined and sampled to get the final coordinates of the stroke. The neural network is defined as

$$f(S_{t-51:t}) = (\mu_t, \sigma_t, \rho_t). \tag{4.10}$$

The definition follows the common notation, meaning $f(\cdot)$ is neural network, $S_{t-51:t}$ is the input signal, $t \in \mathbb{N}$ is the temporal index, $\mu_t \in \mathbb{R}^2, \sigma_t \in \mathbb{R}^2, \rho_t \in \mathbb{R}$ are the mean, standard deviation and correlation, respectively. Whereas each variable is a two-dimensional vector, for both spatial axes.

The standard deviation is furthermore passed through the exponential function, e^x or $\exp(x)$, to ensure positivity. The correlation is passed through the hyperbolic tangents function to enforce the range of $(-1, 1)$.

The probability density function of the next coordinates (x_{t+1}, y_{t+1}), given the current input $S_{t-51:t}$ is then

$$Pr(x_{t+1}, y_{t+1}|S_{t-51:t}) = \mathcal{N}(x_{t+1}, y_{t+1}|\mu_t, \sigma_t, \rho_t). \tag{4.11}$$

The definition is additionally dependent on

$$\mathcal{N}(x, y|\mu, \sigma, \rho) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left(\frac{-Z}{2(1-\rho^2)}\right) \tag{4.12}$$

as well as on

$$Z = \frac{(x-\mu_1)^2}{\sigma_1^2} + \frac{(y-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x-\mu_1)(y-\mu_2)}{\sigma_1\sigma_2}. \tag{4.13}$$

When $(x_{t+1}, y_{t+1}|S_{t-51:t})$ is known, i.e., during training, one can minimise the negative log likelihood of the observed trajectory to accommodate learning.

Variance for Polynomial Coefficients

In the polynomial representation, the network predicts coefficients instead of position points, thus the above formulation can not be directly applied. Maintaining the logic of the variance framework, μ represents the mean of the coefficients instead of the positional mean. The definition of σ is matched accordingly, i.e., the estimated variance w.r.t the coefficients themselves can now be predicted. However both the application as well as the loss function require the variance w.r.t the spatial coordinates. The predicted uncertainty should thus be propagated from the coefficients to the trajectory.

We start by assuming that the axis-wise coefficients are non-correlated. The simplest example for this assumption is the constant velocity case, where only the first-degree coefficient is larger than zero. Moreover, accounting for the full covariance matrix renders the loss function computationally intractable, setting a strong practical motivation for the approximation.

The second assumption is required to use the variation estimation framework as defined in [181]. It states that the coefficients follow a normal distribution.

Under both assumptions, the covariance matrix for the longitudinal axis, x is denoted as

$$cov(A) = \begin{bmatrix} \sigma_{a_1}^2 & 0 & \dots & 0 \\ 0 & \sigma_{a_2}^2 & \dots & 0 \\ \dots & \dots & \dots & 0 \\ 0 & 0 & 0 & \sigma_{a_{d_x}}^2 \end{bmatrix}. \tag{4.14}$$

The polynomial evaluation is a mere linear combination of the coefficients with the input variable from Equation 4.9. The linear case of uncertainty propagation could thus be used to get the variance from the covariance. It is defined as

$$var(x(t|A)) = I_T cov(A) I_T^\top, \tag{4.15}$$

where $I_T = [t^1, t^2, \ldots, t^{d_x}]$ follows the definition in Algorithm 1. Hence, the polynomial form of the variance estimation consists of the coefficient-wise variance predictions. These are propagated to the spatial domain and evaluated similar to Equation 4.9. The adjusted form of Equation 4.9 for the variance values is hence

$$\sigma_x^2(t|A) = \sum_{j=1}^{d_x} 10^{-2j} \sigma_{a_j}^2 t^{2j}. \tag{4.16}$$

Finally, while the definitions throughout this section focus on the longitudinal axis, the formulations for the lateral axis are analogous.

4.5 Evaluation

With the scope of this project set to motorway scenarios, we used the NGSIM dataset for the evaluation [162]. Due to the industrial affiliation, we were prevented from using other, more extensive datasets from the likes of HighD [164]. The dataset covers two motorway segments in the USA, observed by static traffic control cameras which record at 10 Hz. The collected data was manually filtered to include three times of day: dawn, daylight and dusk. It was manually labelled and segmented such that each track is 200 frames long, i.e., 20 seconds. There is also a proposed train to test split at a $3:1$ ratio.

While the test set was left at its original size of $\sim 3,400$ tracks, the training set was filtered to reduce the inherent redundancy which characterises a straight motorway. The amount of constant velocity, straight driving tracks was randomly halved, resulting in a training set of $\sim 7,500$ tracks. Straight driving was defined as the lack of change in the assigned lane. Constant velocity was considered as remaining within the range of ± 10 % of the track's average velocity.

The results are reported either in Average Displacement Error (ADE) or in Root Mean Squared Error (RMSE) in metres, whereas lower values correspond to more accurate trajectories. The time is named in seconds.

4.5.1 Architecture Design

For the evaluation of our polynomial prediction layer, a network architecture had to be selected. The requirements dictate a model which could be trained in reasonable time, uses off-the-shelf layers and at the same time delivers near SotA results, in order to make the results comparable with other works.

For the matter of multi-modality, we experimented with the 'min of k' class of methods [159]. However, while the method showed visually and empirically appealing results, it also lead to a new challenge. Rather than properly addressing multi-modality, the 'min of k' scheme exchanges the difficulty of multi-modal predictions for the challenge of modality selection. While this practice is acceptable during training, during testing it becomes problematic.

The option of conditioning the model on the ground truth modality [155] is also problematic, as it translates to telling the network what it is evaluated on. Imagine training a model to predict three modalities of three possible crossings of an intersection: turning left, going straight and turning right. There will always be a modality which predicts a right turn, rendering a comparison with single-modality algorithms obsolete. Moreover, one could artificially increase the potential accuracy by predicting a larger number of modalities. Evidentially, while [158] discussed the benefits of $3-13$ modalities, in [195] the authors already consider as much as $3{,}342$ modes.

Phan-Minh et al. discussed adding a modality classification output layer to their model [195]. Since the ground truth modality is known, using it as a parallel classification task is a good option. Unfortunately, this only partially solves the modality selection issue. Since this chapter mainly concerns the quality of the predicted trajectories, it was desired to isolate the evaluation and not skew the results through possible classification errors.

In the absence of better options, we resolved to using the architecture proposed in [155] which conditions the decoder on the desired modality, generating only a single prediction. In contrast to other works, modest three modalities were considered: a lane change to the right, keeping straight and a lane change to the left. The elegance of such a framework comes from accounting for multi-modality while not forcing it on the model. It consists of a one-hot vector which is provided to the model as an input, telling it which modality to predict. By conditioning the model on the ground truth modality, one enjoys the benefits of multi-modality while not having to directly handle the modality selection itself. Notice that the model is always conditioned on the desired, i.e., correct, modality. It allows the network to learn the concept of different modalities while allowing us to focus the evaluation on the quality of the trajectories.

Our model follows the encoder-decoder, GRU based architecture, initially proposed in [196]. All three encoder layers and two decoder layers use 32 neurons with hyperbolic tangent ($tanh$) as their activation function. For an immediate reference two versions of this model are considered, one with the common coordinates and variance based output and the other with our proposed polynomial output layers. I.e., both configurations are equipped with four fully connected output layers, two for the spatial representation and two for their respective variance values.

The input sequence $S_{0:51}$ is composed from the target agent and the spatially next eight neighbours. The neighbours are selected to include three agents from the immediate right lane, another three from the immediate left, the leading vehicle and the following one. I.e., each matrix row represents a position relative to the ego agent. Rows without respective agents are set to zero. The sorting of the left and right agents is interleaved, such that both leading and following vehicles are used. The result is a 9×51 matrix in which each cell is defined as $s_t^j \in S_{0:51} = [\delta x_t^j, \delta y_t^j, v_t^j, \alpha_t^j, \theta_t^j, l_t^j, \psi_t^j]$.

