Aspects of Active Learning, Architecture Search and Lidar Panoptic Segmentation towards Pedestrian Perception in Autonomous Driving

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

Dissertation

---

zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften

von

Lukas Hahn

aus Wuppertal

Wuppertal, 2022

Tag der Prüfung: 02.02.2022
Hauptreferent: Prof. Dr.-Ing. Anton Kummert
Korreferent: Prof. Dr.-Ing. Bela Gipp

Abstract

This thesis addresses and develops several new aspects of machine learning which ultimately contribute to the development of a novel pedestrian perception system for autonomous driving in urban environments. After explaining the problem in more detail in chapter 1, chapter 2 gives a brief introduction to the main principles of machine learning. Chapter 3 presents a study on the robustness of different pool-based active learning approaches and presents a new query method, as well as drawing conclusions for the practical applicability of these methods in an industrial setting. The search for the best architecture of a convolutional neural network is the subject of chapter 4, in which a heuristic for the fast and reliable evaluation of such configurations is presented and combined with global optimisation. Chapter 5 includes the description of a new system for real-time panoptic segmentation of lidar point clouds, which combines a clustering approach with efficient classification. After highlighting further aspects of pedestrian feature extraction in chapter 6, chapter 7 finally describes a system for their localisation and motion prediction. A final conclusion and an outlook on possible extensions are given in chapter 8.

Zusammenfassung

Die vorliegende Arbeit behandelt und entwickelt neue Aspekte maschinellen Lernens, die letztendlich zur Entwicklung eines neuartigen Systems zur Wahrnehmung und Einschätzung von Fußgängern für das autonome Fahren im urbanen Raum beitragen. Nachdem in Kapitel 1 die Problemstellung näher erläutert wird, gibt Kapitel 2 eine kurze Einführung in die wichtigsten Grundlagen des Machine Learning. In Kapitel 3 wird eine Studie über die Robustheit verschiedener Pool-Based Active Learning-Ansätze vorgestellt und eine neue Abfrage-Methode präsentiert, sowie Schlüsse für die praktische Anwendbarkeit dieser Verfahren in einem industriellen Umfeld gezogen. Die Suche nach der besten Architektur eines Convolutional Neural Network ist Gegenstand von Kapitel 4, in dem ein heuristisches Verfahren zur schnellen und zuverlässigen Bewertung solcher Konfigurationen vorgestellt und mit einem Ansatz der globalen Optimierung verbunden wird. Kapitel 5 umfasst die Beschreibung eines neuen Systems zur echtzeitfähigen panoptischen Segmentierung von Lidar-Punktwolken, welches einen Ansatz zum Clustering mit einer effizienten Klassifizierung kombiniert. Nachdem in Kapitel 6 weitere Aspekte zur Extraktion der Merkmale von Fußgängern beleuchtet werden, beschreibt Kapitel 7 schließlich ein System für deren Lokalisierung und Bewegungs-Prädiktion. Ein abschließendes Fazit und ein Ausblick auf mögliche Erweiterungen erfolgen in Kapitel 8.

List of Abbreviations

ADAS Advanced Driver Assistance System

AHC Automatic Headlight Control

AL Active Learning

CNN Convolutional Neural Network

CPU Central Processing Unit

DBSCAN Density-Based Spatial Clustering of Applications with Noise

DNN Deep Neural Network

FN False Negative

FP False Positive

GPS Global Positioning System

GPU Graphics Processing Unit

IMU Inertial Measurement Unit

Lidar Light Detection and Ranging

MBO Model-Based Optimisation

ML Machine Learning

ReLU Rectified Linear Unit

RGB Red, Green, Blue

SGD Stochastic Gradient Descent

SLAM Simultaneous Localisation and Mapping

SVM Support Vector Machine

TN True Negative

TP True Positive

Contents

1 Introduction

> *If everyone is moving forward together, then success takes care of itself.*
>
> — **Henry Ford**
> *(attributed to)*

1.1 Motivation

The concept of automated vehicles and driverless traffic is nearly as old as the invention of the car itself. It took less than two decades from its commercialisation, with Ford's *"Model T"* in 1908, for a company to present a radio controlled automobile to the public in the year 1925 (Time Magazine, 1925). While this showcase merely transferred the controls to the following car equipped with a radio transmitter, it demonstrates how present the thought of being driven by an automated vehicle was for people even at that time.
Approximately 60 years later, the programme *"PROMETHEUS"* was launched by the European research initiative EUREKA in 1986 to spearhead endeavours in the automotive industry's development towards the first real machine-controlled autonomous cars. While the actual scope of automated vehicle functionality within the project can be regarded as rudimentary compared with present requirements, the initiative created foundations for many advanced driver assistance systems (ADAS) widely used today. These included prototypes of adaptive cruise control systems, first blueprints for automated lane change and emergency breaking systems, as well as preliminary stages of early on-board navigation devices. With both academic and industrial activities in 11 European countries over a period of 96 months and total funding of more than 700 million Euros equivalent value, the programme marks a unique effort in this field (EUREKA, 2012). In the decades that followed, many succeeding initiatives were rolled out with *"@CITY"* being the current German government project in this legacy. As the title indicates, more than another thirty years later, focus of automated driving research and predevelopment now eventually arrived on the most complex setting of road traffic; inner city scenarios.

Automated driving in the city is a demanding challenge for two major reasons. First and foremost, it must be concerned with the safety and integrity of vulnerable road users like pedestrians and cyclists. Possible malfunctions could be especially severe in this environment, as accidents even at very low velocities will likely lead to personal injuries. Secondly, vehicle traffic itself is much more complex. Particularly in narrow streets with increased pedestrian traffic and residential areas away from the main road.
Semi-automatic assistance systems, which started to make their way into production vehicles in recent years, are primarily designed to assist the driver in narrowly defined scenarios on the highway. Here, traffic flow is separated by directions, free of intersections, roundabouts and traffic lights and the layout of the road is widely standardised. While making a machine follow a large set of traffic rules in a non-standardised environment in itself calls for considerable efforts to be made, it is the dynamism of these surroundings that immensely expands system requirements. Furthermore, unpredictable behaviour of all traffic participants needs to be accounted for, as by nature, pedestrians and human drivers alike will at some point exhibit disregard of traffic regulations, either deliberately or unintentionally.

The field of machine learning offers a variety of techniques that can be used to develop solutions to these problems. By design, this class of algorithms plays to its strengths whenever a manual formulation of a model to solve a task is far too complex or impossible due to unknown underlying feature characteristics. This makes it a tool of paramount importance for the concept of autonomous driving.

1.2 Outline and Context

The present thesis is concerned with providing new methods and considerations targeted towards the development of an experimental system for pedestrian detection, understanding and behaviour prediction for autonomous driving in urban scenarios. It describes findings in both machine learning fundamentals and its applications to automotive sensor data.

This work is subdivided into the following chapters and parts:

In a first part, fundamentals of machine learning algorithms and utilised sensors are described in chapter 2.

Chapter 3 covers active learning and its application to supervised classification problems. Its practicality to reduce human annotation effort and time and cost requirements accordingly is analysed and validated on public and private data sets. Here, special attention is paid to how robust existing and new methods are against the influence of changing hyperparameters and whether a selection made with one neural network architecture remains optimal for another slightly different network layout. The latter can become particularly relevant when models need to be scaled down after a research or predevelopment period, to be implemented in a product under computing power restrictions.

A new method for architecture selection of convolutional neural networks (CNN) is the topic of chapter 4. With a focus on balancing time consumption and precision, a heuristic approach to evaluate CNN architectures is developed and compared to other methods. This is then combined with aspects of random search and Bayesian optimisation to create a complete selection algorithm.

Chapter 5 describes a system for real-time panoptic segmentation in lidar sensor data. First, a novel clustering algorithm, leveraging methods to preserve three-dimensional information after reduction to a two-dimensional representation for fast computation, is introduced. Hereafter, a concept for optimised data representation and computationally efficient classification of identified object clusters is presented. Additional remarks are made to underline the capability of both building blocks to be combined offering a competitive solution for real-time object detection in lidar point clouds.

Chapter 6 treats further considerations regarding pedestrian behavioural feature extraction. It proposes an extension to proven methods for human body key point detection. Here the target is to make performance more robust for automotive

scenarios where low camera resolution and longer distances from vehicle to pedestrian can lead to very limited image details. More importantly, the proposed network consist of significantly fewer parameters than leading approaches utilising very deep architectures. In addition to this, remarks on the detection of pedestrian awareness are given.

The combination of aforementioned functions with additional building blocks for pedestrian localisation and movement prediction into a complete system is explained in chapter 7. A framework is presented, describing information flow from raw sensor data through various algorithm blocks towards generating an output useful to subsequent vehicle path planning and risk assessment systems. Additional examples for applications of concepts introduced in this work are also given.

Finally, the major findings of this thesis are summarised in a conclusion in chapter 8 along with thoughts on further development in this field and meaningful combination of machine learning algorithms and deterministic programming in the context of autonomous vehicles.

This thesis is the result of the scholarship *"Detection of Interaction between Traffic Participants"* provided by the University of Wuppertal through funding from Aptiv Services Deutschland GmbH.
Its contents where in parts developed in cooperation with the project *"@CITY - Automatisierte Fahrfunktionen"*[1], a research initiative supported by the Federal Ministry for Economic Affairs and Energy on the basis of a decision by the German Bundestag. The programme, with partners from both the automotive industry and academia taking part, is organised into seven subprojects with activities started between September 2017 and July 2018. The project is scheduled for completion in June of 2022.

[1] https://www.atcity-online.de/

The contents of this thesis have been partly published in the following articles:

- **Lukas Hahn, Frederik Hasecke, Anton Kummert**
  "Fast Object Classification and Meaningful Data Representation of Segmented Lidar Instances",
  Proceedings of the 23rd IEEE International Conference on Intelligent Transportation Systems (IEEE ITSC), pages 1-6, DOI: 10.1109/ITSC45102.2020.9294217, Rhodos (Greece), 2020

- **Lukas Hahn, Lutz Roese-Koerner, Peet Cremer, Urs Zimmermann, Ori Maoz, Anton Kummert**
  "On the Robustness of Active Learning",
  Proceedings of the 5th Global Conference on Artificial Intelligence (GCAI), EPiC Series in Computing, Volume 65, pages 152-162, DOI: 10.29007/thws, Bolzano (Italy), 2019

- **Lukas Hahn, Lutz Roese-Koerner, Klaus Friedrichs, Anton Kummert**
  "Fast and Reliable Architecture Selection for Convolutional Neural Networks",
  Proceedings of the 27th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), pages 179-184, ISBN: 978-2-87587-065-0, Bruges (Belgium), 2019

and

- **Frederik Hasecke, Lukas Hahn, Anton Kummert**
  "FLIC: Fast Lidar Image Clustering",
  Proceedings of the 10th International Conference on Pattern Recognition Applications and Methods (INSTICC ICPRAM), Volume 1, pages 25-35, ISBN: 978-989-758-486-2, DOI: 10.5220/0010193700250035, Online, 2021

as well as

- **Lukas Hahn, Maximilian Schäfer, Kun Zhao, Frederik Hasecke, Yvonne Schnickmann, André Paus**
  "Detection System for Predicting Information on Pedestrian",
  European Patent Application EP21154871.4, 2021 and
  US Patent Application PCT/US/17/649,672, 2022.

2 Fundamentals

This chapter aims to provide the reader with a description of the field of machine learning in general, convolutional neural networks (CNNs) in particular and the underlying mathematical methods employed. It by no means claims to be complete or give a full introduction to this complex and constantly evolving topic. All references to real world application of the methods presented in the following have been researched carefully but notwithstanding this fact remain selective through their sources' perspective on the overall picture.
These fundamentals are concluded with descriptions of the technical characteristics of sensors that are of crucial importance for automotive applications, namely camera and lidar.

2.1 Machine Learning

Machine learning (ML) is one of, if not the fastest growing field of computer science in recent years. Resulting from striving for artificial intelligence it is not only subject of vivid discussions within the research community, but increasing industrial application and even coverage in mainstream media.
It originates from the pioneer work of Alan Turing, as well as Marvin Minsky in the early 1950s. In 1951, Minsky build a neurocomputer called "SNARC" (Stochastic Neural Analog Reinforcement Computer) with the help of Dean Edmonds (Minsky, 1952). It was the first machine to resemble a neural network using 40 "neurons". Inspired by the experiments of B.F. Skinner on reinforcement learning with laboratory rats and hand build from tubes, motors, capacitors and clutches, the machine was able to learn the same behaviour a rat trying to get food would. Only one year later, Arthur Samuel developed a machine at IBM which learned to play the game of checkers (Samuel, 1959).

One of the milestones on the way towards today's understanding of machine learning was Frank Rosenblatt's invention of the perceptron in 1958 (Rosenblatt, 1958). His algorithm was implemented in the "Mark I Perceptron" machine, build at the Cornell Aeronautical Laboratory. Although weight updates were still performed mechanically

by electrical motors moving potentiometers, the machine's application already was what today is referred to as image recognition. It was connected to a camera using an array of 20×20 cadmium sulfide photocells resulting in an 400-pixel image. The concept of the perceptron is still the underlying principle of modern classification algorithms.

After exaggerating the capabilities of his invention, Rosenberg's predictions were corrected by Marvin Minsky and Seymour Papert in their 1969 book "Perceptrons" (Minsky and Papert, 1969). This led to a period of reduced funding for artificial intelligence, and in combination with the limited computer power available at the time caused the field to stagnate for many years.

Another important method still frequently used in machine learning today is the support vector machine (SVM), invented by Vladimir Vapnik and Alexey Chervonenkis in 1963 (Vapnik and Lerner, 1963; Vapnik and Chervonenkis, 1964) and revised for its contemporary form by Corinna Cortes and Vapnik in 1995 (Cortes and Vapnik, 1995). The invention to definitively re-popularise intelligent was IBM's "Deep Blue", a chess-playing computer that defeated world champion Garry Kasparov in 1997 (Campbell et al., 2001). Although mainly benefiting from brute computational force to evaluate millions of positions per second, the win had symbolic significance in showing how artificial intelligence was becoming capable of challenging humans in specific tasks. In 2016, Google's "AlphaGo" set a new landmark by beating professional player Lee Sedol in a game of Go. Although both tasks show similarities, the ancient Chinese game has a far higher branching factor than chess, resulting in a much larger number of possible moves per turn. A brute force approach was therefore rejected and the system used reinforcement learning of a deep neural network in combination with a tree search technique (Silver et al., 2016).

The field of machine learning addresses the question of how a computer can learn to perform a certain task, generating an appropriate response or output when receiving a given input. The main objective is to generalise from data observed during the learning process to new, unseen data. Thus, it is not about simply remembering the training data. To approach this problem, one must decide upon a few points.
The first being the actual task the machine is asked to learn. It defines how any example data presented as input to the system has to be processed. The data itself can be regarded as a collection of features representing key information of the underlying problem. Whether distinct features have to be selected by hand or can be extracted by the algorithm itself is depending on the particular learning method. A large portion of ML tasks can be broken down to the function of classification. Here, the system is asked to assign a single input to one of n distinct classes.

The second aspect of designing a machine learning algorithm is to choose what kind of experience the machine is allowed to have. This refers to the way the algorithm is trained to perform its task. Learning with fixed datasets can be divided into two categories. In the case of unsupervised learning, a set of data is presented to the system which is asked to learn useful features from these inputs independent from additional information. A possible application for this method could be clustering to find hidden underlying similarities within partitions of the entire input data space. In contrast to that, supervised learning provides a target or label with every input. If the task is to classify pieces of furniture, every input representing a chair will receive the same label, while every table is labelled as such, and so on. Other learning paradigms, like reinforcement learning are beyond the scope of this work and therefore will be left unexplained at this point.

Lastly, it is to be determined how the performance of the specified machine learning algorithm should be measured. For classification tasks, this is either the accuracy or the error rate. A percentage value describing how many inputs have been classified correctly or stating the portion of false classifications respectively. In general, this value is measured not only on the data used to train the classifier, but more importantly on a separate partition kept solely to validate the performance.
In his 1997 book "Machine Learning", Tom Mitchell summarised this in the following often cited quote: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."* (Mitchell, 1997a).
Following the presented requirements, a single input for supervised learning of a classification task must at least contain the following information: 1) The actual input, for example an image represented by a matrix. 2) A label, for example the number of the associated class in the form of an integer. And 3), a value expressing the affiliation to either the training or the validation partition of the dataset.

2.1.1 Neural Networks

In the last two decades, neural networks have possibly become the most popular method for machine learning. As abstract models of neurons in the brain, they aim to imitate the way we as humans learn to process information (Rojas, 1996). Although our brain achieves its capabilities through massive parallel and hierarchical networking using billions of neurons, a single one can be considered a distinct "computational" unit. Input signals are electric impulses carried along the dendrites towards the cell body (see figure 2.1). The output information is then forwarded

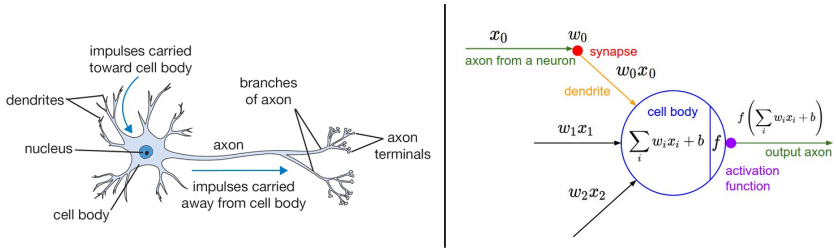


Fig. 2.1.: A biological neuron and its mathematical model (Fei-Fei et al., 2017).

onward the axon, where the message will eventually be transmitted to the dendrites of other neurons through the axon terminals. In the computational model trying to mimic this biological phenomenon, an input signal x_i is multiplied with a weight w_i resembling the strength of a synapse[1]. The cell body then sums up all weighted inputs $\sum_i w_i x_i + b$ adding a bias value b (Mitchell, 1997b). If the final sum is above a certain threshold, the neuron is activated, it *fires*. In practice, non-linear activation functions are used instead of fixed thresholds. The output function of such a neuron can therefore be expressed as $f\left(\sum_i w_i x_i + b\right)$.

A few dozen up to thousands of neurons together can form a network. Therefore, they are organised into a hierarchical structure of layers, where each single neuron from the previous layer is an input to every neuron of the next one. Neurons within one layer share no connections. In contrast to other learning methods, neural networks are not dependent on feature characteristics being extracted prior to training but attempt to detect relevant features in the data on their own during the training process. This fact is the key to their enormous success in the recent years and suitability for a broad variety of tasks.

Classification and Loss Function

At the output of every neural network, one would like to calculate a value to determine whether a given input was classified correctly using the current set of weights. A value on the basis of which a training process could be started to optimise the classification result. For this purpose, the term classification should be specified by a simple example. Considering a Cartesian plane with two groups of points as pictured in figure 2.2, these could be regarded as members of two classes,

[1] N.B.: In the following, the running index i is used to indicate single elements $x_i \in \{x_1, \ldots, x_n\}$, whereas j represents whole input samples $\vec{x}_j \in \{\vec{x}_1, \ldots, \vec{x}_m\}$. Therefore, a sample vector, consisting of elements x_i, will take the form $\vec{x} = (x_1, \ldots, x_n)^T$.