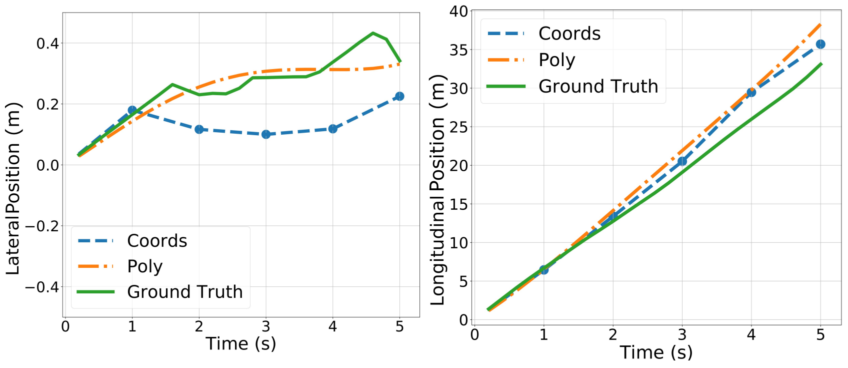
The agents are indexed using j and the time step index is t. Instead of the world coordinates, we normalise the position to position increments $[\delta x_t^j, \delta y_t^j] = [x_t^j, y_t^j] - [x_t^{j-1}, y_t^{j-1}]$. The first time step is artificially set as the origin. The other dimensions are the velocity, acceleration, heading angle and, finally, the polar coordinates to the target agent, respectively.

4.5.2 Ablation Study

We start the evaluation with an ablation study of the different aspects of the framework. Notice that the variance prediction can only be visually inspected, as it is an error estimation of the model for its own performance. Ideally the variance would be zero all throughout and the mean, i.e., the actual prediction, would always be accurate.

Random Anchoring

As a first evaluation step we examine the effect of the random anchoring scheme. The main challenge of this experiment was the evaluation of intermediate time steps in the coordinates-predicting configuration. We have therefore decided to



(a) Lateral predictions (b) Longitudinal predictions

Fig. 4.8: An axis-wise visualisation of a random test sample. One can see the smoother and more reliable matching to the ground truth trajectory by the polynomial model. In contrast the rough estimation of the coordinates model is also visible. It presents a partially unrealistic movement along the lateral axis. As expected from a motorway only dataset, the longitudinal movement is close to constant velocity. This property is clearly reflected by the predictions of both models.

approximate the coordinates configuration using a polynomial network which was trained on fixed anchors.

First, the validity of the previously discussed assumptions is evaluated. This was achieved by training the same model backbone with both output configurations on 5 and 25 anchors. Recalling Figure 4.2b, one can see the configuration-wise improvement with the number of anchors. Additionally, for a given number of anchors, the polynomial configuration outperforms its coordinates counterpart. Finally, with the built-in temporal coherence of the polynomial model, one can see that the 5 anchors polynomial configuration is roughly on par with the 25 anchors coordinates one. The hypothesis of a fair comparison is thus validated and we continue with evaluating the effects of randomness as discussed in Subsection 4.4.3.

The over-fitting of the model configuration with two fixed anchors (Figure 4.2a) was already discussed in Section 4.3. As a reference, a fixed anchor model with 25 anchors is also provided. Here again, one can clearly see that the over-fitting decreases with the number of anchors. Last, the configuration with 2 random anchors is provided and exhibits marginally better performance even comparing to the 25 anchors model. We conclude that the randomness plays an important role in generalisation. This matches the observation from [197] which discussed an image denoising framework. In the absence of information about the type of noise in the input, i.e., randomness w.r.t noise, their model learnt to denoise all types of images, regardless of their noise. That is, the randomness has proven a strong regulariser, outperforming other noise-specific methods.

We have further picked a validation sample at random and visualised the respective axis-wise prediction. The results are shown in Figure 4.8. Unlike the other figures in this chapter, the figure shows the course of a track over time and not the error. Here as well, one sees that the polynomial lateral prediction surpasses that of the coordinates network. We attribute the jittery ground-truth trajectory to measurement and acquisition noise.

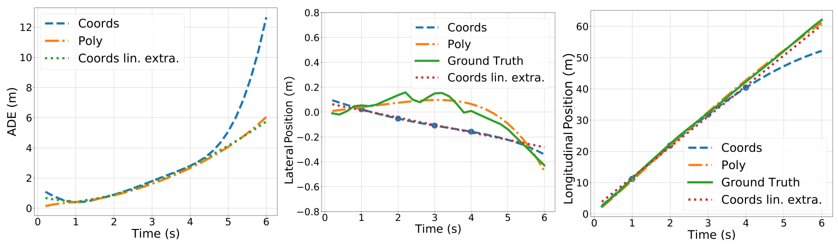
Extrapolation

One of the best ways to evaluate generalisation is by extrapolating the outputs. Extrapolation does not only evaluate the model on unseen samples, it rather does so while increasing the complexity. In the case of trajectory prediction, it means predicting for a longer temporal horizon than the one used for training.

The experiment setup included a polynomial and a coordinates model, each trained with four anchor points up to an offset of four seconds. The converged models were then tested on up to six seconds in the future, i.e., $150\ \%$ of the trained prediction range. While the polynomial functions are inherently flexible in terms of prediction offsets, the coordinates model required an additional step. Using Numpy's `polyfit` function [198], we fitted two curves to the original prediction. A linear curve, resembling a Kalman filter progression [199], and a fourth order polynomial, matching our model's predictions.

The results are visualised in Figure 4.9a and show how the fourth order coordinates extrapolation keeps track for about half a second before completely diverging from the ground-truth. An unexpected observation came from the linear extrapolation which roughly matches the error rate of the polynomial trajectory. Picking a single sample from the evaluation set, at random, and analysing its axis-wise performance provided a possible explanation.

Figure 4.9b and Figure 4.9c show the position over time. Unlike most figures in this chapter, they do not represent an error, but rather the mere course of the track. One can see that the polynomial trajectory provides a significantly better match for lateral axis. Yet, at the same time, the scale of both axes is not comparable. For example, $1\ \%$ error on the lateral axis at 3 seconds would correspond to 0.002 metres, the same error would correspond to 0.3 metres on the longitudinal axis. I.e., the longitudinal axis dominates the loss by two orders of magnitude. Since the forwards movement



(a) Overall performance in ADE (b) Lateral prediction, single sample, in position (c) Longitudinal prediction, single sample, in position

Fig. 4.9: Both models were trained on up to four seconds and evaluated on up to six seconds. To extrapolate the coordinates prediction, two functions were fitted, a linear and a fourth degree polynomial. The trajectory polynomials provide a more accurate, smooth match to the observed trajectory.

Offset (sec)	Coords baseline	Poly (ours)	CS-LSTM (M) [155]	MFP-1 [200]
1	**0.43**	0.55	0.62	0.54
2	1.00	**0.93**	1.27	1.16
3	1.72	**1.64**	2.09	1.90
4	2.76	**2.64**	3.10	2.78
5	3.98	3.85	4.37	**3.83**

Tab. 4.1: Results in RMSE of coordinates vs. polynomial training on NGSIM. For reference, two other SotA results are provided.

in the NGSIM dataset is mostly linear, the linear extrapolation option manages to score well in the overall performance. This matches the observation from [174] where acceptable baseline results using a Kalman tracker were demonstrated.