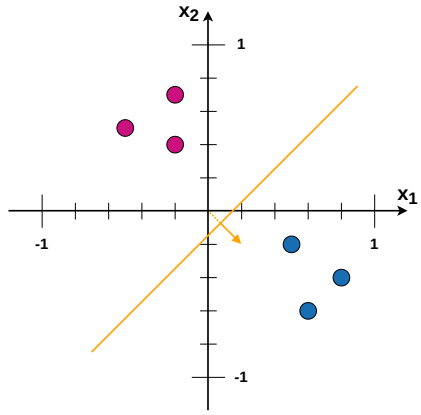


Fig. 2.2.: An example of linear classification of two classes.

distinguishable from each other through characteristic features, that one would like to separate. The most straightforward way to do so, would be to draw a straight line between the two groups. Instead of the smallest possible distance from the points to the line, which is something one would want to achieve in regression analysis when fitting a line using a linear least squares approach for example, this margin is to be maximised. This is what a linear support vector machine would do using the well-known hinge loss function

$$\Phi\left(\vec{w}, b, X, Y\right) = \left[\frac{1}{m}\sum_{j=1}^{m} f_j\left(\vec{w}, b, \vec{x}_j, y_j\right)\right] + \lambda||\vec{w}||^2 \tag{2.1}$$

with

$$f_j\left(\vec{w}, b, \vec{x}_j, y_j\right) = \max\left(0, 1 - y_j\left(\vec{w}\vec{x}_j + b\right)\right) \tag{2.2}$$

with the weight vector $\vec{w}$, the bias b, the inputs $\vec{x}_j \in X = \{\vec{x}_1, \ldots, \vec{x}_m\}$ and the corresponding class labels $y_j \in Y = \{y_1, \ldots, y_m\}$ (James et al., 2013). The additional parameter λ defines the relation of the data loss to the regularisation loss, a trade-off between increasing the margin size and the correct separation of the classes. Returning to the example, every j-th input $\vec{x}_j$ is a pair of values containing the coordinates. Each y_j can be either 1 or -1, depending on the class the j-th point $\vec{x}_j$ belongs to. The separating line, which rather is a plane mathematically speaking, is determined by the weights, in this case a two-dimensional column vector. The values of the weights describe a normal vector to the plane, the bias specifies its distance from the origin. Without this additional parameter, the SVM could only generate solutions which pass through the origin of the coordinate system.

In the given example, a good solution could be achieved by setting the weights $\vec{w} = (1, -1)$ and the bias $b = -0.1$. For these values, the argument of the max function for $i = 1$ would be $f_1 = 1 - 1 \cdot (1 \cdot 0.5 - 1 \cdot (-0.2) - 0.1) = 0.4$ and the overall loss would become

$$\phi = \left[\frac{1}{6}(0.4 + 0 + 0 + 0.3 + 0 + 0)\right] + 0.1 \cdot \left(1^2 + -1^2\right) \approx 0.3167 \quad \text{with} \quad \lambda = 0.1.$$

Although the hinge loss could theoretically be used for multi-class classification with neural networks, it is common to use more sophisticated functions. In practice, the softmax log-loss

$$l(\vec{z}) = -\log(\vec{z}), \tag{2.3}$$

which is based on the output of the softmax function

$$z_i = \sigma(\vec{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \quad \text{for } i = 1, \ldots, n \text{ and } \vec{x} = (x_1, \ldots, x_n)^T \tag{2.4}$$

for elements of $\vec{z} = (z_1, \ldots, z_n)$, is a very popular option (Vedaldi et al., 2015). The softmax function maps arbitrary real inputs on real values in the range $[0, 1]$ that add up to 1, which is ideal for multi-class classification with neural networks. It enables a clear and normalised interpretation of the activation of the final class-neurons, a percentage value for each class, and allows for the possibility to easily add an optional threshold to this activation.
In practice, this function is widely used in combination with the negative log-likelihood, as the loss will approach 0 when the confidence approaches 1 and will grow logarithmically towards infinity the smaller the confidence becomes.

In applications the terms loss function, cost function or objective function are often used interchangeably. When assessing or comparing the performance of machine learning algorithms, the loss itself plays the minor part. As described before, the capability to solve a certain task is usually expressed through a percentage value, characterizing either the partition of correctly classified or misclassified validation samples.

Backpropagation

While at this point it should be clear, how a neural network can be formed and what kind of objective or loss functions can be used to measure the classification result, this introduction still misses a very important aspect. How can the parameters of a classification method be adapted to yield better results? Or, figuratively speaking;

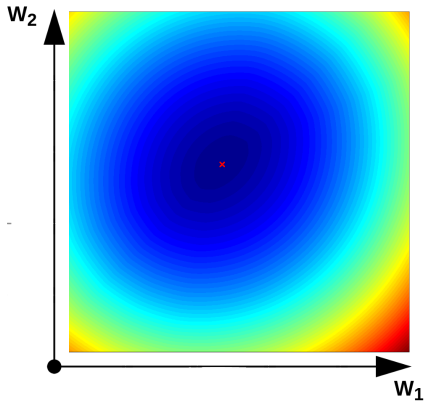


Fig. 2.3.: Example of an error surface.

how can a machine be made to learn? How can the point of minimal loss, denoting the smallest training classification error, be reached?

With the example from the previous paragraph it is easy to visualise the surface of the loss function as seen in figure 2.3. This graphic account was created by parameterising both weights w_1 and w_2 with values between -10 and 10 and a fixed bias $b = -0.1$. The red cross indicates the point $(1, -1)$ previously evaluated. It shows this point to already describe an optimal combination of weights for the given bias. Assuming one would have to solve a more complex classification problem, how could the weights of the learning algorithm be optimised? It is obvious that one would like to use an iterative process of refining the weights to minimise the given loss. It is desired to reach the error surface's global minimum starting from an arbitrary initial point. Since the dimension of the error surface is equal to the number of parameters in the learning algorithm, real world problems imply a multidimensional error surface impossible to imagine. Therefore, the three dimensional bowl-like error surface from the previous example is used and ways to descend along its incline down towards the (global) minimum are discussed.
The gradient is the mathematical operator used to gain information on the slope of a multi-variable function. For a function f, it can be denoted as

$$\nabla f(\vec{x}) := \frac{\partial f}{\partial x_1}\vec{e}_1 + \cdots + \frac{\partial f}{\partial x_n}\vec{e}_n \tag{2.5}$$

with the partial derivative

$$\frac{\partial f(x_1, x_2, \ldots, x_n)}{\partial x_i} := \lim_{h \to 0} \frac{f(x_1, x_2, \ldots, x_n + h, \ldots x_n) - f(x_1, x_2, \ldots, x_n)}{h} \tag{2.6}$$

and the unit vectors $\vec{e}_i$. As a vector field of partial derivatives, the gradient points into the direction of the greatest rate of increase of the function, with the magnitude of the slope in that direction. When the steepest descent is to be found, it is hence logical to use the negative gradient for weight optimisation (Rumelhart et al., 1986a).To calculate the next weights $\vec{w}_{t+1}$ from the previous $\vec{w}_t$ at time t, a weight-update can be expressed as

$$\vec{w}_{t+1} = \vec{w}_t - \epsilon_t \nabla f(\vec{w}_t), \tag{2.7}$$

where ϵ_t is the step size, mostly referred to as *learning rate*. This value is one of the most important hyperparameters in machine learning and determines how big a step in the direction of the calculated gradient will be. Relating to the bowl-like error surface, it is easy to conceive that very big steps are likely to overshoot the minimum, while too small steps require an inefficient total number of iterations.
The whole process of "leading" the error back layer by layer to optimise the individual weights is named *backpropagation* and was described by Rumelhart et al. in 1968 (Rumelhart et al., 1986a). In practice, the calculation of the gradient for a whole neural network makes use of the chain rule, which can be generally express as

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}, \tag{2.8}$$

to recursively calculate local parts of the gradient. The example from the previous paragraphs can be used again for a small illustration of this method. With inputs $\vec{x} = (x_1, x_2)^T$ and weights $\vec{w} = (w_1, w_2)$ as well as the bias b, the gradient on the function $f(\vec{w}, b, \vec{x}, y)$ is sought. The calculation of updates to the weights w_i and bias b for a single input $\vec{x}$ with the loss from equation 2.2, can therefore be denoted as

$$\frac{\partial}{\partial w_i} f(\vec{w}, b, \vec{x}, y) = \frac{\partial}{\partial w_i} \max\left(0, 1 - y(\vec{w}\vec{x} + b)\right) = \frac{\partial f(q)}{\partial q}\frac{\partial q(r)}{\partial r}\frac{\partial r(\vec{w})}{\partial w_i} \tag{2.9}$$

and

$$\frac{\partial}{\partial b} f(\vec{w}, b, \vec{x}, y) = \frac{\partial f(q)}{\partial q}\frac{\partial q(r)}{\partial b} \tag{2.10}$$

respectively, with

$$f(q) = \max(0, 1 - yq(r)) \rightarrow \frac{\partial f(q)}{\partial q} = \begin{cases} -y & \text{if } y(\vec{w}\vec{x} + b) < 1 \\ 0 & \text{if } y(\vec{w}\vec{x} + b) \geq 1 \end{cases}$$

$$q(r) = r(\vec{w}) + b \rightarrow \frac{\partial q(r)}{\partial r} = 1, \quad \frac{\partial q(r)}{\partial b} = 1$$

$$r(\vec{w}) = \vec{w}\vec{x} \rightarrow \left(\frac{\partial r(\vec{w})}{\partial w_1}, \frac{\partial r(\vec{w})}{\partial w_2}\right)^T = \vec{x}.$$

Which results in

$$\frac{\partial}{\partial w_i} f(\vec{w}, b, \vec{x}, y) = \begin{cases} -yx_i & \text{if } y(\vec{w}\vec{x} + b) < 1 \\ 0 & \text{if } y(\vec{w}\vec{x} + b) \geq 1 \end{cases} \tag{2.11}$$

and

$$\frac{\partial}{\partial b} f(\vec{w}, b, \vec{x}, y) = \begin{cases} -y & \text{if } y(\vec{w}\vec{x} + b) < 1 \\ 0 & \text{if } y(\vec{w}\vec{x} + b) \geq 1 \end{cases} \tag{2.12}$$

as possible cases. Let the first input be $\vec{x} = (0.5, -0.2)^T,\ y = 1$ (cf. figure 2.2) and initial parameters $\vec{w} = (0.5, 0.5)$ and $b = 0$, then the calculated updates become $\frac{\partial}{\partial w_1} = -0.5$, $\frac{\partial}{\partial w_2} = 0.2$ and $\frac{\partial}{\partial b} = -1$. With respect to equation 2.7 and a learning rate of $\epsilon = 0.1$, the new weights would be $\vec{w} = (0.55, 0.48)$ with a bias $b = 0.1$. In this case the overall loss would decrease from 1.05 to 1.02066 with this single step. This process would of course have to be repeated a number of times for all m input samples. A cycle of several updates is often referred to as an *epoch*. After five epochs, which for six inputs equals 30 updates, the total loss would be reduced from 1.05 to 0.3885 in the given example and would produce final weights of $\vec{w} = (1.41, -0.49)$ and a bias $b = -0.1$.

When learning on datasets which contain a separate training and validation partition, backpropagation is only carried out on training data and weights are obviously not updated on the validation data. Given a sufficient total number of learnable parameters, often referred to as *capacity*, the network will eventually start to overfit on the training data, learning the samples "by heart", after a large count of epochs. In extreme cases, this could be visualised as drawing an area around every single member of a class instead of one area for all members of the class. Normally, this leads to poor generalisation and subsequently not ideal performance on the validation data. Methods to prevent overfitting are presented in 2.1.3.

Convolutional Neural Networks

Convolutional neural networks (CNNs) are very closely related to regular neural networks. Their particular characteristic is the eponymous convolution. Although there are other applications, image recognition and classification tasks are the most popular. Details on the use of the convolution and other building blocks can be found in the following subsection. An exemplary illustration of a CNN can be seen in figure 2.4.

Their invention derives from early work on understanding the receptive field in the visual cortex of living organisms. In 1968 Hubel and Wiesel found that two

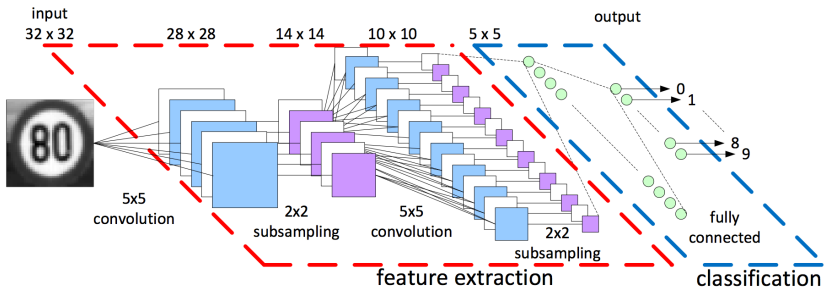


Fig. 2.4.: Example structure of a CNN classifying a traffic sign image patch (derived from Peemen et al., 2011).

different cell types in the brain extract diverse information from the visual input (Hubel and Wiesel, 1968). Fukushima's neural network model "Neocognitron" was the first to use this very idea for pattern recognition in two dimensional inputs, laying the foundation for modern CNNs (Fukushima, 1980). Most of today's CNN implementations are based on the now famous work of LeCun et al. on recognition of handwritten digits in 1998 (LeCun et al., 1998). Besides popularising the technique as it is used today, their work produced the MNIST (abbr. for Modified National Institute of Standards and Technology) database which has been and continues to be used in countless works in machine learning.

2.1.2 CNN Building Blocks

Both CNNs in particular and neural networks in general are usually represented through an acyclic graph of distinct layers. CNNs mainly consist of four basic building blocks; convolutions, non-linearities, pooling and fully connected layers.

Convolution

As the name suggests, convolution layers are the most distinct feature of CNNs. They aim to resemble the so-called local receptive field, inspired by the brain's visual cortex. Rather than connecting each input with every output, they connect one neuron to a small region of the input. In this way, they play the part of extracting abstract features from the input images and finding similarities within members

of a class. While the convolution of two signals f and g in the one-dimensional continuous case is generally defined as

$$(f * g)(x) := \int_{-\infty}^{\infty} f(\tau)g(x-\tau)\,\mathrm{d}\tau \tag{2.13}$$

it can be simplified for one-dimensional discrete functions to

$$(f * g)(n) := \sum_{k\in\mathbb{Z}} f(k)g(n-k). \tag{2.14}$$

For convolution over two axis given a two-dimensional image I and a two-dimensional kernel K, it can be written as the following (Goodfellow et al., 2016):

$$(I*K)(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n) = \sum_m \sum_n I(i-m,j-n)K(m,n). \tag{2.15}$$

For practical implementation, many machine learning libraries use the cross-correlation function

$$(I * K)(i,j) = y(i,j) = \sum_m \sum_n I(i+m,j+n)K(m,n) + b, \tag{2.16}$$

which applies a similar operation but without flipping the kernel and usually also adds an adaptable bias b as described in 2.1.1.

The values of the convolution kernel are the weights adapted during the training process. Nearly all implementations use the valid convolution, which places the kernel only on positions within the image, rather than the circular, resulting in smaller image sizes after processing. Padding methods like zero-padding can be used to retain the original input size. Besides the kernel size F and the amount of zero padding P, a convolutional layer requires two more hyperparameters; the number of filters N as well as a stride value S. The latter defines how many pixel a kernel is moved after each step of the convolution and is normally set to 1. The number of filters is also equal to the number of resulting images, often referred to as *feature maps*. With an input tensor of size $W_1 \times H_1 \times D_1$, the layer will produce an output of the size $W_2 \times H_2 \times D_2$, where

$$\begin{aligned} W_2 &= \frac{W_1 - F + 2P}{S} + 1 \\ H_2 &= \frac{H_1 - F + 2P}{S} + 1 \\ D_2 &= N. \end{aligned}$$

Non-linearity

A non-linearity is used as an activation function for the neurons within the network. Every input value is mapped to a certain output value following a fixed mathematical operation. In practice, there are three common functions, as pictured in figure 2.5. The *sigmoid* originates from the biological model and most closely resembles the firing rate of an actual neuron. With the mathematical expression

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.17}$$

it saturates values into a range between 0 and 1. The *tanh* non-linearity attains the same saturation effect, but within the range $[-1, 1]$ making the output zero-centred. The equation can also be expressed directly through the sigmoid function

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} = 2\sigma(2x) - 1. \tag{2.18}$$

The last and most popular function in recent years is the rectified linear unit, usually shortened to *ReLU*. First introduced by (Hahnloser et al., 2000) and popularised by (Jarrett et al., 2009a). The ReLU simply applies a threshold at 0 for all negative inputs, using the function

$$f_{ReLU}(x) = \max(0, x). \tag{2.19}$$

Input values above 0 remain unchanged without any kind of saturation. Variations of the ReLU, often called “leaky”, do not completely cut off inputs below 0 but rather diminish them to very small values.
Non-linearities are typically placed behind convolutional and fully connected layers, as otherwise neighbouring ones could be directly summarised as a linear combination.

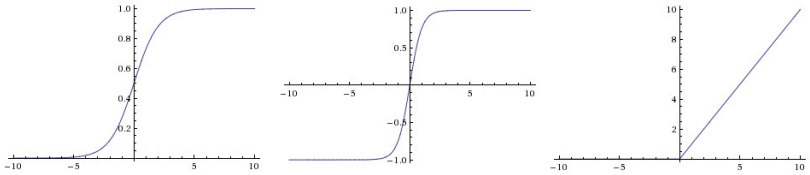


Fig. 2.5.: Non-linear functions: sigmoid *(left)*, tanh *(middle)* and ReLU *(right)*.

Pooling

Pooling is another type of layer very commonly used in CNNs. It is applied to make the network's representation invariant to small translations of the input and helps to better learn whether a feature is actually present, rather than its exact position. This is achieved by selecting a rectangular neighbourhood with the size of an integer factor k from the input and determine a single representation. It can thus also be understood as a sampling reduction of the pixel resolution. The best known pooling operations are to simply select the maximum value from the neighbourhood or calculate an average value (Zhou and Chellappa, 1988; Nagi et al., 2011). Like exemplified in figure 2.6, it is common practice to select non-overlapping pooling regions by choosing a filter stride $s = k$ or $s > k$ resulting in reduced image resolution by the factor s. Hence, this characteristic pooling is sometimes referred to as subsampling.

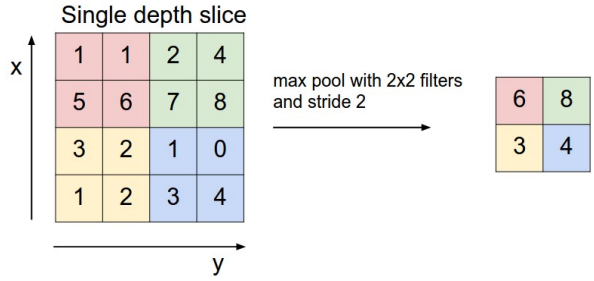


Fig. 2.6.: Example of the max pooling operation (Fei-Fei et al., 2017).

Fully Connected Layers

Fully connected (FC) layers owe their name to the fact that each single input that this type of layer receives is directly connected to every output. One or more fully connected layers are normally placed at the end of a CNN's architecture and often referred to as the classifier part of the network (cf. figure 2.4). The final FC layer is generally obliged to have the same number of outputs as the number of classes the networks wants to classify. The neurons in these layers are often referred to as "logits". Other fully connected layers, especially those between the first and last one, are sometimes called "hidden layers".