4.5.3 Quantitative Results

Finally, we evaluated our method against two other SotA models, the ‘CS-LSTM (M)’ model with social pooling from [155] and the ‘MFP-1’ model with a multimodal RNN from [200]. The results, presented in Table 4.1, are based on their respective publications and were not re-implemented. Here as well, one sees that the polynomial prediction layer leads to a favourable outperformance.

4.6 Limitations and Extensions

The work presented in this chapter supports the main claim of this dissertation that the integration of prior knowledge into a neural architecture acts as a good regulariser, leading to performance improvements and improved generalisation. Nevertheless, we recognised some limitations, leaving a room for future development.

First and foremost is the maximum representable complexity. While vehicle trajectories along a motorway are well represented using fourth-degree polynomials, the reality becomes more complicated in urban scenarios. Furthermore, pedestrians require a higher degree of freedom for common walking scenarios.

Consider the example of a person waiting and pacing outside of a building. They might wait and stand around for a bit, then walk in one direction, turn around and walk back. Predicting spatial polynomials to represent such a movement is often more complicated than predicting polynomials for the velocity or acceleration of the same agent. This is depicted in Figure 4.10, which shows how the velocity and acceleration curves are less complex than the positional curves. This is also one of the main claims in our consecutive publication [171].

Another possible solution would be to use splines. For example, one could use natural cubic splines. These are a class of splines in which each spline consists of multiple polynomials for different ranges. To enforce continuity, the splines are forced to satisfy a smooth transition condition as well as continuous first and second derivatives. Moreover, the model could predict the transition offsets between the splines. A trajectory would thus have the form

$$S_{j_{ref}:j_{ref}+m_M} = \begin{cases} f_0(t) = x(t|A_0), y(t|B_0) & 0 : m_0 \\ f_1(t) = x(t|A_1), y(t|B_1) & m_0 : m_1 \\ \vdots & \vdots \\ f_{M-1}(t) = x(t|A_{M-1}), y(t|B_{M-1}) & m_{M-1} : \infty \end{cases} . \tag{4.17}$$

Here $M \in \mathbb{N}$ represents the number of segments, $m_{0:M-1} \in \mathbb{R}$ represent the transition offsets (knots), in seconds, from one polynomial to the other. $A_{0:M-1}$ and $B_{0:M-1}$ represent the predicted polynomial coefficients for the respective segment. Following this proposed formulation M would be a hyper-parameter of the model.

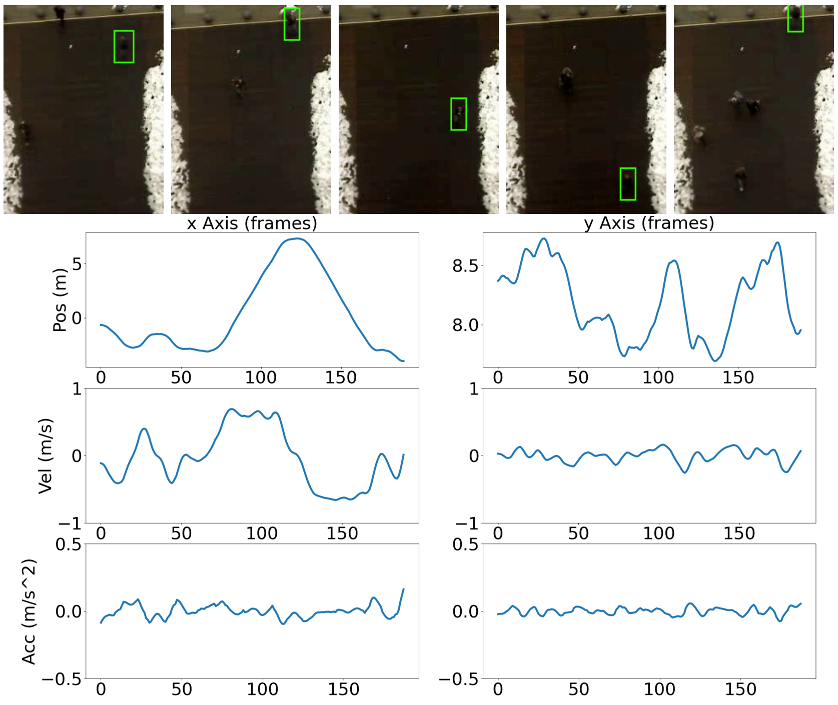


Fig. 4.10: An example pedestrian track from [153] of a pacing person waiting outside of a building. The person is marked by a green bounding box and shown at frames $[0, 25, 100, 125, 175]$, corresponding to the horizontal dimension of all graphs. The top two graphs show the pedestrian's axis-wise position. The middle row shows the axis-wise velocity. The bottom row shows the axis-wise acceleration. All values are smoothed to account for acquisition and labelling noise.

To maintain smoothness, the layer would also have to enforce

$$\begin{aligned}
f_0(m_0) &= f_1(m_0) \\
f_0(m_0)' &= f_1(m_0)' \\
f_0(m_0)'' &= f_1(m_0)'' \\
f_1(m_1) &= f_2(m_1) \\
f_1(m_1)' &= f_2(m_1)' \\
f_1(m_1)'' &= f_2(m_1)'' \\
&\vdots \\
f_{M-2}(M-1) &= f_{M-1}(M-1) \\
f_{M-2}(M-1)' &= f_{M-1}(M-1)' \\
f_{M-2}(M-1)'' &= f_{M-1}(M-1)''
\end{aligned} \quad . \tag{4.18}$$

I.e., for each knot the the zeroth, the first and the second derivatives of both the current and the consecutive polynomials must match.

The second shortcoming is that we only tested our algorithm in the two-dimensional domain. An expansion to the height dimension would benefit trajectory planning for aerial vehicles from the likes of drones and other Unmanned Aerial Vehicles (UAVs). This could be seen as an extension to [189].

Finally, as seen in Figure 4.3, the predicted trajectories cannot represent micro-movements along the smooth trajectory. This is, however, not necessarily a limitation of the polynomial trajectories, as the majority of applications only regards such movements as noise to be ignored. Yet one could imagine some applications which would care about such movements. For example, an algorithm for cut-in predictions might be able to predict a cut-in event a few frames earlier, just by seeing such micro-movement artefacts.

4.7 Conclusions

The previous chapters focused on a single aspect of optimisation. Chapter 2 explored an improvement to the cross entropy loss function, making it more suitable for spatially oriented tasks like semantic segmentation. In Chapter 3, we dissected the convolution operation itself, optimising the processing effort for embedded systems. This chapter, on the contrary, has discussed a plurality of methods, each with its own contribution to the quality of predicted trajectories.

The key observation leading to this work is the temporally incoherent trajectories predict by the prior art. By outputting a series of spatial coordinates, we showed how the inherent continuity of trajectories is indirectly broken. This manifests itself as jittery predictions which over-fit their target offsets.

The main enabler of our solution is the transition from coordinates prediction to the prediction of polynomial coefficient. These coefficients parameterise polynomials of position as a function of time. It was established that this type of an output format is by itself enough to increase generalisation and performance.

Consequentially, the polynomials allow for further improvements to the training scheme. The most significant of which has proven to be the random anchoring scheme. By testing the model on randomly drawn temporal offsets, it is forced to learn the movement rather than a series of fixed offsets. This improved generalisation was mostly demonstrated by evaluating the model on time steps which are 1.5 times larger than the maximal offset during training. Even for these extended offsets, good results were achieved, especially comparing to the coordinates baseline.

Supporting adjustments are the natural loss weighting, the sanity check and the coefficient scaling. All of which stabilise the training process and have proven valuable tools.

Finally, the commonly predicted variance estimation was extended and adjusted to the polynomial form. The variance improves the information gain and allows for a more informed decision making in downstream applications.

With the main weakness of the model being the performance in the first second, enough room is left for future improvement. For this we proposed to explore the option of splines. Splines are more flexible than their counterparts and, if successfully implemented, they could improve our temporally coherent framework even further.