2.1.3 Training Techniques

As a field of ongoing research and constant development, the possibilities to design (convolutional) neural networks have been ceaselessly growing. New methods to optimise convergence, speed up the training process or reduce overfitting for example are presented every year. Since it is not very meaningful to enumerate all of them, the following small selection describe four popular techniques used in this work and likely the majority of others.

Batch Training

Especially in applications with large amounts of data, often containing millions of samples, it would be time consuming to compute and optimise the loss function over the entire dataset just to attain one single parameter update. Therefore, in the vast majority of those cases, an estimation of the gradient is calculated by dividing data into batches. Typically, this size ranges from one to a few hundred samples per batch. The actual size can be considered a separate hyperparameter open for optimisation, since it is correlated with the overall performance. Very small batch sizes promote overfitting, as each parameter update is based on a small amount of samples. *Overfitting* describes the fact, that a learning algorithm learns the training samples by heart, rather than the underlying features. This leads to a stagnating validation error, while the training error is minimised. In contrast, large batches needlessly slow down the training process, commonly resulting in a non-optimal performance in a fixed amount of epochs. Gradient-based optimisation using batches is often called “batch gradient descent” or “stochastic gradient descent”.

Momentum

Momentum is a very popular extension to the backpropagation algorithm described in section 2.1.1. It aims to reduce the tendency of the gradient descent to get stuck in local minima of the objective error surface. Originally specified by (Polyak, 1964), this method simply adds a fraction $\mu \in [0, 1)$ of the previous weight update to the new one. In practice, typical values for μ are 0.5 or 0.9. The momentum update (Sutskever et al., 2013) can be formulated like this:

$$\begin{aligned} \vec{v}_{t+1} &= \mu\vec{v_t} - \epsilon_t \nabla f(\vec{w_t}) \\ \vec{w}_{t+1} &= \vec{w_t} + \vec{v}_{t+1}. \end{aligned} \tag{2.20}$$

This optimisation helps to smooth out variations of the gradient changing direction and achieve a more steady and consistent descent towards the (global) minimum. Furthermore, it hereby helps to speed up convergence. Just like in the mechanic example, were momentum is mass times velocity $\vec{p} = m \cdot \vec{v}$, the weights can be regarded as a particle moved by the force of the negative gradient. When using a lot of momentum (μ close to 1), it is generally necessary to reduce the global learning rate, because otherwise it is likely to rush past the desired minimum not achieving convergence. The actual implementation of this technique may be different, depending on the program library in use.

Dropout

Dropout is a rather plain but very effective method presented by Hinton et al. in 2012 (Hinton et al., 2012; Srivastava et al., 2014). During the training process, this technique randomly drops units in the layer it is applied to with a probability p, setting them to zero. In this way it can be regarded as sampling one of 2^n thinned networks out of the original n-unit network each iteration. An example is pictured in figure 2.7. At test time however, all units remain activated with the outgoing weights multiplied by p ensuring the output of any unit to be the same as the expected output under the distribution used to drop units.

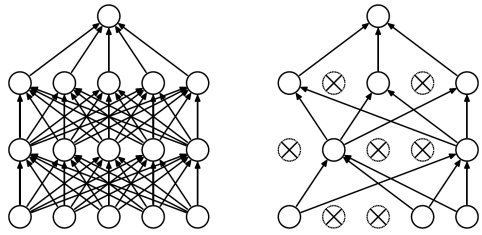


Fig. 2.7.: A neural network before *(left)* and after *(right)* applying dropout (Srivastava et al., 2014).

Popular ML toolboxes like *Tensorflow* or *Caffe* perform this compensation during training, for example by scaling up inputs and gradients of non-zero units by $\frac{1}{1-p}$. Dropout has been reported to demonstrably and significantly reduce overfitting through preventing the network from learning too complex co-adaptions (Warde-Farley et al., 2014). Figuratively speaking, deterring it from learning the training samples by heart.

Batch Normalisation

With the goal to *"accelerate deep network training by reducing internal covariate shift"*, Ioffe and Szegedy presented a technique referred to as batch normalisation (Ioffe and Szegedy, 2015). With internal covariate shift the authors refer to the problem of constantly changing inputs when updating weights in a deep network. This method forces the activations throughout the neural network to take on a stable distribution, trying to reduce the dependency on the weight initialisation and low learning rates to assure convergence. The details of this technique are not expanded here, but can be found in the original paper. Batch normalisation layers are commonly inserted after convolutions or fully connected layers and before non-linearities.

2.2 Sensors

Camera

Cameras are probably the most common type of sensor used in the automotive industry. They can monitor the interior and especially the surroundings of the vehicle. Here, they are employed for tasks like object perception, lane marking recognition or traffic sign detection among others.

In general, camera sensors work with light in the visible spectral range, although systems in the infrared range are in use as well. While cameras for photography or videography usually have a pixel grid of photo diodes separated by sensitivity for the elementary colours red, green and blue (cf. (Bayer, 1976)), those used in the automotive sector often only deliver a grey-scale image for reasons of cost, see figure 2.8. A middle ground may be found in red-pixel cameras which use a red filter for one of every four sub-pixels. This can be useful if limited colour information is of use for object classification (of e.g. traffic signs and tail lights).

For further information on camera calibration, both intrinsically and extrinsically, including details about aspects like focal length, lens distortions, distances between sensor and lens and others, the reader is kindly referred to the comprehensive work of (Zhang, 2000).

Fig. 2.8.: A grey-scale camera image recorded with a module placed on top of a car.

Lidar

Lidar, acronym for *"light detection and ranging"*, is the method of measuring range with beams of light. A lidar sensor targets objects with a laser, usually at a wavelength of the light spectrum invisible to the human eye, and measures the distance d using the time t of flight as

$$d = \frac{c_0 \cdot t}{2} \qquad \text{with} \quad c_0 \approx 3 \cdot 10^8 \frac{m}{s}. \tag{2.21}$$

Beyond the pure range information extracted from this relationship, most sensors of this type also provide measurements of the intensity of the reflected beam. Due to the shape and nature of the object which reflects the light pulse, the backscattered rays lose a fraction of the luminous flux to absorption. The intensity of the returned rays can be used to draw conclusions about the absorption capacity of the object and therefore its surface material.

While this type of sensor finds applications in many fields including geodesy and astronomy, two main designs are used in an automotive context. In *solid state* lidars, each light beam is emitted from a fixed position, while in *rotating* versions a line array of lasers is rotated to scan the surroundings at an angle of up to 360°. For this, each emitter-receiver pair fires multiple times per revolution to create a circular measuring line in a given angular position. An image representation of such a scan frame can be seen in figure 2.9. For applications in the automotive industry, lidar provides invaluable three-dimensional information of a vehicle's surroundings for object detection, self-localisation and range estimation. This can be used to serve functions like brake assistance, adaptive cruise control or path planning for autonomous driving.

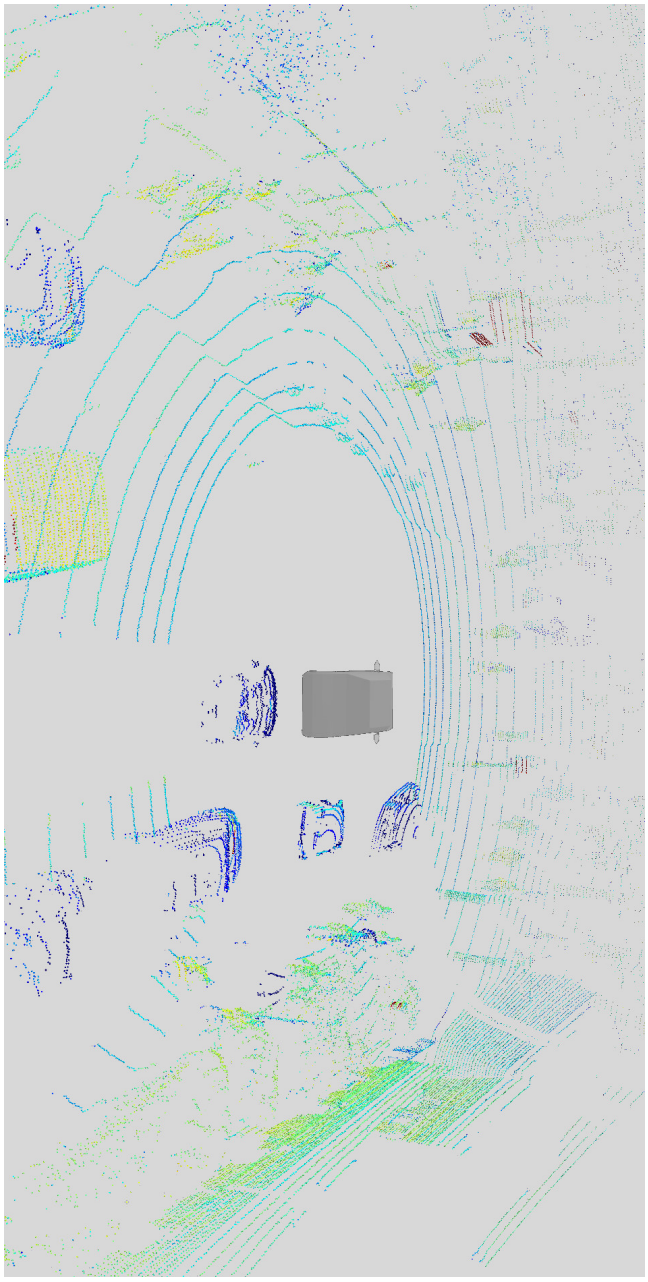


Fig. 2.9.: A 360° lidar scan captured from the top of a vehicle.

3 A New Study on the Robustness of Active Learning

This chapter discusses the field of active learning (AL) and its practical applicability. While section 3.1 gives a short introduction to the concept and pool-based AL in particular, section 3.2 presents different data query strategies and possible implementations derived from that. A new strategy of this kind is subsequently introduced in section 3.2.1. The following passage cover aspects of the robustness of active learning. Focus is brought to the influence of hyperparameter adjustments in section 3.3.1. Section 3.3.2 expands these considerations to the transferability of query selections between different network architectures. A conclusion on the findings is given in section 3.4 in conjunction with practical application to a task using a hierarchical classification network.

3.1 Pool-Based Active Learning

Data is the foundation for any form of machine learning. Over the past two to three decades, a multitude of data scientists have been proposing an even larger number of methods to optimise the manner in which a model can learn to solve a task from a given collection of data. Efforts have been made to find approaches that balance class distributions in data sets or compensate for such imbalance in the loss function. To determine the optimal validation set to achieve the highest generalisation or improve it through countless regularisation techniques, just to mention a few.

The question that lies at the core of the field of active learning is not much different from this after all; how to identify the most useful samples for a Machine Learning algorithm to be trained with? Compared to other concepts, however, it relies on *active* response from the model as an input to create a selection.
Different scenarios to implement this intention have been researched, but especially from the standpoint of practical application, the so-called *pool-based* approach seems to be the one most widely studied (Settles, 2009). Although this makes

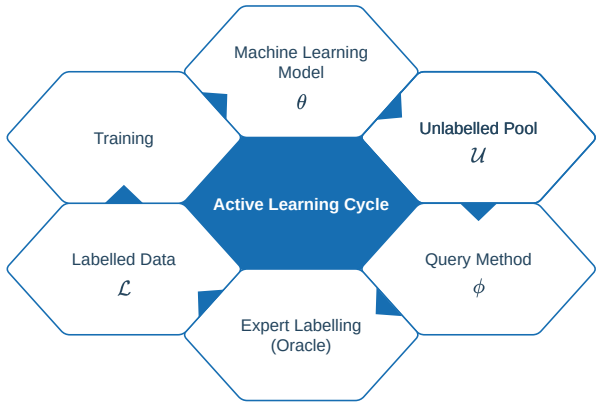


Fig. 3.1.: The pool-based active learning cycle.

for an important motivation, it is not the only reason it is chosen for all further considerations in this chapter.

Pool-based active learning follows a cycle, as depicted in figure 3.1. Given a set of data $\mathcal{D}$ consisting of a number of feature samples $x \in X$, an initial labelled set $\mathcal{L}$ is defined (heuristically or at random) for training. For every data point in this subset, a paired label $y \in Y$ is annotated. A machine learning algorithm θ is then trained on $\mathcal{L}$. Every remaining unlabelled sample is then inferred with the trained model. Using a query metric ϕ, a number of new data points are selected to be labelled. This annotation process requires input from an *oracle*, normally a human, assigning the correct class. All new feature/label pairs $\langle x, y \rangle$ are then added to the training set and the cycle starts again. A pseudo code representation of a pool-based active learning algorithm is noted in algorithm 1 below.

The other two main AL scenarios should be mentioned here, to underline their difference to this concept: *stream-based* sampling (Lewis and Gale, 1994) looks at single instances one at a time and is suitable in situations where data points can be generated on demand. Here, query functions are designed to evaluate a one item and decide whether to directly assign a label or ask the oracle for annotation. A third scenario called *Membership Query Synthesis* (Angluin, 1988) is primarily interesting for circumstances where the amount of data available is very small. The query algorithm generates new instances from the underlying distribution, e.g. small excerpts or cropped sections of other samples, which can lead to results comparable with methods for data augmentation.

This form of AL is particularly interesting for application to machine learning algorithm development in the automotive and other industries due to two major reasons:
First, it can most importantly be used to decide on a subset of collected data to be annotated in order to create training and validation set for a supervised machine learning task. While it can be comparatively easy and inexpensive to record and gather vehicle sensor data and ever decreasing cost makes it affordable to possibly neglect storage expenses, reliable ground truth annotation still requires manual labour and is therefore the crucial factor. In industrial application of machine learning to various tasks, budget and time constraints play a significant role and performance can depend on choosing the best n samples to train on.

Secondly, one could think of using active learning methods as a form of regularisation. While increasing the number of available training samples is in general regarded as helpful, certain factors can lead to an impaired performance when doing so. The more objects in a recognition task are standardised, the more redundant information is potentially added to the dataset with each new sample, which can result in worsened generalisation. Active learning methods can also be applied to sanitise a dataset from falsely labelled samples, as a suitable strategy will not pick data points with a conspicuous difference between label and prediction.

When applied correctly, active learning can be a very powerful tool to counteract the immense data requirements of (deep) artificial neural networks. However, the following sections of this study should analyse how robust it is to changes in certain hyperparameters or mislabelled data. This can become especially important if it was to be applied without profound domain experience, as some publications claim their methods to be universally employable, seemingly even without prior knowledge. It then may be vulnerable to the risk of a sub-optimal sample selection, which, in the worst case scenario, could render the entire ML task unsuccessful.

Algorithm 1: Pool-Based Active Learning.

Input : $\mathcal{L} \subset \mathcal{D} = \{\langle x, y\rangle\}$: Labelled set,
$\mathcal{U} = \mathcal{D} \setminus \mathcal{L} = \{\langle x, ?\rangle\}$: Unlabelled set,
θ : Classification model,
ϕ: Active learning query function

while $|\mathcal{U}| > 0 \wedge$ *no stopping criterion* **do**
  $\theta = train(\mathcal{L})$;
  for ***all*** $\langle x, ?\rangle \in \mathcal{U}$ **do**
    compute active learning metric $\phi(x)$ under θ;
  end
  Choose $x^\star$ with highest magnitude of $\phi(x^\star)$;
  Annotate $y^\star$;
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{\langle x^\star, y^\star\rangle\}$;
  $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\langle x^\star, y^\star\rangle\}$;
end

3.2 Active Learning Query Strategies

While the previous section introduced the basic principle behind the term active learning and its process, query strategies were only regarded as a black box providing the desired output, i.e. a selection of data samples to be labelled.
In contrast to other methods, active learning query strategies make use of information from the response of a machine learning model θ to samples from an unlabelled set $\mathcal{U}$. Of course, well known statistical methods like stochastic neighbour embedding (SNE) (Hinton and Roweis, 2003; van der Maaten and Hinton, 2008) or even principal component analysis (PCA) (Pearson, 1901) can be used to analyse high-dimensional data and one could make selections based on outliers and/or ambiguous samples in the respective representation. But having the option to use "feedback" from the very algorithm whose performance is to be improved, offers a considerable advantage in terms of usable information. In the following paragraphs, various concrete formulations of AL query strategies are derived on the basis of known metrics.

One of the most valuable pieces of information that can be obtained from a machine learning model with regard to a given input, is the (un)certainty of the corresponding output. The group of methods to be applied here is therefore called uncertainty sampling (Lewis and Gale, 1994). Prediction from a classification algorithm θ is

given by $\hat{y} = \arg\max_y P_\theta(y|x)$, reflecting the class label with the highest posterior probability. Accordingly, it could be a straightforward approach to select the sample

$$x^\star_{LC} = \arg\max_x \left(1 - P_\theta(\hat{y}|x)\right) \tag{3.1}$$

that the classifier is *least confident* about. Several strategies can be derived from this measure and in the further course of this work, this family of strategies basing the decision mainly on the one neuron with the highest activation is referred to as *naive certainty (NC)*. With no further extension to this concept, the strategy **NC Low** is formed. Here, n samples with the minimal maximal activation in the neurons of the final layer of the classifier are selected. These neurons are often referred to as "logits"[1]. The same could be done for a certain range in the logits' activation (e.g. $[0.2, 0.7]$), **NC Range**, offering to define a lower and upper bound[2] to exclude very hard/easy samples.

The third strategy of this kind, **NC Balanced**, further aims to balance the class distribution of the n samples to be selected. A confusion matrix, where each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class, therefore showing true predictions on the main diagonal and class-wise errors in the off-diagonal elements, is calculated for the previous training set. The class distribution of those wrongly predicted samples is taken into account to determine the class-wise share of new sample to be added. This strategy terminates if one class contains no more samples to be drawn.

All methods described so far in this section consider the final layer of a neural network to make decisions about data points to annotate. Although they are not as easy for humans to interpret as the normalised probability distribution of the output, previous fully connected layers hold more information. They are often referred to as *embedding* layers, since they contain a low-dimensional continuous vector representation, an embedding, of the discrete categorical input.
Another extension to the pure *least confident* calculation is to not only select n samples with minimal maximal activation, but additionally compare them with those already in the training set to prevent too similar samples from being chosen. In this **NC Diversity**, a similarity measure on basis of the embedding vectors is added to be calculated. Here, the L_1 distance, sometimes referred to as *Manhattan distance*

[1] Not to be confused with the logit function, the inverse of the standard logistic function, used in the field of statistics.
[2] N.B.: Here, the smallest sensible value is directly dependent on the number of classes c, since no normalised single class prediction value can be smaller than $1/c$, as this would already represent a probability equipartition.

or *taxicab geometry*, is used to measure the distance $d = \sum_i |x_i^\star - x_i|$ between the vector representation of every candidate sample $x^\star \in \mathcal{U}$ in the unlabelled pool and the training set $x \in \mathcal{L}$. The larger d becomes, the more different a possible new sample is from the data points already present in the labelled set, at least from the perspective of the network's internal representation.

Instead of only looking at the least confident predictions, another way to determine data points for inclusion into the training set, is to identify candidates which produce the smallest difference between the two highest activations in logits. This **Margin** can be expressed as

$$x_M^\star = \arg\min_x \left(P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x)\right) \tag{3.2}$$

and should focus data pool queries on instances that, figuratively speaking, leave the classification algorithm undecided between two nearly equally likely options. In doing so, the decision boundary could be refined through these crucial points.