Conclusions 5

This work was written to exist in two planes. First, the algorithmic plane. Over the course of its three chapters, we proposed three new schemes and frameworks which improve the performance of neural network algorithms for automotive applications. Second, the methodic plane. Each of our proposed algorithms was constructed under a common denominator, the integration of a-priori world knowledge into SotA models.

5.1 Algorithmic Conclusions

In Chapter 2 we covered the potential of structured loss functions. The reasons for the incompatibility of classical classification loss functions to the field of pixel-wise classification, i.e., semantic segmentation, were discussed and evaluated. We showed how by accounting for these inherent structures one could not only drastically improve runtime performance, but also the segmentation quality. Our Spatio-focal Loss outperformed the reference algorithms, as well as the baseline, in handling small objects. The utilisation of the same logic as a binary filtering mask lead to a runtime reduction by a factor of ~ 5 while maintaining a results-agreement rate of $99.61\ \%$ with the baseline.

In Chapter 3, we analysed the reasons which cause neural networks run slower on embedded hardware. We established that the vanilla convolution layer causes multiple bottlenecks on Texas Instruments' TDA3 board. The results were then used to propose a novel convolution block architecture which approximates the classical layer. It does so in a more streamlined manner while solving most bottlenecks. Furthermore, compared to other optimised architectures, it delivers a higher value per computation with better accuracies. Additionally, it surpasses even the baseline's accuracy when inflated to its size.

Finally, in Chapter 4, we proposed to predict trajectories in the form of temporally coherent polynomials. We showed the multiple benefits of this representation. It generalised better, predicted more accurate trajectories and more realistically looking ones. Moreover the polynomial representation was demonstrated to be more flexible and extrapolate better to unseen offsets.

5.2 Methodic Conclusions

The exact scope of this work was not set in advance. It has rather emerged during the course of the programme and had manifested itself in a similar way across the different projects. Regardless of the domain, most current SotA models were developed to be as general and non-restricted as possible. An attribute which does have a certain appeal, as reproducibility and applicability to similar domains are generally a desirable feature. Yet these algorithms often overshadow years worth of research and accumulated experience.

Our proposed spatio-focal loss, discussed in Section 2.4, is based on the same concepts as the 1998 bilateral filter [80]. It accounts not only for the spatial distance between two pixels, but rather also their colour, i.e., intensity, agreement.

Our EffNet block from Chapter 3 utilises the very nature of convolution. It states that given a symmetric kernel, a multi-dimensional convolution equals its respective single-dimensional decomposition [134, p.101]. The successful reintroduction of this property to CNNs was the main novelty of this work.

Finally, the polynomial output layer of Chapter 4 is vaguely based on the well established bicycle model. Luckily, works in this domain have not completely neglected their kinematic background, as covered in Subsection 4.2.5.

The methodic conclusions of this work, ultimately aims to inspire future works to acknowledge the valuable knowledge gained, before but also since the resurrection of ML. It does not argue in any way against ML. It is a truly disruptive technology which has not yet reached its full maturity and potential. Nevertheless, domain adaptations and inter-domain inspiration are evidentially a good way to improve models.

List of Acronyms

ACC	Adaptive Cruise Control
ADAS	Advanced Driver Assistance System
ADE	Average Displacement Error
AEB	Automatic Emergency Breaking
AI	Artificial Intelligence
aka	also known as
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CRF	Conditional Random Field
DL	Deep Learning
DNN	Deep Neural Network
DSP	Digital Signal Processor
FCN	Fully Convolutional Network
FFT	Fast Fourier Transform
FLOP	Floating Point Operation
FPS	Frames per Second
GAN	Generative Adverserial Network
GAT	Graph Attention Network
GCN	Graph Neural Network
GPU	Graphics Processing Unit

GRU	Gated Recurrent Units
GTSRB	German Traffic Sign Recognition Benchmark
HMM	Hidden Markov Model
IoU	Intersection over Union
LSTM	Long Short Term Memory
ML	Machine Learning
MLP	Multi Layer Perceptron
MNIST	Modified National Institute of Standards and Technology
MSE	Mean Squared Error
NLP	Natural Language Processing
NN	Neural Network
RAM	Random Access Memory
ReLU	Rectifying Linear Unit
RGB	red, green and blue
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks
ROI	Region of Interest
SoC	System on a Chip
SotA	state-of-the-art
SSD	Solid State Drive
TTS	Text to Speech
UAV	Unmanned Aerial Vehicle
VAE	Varational Auto Encoder
VOC	Visual Object Classes
WHO	World Health Organisation
w.r.t	with respect to

dw	depth-wise
s	stride
mp	max pooling
gc	grouped convolutions
dm	depth multiplier
FC	Fully Connected

List of Figures

List of Tables

Bibliography

[1] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. "Hierarchical text-conditional image generation with clip latents". In: *arXiv preprint arXiv:2204.06125* (2022) (cit. on p. iv).

[2] Alan M Turing. "Computing machinery and intelligence". In: *Parsing the turing test*. Springer, 2009, pp. 23–65 (cit. on p. 1).

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on pp. 1, 9, 14, 15, 47, 54).

[4] Olga Russakovsky, Jia Deng, Hao Su, et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252 (cit. on p. 1).

[5] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995 (cit. on pp. 1, 15, 54).

[9] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297 (cit. on p. 2).

[11] Tomas Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013 (cit. on p. 2).

[12] Aäron van den Oord, Sander Dieleman, Heiga Zen, et al. "WaveNet: A Generative Model for Raw Audio". In: *Arxiv*. 2016 (cit. on p. 2).

[13] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 2).

[14] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734 (cit. on p. 2).

[15] Omer Sezer, Murat Ozbayoglu, and Erdogan Dogdu. "An Artificial Neural Network-based Stock Trading System Using Technical Analysis and Big Data Framework". In: Apr. 2017, pp. 223–226 (cit. on p. 3).

[16] Nowrouz Kohzadi, Milton Boyd, Ken Nagasaka, and Iebeling Kaastra. "A comparison of artificial neural network and time series models for forecasting commodity prices". In: *Neurocomputing* 10 (Mar. 1996), pp. 169–181 (cit. on p. 3).

[17] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *Computer Vision – ECCV 2014*. Springer International Publishing, 2014, pp. 740–755 (cit. on pp. 3, 9).

[18] Marius Cordts, Mohamed Omran, Sebastian Ramos, et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on pp. 3, 10, 17–19, 28, 29, 31, 34, 42).

[21] Christian Szegedy, Wei Liu, Yangqing Jia, et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9 (cit. on pp. 3, 47).

[25] Curtis Hawthorne, Andrew Jaegle, Cătălina Cangea, et al. “General-purpose, long-context autoregressive modeling with perceiver ar”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 8535–8558 (cit. on p. 4).

[26] Or Sharir, Barak Peleg, and Yoav Shoham. “The cost of training nlp models: A concise overview”. In: *arXiv preprint arXiv:2004.08900* (2020) (cit. on pp. 4, 47, 48).

[28] Li Deng. “The mnist database of handwritten digit images for machine learning research”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142 (cit. on p. 9).

[29] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009 (cit. on pp. 9, 68, 69, 77).

[30] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*. 2007 (cit. on pp. 9, 16).

[31] Ido Freeman and Jan Siegemund. “Realtime Semantic Segmentation Using Masked Conditional Random Fields”. U.S. pat. US16143741. Aptiv Limited Ltd. Oct. 2017 (cit. on pp. 9, 26).

[32] Ido Freeman and Pascal Colling. “Spatio-Focal Loss Adaptive Weighting of Semantic Segmentation Loss”. U.S. pat. EP21154378. Aptiv Limited Ltd. Jan. 2021 (cit. on pp. 9, 33, 35).