As a wide range of classification tasks are multi-class problems, considering all given prediction confidences could be beneficial to make valid sample selections. The Shannon entropy (Shannon, 1948) represents the average level of information inherent in the possible outcomes of a random variable. It can therefore be used to measure how peaked or equipartitioned such a distribution is. The more evenly distributed the activation of the logits, the greater the entropy. To identify a valuable sample to be labelled, a query strategy could make a selection through the highest **Entropy** value;

$$x_H^\star = \arg\max_x \left(-\sum_i P_\theta(y_i|x) \log P_\theta(y_i|x)\right). \tag{3.3}$$

A new variant of this consideration factoring in the output distribution is presented in the following section 3.2.1.

In their work (Sener and Savarese, 2018) proposed an approach for active learning with CNNs, where they create a core set by approximating the problem of distributing k centres in m points, such that the minimal distance of all points to the nearest centre is maximised. This can be applied as follows, using similarity measures in the embedding space to constitute a method named **Core Set Greedy** for this work: Through inference of the trained classification model θ with both the labelled training set $\mathcal{L}$ and the unlabelled pool $\mathcal{D}$, embedding representations are obtained. Then a matrix is calculate denoting the closest sample in $\mathcal{L}$ to any sample in $\mathcal{D}$. To this purpose, the closest distance is defined as the minimum of the sum of least

square difference of the embedding vectors. A greedy algorithm is lastly used to select the sample to be added to $\mathcal{L}$, by finding the point for which the minimum distance to all $x \in \mathcal{L}$ is maximised. Eventually, the minimum distance matrix is updated to factor in the new training sample $x^{\star}_{CSG}$.

3.2.1 A New Method for Pool-Based Active Learning

In Active Learning, we require a measure of how sure the classifier is that its class decision during inference is accurate. One possibility for such an accuracy-of-inference measure is to analyse the distribution of the network's logits. Within the trained model of the classifier, the logits can be interpreted as probabilities that the inferred sample belongs to the class associated with the respective logit. If the logits are strongly biased in favour of a certain class, it is very likely that the given sample belongs to the class corresponding to the strongest logit. On the contrary, if the logits do not show a clear preference for a certain class, there is a high risk that taking the class of the strongest logit results in a false prediction. In other words, to which degree the distribution of logits tends towards peaks rather than an equipartition indicates how accurate the inference is going to be.

In previous literature, the Shannon entropy (Shannon, 1948) has been frequently used as a measure of how peaked or equipartitioned a distribution is. A valid strategy for active learning could then be to query those samples for annotation, for which the Shannon entropy $H = -\sum_i l_i \log(l_i)$, with l_i being the values of the logits, is particularly high. However, a shortcoming of this approach is that it does not adequately account for the situation when the distribution of logits is admittedly strongly peaked, but with peaks on more than one class logit. Such a situation can easily arise in samples, when they belong to classes showing similarities and the classifier's model does not yet feature a clear decision boundary between them. In such a case, the distribution of logits is still far away from an equipartition, resulting in a relatively low value for the Shannon entropy H. Thus, although labelling these samples would be particularly valuable for fleshing out the decision boundary and allowing the classifier to better separate between classes, they would not be added to the training set.

To overcome these shortcomings of the Shannon entropy H as a measure for characterising the distribution of logits l_i, it might be more beneficial to use the Simpson diversity index $D = 1 - \sum_i (l_i)^2$ (Simpson, 1949) instead. The closer the distribution l_i is to an equipartition, the larger D becomes. If the l_i shows a strong peak at a certain i, D is close to zero. Finally, if the l_i are strongly peaked among

several classes, D will have a small-to-moderate value between zero and one[3]. The latter property of D in particular allows to select those samples for labelling, for which the classifier can narrow the class decision down to a few classes, among which it is still unsure. The query strategy, referred to as **Sum of Squared Logits (SOSL)** in the further course of this work, is then to select in each iteration the n samples with the highest D. To selection of a sample could therefore be noted as

$$x^{\star}_{SOSL} = \arg\max_{x} \left(1 - \sum_{i} P_{\theta}(y_i|x)^2 \right). \tag{3.4}$$

3.3 Robustness Evaluation

Before moving on to considerations regarding robustness, the performance of the presented query strategies for pool-based active learning is evaluated on different data sets for image classification. These consist of the well-known handwritten digit classification set *MNIST* (Lecun et al., 1998) and the thereof inspired data sets of handwritten samples of the Latin alphabet *CoMNIST* (Vial, 2017) and clothing item classification *Fashion-MNIST* (Xiao et al., 2017). Furthermore, sets for general object classification *CIFAR-10* (Krizhevsky et al., 2009) and the house number collection *SVHN* (Netzer et al., 2011). Strategies are also tested on a private data set of 33 different classes of traffic signs (*TSR*) represented through small grey scale images.

For every data set, a distinct but similar plain feed-forward convolutional neural network is used as classification algorithm. As this experiment is not designed to set new benchmarks in terms of performance or find the optimal architecture, but to identify the most promising data samples, the number of layers and feature map channels are chosen according to the approximate complexity of the task. The popular Adam algorithm (Kingma and Ba, 2015) is employed as optimisation method. An exception is made for *CIFAR-10*, where due to its higher complexity an implementation of the deep CNN *ResNet50* (He et al., 2016) is used.

As table 3.1 shows, all data sets are initially split into a training and validation set. To obviate any overfitting-like behaviour in training iterations with an early stopping criterion and any other bias, 10% of the original training set is split into an additional

[3]Assuming a scenario with 20 logits/classes for example, an equal distribution of the l_i will result in $D = 0.95$, two peaks will be $D \sim 0.662$ and one clear single peak could be $D \sim 0.342$. Entropy values are of course not directly comparable, as the value of H increases with the number of l_i. But if normalised to the same scale as D, the values for the same scenario would be $H \sim \{0.950, 0.494, 0.332\}$. With an increasing number of l_i, the relative value of H for two peaks will be even closer to that for one peak.

development set. Used in training as validation after every epoch and balanced in class distribution. The original validation set is regarded as the test set, only to be applied once a model is completely trained to determine a final performance score with the best weights acquired during training according to the development set accuracy.

For all of these experiments, the initial AL training set $\mathcal{L}$ is composed from 100 randomly selected samples per class of the particular dataset. The CNN is trained for up to 1000 epochs with an early stopping of 200. If the validation accuracy on the development set does not improve for 200 consecutive epochs, training is ended. Once a model is trained, it is used to then select new samples to be added to the training set utilising one of the eight query strategies introduced in 3.2. With each iteration, the oracle is asked to annotate a number of samples so that the size of the training set is increased by 20%. As all the original labels are known for these datasets, the expert labelling step does not require actual human interaction for these experiments but can be accomplished by the computer automatically.

For statistical significance, every combination of the data sets and query strategies is tested completely five times. This should allow to suppress any influence of weight initialisations or positive or negative bias of a particularly good or underperforming initial training set. To reduce the computational burden of this large-scale experiment, the AL cycle only queries new samples to be labelled until approximately a third of the full size of the respective original training set is reached. If no promising performance result is achieved by this point, the active learning approach could be considered to have failed anyway. To be able to put the benefit of AL into perspective, all query strategies are compared to the plain strategy of selecting samples to be added purely at random. Furthermore, a baseline is created for every data set by training on the full original training set.

Figure 3.3 illustrates the results of the evaluation of all query strategies on four of the data sets. On three of those, a benefit provided through the AL methods is clearly visible. All or at least some of the query strategies produced a training selection which can result in a performance of the ML model that is comparable to the baseline already around the 30% mark. This is an important advantage and shows that a significant portion of the original data contains redundant information. The fact that most of the strategies also sample data points in a way more advantageous than random selection is also confirmed. With data from the *Fashion-MNIST* base this difference is very clear, although the embeddings-based *Core Set* approach generates a visibly underperforming choice. For *TSR* and *MNIST* findings are comparable. Whereas the former shows larger fluctuations in accuracy, possibly due to the

	CIFAR-10	CoMNIST	Fashion-MNIST	MNIST	SVHN	TSR
Classes	10	26	10	10	10	33
Image Size	32×32	32×32	28×28	28×28	32×32	34×34
Channels	3	1	1	1	3	1
Training Samples	50 000	9 918	60 000	60 000	73 257	265 774
Validation Samples	10 000	1 300	10 000	10 000	26 032	66 443

Tab. 3.1.: Characteristics of the datasets used for the active learning experiments.

basic population being greater than in the other sets, the latter also seems to be understandable for the *Core Set* strategy. The other two datasets, *CoMNIST* and *SVHN*, did not produce significantly different results.
For *CIFAR-10* however, this is not true. None of the methods show any profit for this dataset and are in line with the random sample selection, resulting in a nearly perfectly linear increase in accuracy. This does not come as a surprise, as *CIFAR-10* has very diverse representations of its classes and seems to contain no redundant information. As the baseline is also clearly not reached at the experiment's endpoint, no performance gain is provided through AL overall in this case, making it a first clear indication that AL should be used with knowledge about the context of the application.

3.3.1 Hyperparameter Robustness

After this initial evaluation, a dedicated consideration of the robustness of AL query strategies to external influence is sensible to investigate their applicability in the real world. This becomes particularly relevant when these methods are considered for application in industry projects.

First, hyperparameters for the neural network training can be taken into account, as they are generally important for fine-tuning the performance. In theory these should be decoupled from the AL cycle and not change performance of a sample query, as long as the classification algorithm itself is not prevented from converging. To confirm this notion, alterations of the learning rate and training batch size were tested. The test was carried out on the *MNIST* set, since it represents a stable starting point for all strategies. Adaptions of the learning rate span two orders of magnitude from 10^{-3} to 10^{-5} with five steps in between and the batch size is altered in powers of two from 2^5 to 2^9.
All methods behave very robustly under these influences as figure 3.4 illustrates. No negative effects on performance are shown and all strategies share a common result quality. While not directly related to this experiment's settings, the side-by-side

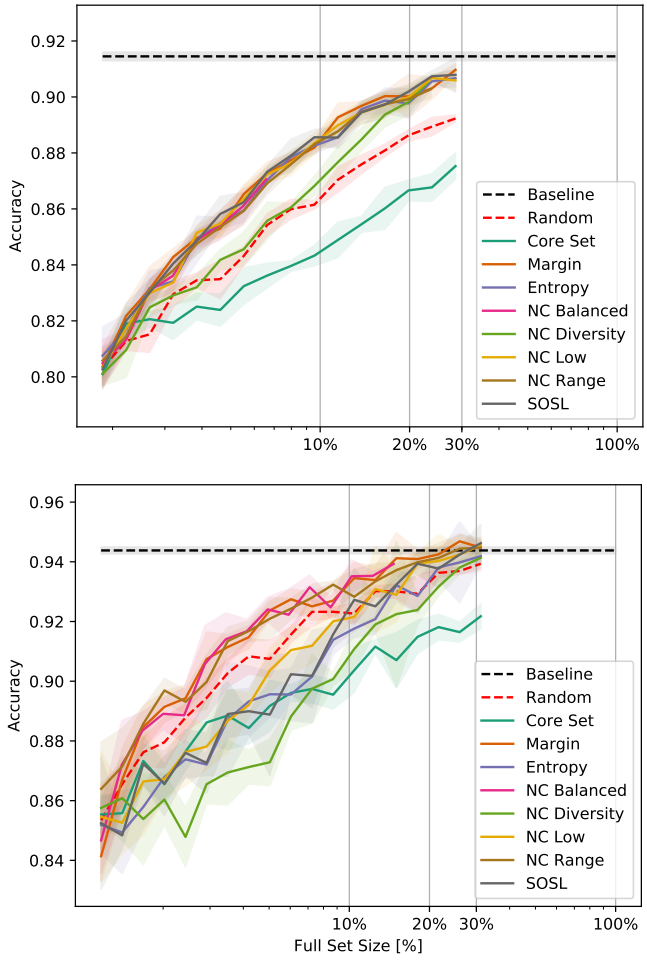


Fig. 3.2.: Classification accuracy over training set size for all strategies on Fashion-MNIST *(top)* and TSR *(bottom)*. The plotted value is the median of five runs and the shaded area denotes one standard deviation. Please note the logarithmic scale of the x axis.

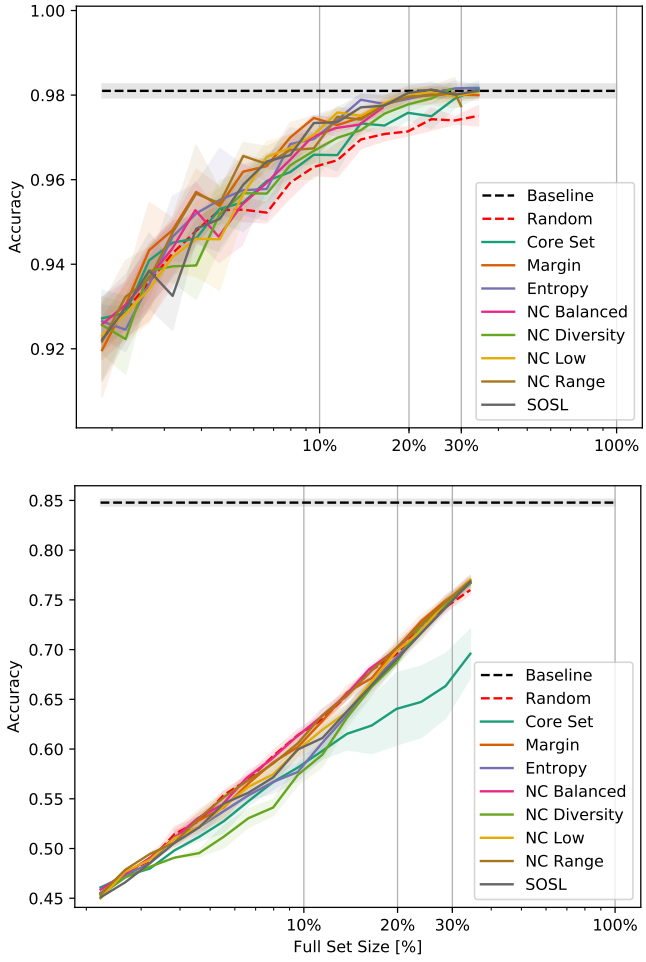


Fig. 3.3.: Classification accuracy over training set size for all strategies on MNIST *(top)* and CIFAR10 *(bottom)*. The plotted value is the median of five runs and the shaded area denotes one standard deviation. Please note the logarithmic scale of the x axis.

comparison displays clearly how *NC Balanced* reaches its abort criterion and stops the iterative cycle prior to the other approaches. This has no negative impact, on the contrary it shows a saving of computing time while the classification performance remains on par with the other strategies.

In a second experiment the *dropout* method (cf. sec. 2.1.3) is tested representative for regularisation approaches. Since this technique stochastically eliminates weights in (fully connected) layers during training of the network to reduce overfitting, an impact on the performance of AL query strategies using those to make decisions is to be expected. All strategies were applied to a network being trained with *dropout* rates of 0.3, 0.5 and 0.7. Figure 3.5b shows the outcome of this experiment.
For a factor of 0.3, neither a difference between the methods nor an influence on the overall classification performance can be identified. If this is increased to 0.5, performance is slightly impaired in all cases but more noticeably for *NC Diversity*. Here, the decrease in validation accuracy becomes clearly visible when plotted. The other strategies only reach this level of decline by the time dropout is set to 0.7. Then again, *NC Diversity* and also the *Core Set* strategy now exhibit a very severe performance loss. This underlines a clear disadvantage for these strategies based on information from embedding layers when dropout is applied, as it understandably influences their decision-making. These results conclusively do not come as a real surprise to someone with domain knowledge and experience. However, even this small example makes it clear that the applicability of AL methods is not universally valid and is influenced by the use case and training parameters, not only the nature of the data.

With regard to the data itself, these can of course also have different influences on the results of the AL task. Achieving high quality in data annotations is an important cornerstone of solving any machine learning problem. With an increasing size and complexity, any data set will sooner or later contain a number of incorrectly labelled samples. These will not only undoubtedly impair the classification accuracy, but can additionally confuse AL query strategies.

Accordingly, the next experiment introduced synthetic labelling errors by randomly changing the label of an queried sample, after the oracle annotated it. The results for error rates of 0%, 1%, 2%, 5% and 10% are given in figure 3.5a. While incorrectly labelled data in the training set in itself generally will impair training performance, there are also again differences between the strategies visible. Several of them, such as *NC Low* or the proposed *SOSL* for example, exhibit good results and also behave robustly in relation to the erroneous data. Yet again, the methods based on embedding layer information, especially *NC Diversity*, clearly lose out by comparison.

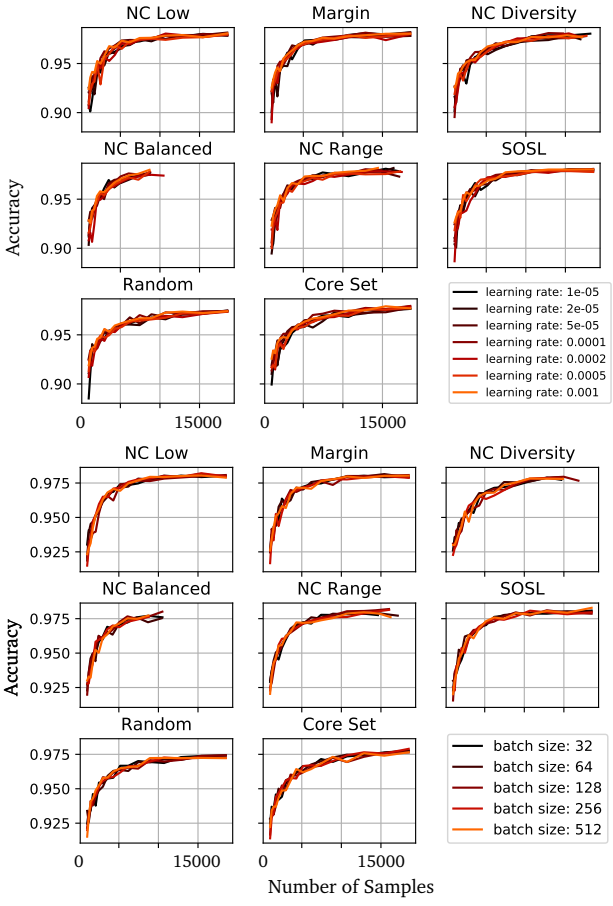
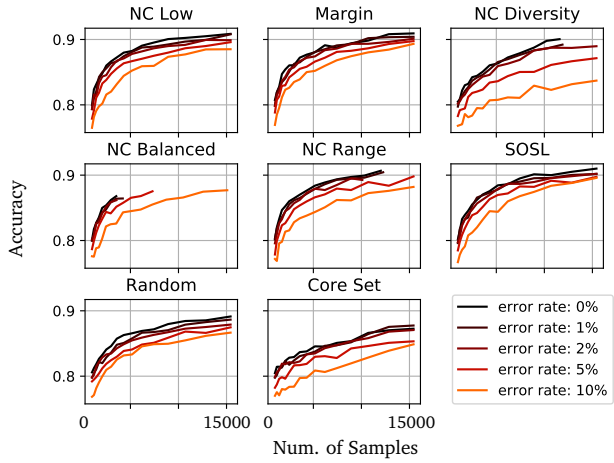
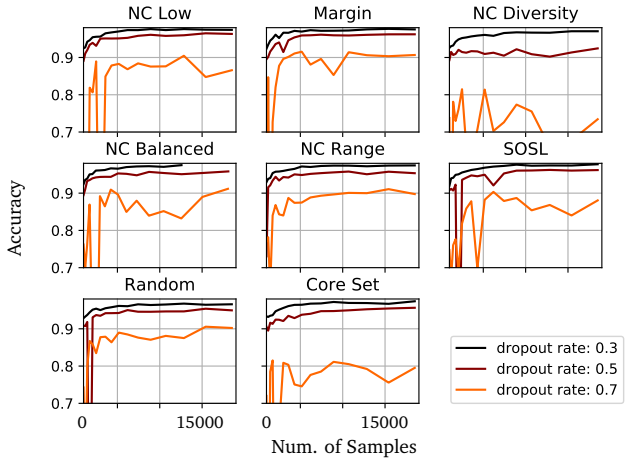


Fig. 3.4.: Validation accuracy influence of changes in learning rate *(top)* and batch size *(bottom)* for various active learning strategies on MNIST. The plotted value is the median of five runs per method.



(a) Synthetic annotation errors.



(b) Dropout.

Fig. 3.5.: Active learning strategy validation accuracy results for various synthetic labelling error rates on Fashion-MNIST *(top)* and different dropout regularisation rates on MNIST *(bottom)*. The plotted value is the median of five runs per method.

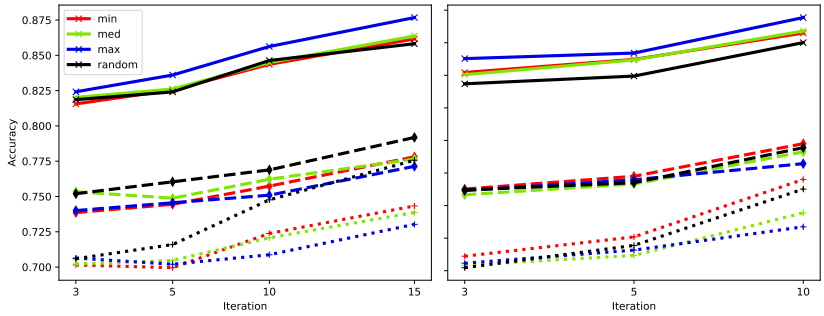
Relying on this diversity criterion negatively impacts the outcome, since the selection process should prevent similar samples from being chosen. Therefore, it may be more difficult to correct the negative effects that selecting a mislabelled sample would have and falsely annotated data points make for an even more attractive query, as they will likely be prominent outliers in the feature space.

3.3.2 Network Architecture Influence and Generalisation

The previous section underlined how different query strategies for pool-based active learning can be influenced differently by various external factors. In the application of machine learning, especially in a product context, successive refinement of the algorithm is very common. A CNN architecture might be adjusted several times over the course of development or a production process, to optimise the performance or to adapt to changes in the dataset or external restrictions like computational resources. Therefore, it is important to investigate to what extent the selection of samples depends on a specific network architecture and how the usability of AL might be influenced, if data selection is done by a different network than the one eventually used.

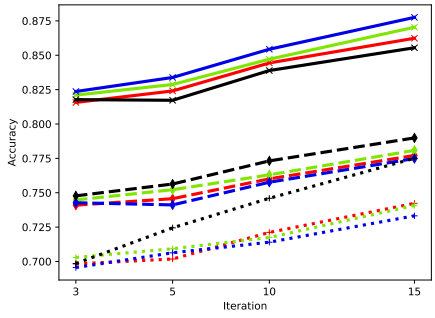
For this purpose, three CNNs with different numbers of parameters are implemented ceteris paribus. To reflect their varied capacity, they are in the following simply referred to as *Min*, *Med* and *Max*. Samples are iteratively selected from the Fashion-MNIST data set with the query strategies as described before. To ensure comparability of the results, the same initial dataset of 100 samples per class is used for all classifiers. Cross-training is then performed, where every network is trained with the selections of the others but of course also its own. A random sample selection is also compared. To keep the computational effort within acceptable limits, this is only done after 3, 5, 10 and 15 iterations of the AL cycle. For statistical certainty each training is repeated five times.

Figure 3.6 depicts the result of three selected strategies for this experiment. Apart from information about the replaceability of classifiers, these results can show how the classifier capacity itself influences the applicability of active learning strategies. For the example of *NC Balanced* it can be noted that there is a bias for the own selection performing best with the *Max* and *Min* classifier, while the medium-sized one shows indifference. The “weaker” the network gets, the better the performance of the random selection becomes. For the *SOSL*, this becomes even more clear. While the selection of the *Max* classifier is still definitely the best for itself, the smaller networks show the best performance with the randomised set. The results with the



(a) Entropy

(b) NC Balanced



(c) Sum of squared logits

Fig. 3.6.: Results (arith. mean of five runs) of cross training of different classifiers, Max *(solid line)*, Med *(dashed)*, Min *(dotted)*, with the sample selection of the other networks and its own *(blue, green, red)*, compared against a random selection *(black)*, for three strategies. Evaluations are made after 3, 5, 10 and 15 Iterations of querying new samples, except for NC Balanced which already terminated before the 15th iteration.

Entropy strategy are very similar, but the gaps become even more obvious. *Max* now shows a very clear preference for the own selection compared to any other and the performance of the active learning strategy selection on the *Min* network is now more than three percentage points behind random.

The tendency is that, although preferences are usually visible from the beginning, they become stronger over the course of the AL cycle. This amplification would mean that in a continuous application of AL, e.g. in a production programme, a later change to the classifier could result in significant performance losses. At this point, even the selection of an otherwise generally well-performing query strategy might be so biased that a random choice would have been the better method to build a training data set.

3.4 Conclusions on the Practical Application

Before drawing a conclusion from the experiments in the previous sections, it is reasonable to look at a slightly different machine learning task and compare differences and similarities in the findings. Therefore, a neural network structure different from the straightforward CNNs in the preceding sections is examined.

3.4.1 Application to Hierarchical Data

In contrast to rudimentary classification networks, hierarchical or cascaded classifiers do not use a single label per sample but a whole label tree (Weyers et al., 2018). Consequently, labels are represented as vectors and consist of one of the three following options per class: "1, 0 or not applicable" and each sample belongs to exactly one class per hierarchy level.

In this experiment, a private dataset for driver hand gesture recognition is used. It consists of 12 classes which depict different poses made by a human hand that could be used to as input for an infotainment system (e.g. "One finger", "Two fingers", "Fist Thumb Up", etc.). As depicted in figure 3.7, these are structured into three levels of hierarchy: 1.) "Hand / No Hand", 2.) a "Hand" class and 3.) a subclass, if applicable. A sample containing a thumbs up gesture would have the labels "Hand" and "Fist + Thumb" and "Fist + Thumb Up" for example, while an instance of "No Hand" would have the other levels beyond the first marked as "not applicable".

The neural network used for this task consists of three convolutional and pooling layers in alternating order and two fully connected layers at the end. An intermediate flatten operation is applied to transfer the tensor into a vector shape. In order to take the hierarchical nature of the labels into account, non-applicable logit neurons are masked out during training. This is achieved through a modification of the loss function and masked-out neurons do not contribute to the loss.

The given data comprise a total of $753\,000$ samples represented through grey scale images of size 22×46. For the AL experiment, this is split into a pool of $670\,000$ samples of possible candidates for the training set, as well as a balanced development set of $75\,000$ and a balanced test set of $8\,000$ images. The size of the initial training set is selected to be $2\,000$ instances which are also equally distributed among classes. As in previous trials, every active learning cycle should increase the size of the labelled training set by 10%. A baseline is created by training on all available

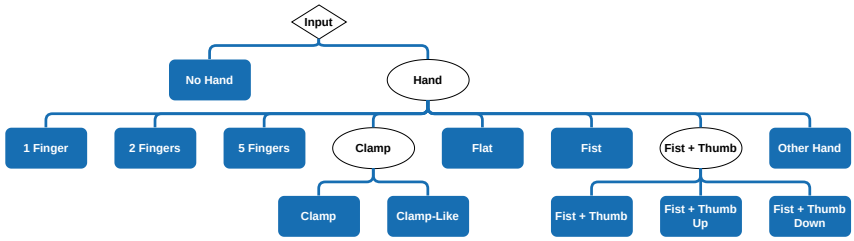


Fig. 3.7.: Hierarchical label structure for the hand gesture recognition task. Blue boxes denote the 12 classes.

samples and a random selection is used as comparison for the AL query strategies. All calculations are repeated ten times per method to achieve statistical significance.

The resulting graphs for all strategies are plotted in figure 3.8. They exhibit a generally very mixed performance as several strategies perform worse than random sampling. Especially approaches which rely on quantifying the uncertainty of the logits are rendered useless, showing a classification accuracy nearly ten percentage points lower than the one achieved with data selected without any specific decision criterion at all.

This does not come as a surprise, since these strategies implicitly rely on the assumption that a single label per sample is used and the neural network learns to output a probability distribution of classes. In the hierarchical label structure used here, this is not the case. For certain classes, up to three logits are expected to take on high values to constitute are correct classification. Output neurons that belong to classes marked as ”not applicable” in some cases are not considered during backpropagation and can therefore theoretically take arbitrary values and thus potentially confuse the AL strategies.

The neurons in the first fully connected layer, which in this example forms the embedding space, are less strongly affected by this training scheme. This at least suggests the above-average performance of the *Core Set* method. In marked contrast to the preceding observations, it is now the only strategy to outperform random selection. It is also is the only approach to reach the baseline accuracy early on even before taking 20% of the available data, making it considerable with regard to real-world use.

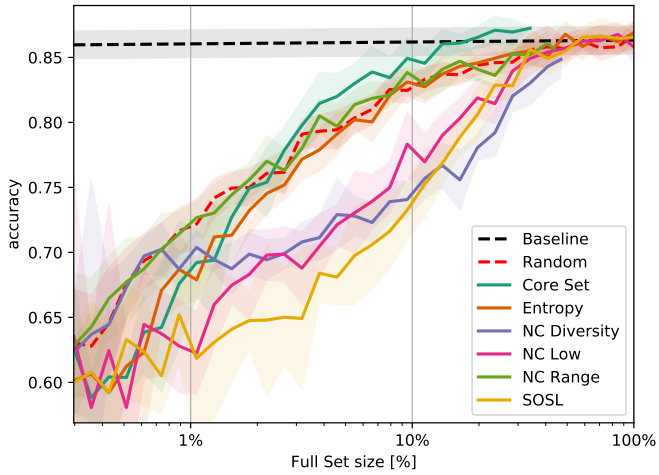


Fig. 3.8.: Classification accuracy over training set size for different active learning methods applied to hierarchical neural network for hand gesture classification. The plotted value is the median of ten runs per strategy and the shaded area denotes one standard deviation. Please note the logarithmic scale of the x axis.

3.4.2 Summation of the Findings

The different experiments in the preceding sections showed that even within a comparably narrowly defined selection of applications, in this case image classification with CNNs, the usability of strategies for pool-based active learning can vary greatly.

Starting with the nature of the data itself, the conducted test showed that the performance of query strategies is not independent of this. As seen in section 3, methods that work well on a number of datasets might suddenly fail on a different one and certain data collections might be inherently unsuitable for this kind of active data selection. When the pool of unlabelled data contains no redundant information, AL will likely yield no benefit.

Although many changes to hyperparameters and erroneous labels on the one hand have no impact on the performance of certain strategies, changes to the classifier on the other hand may very well. If AL is considered for application in an industrial environment where it has a high potential to help to save time and money, caution is required. Should the classifier be replaced at a later stage due to stricter requirements or architectural advantages, the performance may be reduced. As the experiments

clearly displayed a bias for a network architecture to perform best with its own training data selection.

Changing the output format of a classifier from a straightforward probability distribution might entail even greater risks. The concluding experiment demonstrated that using a hierarchical label structure can turn all previous findings upside down. An even more drastic outcome is to be expected if, for instance, a regression task was considered. Almost none of the strategies tested here were designed to understand such a numerical output, and using those that rely on information from the embeddings would be the best remaining option.

In the end, it can be summarised that active learning can be a helpful tool in data science, but has to be used with knowledge about the targeted utilisation. Claiming that it can be applied without domain experience and regardless of the context would be dangerous. As the experiments underline, in some cases accuracy could even be negatively affected and an uninformed user would be better advised to use random sampling.

4 A New Method for Architecture Selection of Convolutional Neural Networks

This chapter presents a new method for architecture selection of convolutional neural networks (CNN). First, aspects of the field of hyperparameter optimisation are described in section 4.1 and a motivation for the problem is given. To evaluate the performance of a CNN architecture, in section 4.2 a heuristic approach is developed with a focus on balancing time consumption and prediction reliability. In section 4.3 this is then combined with aspects of random search and Bayesian optimisation to create a complete selection algorithm.

4.1 Hyperparameter Optimisation

Selecting the optimal network architecture and corresponding hyperparameters when trying to solve a new machine learning task remains one of the core problems even for experts in the field. This is a topic that has received a lot of attention in the research community in recent years, and it exists for three major reasons:

First of all, strategies that are considered best practice and have been shown to effectively accomplish particular tasks may not be applicable to a different problem. This stems mostly from the unique features of the given data and is difficult to standardise. These challenges are also the origin of the independent field of transfer learning (Zhuang et al., 2021).

Second, as a result of an exponential increase in the number of publications on machine learning throughout the last decade, new building blocks for designing neural networks have been introduced to the toolbox, with the state of the art quickly and continuously changing. Keeping up with this rapid growth can be challenging, even for those that have prior experience. Last but not least, the time intensive

difficulty of determining the right solution by analysing a wide range of different architectures can become the most significant issue. Although this constraint may not be as serious in research, it most certainly restricts the future success in product development in business implementations where time and, therefore, money are key.

Considering these challenges, it stands to reason that one would want to create methods for determining the best network architecture for a given problem. This system should not only be effective in forecasting real outcomes, but it should also be fast enough to clearly outperform manual methods in terms of time consumption. Since final optimisation through human supervision is unlikely to become entirely obsolete in the near future, the quicker decision support is given, the more time exists for fine tuning the result.

In the last years, automated machine learning (AutoML) became a very prominent research area in computer science. Driven by leading companies in the software and technology industries, a variety of toolboxes were developed (Zöller and Huber, 2021). These methods mainly target non-experts and are offered as all-in-one solutions, often with integration into cloud computing and storage products. A very high level of automation handles not only a model selection process and hyperparameter optimisation, but offers the user approaches for automatic input data preprocessing, feature engineering and choice of evaluation metrics and validation procedures.
Although this technological progress is impressive and can open the doors to the machine learning world for an even larger number of people, it is largely based on very deep network architectures, benefits from vast computational resources in server clusters and employs a multitude of different algorithms for optimisation. While there a certainly commonalities to this high-level approach, the methods presented on the following work on a more fundamental level and are tailored to applications with CNNs. They are therefore not directly comparable.

4.2 A Heuristic Approach

This section should highlight the influence of certain aspects and parameters on the convergence of training a neural network, especially in an early phase. Furthermore, references to previous work analysing the significance of the use of random weights are made. A heuristic approach to the quick evaluation of network architectures is derived from this.

4.2.1 Initialisation Influence

Weight initialisation plays an important role in preventing layer activation outputs from becoming too large ("exploding gradient problem") or infinitesimally small ("vanishing gradient problem") during the course of a forward pass through a neural network, especially in the initial/early phase of the training process. If one of those cases occurs, loss gradients will either be too large or too small to be propagated backwards in a manner beneficial for the convergence of the network, if this is still possible at all.
Accordingly, it is desirable to initialise neural network weights with small values around a mean of 0. While using a uniform distribution $\mathcal{U}(a, b)$ would not be completely out of the ordinary, a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with $\mu = 0$ and $\sigma^2 \leq 1$ was a common standard.
In their work, Glorot and Bengio created a now classic initialisation scaling with the aim of improving on the standard methods of the time. Their target was to maintain the variance of activations and gradients in all layers of the network (Glorot and Bengio, 2010). They proposed the uniform distribution scaling

$$W_{Xavier} \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right] \tag{4.1}$$

taking into account the number of incoming n_j and outgoing connections n_{j+1} of a layer. It is now natively implemented in many machine learning libraries and its scaling can of course be transferred to other distribution functions.

It is easy to understand, that the aspect of initialisation and its influence on convergence becomes particularly important for a process that should allow for a very fast evaluation of the performance of an architecture. But even beyond that there is an influence that should not be neglected, as an initial experiment showed. The vast majority of random number generators (RNG) used in computer programs do not produce truly random numbers but pseudo-random ones. An algorithm, like the widely used "Mersenne Twister" (Matsumoto and Nishimura, 1998) for example, generates a sequence of numbers whose properties approximate those of a sequence of random numbers. To do so, a starting value or *seed* is needed. The output is therefore entirely dependent on this value and one specific seed will always create an identical series of numbers.
Figure 4.1 depicts a histogram of the outcome of training the same architecture with the same data set 15 times for 100 epochs using different seeds to initialise the RNG and subsequently the network's weights. The value of the classification error on validation data ranges over more than one percentage point. Compared with the

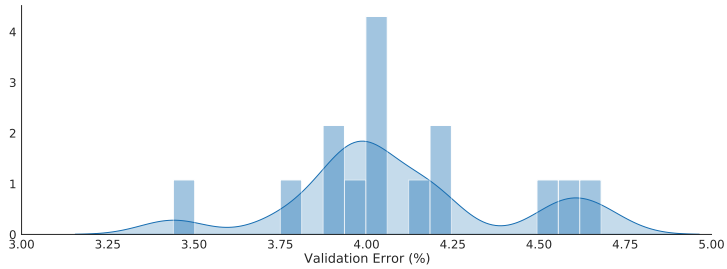


Fig. 4.1.: Distribution of the validation error over 15 weight initialisations of one architecture, ceteris paribus.

magnitudes of improvement discussed in many publications, especially in regard to competitive benchmarks, this is a quite significant magnitude. Accordingly, this shows the necessity to train multiple initialisations of every candidate architecture for reliable testing and at the same time the need to use the same specific seeds for reproducibility.

4.2.2 Random Weights and a Heuristic

Methods which sufficiently approximate the outcome of a computationally extensive task in a short amount of time are used in many areas. Apart from the pure time saving potential, the reliability of such a prediction is crucial. This is particularly true for the task of CNN architecture selection. Here, the required amount of time can become a major problem, especially against the background of uncertainty in results and the possible need to evaluate a single architecture more than once.

Previous works made use of random weight values to estimate the performance of learning algorithms or generally make remarks on the training process. (Jarrett et al., 2009b) were surprised by their own findings about the performance of random filter weights for feature extraction and stated that “architectural sophistication seems to compensate for lack of training”. In their 2011 paper, (Saxe et al., 2011) analysed how random weights perform when used in convolutional and non-convolutional feature extraction architectures before using a linear support vector machine (SVM) for classification. They reported a significant correlation with the results using normal training on these layers and motivated use in the context of architecture search. Nevertheless, they admitted the need for several random initialisations per architecture naming the performance difference “not generally negligible”. They

further said that the correlation between random and trained performance is not persistent in regard to high performing initialisations.

Given these earlier findings it is clear that random weights could also be considered for evaluation of CNN architectures. Then again, another methodology which quickly comes to mind is training the network normally but for a fraction of the normal period of time. It is natural to expect that this would show a high correlation with the actual outcome.

A new heuristic approach can be derived against this background combined with experience gained from training neural networks. Both previously mentioned works distinguish between a feature extraction and a classification part of their algorithms. Even without valuable contributions to help understand feature representations in neural networks (Yosinski et al., 2015), it is clear that even having only roughly adapted feature extraction filters is preferable to those layers outputting random distributions. It seems reasonable to assume, that training of the fully connected layers could benefit from this, since one or two updates to the filter weights could compensate for far more epochs of training the fully connected part. The initial phase of training a classifier also normally provides the highest relative performance improvement, as gradients and subsequently weight updates are large.
Therefore, the following heuristic is proposed:

1. Only compute two epochs of back propagation updates for the feature extraction layers of the network.
2. After the third forward pass of the training samples through the convolutional layers, save the feature maps.
3. Based on those intermediate result, train the fully connected classification part of the network for a few epochs.