[34] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. 2012 (cit. on p. 10).

[35] Bolei Zhou, Hang Zhao, Xavier Puig, et al. “Scene parsing through ade20k dataset”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 633–641 (cit. on p. 10).

[38] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez. “Can Semantic Labeling Methods Generalize to Any City? The Inria Aerial Image Labeling Benchmark”. In: *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE. 2017 (cit. on p. 11).

[40] Debesh Jha, Sharib Ali, Krister Emanuelsen, et al. “Kvasir-instrument: Diagnostic and therapeutic tool segmentation dataset in gastrointestinal endoscopy”. In: *International Conference on Multimedia Modeling*. Springer. 2021, pp. 218–229 (cit. on p. 11).

[41] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587 (cit. on p. 11).

[42] Ross Girshick. “Fast r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448 (cit. on pp. 11, 13).

[43] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems* 28 (2015) (cit. on p. 11).

[44] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788 (cit. on p. 11).

[45] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. “Yolov4: Optimal speed and accuracy of object detection”. In: *arXiv preprint arXiv:2004.10934* (2020) (cit. on p. 11).

[46] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML ’01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289 (cit. on pp. 12, 19, 20).

[47] Radhakrishna Achanta, Appu Shaji, Kevin Smith, et al. “SLIC Superpixels Compared to State-of-the-Art Superpixel Methods”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 34.11 (Nov. 2012), pp. 2274–2282 (cit. on pp. 12, 18, 19).

[48] Philipp Kraehenbuehl and Vladlen Koltun. “Efficient inference in fully connected crfs with gaussian edge potentials”. In: *Advances in neural information processing systems*. 2011, pp. 109–117 (cit. on pp. 12, 19, 21, 22, 24, 27).

[49] Nadia Payet and Sinisa Todorovic. “(RF)ˆ2 – Random Forest Random Field”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. Vol. 23. Curran Associates, Inc., 2010 (cit. on p. 12).

[50] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully Convolutional Networks for Semantic Segmentation”. In: *CoRR* abs/1411.4038 (2014). eprint: `1411.4038` (cit. on pp. 12, 16, 17, 20, 29, 42).

[51] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs”. In: vol. 40. 4. IEEE, 2017, pp. 834–848 (cit. on pp. 12, 28, 33, 42, 43).

[52] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. “Rethinking Atrous Convolution for Semantic Image Segmentation”. In: *ArXiv* abs/1706.05587 (2017) (cit. on pp. 12, 33).

[53] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, et al. “Conditional random fields as recurrent neural networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1529–1537 (cit. on pp. 12, 23).

[54] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241 (cit. on pp. 12, 17, 33).

[55] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016 (cit. on pp. 12, 94, 101).

[56] Anurag Arnab, Shuai Zheng, Sadeep Jayasumana, et al. “Conditional random fields meet deep neural networks for semantic segmentation: Combining probabilistic graphical models with deep learning for structured prediction”. In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 37–52 (cit. on pp. 12, 24).

[57] Andrew Tao, Karan Sapra, and Bryan Catanzaro. “Hierarchical multi-scale attention for semantic segmentation”. In: *arXiv preprint arXiv:2005.10821* (2020) (cit. on p. 12).

[58] Peter J Burt. “Fast filter transform for image processing”. In: *Computer graphics and image processing* 16.1 (1981), pp. 20–51 (cit. on p. 12).

[59] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. “Mask r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969 (cit. on p. 13).

[60] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, et al. “Simple copy-paste is a strong data augmentation method for instance segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2918–2928 (cit. on p. 13).

[61] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133 (cit. on p. 14).

[62] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957 (cit. on p. 14).

[63] Christopher M Bishop et al. *Pattern recognition and machine learning*. Vol. 4. springer New York, 2006 (cit. on pp. 14, 19, 36, 37, 99).

[64] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004 (cit. on p. 14).

[65] Yann LeCun, Bernhard Boser, John S Denker, et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551 (cit. on pp. 14, 21).

[66] David H Hubel and Torsten N Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. In: *The Journal of physiology* 160.1 (1962), p. 106 (cit. on pp. 14, 15).

[68] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 14).

[69] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. “Benchmark analysis of representative deep neural network architectures”. In: *IEEE Access* 6 (2018), pp. 64270–64277 (cit. on pp. 15, 47, 48).

[70] Michael A. Nielsen. *Neural Networks and Deep Learning*. misc. 2018 (cit. on p. 15).

[71] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *ICML 2010*. 2010, pp. 807–814 (cit. on p. 15).

[72] Sebastian Lapuschkin, Alexander Binder, Grégoire Montavon, Klaus-Robert Muller, and Wojciech Samek. “Analyzing classifiers: Fisher vectors and deep neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2912–2920 (cit. on p. 16).

[73] Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, et al. “Unmasking Clever Hans predictors and assessing what machines really learn”. In: *Nature communications* 10.1 (2019), pp. 1–8 (cit. on p. 16).

[74] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. “Image classification with the fisher vector: Theory and practice”. In: *International journal of computer vision* 105.3 (2013), pp. 222–245 (cit. on p. 16).

[75] Pushmeet Kohli Nathan Silberman Derek Hoiem and Rob Fergus. “Indoor Segmentation and Support Inference from RGBD Images”. In: *ECCV*. 2012 (cit. on p. 16).

[76] Ido Freeman. *Using Continuous Graphical Models to Structure and Improve Disparity Estimations*. Tech. rep. University of Tuebingen, Germany, 2017 (cit. on p. 19).

[77] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations*. 2021 (cit. on pp. 19, 26).

[78] Xiang Zhang, Junbo Zhao, and Yann LeCun. “Character-level convolutional networks for text classification”. In: *Advances in neural information processing systems* 28 (2015) (cit. on p. 20).

[79] Renfrey Burnard Potts. “Some generalized order-disorder transformations”. In: *Mathematical proceedings of the cambridge philosophical society*. Vol. 48. 01. Cambridge Univ Press. 1952, pp. 106–109 (cit. on p. 21).

[80] Carlo Tomasi and Roberto Manduchi. “Bilateral filtering for gray and color images”. In: *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*. IEEE. 1998, pp. 839–846 (cit. on pp. 21, 120).

[81] Andrew Adams, Jongmin Baek, and Myers Abraham Davis. “Fast high-dimensional filtering using the permutohedral lattice”. In: *Computer graphics forum*. Vol. 29. 2. Wiley Online Library. 2010, pp. 753–762 (cit. on pp. 22, 23).

[82] Kaiming He, Jian Sun, and Xiaoou Tang. “Guided image filtering”. In: *European conference on computer vision*. Springer. 2010, pp. 1–14 (cit. on p. 22).

[83] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. “Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs”. In: *ICLR*. 2015 (cit. on pp. 23, 27).

[84] Monika Heift, Klaus Friedrichs, and Andre Paus. “Device and a method for detecting vehicle lights in an image”. U.S. pat. US11042764. Aptiv Limited Ltd. July 2019 (cit. on p. 26).

[85] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. “Design of an image edge detection filter using the Sobel operator”. In: *IEEE Journal of solid-state circuits* 23.2 (1988), pp. 358–367 (cit. on p. 30).

[86] Towaki Takikawa, David Acuna, Varun Jampani, and Sanja Fidler. “Gated-scnn: Gated shape cnns for semantic segmentation”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 5229–5238 (cit. on p. 33).

[87] Ke Sun, Yang Zhao, Borui Jiang, et al. “High-Resolution Representations for Labeling Pixels and Regions”. In: *ArXiv* abs/1904.04514 (2019) (cit. on p. 33).

[88] Yuhui Yuan, Jingyi Xie, Xilin Chen, and Jingdong Wang. “Segfix: Model-agnostic boundary refinement for segmentation”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 489–506 (cit. on pp. 33, 34).