Including the second step to transfer the training data into this new format saves computation time, as otherwise the forward pass of the samples through the convolutions would need to be computed every time even though only the fully connected layers are trained. The number of epochs for this training set to 30 on the basis of an otherwise 100 epoch long full training.

	Random Weights		Short Training		Proposed Heuristic	
	Mean	Best	Mean	Best	Mean	Best
Mean	0.813	0.769	0.832	0.774	0.641	0.852
Best	0.791	0.750	0.757	0.720	0.567	0.843
Time Consumption	89.3%		50.0%		22.6% / 45.2%	

Tab. 4.1.: Comparison of three different evaluation methods for architecture performance as correlation coefficient with the outcome of a reference training and time required. Correlation is given for mean and best values of the respective methods and the full training. The time required is given relative to that of one full training.

4.2.3 Evaluation

In a first experiment the random weights approach, short time training and the proposed heuristic are compared. Short training is considered to be five epochs of full training. A small dataset of traffic signs, named TSR-8, was selected. It consists of eight classes; seven different speed limit signs and a negative "no sign" class. The training set is balanced comprising $2\,500$ samples of each speed limit sign and $6\,000$ of the latter. Validation data includes 200 samples per class. Based on a small CNN with four convolutional and two fully connected layers, 24 combinations of changes to this architecture are considered as candidates. To measure how good an estimate generated by one of the methods approximates the actual training performance, the Pearson correlation coefficient

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 (y_i - \overline{y})^2}} \qquad \text{with} \quad \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.2}$$

is used. Random weights and short time training compute results of ten initialisations per candidate architecture, the proposed heuristic only gets five of those performance samples. The reference training is run five times per network configuration.

Table 4.1 lists the outcome of this experiment. All three approaches exhibit a generally good correlation with the actual training performance. The proposed heuristic shows the most significant correlation when looking at the best performing initialisation of an architecture. It is also clearly the fastest method, taking on average only 22.6% of the time of single full training (which given the explanations in subsection 4.2.1 is not entirely reliable on its own). Even if ten initialisations are calculated per configuration, which brings no performance advantage but an even fairer comparison with the other approaches, it is still the fastest.

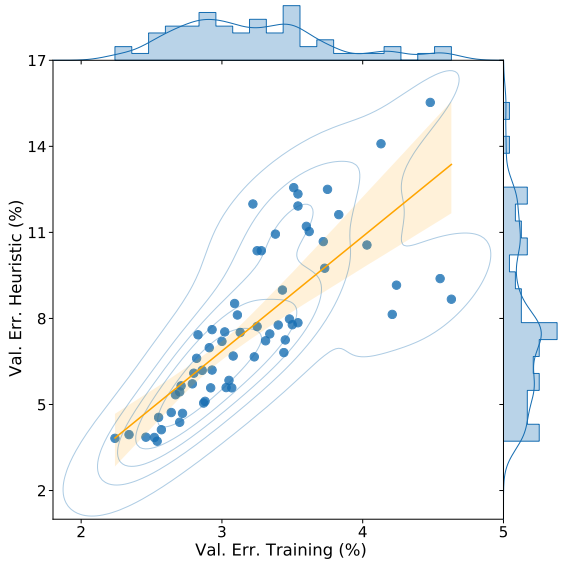
	MNIST	USPS	CoMNIST	SVHN	AHC	OD
Mean	0.785	0.828	0.679	0.353	0.875	0.800
Best	0.789	0.742	0.597	0.319	0.878	0.818

Tab. 4.2.: Correlation coefficients of the validation error projections from the proposed heuristic and reference training (arithmetic mean and best) on six data sets.

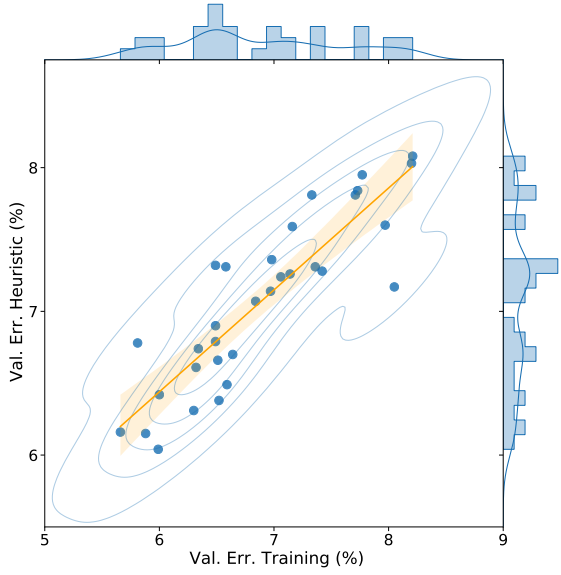
To further validate the proposed heuristic, its capability to predict a neural network's performance is tested on more data sets. These include four public ones; MNIST (Lecun et al., 1998), USPS (Hull, 1994), CoMNIST (Vial, 2017), SVHN (Netzer et al., 2011), as well as two private sets; AHC, a data set for light source classification for high beam control, and OD, a set for the classification of road users. Detailed information on these data can be found in the appendix table A.1. For each architecture a number of small CNN architectures are tested in a grid search. Variations to be evaluated include the number of layer, channels, fully connected neurons and more, as well as different global hyperparameters. Each architecture is initialised five times, both for the heuristic in the reference training.

The experiment generally confirmed the significant correlation between performance projections from the heuristic and the actual training outcome. This is equally true for the arithmetic mean and best result of the reference training, as table 4.2 shows. Four out of six data sets verify this without any restriction. Figure 4.2 depicts plots for all tested configurations on MNIST and AHC. For CoMNIST a reduced correlation of the results is measured. Nonetheless, the best overall network can still be found within the first three architectures proposed by the method. These three would be among the five best networks within the whole search and therefore provide a useful result in practice. When using the SVHN data for validation, the outcome is not correlated with the training results. This occurred due to general overfitting in all architectures, which led to results that were difficult to distinguish even in reference training.

All things considered, the proposed heuristic approach produced fast and reliable results for performance evaluation of CNN architectures. It can therefore be considered a valuable asset to a search for the best network configuration.
To some surprise, the magnitude of the validation error scores produced by the heuristic is very close to the ones of the reference training in some cases. Validation errors sometimes deviate by less than one percentage point. While this could be considered to be proof of the notion expressed in previous works that architecture sophistication prevails even against a lack of training to some extent, it also rises interesting questions on how training of neural network layers should be conducted in general.



(a) Results for the MNIST data set.



(b) Results for the AHC data set.

Fig. 4.2.: Correlation of validation error performance for full training *(x-axis)* and the proposed heuristic *(y-axis)* for all tested network architectures.

4.3 Bayesian Optimisation Framework

While the previous section showed, that the proposed heuristic approach offers a fast and robust way to evaluate candidate architectures without the need of full training, the underlying grid search, although being good for a comprehensible evaluation, is not as efficient in covering a vast search space. To further optimise an architecture search in terms of result quality and computation time, one might want to use a more sophisticated method. Even though the exhaustive properties of a grid search can be beneficial, it produces an exponentially growing number of sampling points to evaluate. As presented in the work of Bergstra et al., random search outperforms grid search when it comes to hyperparameter optimisation. They compared both approaches on architecture search for neural networks and discovered, that random search not only necessitates a fraction of the computation time for equal results, but by trend finds even better solutions. They stated: “Grid search experiments allocate too many trials to the exploration of dimensions that do not matter and suffer from poor coverage in dimensions that are important.” (Bergstra and Bengio, 2012). This is understandably sensible, as a large-scale multi-parameter search would likely not be needed in the first place, if the sensitivity of specific parameters was known beforehand.

To further increase the performance capability of a search method, one might not want to solely rely on a random process but add an algorithmic component for more plausibility and focus. Here, approaches in the field of global optimisation can be considered. Model-based optimisation (MBO) is a type of global optimisation (cf. Efficient Global Optimisation (EGO) (Jones et al., 1998)) and plays to its full strength especially when the number of parameter combinations is large and too expensive to evaluate in respect of the computation time. This method was created to find solutions for problems with a great variety of input parameters and missing information of their internal working, so called expensive black-box functions. This relates to the given task of finding the best architecture for a CNN concerning a vast number of hyperparameters and their respective possible values. Lastly the principle of a black-box describes the state of knowledge about internal processes in neural networks to some extent.

This work makes use of the “mlrMBO” library by (Bischl et al., 2017). The sequential cycle of this approach to estimate the best parameters for an expensive black-box function $f(\vec{x})$ can be structured into the following steps:

1. To establish a reliable data basis that can act as a starting point, an initial design can be found using a method to one's choice. As described before, this is normally a random search since a grid search is deemed to be too expensive. Here Latin hypercube sampling, originally described by (McKay et al., 1979), is a good option to refine an otherwise purely random approach. It generalises the concept of the Latin square to an arbitrary number of dimensions. Each of the n_{init} points sampled from this multidimensional space is the only one in each axis-aligned hyperplane containing it. In this way, the method tries to cover as much of the valid configuration space as possible with a minimal amount of samples taken.
   Say $\vec{x}$ is a parameter configuration within the space of all valid configurations $\mathcal{X}$ of the CNN algorithm. Then we are searching for an optimal configuration $\vec{x}^*$ minimizing the cost y, which in this example is the validation error. The result of this initial step are tuples $(\vec{x}_i, y_i)$ with $i = 1, ..., n_{init}$.

2. With the starting points evaluated in step 1, a surrogate model $\hat{f}(\vec{x})$ to approximate the expensive black-box function $f(\vec{x})$ is estimated. The given library uses Kriging models, also known as Gaussian process regression, to estimate a result y_i from a not tested configuration $\hat{f}(\vec{x}_i)$ (Krige, 1951). As far as this work is concerned, it is sufficient to note that this is a type of interpolation modelled by a Gaussian process. Essential for this application within MBO is its characteristic to not only provide result prediction, but also estimate the corresponding uncertainty $\hat{\sigma}(\vec{x}_i)$.

3. Given the surrogate model function $\hat{f}(\vec{x})$ promising points for optimisation are selected using an infill criterion. While other methods could be used, the expected improvement

$$EI(\vec{x}) := E(\max(y_{min} - Y(\vec{x}), 0)) \tag{4.3}$$

   is a tested choice (Bischl et al., 2014). Let $Y(\vec{x})$ be a random variable, which for the assumed Gaussian process is normally distributed with $Y(\vec{x}) \sim \mathcal{N}(\mu(\vec{x}), \sigma^2(\vec{x}))$, and y_{min} the best result obtained so far. With this premise, the expected improvement can be expressed as

$$EI(\vec{x}) = (y_{min} - \hat{\mu}(\vec{x}))\ \Phi\left(\frac{y_{min} - \hat{\mu}(\vec{x})}{\hat{\sigma}(\vec{x})}\right) + \hat{\sigma}(\vec{x})\ \phi\left(\frac{y_{min} - \hat{\mu}(\vec{x})}{\hat{\sigma}(\vec{x})}\right) \tag{4.4}$$

   in its full form, where Φ and ϕ are the distribution and density function of the standard normal distribution. Accordingly, a desired candidate point is defined as $\vec{x}^* = \arg\max_{\vec{x} \in \mathcal{X}} EI(\vec{x})$. This formula aims to balance aspects of

exploitation and exploration in the parameter space using the prediction and uncertainty provided by the surrogate model estimate. Exploitation describes pure focus on the lowest estimated mean $\hat{\mu}(\vec{x})$ searching for a minimum with high expectancy. Exploration on the contrary targets regions with large estimated standard deviation $\hat{\sigma}(\vec{x})$ and consequently high uncertainty in the model $\hat{f}(\vec{x})$. In the applied library "mlrMBO" the point to be evaluated eventually is determined with an infill optimisation algorithm called "focus search". Other optimisation methods could be chosen. The specifics of this algorithm can be found in the original publication. Its function can be summarised to using the best point obtained so far to focus the search space around it and identifying the next promising candidates using the expected improvement.

4. The proposed configuration $\vec{x}_j^*$ is then evaluated on $f(\vec{x})$ and the surrogate model $\hat{f}(\vec{x})$ is thus updated with the new tuple $(\vec{x}_{(init+j)}, y_{(init+j)})$.

5. If a defined termination criterion is not reached, to process continues with step 2. Otherwise the cycle ends. For this application the termination criterion is chosen to be fixed number of n_{iter} iterations. As long as this budget is not exceeded, no configurations are determined and evaluated and the surrogate model is subsequently updated. Other criteria like reaching a lower objective value bound might be a chosen.

6. If the termination criterion in step 5 is reached, the best configuration $\vec{x}^*$ yielding y_{min} when evaluated on $f(\vec{x})$ is returned.

In combination to the heuristic architecture evaluation approach proposed in section 4.2, this application of MBO has the potential for a strong synergetic effect. Both techniques are designed for economy of time but also provide in-depth exploration. MBO aims to cover the whole search space with a small number of sample configurations and focuses on areas with high potential improvement during optimisation steps. The heuristic adds a reliable measurement basis for evaluation through multiple initialisations of each architecture.

4.3.1 Evaluation

The significant question to evaluate the usability of the proposed method combination for architecture search is once again not only about speed, but reliability. Since it is expected from the beginning, that the introduced heuristic and MBO-based search will require less runtime than MBO search combined with normal training, it is important to see whether the former can produce results that are comparable.

	Training + MBO	Heuristic + MBO
Runtime	4706 min.	2586 min.
Val. Err. TSR-8	2.56%	2.37%
Val. Err. TSR-20 (best/mean)	0.84% / 0.90%	0.63% / 0.84%

Tab. 4.3.: Performance indicators in comparison for the MBO search using normal training and the proposed heuristic approach.

To examine both approaches, a search using the TSR-8 data set was conducted. (Bischl et al., 2017) suggest to acquire five times as many data points for the initial design as parameters in the optimisation problem. With $p = 10$ parameters, the number of initial trainings of architectures determined by Latin hypercube sampling is set to $n_{init,T} = 5p = 50$. The budget for the optimisation iterations is selected to be $n_{iter,T} = 100$ and every network is trained normally for 100 epochs. Since the time consumption when using the proposed heuristic is expected to turn out significantly lower, the contingent for initial and iteration steps is determined as $n_{init,H} = 100$ and $n_{iter,H} = 200$ from the start.

As the results presented in table 4.3 show, even with twice as many configuration sets to test, the search employing the proposed heuristic only needs ~ 43 hours compared to ~ 78 when using full training, which is approx. 55%. More importantly, with the proposed search method there are not only no significant losses in performance, the best candidate architecture actually slightly outperforms the parameter setting found with normal training, when fully retrained. Even when just using the direct result from the heuristic approach, the validation error is only 2.88%. As already noted with other experiments in the previous section, this is surprisingly close in magnitude to a full training performance. To confirm this finding, both best network architecture configurations were trained from scratch on the extended TSR-20 data set, including 12 additional sign classes compared to the TSR-8 representation. Each network was trained five times using different weight initialisation seeds. The architecture found with the proposed heuristic combined with MBO again reaches a slightly better performance on the validation set.

In summary, it could be shown that the presented heuristic and MBO develop a helpful synergy when applied to the problem of neural network architecture search. Through multiple initialisations of a single parameter configuration, the heuristic provides the search optimisation algorithm with results, which are more robust and reliable than the outcome of one single training, while also offering a clear runtime benefit.

5 A System for Real-Time Panoptic Segmentation in Lidar Data

To create a fast approach for three-dimensional in-vehicle object recognition, new methods to facilitate object segmentation in lidar point clouds in real-time are presented in this chapter. Section 5.1 outlines the existing state of the art for detection and classification approaches in lidar. The motivation to use a clustering algorithm to create class-agnostic object instance detection is explained in section 5.2. Here, a new approach is considered to leverage the properties of the Euclidean distance to retain three-dimensional measurement information, while narrowing down the point cloud to a two-dimensional representation for fast computation. It is shown, how this and other optimisation steps help to enable real-time execution on CPU.
Section 5.3 clarifies how the output of this clustering block can be represented to form an efficient and meaningful data representation for classification of the object instances. A neural network to use this format and realise classification is subsequently introduced in section 5.4. An evaluation regarding different metrics for object detection and segmentation is eventually presented in section 5.5 alongside timing measurements. The concept of panoptic segmentation is also introduced and analysed here.

5.1 Lidar Object Detection Algorithms

There is a multitude of publications concerned with object detection and classification on lidar data. In an attempt to provide a short overview, they can be roughly divided into three categories; algorithms that work on unregularised point clouds, those that use regularised ones, and algorithms that use fusion approaches combining lidar with other sensors, mostly camera data.
A lot of methods process raw, unregularised point clouds and usually generate per-point features, therefore being sometimes referred to as point-wise algorithms. A pair of publications introducing *PointNet* (Qi et al., 2017b) and its successor PointNet++

(Qi et al., 2017a) established the baseline of this category. While the original network generated features on a global level at one scale, the latter improvement cascaded several instances of these processing layers of the aforementioned approach at different scales for more localised features.
Several other publications, including PointRCNN (Shi et al., 2019), followed this direction. While facilitating good performance, these approaches are computationally intensive and designed to be executed on large GPUs. This rules them out for embedded application.

A second group of algorithms regularises lidar point clouds before processing them. To do so, grid structures are used in fixed (Zhou and Tuzel, 2018; Shi et al., 2019; Chen et al., 2019) or variable sizes (Alsfasser et al., 2020) to preprocess the data. Zhou and Tuzel laid the foundation for these approaches by introducing what they called "Voxel Feature Encoding". Per grid cell, one feature vector with a fixed length is generated and padded with zeros if not enough data points are available. Taken together, these then form sparse four-dimensional tensors, which are fed into a convolutional network.
To alleviate the negative performance influence of many empty grid cells created by this methods, works like (Yan et al., 2018b) established specific network layers to exploit this sparsity, providing a significant speed-up. (Lang et al., 2019) reduce grids to a two-dimensional image-like representation while preserving height information through an encoding. While this enables the omission of costly 3D convolutions for faster runtime, these networks are still very demanding in computational resources and require graphic cards.

The third and last set of works on object detection in the lidar space adds information from camera sensors to enrich point cloud data. Some of them use camera images to create region proposals, which are refined by a lidar algorithm in a subsequent step. Here, a popular approach is the use of a frustum for projection into the point cloud, as shown in (Qi et al., 2018; Zhao et al., 2019) and (Wang and Jia, 2019) for example. Comparable methods utilise full three-dimensional object proposals from image inputs to refine the frustum cutout (Shin et al., 2019) or more complex deep learning fusion networks (Chen et al., 2017).

All of the approaches mentioned above are highly computationally intensive, with many of them not being real time capable even on high-end GPUs. There is not much work to be found concentrating on real-time application of lidar algorithms on systems with less computational power. (Minemura et al., 2018) claim such capabilities using specific optimisations for powerful CPUs, but report sub-par results.

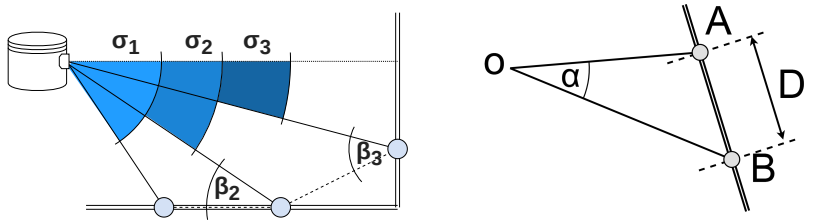
5.2 Fast Clustering by Density and Connectivity

As motivated in the previous section, there are many powerful methods to accomplish the task of object detection in lidar point clouds. Since they require equally powerful hardware to be computed, an alternative approach might be sensible to implement this function into an experimental test vehicle.

While machine learning can be a versatile and helpful class of algorithms for many problems, especially when it is not possible to explicitly formulate a model for a given task, it is not necessary to disregard tried and tested classical methods because of this. When it comes to separating, or segmenting for this matter, data points into groups, clustering algorithms are a good and obvious choice. Well-known representatives include the *DBSCAN* (Ester et al., 1996), *Mean Shift* (Fukunaga and Hostetler, 1975; Comaniciu and Meer, 2002) and *OPTICS* (Ankerst et al., 1999) algorithms for example. Although these are comparatively old by the standards of the rapidly developing machine learning world, especially the first one remains widely used and was proven to remain relevant especially in data mining (Schubert et al., 2017). However, it is still clearly too complex for real-time execution on sensor data with limited computational resources.

Previous works like (Moosmann et al., 2009) recognised this problem and aimed for approaches more specific to lidar application. They combined a graph-based approach with analysing the surface of a point cloud locally with regard to the convexity. Two neighbouring surfaces are termed locally convex to each other, if their centre points are mutually below each other's surface. While showing good results in urban environments, the algorithm fails to achieve real time capability. (Bogoslavskyi and Stachniss, 2016) targeted mobile application on a robot and therefore required a very fast execution time. They also regarded surface convexity, but used angles related to the sensor origin for calculation and more importantly transferred many operations to a depth image representation.

These publications form a good basis for further considerations in terms of highly efficient clustering of lidar point clouds. The aspect of maintaining spatial features while moving operations to a two-dimensional plane is particularly interesting here. Although data from a lidar sensor is most recognisable when represented as a three-dimensional animation, raw data packages are of course transmitted as a list of values. These are normally ordered by channels, which themself are for most applications stacked in vertical direction. For each scanning position of a channel, a range measurement is returned alongside a value representing the reflectivity. With



(a) The angle measurement of lidar points in the vertical direction is used to define a horizontal orientation for the ground plane extraction. (Note that there is no β_1, as the first lidar channel has no previous channel. β_2 is therefore extended to the first channel.)

(b) With the lidar sensor in O, the lines OA and OB show two neighbouring distance measurements. The distance between the two measurements is calculated using the spanned angle α between the points.

Fig. 5.1.: Trigonometric relationships used in the ground segmentation *(a)* and the cluster separation *(b)*.

the channel number as y-axis and the value order per channel as x-axis a range image, sometimes also called depth image, can be readily created.

Ground Plane Extraction

For any ground-based vehicle, like a car, it is helpful to extract and ignore the points belonging to the ground plane from the segmentation. This will prevent the algorithm from connecting two instances via this plane due to the nature of the lidar scan lines in horizontal direction. Here, a simple height based threshold clipping values below a certain z-value is not sufficient, as the road surface itself can be uneven. Pitching and rolling of the ego vehicle can also influence the way the ground is perceived in the sensor data.

Given detailed information about the lidar sensor, the exit angle σ_r for each channel r can be used to determine the angle β_r at which the laser beam hits the surface (see figure 5.1a). Let $d_{n,r}$ be a position in a range image at the n-th cell (on the y-axis of the image plane) of channel r (on the x-axis). Then, an angle image, representing the angle values of the lidar beam in relation to the point cloud surface spanned between the current measurement and the measurement of the channel below, can be created in which each new cell is calculated using the relationship

$$\beta_{n,r} = \arctan\left(\frac{d_{n,r} \cdot \sin\left(\sigma_{r+1} - \sigma_r\right)}{d_{n,r+1} - d_{n,r} \cdot \cos\left(\sigma_{r+1} - \sigma_r\right)}\right), \tag{5.1}$$

comparing vertically neighbouring range image cells. From this angle image, all cells below a threshold of $\pm 10°$ to a full horizontal surface are selected and their

respective counterparts at the same position in the range image are deleted. This leaves only objects above the ground plane for further processing.

Clustering

Figure 5.1b shows the trigonometric relationship of two adjacent points measured by a lidar sensor. The range values $||OA||$ and $||OB||$ correspond to neighbouring cells d in the depth image. To calculate the Euclidean distance D between the two points, the cosine law can be applied:

$$D = \sqrt{||OA||^2 + ||OB||^2 - 2 \cdot ||OA|| \cdot ||OB|| \cdot \cos\alpha} \; .$$

For any n-th cell of channel r in the range image and using the lidar sensors exit angles σ_r this translates to

$$D^{\text{vert}}_{n,r} = \sqrt{d^2_{n,r+1} + d^2_{n,r} - 2 \cdot d_{n,r+1} \cdot d_{n,r} \cdot \cos\left(\sigma_{r+1} - \sigma_r\right)} \tag{5.2}$$

for vertical connections. Accordingly, for horizontal connections it can be calculated as

$$D^{\text{hor}}_{n,r} = \sqrt{d^2_{n+1,r} + d^2_{n,r} - 2 \cdot d_{n+1,r} \cdot d_{n,r} \cdot \cos\theta} \tag{5.3}$$

with an constant horizontal resolution θ, which for most lidar sensors is between $0.2°$ and $0.05°$.

Using this physical distance between two measured points, a threshold value can be defined between those, which are close enough together to belong to the same object, or too far apart to be considered neighbours on the same object. The distance of adjacent measurements on a given object is in general relatively close. The distances of those points in the range image from two separate objects are substantially larger. As far as this work is concerned, this threshold is set to 0.8 metres.

By using this clustering criterion with the range image, one horizontal and one vertical connection image can be created. After applying the selected distance threshold, these images can be brought to a binary representation where every valid connection below the threshold value becomes 1 and everything else is 0. In a subsequent step, the range image is also brought to a binary from, representing the presence and absence of lidar measurements for the corresponding pixels in the image.

These three created binary images contain all the required information to segment the lidar measurements of the whole frame into clusters and background points. For this, the images are all dilated and combined into one large grid image as illustrated in figure 5.2. Now it is possible to apply a simple and efficient image processing algorithm; connected-component labelling (CCL). Here, the “one component at a time” CCL implementation of (Abubaker et al., 2007) is used in the fast and straightforward version available in the *scipy* library "label" (Virtanen et al., 2020). The 4-connected pixel connectivity, also known as von Neumann neighbourhood (Toffoli and Margolus, 1987), is defined as a two-dimensional square lattice composed of a central cell and its four adjacent cells. Using the CCL algorithm with a 4-connectivity on the grid image makes it possible to separately label each “island” of interconnected measurements as a different cluster. The resulting segmented image is then eventually subsampled to the original range image size. These images are visualised in figure 5.4.

This ultimately enables the generation of three-dimensional cluster labels directly from the connected-component image, as each pixel corresponds to a given lidar point in the three-dimensional point cloud. The lidar sensor measurements are thus segmented into connected components of separate objects and non-segmented points, which correspond to the ground plane and background noise. In a final step all clusters with less than 100 points are rejected to reduce false instances resulting from noise in the sensor, the ground plane extraction or very small static objects such as poles and debris on the road.

Map Connections

All lidar segmentation algorithms are to some extent prone to under- and over-segmentation, due to the characteristics of the sensor. Namely its sparsity (especially in vertical direction for most 360° rotating lidars) and missing measurements resulting from deflected laser beams, which have no remission value back to the sensor. Missing values result in missing connections between areas of the same object, due to which the direct neighbourhood approach described above will over-segment a single object into multiple clusters. Examples of such challenging instances are shown in figure 5.5.

To overcome the limitations of the direct neighbourhood approach and to ensure a more robust segmentation, the two-dimensional Euclidean clustering is extended by what is referred to as *Map Connections* (MC) in the further course of this work. For this, the combined grid image shown in figure 5.2 is reduced to a sparse matrix,

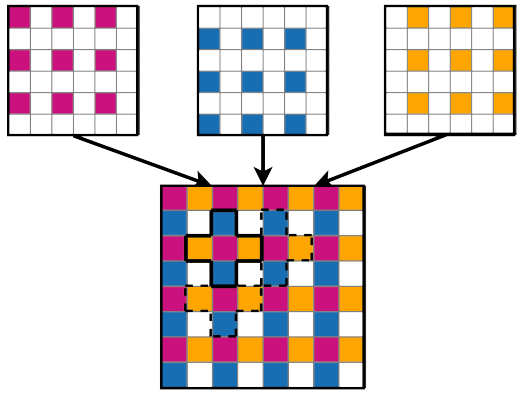


Fig. 5.2.: Combination of defined image representations for instance segmentation. The red squares represent the binary value of present lidar measurements, the yellow and blue squares represent the horizontal and vertical connections of these measurements respectively. The kernel shown on the combined image should indicate the 4-connectivity.

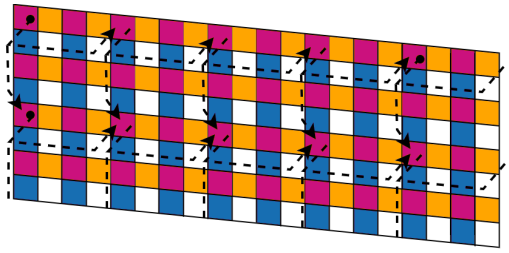


Fig. 5.3.: Additional map connections *(dotted lines)* between non-neighbouring lidar points on top of the direct connections to neighbouring points *(yellow and blue squares)*.

connecting only every n-th point in the vertical and horizontal direction, thus connecting a subset of original points. The schematic visualisation in figure 5.3 displays a connection of each measurement with its second neighbour ($n = 2$). Higher values for n are also possible and evaluated later in this chapter. This allows for a robust connection of segments of the same object which have no direct connection due to missing measurements or obstruction by other objects in the range image.

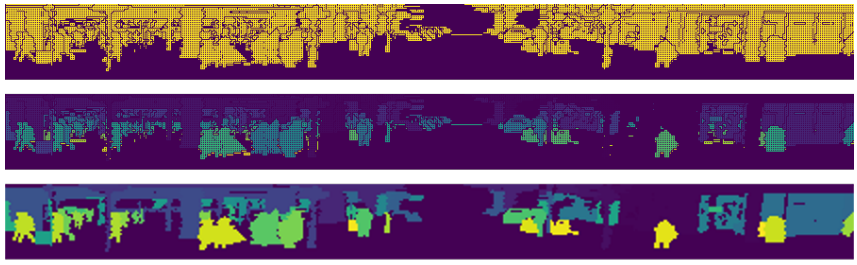


Fig. 5.4.: Visualisation of the combined binary grid image representation *(top)*, the applied 4-connectivity *(middle)* and the result reprojected to the lidar range image resolution *(bottom)*.

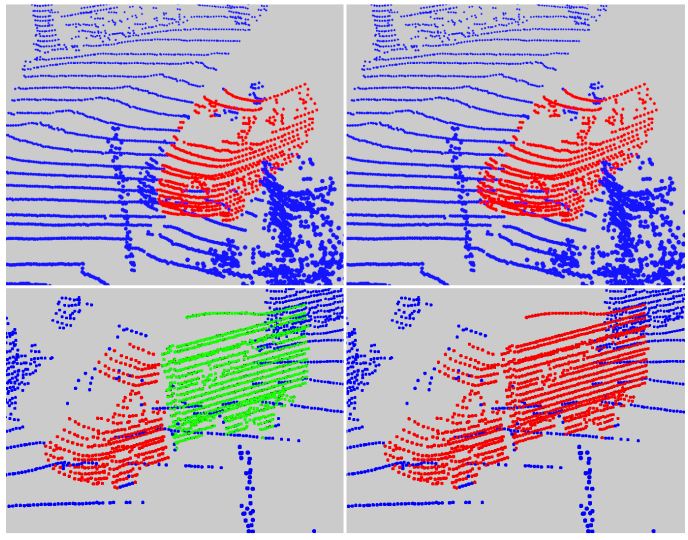


Fig. 5.5.: *Left*: Results using only the direct connectivity between neighbouring lidar points. *Right*: A single additional MC between every second lidar measurement. The proposed MCs enable a more accurate segmentation of the car *(top)* and reduce the over-segmentation of partially occluded objects, as the truck in the bottom images shows.

5.3 Meaningful Data Representation of Segmented Lidar Instances

To prepare and optimise lidar data representations of object instances obtained from a clustering algorithm for classification in a subsequent step, several operations can be considered.

5.3.1 Image Layers

From the raw sensor data, different information aspects can be used to describe a measured point. The obvious first elements are the x, y and z coordinate values of this point, usually given in Cartesian coordinates with the origin of the coordinate system in the location of the lidar sensor. Derived from this, the distance or range of the point from the origin can be calculated, for example as Euclidean distance $r = \sqrt{x^2 + y^2 + z^2}$. Furthermore, the measured intensity of a point is normally given in the raw data, enabling indication of the surface reflectivity of an object. When thinking about preparing lidar data for image classification, these properties already create five possible image layers.

To gain more knowledge about an object's surface in the lidar space, normal vectors are a tried and tested representation. Their application is particularly popular in the field of odometry or SLAM (Moosmann and Stiller, 2011; Behley and Stachniss, 2018; Li et al., 2019). The normal in geometry is a vector which is perpendicular to a given object. In three dimensions, the normal of a surface in a given point is a vector pointing outwards of the surface. To remain in a two-dimensional space for this use case, it is best to calculate an image representation of the horizontal and vertical component of the normal vector respectively. The normal vector for a measured lidar point can be determined using the angle relationships shown in figure 5.6. α and β are the angles between the line from the sensor origin to the given point and the line from the given point to its respective neighbour. The angle bisector $\frac{\Phi}{2}$ of the combined angle $\phi = \alpha + \beta$ equals one component of the normal vector.

To further leverage the two-dimensional nature which is the target for all of this data preparation, it is reasonable to use the scalar values of the angles instead of the more common cross product calculation between neighbouring points in the three-dimensional space. This strategy reduces the calculation of the normal vector image to a simple element-wise matrix subtraction. To compute the horizontal component,

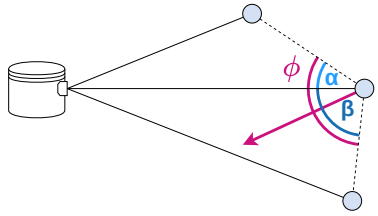


Fig. 5.6.: Relationship of angles between lines connecting adjacent lidar measurement points and the line from the respective point to the sensor origin for calculating the a normal vector component. With $\phi = \alpha + \beta$, the angle bisector can be determined as $\frac{\phi}{2} = \frac{\alpha+\beta}{2}$.

the neighbouring points left and right of the point in question are used. For the vertical component, the neighbour above and below are considered respectively. Using just one neighbouring point here would allow for a potentially larger error. This can be implemented efficiently by creating a total of four copies of the range image, two per component, with shifts of one pixel in each direction respectively. With a range image R and two shifted images S_l and S_r, the horizontal component image NC_H can be calculated as

$$\begin{aligned} A &= \text{atan2}(S_l \cdot \sin(\alpha),\ R - S_l \cdot \cos(\alpha)) \\ B &= \text{atan2}(S_r \cdot \sin(\alpha),\ R - S_r \cdot \cos(\alpha)) \\ NC_H &= A - B \end{aligned} \tag{5.4}$$

given the horizontal resolution α of the lidar sensor in radians. The vertical normal component image can be determined accordingly. While this implementation requires a bit more memory, it can be computed very efficiently.
In result, these add two more possible image layers for image classification. An exemplary representation of both component images can be seen in figure 5.7, with an additional combined view for illustration purposes.

As this classification approach is build on top of segmented object instances from the clustering step, this facilitates an additional advantage. Although it would be conceivable to design a larger end-to-end network that learns to both detect and classify objects from full frames of these image representations, already having instances detected is a great advantage.
Since a clustering algorithm produces pointwise results, this information can be used beyond simply cropping a rectangular image patch from the lidar frames. Each point that has been labelled as belonging to a cluster can be identified in the image plane. If an image patch is then created to become the input of a classification algorithm, the width and height can be derived from the selected points. More importantly,

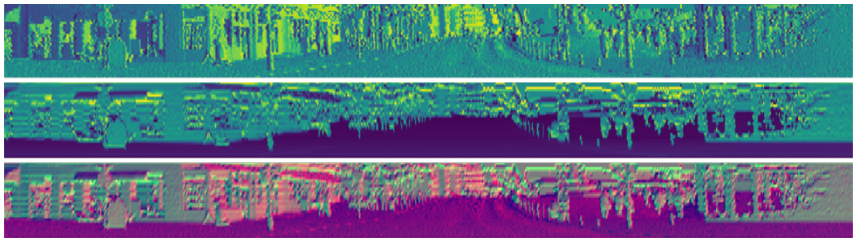


Fig. 5.7.: Visualisation of the horizontal *(top)* and vertical *(middle)* normal vector component lidar image, as well as a combined view of both components, using the red colour channel for the horizontal component and the green channel for the vertical, for representative purpose. Although it may be difficult for the layman to identify objects such as house walls, posts and people in this illustration of a city scene, the division of horizontal and vertical surfaces in the two pictures is clear. The absence of the entire ground, including the roadway, in the vertical component image makes this particularly obvious.

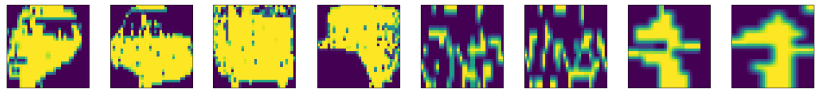


Fig. 5.8.: Exemplary image patches of object instance masks for different classes: car, truck, bicycle, person (*two each from left to right*).

they can be used to mask out any possible background in this rectangle putting more emphasis on the object. This should improve classification accuracy, as the shape of different objects in the automotive context is very distinctive. A few examples of these instance masks applied to object image patches can be seen in figure 5.8.

5.3.2 Object Statistics

As explained in the preceding section, there is a clear advantage of having object instances provided by a clustering algorithm before targeting classification. The again, it also has a downside since clustering itself is class-agnostic. All object instances above the ground plane will be segmented, independent of whether they belong to a supported class relevant for automotive application. Hence, classifier will be presented with a majority of "None" objects along the roadside. For example; walls of shops and houses can be confused with trucks, street lights or other poles can resemble pedestrians and in some cases, cars might even be confused with larger bushes.

To remedy this situation, additional handcrafted features can be created. These should be able to convey further information about an object's geometric nature to the neural network. From the clustering output, it is straightforward to provide the number of points belonging to an instance and determine its distance to the sensor origin. For simplicity, the smallest x and y values of points in the respective cluster are selected here and the distance is expressed as Euclidean and pure x- and y-axis lengths. A sense if the object's size is already conveyed in this manner, as the number of reflected points per area decreases with distance due to the fanning out of most lidar sensor's channels. Further, the width, length and height of the object are calculated, completing a static vector with seven values.

While the influence of this statistics vector on the overall classification performance score later revealed itself to be not particularly substantial, its use proved to offer valuable decision support for critical edge cases. Examples of this advantage in correcting both false positives and false negatives in the classification can be seen in figure 5.11.