[89] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. “Focal Loss for Dense Object Detection”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2999–3007 (cit. on pp. 34, 37, 38, 41, 43, 102).

[90] Shubhankar Borse, Ying Wang, Yizhe Zhang, and Fatih Porikli. “Inverseform: A loss function for structured boundary-aware segmentation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 5901–5911 (cit. on p. 34).

[91] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, et al. “Generalized intersection over union: A metric and a loss for bounding box regression”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 658–666 (cit. on p. 35).

[92] MTCAJ Thomas and A Thomas Joy. *Elements of information theory*. Wiley-Interscience, 1991 (cit. on p. 36).

[93] Solomon Kullback and Richard A Leibler. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86 (cit. on p. 37).

[94] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. “V-net: Fully convolutional neural networks for volumetric medical image segmentation”. In: *2016 fourth international conference on 3D vision (3DV)*. IEEE. 2016, pp. 565–571 (cit. on pp. 37, 47).

[95] Xianwei Zheng, Linxi Huan, Gui-Song Xia, and Jianya Gong. “Parsing very high resolution urban scene images by learning deep ConvNets with edge-aware loss”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 170 (2020), pp. 15–28 (cit. on pp. 38, 41, 43).

[96] Meng Chang, Huajun Feng, Zhihai Xu, and Qi Li. “Low-light image restoration with short-and long-exposure raw pairs”. In: *IEEE Transactions on Multimedia* (2021) (cit. on p. 38).

[97] Irwin Sobel and Gary Feldman. “A 3x3 isotropic gradient operator for image processing”. In: *a talk at the Stanford Artificial Project in* (1968), pp. 271–272 (cit. on p. 39).

[98] Jia Deng, Wei Dong, Richard Socher, et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255 (cit. on pp. 42, 77).

[99] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. “Rangenet++: Fast and accurate lidar semantic segmentation”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 4213–4220 (cit. on pp. 45, 47).

[100] Tiago Cortinhal, George Tzelepis, and Eren Erdal Aksoy. “SalsaNext: Fast, uncertainty-aware semantic segmentation of LiDAR point clouds”. In: *International Symposium on Visual Computing*. Springer. 2020, pp. 207–222 (cit. on p. 45).

[101] Pauline Luc, Natalia Neverova, Camille Couprie, Jakob Verbeek, and Yann LeCun. “Predicting deeper into the future of semantic segmentation”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 648–657 (cit. on p. 45).

[102] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Vol. 1. 2016, pp. 770–778 (cit. on pp. 47, 57, 66, 67).

[103] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660 (cit. on p. 47).

[104] Roger Ariew. “Ockham’s Razor: A Historical and Philosophical Analysis of Ockham’s Principle of Parsimony”. PhD thesis. University of Illinois at Urbana-Champaign, 1976 (cit. on pp. 47, 85).

[105] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* 1 (2014) (cit. on pp. 47, 52).

[106] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. “Reconciling modern machine-learning practice and the classical bias–variance trade-off”. In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854 (cit. on pp. 47, 50).

[107] Mingxing Tan and Quoc Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6105–6114 (cit. on pp. 48, 67).

[108] Colin Raffel, Noam Shazeer, Adam Roberts, et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *arXiv preprint arXiv:1910.10683* (2019) (cit. on p. 48).

[109] Ido Freeman, Lutz Roese Koerner, and Anton Kummert. “Effnet: An efficient structure for convolutional neural networks”. In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. Vol. 1. IEEE. 2018, pp. 6–10 (cit. on pp. 48, 70, 74, 78–82).

[110] Ido Freeman, Lutz Roese-Koerner, Christoph Petig, and Peet Cremer. “Robust Convolution Block for Small and Efficient Convolutional Neural Networks”. U.S. pat. US16241091. Aptiv Limited Ltd. Jan. 2018 (cit. on p. 48).

[111] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, eds. *Automated Machine Learning - Methods, Systems, Challenges*. Springer, 2019 (cit. on pp. 49, 50).

[112] Jeff Bilmes. *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*. Tech. rep. 1998 (cit. on p. 50).

[113] Amir Gholami, Sehoon Kim, Zhen Dong, et al. *A survey of quantization methods for efficient neural network inference*. Tech. rep. 2021 (cit. on pp. 50, 51).

[114] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. *What is the state of neural network pruning?* Tech. rep. 2020 (cit. on p. 50).

[115] Benoit Jacob, Skirmantas Kligys, Bo Chen, et al. “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018 (cit. on p. 51).

[116] Andrew G. Howard, Menglong Zhu, Bo Chen, et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. cite arxiv:1704.04861. 2017 (cit. on pp. 52, 68, 70, 73, 74, 78–80, 82).

[117] Andrew Howard, Mark Sandler, Grace Chu, et al. “Searching for mobilenetv3”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1314–1324 (cit. on pp. 52, 82).

[118] Laurent Sifre and Stéphane Mallat. “Rigid-motion scattering for texture classification”. In: *arXiv preprint arXiv:1403.1687* (2014) (cit. on p. 52).

[119] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6848–6856 (cit. on pp. 52, 61, 67, 68, 70, 73, 74, 78–80, 82).

[120] J.L.R. d’Alembert. *Recherches sur differens points importans du systême du monde*. Recherches sur differens points importans du systême du monde v. 2. Chez David l’aîné, 1754 (cit. on p. 53).

[121] Ronald Newbold Bracewell and Ronald N Bracewell. *The Fourier transform and its applications*. Vol. 31999. McGraw-Hill New York, 1986 (cit. on p. 54).

[122] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. “Feature visualization”. In: *Distill* 2.11 (2017), e7 (cit. on p. 55).

[125] Ulrich Drepper. “What every programmer should know about memory”. In: *Red Hat, Inc* 11 (2007), p. 2007 (cit. on p. 60).

[126] Forrest N Iandola, Song Han, Matthew W Moskewicz, et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and$<$ 0.5 MB model size”. In: *ArXiv e-prints* 1 (Apr. 2016), pp. 1–9. eprint: `1602.07360` (cit. on pp. 61, 66).

[127] Vincent Dumoulin and Francesco Visin. “A guide to convolution arithmetic for deep learning”. In: *ArXiv e-prints* 1 (Mar. 2016). eprint: `1603.07285` (cit. on p. 62).

[128] J.T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. “Striving for Simplicity: The All Convolutional Net”. In: *ICLR (workshop track)*. 2015 (cit. on p. 62).

[129] Harry Nyquist. “Certain topics in telegraph transmission theory”. In: *Transactions of the American Institute of Electrical Engineers* 47.2 (1928), pp. 617–644 (cit. on p. 63).

[130] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. “Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark”. In: *International Joint Conference on Neural Networks*. 1288. 2013 (cit. on pp. 65, 68, 78).

[131] Antônio H Ribeiro and Thomas B Schön. “How Convolutional Neural Networks Deal with Aliasing”. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 2755–2759 (cit. on p. 65).

[132] Richard Zhang. “Making convolutional networks shift-invariant again”. In: *International conference on machine learning*. PMLR. 2019, pp. 7324–7334 (cit. on p. 65).

[133] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. “Visualizing the Loss Landscape of Neural Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, et al. Vol. 31. Curran Associates, Inc., 2018 (cit. on pp. 66, 67).

[134] Chris Solomon and Toby Breckon. *Fundamentals of Digital Image Processing: A practical approach with examples in Matlab*. John Wiley & Sons, 2011 (cit. on pp. 67, 120).

[135] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826 (cit. on p. 67).

[136] Yuval Netzer, Tao Wang, Adam Coates, et al. *Reading digits in natural images with unsupervised feature learning*. 2011 (cit. on pp. 68, 77).

[137] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958 (cit. on p. 69).

[138] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. “Empirical Evaluation of Rectified Activations in Convolutional Network”. In: *CoRR* abs/1505.00853 (2015). arXiv: 1505.00853 (cit. on p. 73).

[139] Le Dong, Ling He, Mengdie Mao, et al. “CUNet: A compact unsupervised network for image classification”. In: *IEEE Transactions on Multimedia* 20.8 (2017), pp. 2012–2021 (cit. on pp. 73, 75).

[140] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520 (cit. on pp. 74, 78, 80–82).

[141] François Chollet. “Xception: Deep learning with depthwise separable convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258 (cit. on p. 75).

[142] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995 (cit. on p. 78).

[143] Martín Abadi, Ashish Agarwal, Paul Barham, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015 (cit. on p. 78).

[144] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015 (cit. on p. 78).

[145] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. “Shufflenet v2: Practical guidelines for efficient cnn architecture design”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 116–131 (cit. on p. 82).

[146] Stone Yun and Alexander Wong. “Where should we begin? a low-level exploration of weight initialization impact on quantized behaviour of deep neural networks”. In: *arXiv preprint arXiv:2011.14578* (2020) (cit. on p. 82).

[147] Stone Yun and Alexander Wong. “Do All MobileNets Quantize Poorly? Gaining Insights into the Effect of Quantization on Depthwise Separable Convolutional Networks Through the Eyes of Multi-scale Distributional Dynamics”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2447–2456 (cit. on p. 82).

[148] Tao Sheng, Chen Feng, Shaojie Zhuo, et al. “A quantization-friendly separable convolution for mobilenets”. In: *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*. IEEE. 2018, pp. 14–18 (cit. on p. 82).

[149] Ido Freeman, Kun Zhao, and Anton Kummert. “Polynomial Trajectory Predictions for Improved Learning Performance”. In: *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2021, pp. 3313–3317 (cit. on pp. 83, 86, 94, 95).

[151] Daniel Kahneman. *Thinking, fast and slow*. New York: Farrar, Straus and Giroux, 2011 (cit. on p. 83).

[152] Dirk Helbing and Peter Molnar. “Social force model for pedestrian dynamics”. In: *Physical review E* 51.5 (1995), p. 4282 (cit. on pp. 84, 87).

[153] Stefano Pellegrini, Andreas Ess, Konrad Schindler, and Luc Van Gool. “You’ll never walk alone: Modeling social behavior for multi-target tracking”. In: *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, pp. 261–268 (cit. on pp. 84, 115).

[154] Ramin Mehran, Alexis Oyama, and Mubarak Shah. “Abnormal crowd behavior detection using social force model”. In: *2009 IEEE conference on computer vision and pattern recognition*. IEEE. 2009, pp. 935–942 (cit. on pp. 84, 87).

[155] Nachiket Deo and Mohan M Trivedi. “Convolutional social pooling for vehicle trajectory prediction”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 1468–1476 (cit. on pp. 84, 91, 109, 113).

[156] Henggang Cui, Thi Nguyen, Fang-Chieh Chou, et al. “Deep kinematic models for physically realistic prediction of vehicle trajectories”. In: (2019) (cit. on pp. 84, 92, 96, 98).

[157] Mohammadhossein Bahari, Ismail Nejjar, and Alexandre Alahi. “Injecting knowledge in data-driven vehicle trajectory predictors”. In: *Transportation research part C: emerging technologies* 128 (2021), p. 103010 (cit. on pp. 84, 92).

[158] Christian Rupprecht, Iro Laina, Robert DiPietro, et al. “Learning in an uncertain world: Representing ambiguity through multiple hypotheses”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 3591–3600 (cit. on pp. 84, 90, 109).

[159] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, et al. “Multimodal trajectory predictions for autonomous driving using deep convolutional networks”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 2090–2096 (cit. on pp. 84, 91, 109).

[160] Wongun Choi, Khuram Shahid, and Silvio Savarese. “What are they doing? : Collective activity classification using spatio-temporal relationship among people”. In: Nov. 2009, pp. 1282–1289 (cit. on p. 84).

[161] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. “Crowds by example”. In: *Computer graphics forum*. Vol. 26. 3. Wiley Online Library. 2007, pp. 655–664 (cit. on p. 84).

[162] Vijay Kovvali, Vassili Alexiadis, and Lin Zahng. “Video-Based Vehicle Trajectory Data Collection”. In: *Transportation Research Board 86th Annual Meeting*. Citeseer. 2006 (cit. on pp. 84, 94, 108).

[163] Benjamin Coifman and Lizhe Li. “A critical evaluation of the Next Generation Simulation (NGSIM) vehicle trajectory dataset”. In: *Transportation Research Part B-methodological* 105 (2017), pp. 362–377 (cit. on p. 85).

[164] Robert Krajewski, Julian Bock, Laurent Kloeker, and Lutz Eckstein. “The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2018, pp. 2118–2125 (cit. on pp. 85, 108).

[165] Frederik Diehl, Thomas Brunner, Michael Truong Le, and Alois Knoll. “Graph neural networks for modelling traffic participant interaction”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 695–701 (cit. on pp. 85, 89).

[166] Hanna Krasowski, Xiao Wang, and Matthias Althoff. “Safe reinforcement learning for autonomous lane changing using set-based prediction”. In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2020, pp. 1–7 (cit. on pp. 85, 89).

[167] Kaouther Messaoud, Itheri Yahiaoui, Anne Verroust-Blondet, and Fawzi Nashashibi. “Attention based vehicle trajectory prediction”. In: *IEEE Transactions on Intelligent Vehicles* 6.1 (2020), pp. 175–185 (cit. on pp. 85, 89).

[168] Maximilian Schäfer, Kun Zhao, Markus Bühren, and Anton Kummert. “Context-Aware Scene Prediction Network (CASPNet)”. In: *arXiv preprint arXiv:2201.06933* (2022) (cit. on pp. 85, 91, 92).

[169] Dominic Spata, Arne Grumpe, Mirko Meuter, and Ido Freeman. “Methods and Systems for Predicting a Trajectory of an Object”. U.S. pat. EP21186073. Aptiv Limited Ltd. July 2021 (cit. on p. 86).

[170] Dominic Spata, Arne Grumpe, and Ido Freeman. “Variance Estimation for Deep-tracker”. U.S. pat. EP21188550. Aptiv Limited Ltd. July 2021 (cit. on pp. 86, 104).

[171] Kun Zhao, Ido Freeman, and Thomas Kurbiel. “Method and Computer System for Controlling the Movement of a Host Vehicle”. U.S. pat. GB2203519.0. Aptiv Limited Ltd. Mar. 2022 (cit. on pp. 86, 99, 114).

[172] Laura Leal-Taixé, Gerard Pons-Moll, and Bodo Rosenhahn. “Everybody needs some-body: Modeling social and grouping behavior on a linear programming multiple people tracker”. In: *2011 IEEE international conference on computer vision workshops (ICCV workshops)*. IEEE. 2011, pp. 120–127 (cit. on p. 87).

[173] Wongun Choi and Silvio Savarese. “A unified framework for multi-target tracking and collective activity recognition”. In: *European Conference on Computer Vision*. Springer. 2012, pp. 215–230 (cit. on p. 87).

[174] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, et al. “Social lstm: Human trajectory prediction in crowded spaces”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 961–971 (cit. on pp. 87, 89, 90, 102, 113).

[175] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. “Social gan: Socially acceptable trajectories with generative adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2255–2264 (cit. on p. 88).

[176] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, et al. “Generative adversarial nets”. In: *Advances in neural information processing systems* 27 (2014) (cit. on pp. 88, 91).

[177] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, et al. Vol. 30. Curran Associates, Inc., 2017 (cit. on pp. 88, 89).

[178] Amir Sadeghian, Vineet Kosaraju, Ali Sadeghian, et al. “SoPhie: An Attentive GAN for Predicting Paths Compliant to Social and Physical Constraints”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019 (cit. on pp. 88, 90).

[179] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. “Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst”. In: *arXiv preprint arXiv:1812.03079* (2018) (cit. on pp. 89, 104).

[180] Christopher M Bishop. “Mixture density networks”. In: (1994) (cit. on p. 90).

[181] Alex Graves. *Generating sequences with recurrent neural networks*. Tech. rep. 2013 (cit. on pp. 90, 106, 107).

[182] Jean Mercat, Nicole El Zoghby, Guillaume Sandou, Dominique Beauvois, and Guillermo Pita Gil. “Kinematic Single Vehicle Trajectory Prediction Baselines and Applications with the NGSIM Dataset”. In: *arXiv preprint arXiv:1908.11472* (2019) (cit. on p. 90).

[183] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013) (cit. on p. 91).

[184] Joey Hong, Benjamin Sapp, and James Philbin. “Rules of the road: Predicting driving behavior with a convolutional model of semantic interactions”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8454–8462 (cit. on p. 91).

[185] Thomas Gilles, Stefano Sabatini, Dzmitry Tsishkou, Bogdan Stanciulescu, and Fabien Moutarde. “Home: Heatmap output for future motion estimation”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 500–507 (cit. on p. 91).

[186] Thomas Kurbiel, Akash Sachdeva, Kun Zhao, and Markus Buehren. “PrognoseNet: A Generative Probabilistic Framework for Multimodal Position Prediction given Context Information”. In: *arXiv preprint arXiv:2010.00802* (2020) (cit. on p. 91).

[187] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011 (cit. on p. 92).

[188] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. “Kinematic and dynamic vehicle models for autonomous driving control design”. In: *2015 IEEE intelligent vehicles symposium (IV)*. IEEE. 2015, pp. 1094–1099 (cit. on p. 92).

[189] Charles Richter, Adam Bry, and Nicholas Roy. “Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments”. In: *Robotics research*. Springer, 2016, pp. 649–666 (cit. on pp. 93, 116).

[190] Alexandr Andoni, Rina Panigrahy, Gregory Valiant, and Li Zhang. “Learning polynomials with neural networks”. In: *International conference on machine learning*. PMLR. 2014, pp. 1908–1916 (cit. on p. 93).

[191] Juan-Manuel Pérez-Rúa, Tomas Crivelli, Patrick Bouthemy, and Patrick Pérez. “Learning how to be robust: Deep polynomial regression”. In: *arXiv preprint arXiv:1804.06504* (2018) (cit. on p. 93).

[192] Grigorios G Chrysos, Stylianos Moschoglou, Giorgos Bouritsas, et al. “P-nets: Deep polynomial neural networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 7325–7335 (cit. on p. 93).

[193] J Bernardo, J Berger, APAFMS Dawid, A Smith, et al. “Regression and classification using Gaussian process priors”. In: *Bayesian statistics* 6 (1998), p. 475 (cit. on pp. 95, 105).

[194] Mark AZ Dippé and Erling Henry Wold. “Antialiasing through stochastic sampling”. In: *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*. 1985, pp. 69–78 (cit. on p. 100).

[195] Tung Phan-Minh, Elena Corina Grigore, Freddy A Boulton, Oscar Beijbom, and Eric M Wolff. “Covernet: Multimodal behavior prediction using trajectory sets”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 14074–14083 (cit. on p. 109).

[196] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. “On the properties of neural machine translation: Encoder-decoder approaches”. In: *arXiv preprint arXiv:1409.1259* (2014) (cit. on p. 110).

[197] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, et al. “Noise2Noise: Learning image restoration without clean data”. In: *arXiv preprint arXiv:1803.04189* (2018) (cit. on p. 111).

[198] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362 (cit. on p. 112).

[199] Rudolph Emil Kalman. “A new approach to linear filtering and prediction problems”. In: (1960) (cit. on p. 112).

[200] Charlie Tang and Russ R Salakhutdinov. “Multiple futures prediction”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 15398–15408 (cit. on p. 113).

Web Pages

[6] Standford University. *An End-to-End Deep Learning Benchmark and Competition*. Accessed on 24.04.2020. URL: `https://dawn.cs.stanford.edu/benchmark/ImageNet/train.html` (cit. on p. 1).

[7] Papers with Code. *Image Classification on ImageNet*. Accessed on 26.05.2022. URL: `https://paperswithcode.com/sota/image-classification-on-imagenet?metric=Number%20of%20params&dimension=Top%201%20Accuracy` (cit. on p. 1).

[8] Thompson Alan. *Selected Models Size Comparison*. Accessed on 02.10.2022. URL: `https://lifearchitect.ai/models/` (cit. on p. 2).

[10] Statista. *Funding of artificial intelligence (AI) startup companies worldwide from 2014 to 2019*. Accessed on 28.04.2020. URL: `https://www.statista.com/statistics/621468/worldwide-artificial-intelligence-startup-company-funding-by-year/` (cit. on p. 2).

[19] TractableAI. *TractableAI*. Accessed on 30.04.2020. URL: `https://www.tractable.ai/` (cit. on p. 3).

[20] DeepMind. *DeepMind*. Accessed on 30.04.2020. URL: `https://www.deepmind.com/` (cit. on p. 3).

[22] WHO. *Death on the roads*. Accessed on 28.04.2020. URL: `https://extranet.who.int/roadsafety/death-on-the-roads/` (cit. on p. 3).

[23] NCAP. *Euro NCAP Roadmap 2016–2020*. Accessed on 28.04.2020. URL: `https://www.euroncap.com/en/about-euro-ncap/timeline/roadmap-2016-2020` (cit. on p. 3).

[24] nVidia. *NVIDIA RTX A6000-GRAFIC CARDS*. Accessed on 15.05.2022. URL: `https://www.nvidia.com/de-de/design-visualization/rtx-a6000/` (cit. on p. 4).

[27] nVidia. *nVidia Autonomous Machines*. Accessed on 30.04.2020. URL: `https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/` (cit. on p. 4).

[33] ByteBridge. *What is Semantic Segmentation, Instance Segmentation, Panoramic segmentation?* Accessed on 21.04.2022. URL: `https://becominghuman.ai/what-is-semantic-segmentation-instance-segmentation-panoramic-segmentation-3bbb03856c12` (cit. on p. 10).

[36] *Segmentation of neuronal structures in EM stacks challenge - ISBI 2012*. Accessed on 07.02.2022. 2012. URL: `https://imagej.net/events/isbi-2012-segmentation-challenge` (cit. on p. 11).

[37] *2021 Cell Tracking Challenge*. Accessed on 07.02.2022. 2021. URL: `http://celltrackingchallenge.net/` (cit. on p. 11).

[39] *Drone Deploy*. Accessed on 07.02.2022. 2021. URL: `https://www.dronedeploy.com/` (cit. on p. 11).

[67] Tim Sainburg. *Visualizing features, receptive fields, and classes in neural networks from "scratch" with Tensorflow 2. Part 2: Visualizing features and receptive fields*. Accessed on 31.01.2023. URL: `https://timsainburg.com/tensorflow-2-feature-visualization-visualizing-features.html` (cit. on p. 15).

[123] Texas Instruments. *TDA3LA*. Accessed on 22.03.2022. URL: `https://www.ti.com/product/TDA3LA` (cit. on pp. 58, 71, 77).

[124] EventHelix. *Why software developers should care about CPU caches*. Accessed on 03.01.2021. Dec. 2021. URL: `https://medium.com/software-design/why-software-developers-should-care-about-cpu-caches-8da04355bb8a` (cit. on p. 60).

[150] Society of Automotive Engineers. *J3016 automated-driving graphic update*. Accessed on 03.12.2021. Jan. 2019. URL: `https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic` (cit. on p. 83).