5.4 A New Architecture for Online Lidar Object Classification

With the underlying meaningful data representation, a suitable starting point for fast lidar object instance classification has been created. To support real-time execution even on CPU, a comparably small convolutional neural network is needed. There is a multitude of specialised works on this topic analysing a variety of optimised calculations, e.g. (Freeman et al., 2018; Howard et al., 2017; Arriaga et al., 2019). In simple terms, they all consider at least one of two important principles. The first being depth-wise separable convolutions. Actually popularised in the deep *Xception* architecture of (Chollet, 2017), these help to save a lot of arithmetic operations by calculating one $n \times n \times 1$ convolution per input channel c_i and expanding to c_o output channels by applying c_o $1 \times 1 \times c_i$ convolutions instead of c_o $n \times n \times c_i$ "full" convolutions[1].

The second principle concerns so called residual connections introduced by (He et al., 2016). By adding the output of a previous layer to the output of the current layer they mitigated the vanishing gradient problem. These occur when in deep

[1]For an exemplary conv. layer with a 3×3 kernel and $c_i = 5$ inputs of size 8×8 and $c_o = 64$ outputs a normal convolution would move $6 \cdot 6$ times, creating $64 \cdot 5 \cdot 3 \cdot 3 \cdot 6 \cdot 6 = 103\,680$ multiplications. A depth-wise sep. conv. would only need $3 \cdot 3 \cdot 5 \cdot 6 \cdot 6 + 64 \cdot 1 \cdot 1 \cdot 5 \cdot 6 \cdot 6 = 13\,140$.

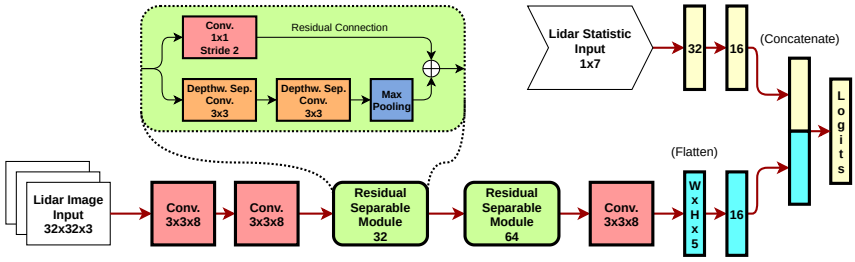


Fig. 5.9.: Architecture of the proposed CNN for fast lidar object instance classification.

layers of large networks the gradient becomes infinitesimally small due to frequent multiplications with weight values smaller than 1. But even in smaller networks they can add helpful non-linearities in a parallel path.

As depicted in figure 5.9, the proposed network architecture for lidar object instance classification consists of two branches. The first one takes the statistics vector of length seven presented in section 5.3.1 as input and comprises two fully connected layers. The second larger branch processes the lidar image representations and is characterised by two residual separable modules in between common convolution layers with 3×3 kernels. Such a module features two depth-wise separable convolutions and maximum pooling in parallel to a residual connection with 1×1 filter kernel. The lidar image patches are expected to have a size of 32×32 as input to the network. This is a very much sufficient resolution, as most widely used lidar sensors offer between 32 and 64 separate channels. In this way, even objects that theoretically occupy the entire height of the scanned area because they are very large or very close to the sensor can be sufficiently resolved.
The output of the last 3×3 conv. layer are five feature maps of size 7×7. With a flatten operation, these are restructured to an vector of length 245 before being fed into a dense layer. Both branches are eventually connected by concatenating their final layers before the output logits.

The complete network architecture has a total of $20\,802$ parameters, and its structure and weights can be stored in a checkpoint file of 398 kBytes in size.
Although it does not make for an entirely fair comparison, state-of-the-art lidar networks which aim to solve basically the same task in an end-to-end fashion have many millions of parameters and need much more memory to store weights accordingly, which is another cost driving factor in addition to their much higher computational requirements.

5.5 Experimental Evaluation

In this section, the newly presented clustering and classification methods will be evaluated concerning both their accuracy on different metrics and speed.

5.5.1 Clustering

To evaluate the performance of the presented clustering method *FLIC*, its quality to accurately segment object instances in a lidar point cloud is measured. The *SemanticKITTI* data set (Behley et al., 2019) is used here to provide object instance segmentation ground truth. It is an extension to the well-known *KITTI* data set (Geiger et al., 2012), a collection of street scenes recorded with different automotive sensors, including a 360° rotating lidar with 64 channels, and offering annotations for a variety of tasks. Its odometry challenge is enhanced with semantic and instance-wise labels for every lidar measurement by this update.

Two popular measures are used for evaluation of the segmentation quality. The first to be calculated is the Intersection over Union (IoU), sometimes referred to as *Jaccard Index*. It measures how exactly two sets (of points) overlap as a value in the closed interval $[0, 1]$. Given a single ground truth instance A and one cluster B, it is defined as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \;\widehat{=}\; \frac{TP}{TP + FP + FN} \,. \tag{5.5}$$

representing the area/volume of overlap divided by the area/volume of union. This corresponds to the number of true positive points (TP) divided by the TP plus the false positive points (FP) and the missing false negative (FN) points.

The second metric is the precision measure P. It is based on the index and shows how many instances are matched with an IoU of at least x. Therefore, it can be defined as

$$P_x = \frac{1}{N} \sum_{n=0}^{N} \sum_{m=0}^{M} a_{n,m} \quad \text{with } a_{n,m} = \begin{cases} 1, \text{ if } J(n,m) >= x, \\ 0, \text{ else} \end{cases} \tag{5.6}$$

for N instances and M clusters in which each instance and cluster are matched via $J(n, m)$. Please note, that due to the definition of the *Jaccard Index* only one cluster can match a ground truth instance with an $IoU > 0.5$.

Method	IoU_μ	P_μ	$P_{0.5}$	$P_{0.95}$	f_μ
Bogoslavskyi et al.	73.93	59.31	83.75	13.18	$\approx$ 152 Hz
FLIC	76.20	63.73	84.30	22.03	$\approx$ **165 Hz**
FLIC (*1 MC*)	77.97	66.68	85.60	27.19	$\approx$ 104 Hz
FLIC (*6 MC*)	81.14	71.92	88.25	36.05	$\approx$ 48 Hz
FLIC (*14 MC*)	**84.25**	74.68	**89.75**	40.63	$\approx$ 26 Hz
DBSCAN	76.21	**76.50**	81.54	**69.25**	$\approx$ 3 Hz

Tab. 5.1.: Comparison of the segmentation quality on *SemanticKITTI* using the mean intersection over union and the precision averaged over all bins and for the bins with an IoU of 0.5 and 0.95. All values are given in percent. The mean processing frequency of the lidar frames is also listed for the respective algorithm.

For comparison, the well-known *DBSCAN* algorithm (Ester et al., 1996) and the comparable state of the art approach of (Bogoslavskyi and Stachniss, 2016) are tested on the same task as the proposed *FLIC*. All listed methods are evaluated on a Intel Core i7-6820HQ CPU @ 2.70 GHz.

For each ground truth object in the data set that consist of at least 100 lidar points, every algorithm's cluster output with the highest IoU is selected. All results are measured instance-wise, so that if two ground truth instances are represented by only one cluster, only the object with the higher IoU is counted, while the second object is marked as not found. For all methods the same ground plane extraction as described in section 5.2 is applied. The intersection over union is calculated for every instance in a lidar frame and averaged over all 2500 frames in the data set to form the mean IoU_μ. For the precision P, ten bins of point-wise overlap of the ground truth and the clusters are defined ranging from an IoU of 0.5 to 0.95 in steps of 0.05. The precision of all bins is averaged into one single metric score P_μ. The precision is also separately reported for the bins with the lowest ($P_{0.5}$) and highest IoU ($P_{0.95}$) in table 5.1. This also lists the average time for clustering one frame as the possible processing frequency f_μ.

Even when compared to the sophisticated and computationally intensive three-dimensional clustering of the *DBSCAN* algorithm, the proposed clustering method achieves a respectable performance. For the mean IoU the segmentation quality is precisely on par, even without the use of map connections. If these are added up 14 ones, *FLIC* becomes the best performing method for this metric by a margin. When it comes to precision, the slow but precise *DBSCAN* still has a visible advantage, most noticeably in the bin with the highest IoU $P_{0.95}$. Yet, its average processing time makes it unusable for real-time application.

The method presented by Bogoslavskyi and Stachniss achieves a very fast runtime, which can only be caught up by the base implementation of *FLIC* without map connections, but lags behind in all other metrics. In the $P_{0.95}$ bracket, it shows an all the more pronounced drop in performance. Even with the maximum of map connections tested here, the presented approach would still be able to run faster than the recording frequencies of most lidar sensors which are typically between 10 and 20 Hz.

5.5.2 Classification

In the following, several different aspects of the proposed classification of segmented lidar object instances are analysed. Before applying known metrics on public data, however, the network architecture and image representation must first be evaluated.

Architecture

To assess its accuracy performance, the CNN architecture proposed in section 5.4 is compared with two plain CNNs that both only apply classic convolutional layers, compared to the residual depth-wise separable modules utilised in the former. One of the plain CNNs simply replaces each of these special blocks with a 3×3 convolutional layer with the same number of channels, 32 and 64 respectively. To keep the sizes of the intermediate feature maps the same as they would be in the residual blocks, maximum pooling layers are added to halve their width and height. The other straightforward CNN tries to resemble a deeper architecture by using two times two 3×3 convolutions with 16 channels in this place.

A private data set is created for this experiment from a database of several lidar point clouds. These were recorded with a 40-channel *Hesai Pandora* sensor which is rotating 360° on top of a vehicle. Object instances were originally annotated as three-dimensional bounding boxes and can thus be transferred in the image representation. These were otherwise created as described in section 5.3 and a statistics vector is also determined for every instance. As object classes cars, trucks, bicycles (including motorbikes) and pedestrians are considered. For a fifth "none" class, samples generated from areas that do not have any overlapping labels from the original annotations. A total of $\sim 46\,000$ training and $\sim 8\,000$ test samples are created in this way.

	Num. Params.	FLOPS	Test Acc.
Proposed	20 802	47 109	0.906
Plain Same	29 066	124 263	0.852
Plain Deep	36 970	167 903	0.893

Tab. 5.2.: Comparison of three CNN architectures for lidar image patch classification in terms of their number of parameters, floating point operations to process one input sample and their accuracy on the created private test set.

Table 5.2 shows the computation properties of the three networks and outcome of training on this data set. While the plain CNN imitating a similar channel structure as the proposed network already has nearly 50% more parameters and requires two and a half times as many floating point operations (FLOPS) to process one input sample it clearly exhibits the inferior performance on the test set. The alternative CNN with the deep conv. layer structure nearly matches the accuracy of the presented architecture, it raises the number of parameters and FLOPS even further.

Overall, it can be seen that in this case too, as already shown in prior literature with other applications, residual depth-wise separable modules can bring great advantages in terms of computing effort to performance ratio.

Image Configuration

Now that a private data set has been created in the form of the representation introduced in section 5.3, it is sensible to analyse the composition of the lidar image patches. On the one hand, to identify possible redundant information in these seven layers and on the other to reduce the number of input layers which later need to be processed. Therefore, an ablation experiment is conducted, testing all possible input combinations (see table 5.3).The binary mask derived from the clustered instance output is not generated as a separate layer but applied to all other ones masking out non-related points with zeros.

The analysis shows that one is able to efficiently select only three layers, namely the intensity values and our horizontal and vertical normal vector component representations, and maintain performance with only a minimal accuracy decrease compared to using all seven of them. Consequently, an image input of the size $32 \times 32 \times 3$ is used in the further course.

Channel Configuration							Test Acc.
X	Y	Z	I	D	HNV	VNV	
							0.835
•	•	•					0.875
•	•	•	•				0.892
•	•	•		•			0.879
•	•	•	•	•			0.895
•	•	•			•	•	0.882
•	•	•	•		•	•	**0.900**
•	•	•		•	•	•	0.890
•	•	•	•	•	•	•	**0.902**
			•				0.887
				•			0.861
			•	•			0.891
			•		•	•	0.**896**
				•	•	•	0.881
			•	•	•	•	0.894
					•	•	0.873

Tab. 5.3.: Results of the input channel ablation experiment. Configuration options include the Cartesian coordinates (X, Y, Z), intensity (I) and depth (D), as well as the horizontal and vertical normal vector component images (HNV, VNV). A binary mask from the points belonging to the clustered instance is applied at all times.

Semantic Segmentation and Object Detection

After proving the advantages of the proposed network architecture and determining a selection for the lidar image input layers, the presented approach is evaluated on public data. The *SemanticKITTI* data set, which was already introduced in a preceding section, is utilised again for this purpose.
While good detection/classification rates are important, the proposed CNN, just like the lidar image clustering, was developed with a focus on real-time capability for CPU-based platforms. Hence, it is not sensible to expect new benchmark high scores and compete directly with magnitudes larger end-to-end lidar networks. Given the strong computational limitations, it can be regarded as sufficient to aspire for a reliable output.

The semantically segmented point clouds in *SemanticKITTI*, as well as the additional instance labels in the dataset, allow for multiple pointwise evaluation approaches. The first is semantic segmentation for which *FLIC* is employed to provide separate class-agnostic object instances from the lidar data. These clusters are then brought into the previously described image and object statistics vector representations and are processed by the classification network.

Input Method	None	Car	Truck	Bike	Pedestrian
FLIC Instances	0.954	0.750	0.472	0.265	0.282
GT Instances	0.994	0.926	0.732	0.525	0.558

Tab. 5.4.: Semantic segmentation results in the intersection over union metric (see formula (5.5)), using either clustered instances from the *FLIC* or ground truth instances as input for the classification for each of the five classes created.

As with the created private data set before, to target is to classify objects into five general automotive classes: “Cars”, “Trucks”, “Pedestrians”, “Bikes” and the “None” class, which embodies all static background classes such as road surface, buildings and vegetation. To achieve this mapping, the *SemanticKITTI* classes “Bicycle”, “Bicyclist”, “Motorcycle” and “Motorcyclist” are combined to “Bike”, as well as “Truck”, “Bus”, “On-Rails” and “Other-Vehicle” to “Trucks”. The classification network has been trained with the annotated point clouds of the available training logs. As suggested in the (Behley and al., 2019) documentation, the 8th log is kept separate for validation.

Table 5.4 shows the results for the class-wise semantic segmentation intersection over union (IoU) metric (see formula (5.5)) of the combined approach of clustering the point cloud and classifying each cluster separately. This metric provides a good impression of pointwise segmentation quality, since the correct predictions and both types of incorrect predictions, false positives and false negatives respectively, for each point are included in the equation.

The score of this metric is computed for two approaches. The first approach is the classification of clustered instances from the *FLIC* as an end-to-end pipeline on point cloud data to show the performance of the proposed method on unseen samples. The second approach applies the classification directly on the annotated ground truth instances in the data set. This allows for a comparison, on how the performance is influenced by the grade of the provided object instances. As the results show, performance of the semantic segmentation directly depends on the quality of the given object clusters and the result could be improved, if a clustering algorithm that is as accurate as the annotated ground truth while offering real-time processing would exist.

The second evaluation method is meant to assess the performance of object detection. For this, as previously introduced, the average precision P_μ is calculated and averaged over ten bins of point-wise overlap of the ground truth and the clusters

Input Method	P_μ	$P_{0.5}$	$P_{0.75}$	$P_{0.95}$
FLIC Instances	0.407	0.441	0.419	0.314
GT Instances	0.554	-	-	-

Tab. 5.5.: Object detection results as average and IoU bin-wise precision on clustered instances and provided ground truth instances.

ranging from an IoU of 0.5 to 0.95 in steps of 0.05. Additionally, the precision for the overlap values of 0.5, 0.75 and 0.95 are listed, in which the evaluation is restricted to objects at or above the denoted IoU. In contrast to the previous object detection evaluation performed solely on cluster instances, the calculation is now no longer class-agnostic. Therefore all matched instances that are not attributed to the right class count against the precision. As only four positive classes out of the 28 classes from *SemanticKITTI* are supported, this visibly impairs the score as seen in table 5.5. This becomes even more clear, when using the ground truth instances as input as these are perfectly matched and the score still only reaches approximately 55%.

5.5.3 Panoptic Segmentation

By combining the fast lidar image clustering, providing class-less separation of object clusters and background, with the proposed efficient object classification, it is possible to perform the task of panoptic segmentation. This term was coined by Kirillov et al. in their work of the same name (Kirillov et al., 2019). According to the authors, this task "unifies the typically distinct tasks of semantic segmentation (assign a class label to each pixel) and instance segmentation (detect and segment each object instance)". The clustered instances provide sufficient information on the separation of objects, while the subsequent classification yields the prediction.

In panoptic segmentation the differentiation between *Stuff* ie. background and *Things*, in this use case active road users, is as important as the separation of *Things* among themselves. After separation into background and clusters and classification of the latter, the predicted class labels are used to remove the instance labels from clusters which are not part of the *Things*, in this case all "None" labels such as utility boxes, road signs and vegetation.

The *SemanticKITTI* data set also added a panoptic segmentation benchmark in 2020 to validate this task on its content (Behley et al., 2020). This challenge uses the panoptic quality (PQ), originally proposed by (Kirillov et al., 2019), averaged over

all classes C, as used by Porzi et al. (Porzi et al., 2019), on the whole test set. For a single class it is defined as

$$PQ = \frac{\sum_{(\mathcal{S},\hat{\mathcal{S}})\in TP} IoU(\mathcal{S},\hat{\mathcal{S}})}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|} \tag{5.7}$$

and is composed from the segmentation quality (SQ) and the recognition quality (RQ) as follows

$$PQ = \underbrace{\frac{\sum_{(p,g)\in TP} IoU(p,g)}{|TP|}}_{\text{segmentation quality (SQ)}} \times \underbrace{\frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}}_{\text{recognition quality (RQ)}}, \tag{5.8}$$

with true positive (TP), false positive (FP) and false negative (FN) classifications.

At the time of the original publication, the presented method was the only approach to be submitted for the panoptic segmentation benchmark. Now at the time of composing this work, several deep learning approaches were entered offering a better performance. This does not come as a surprise, since this method of combining the *FLIC* with a dedicated meaningful data representation and a fast classification network architecture is as previously shown limited to only 4 of the data set's classes and will not yield competitive results summed over all classes. Nevertheless, to the author's knowledge, it is still the only published approach to enable panoptic segmentation in real-time on CPU.

The results in table 5.6 are therefore limited to the four classes described in detail above. Due to the reduced class mapping explained previously, the performance on "Truck" and "Bike" is additionally impaired, as the reported official results for the challenge are evaluated on the full class set. For all four classes, the segmentation quality yields good to very good results. Apart from "Truck" and "Bike", the recognition quality for "Car" is pleasant, but "Pedestrian" recognition could be better. Then again, *SemanticKITTI* distinguishes humans between "Pedestrian", "Bicyclist" and "Motorcyclist" which further deteriorates the score.

5.5.4 Timing

The network architecture illustrated in figure 5.9 is implemented in Python with *Tensorflow* (Abadi et al., 2015), devoid of any further customisation or optimisation. For 100 input instances of object proposals from the point cloud, which can be considered to be representative of a residential area to inner-city scene, inference

Class	PQ	SQ	RQ	IoU
Car	0.754	0.866	0.87	0.792
Truck	0.0534	0.888	0.0602	0.0371
Bike	0.0822	0.723	0.114	0.0462
Pedestrian	0.377	0.905	0.417	0.161

Tab. 5.6.: Panoptic segmentation results as panoptic quality (PQ), segmentation quality (SQ), recognition quality (RQ) and IoU using classification of clustered instances.

time on an Intel i7-6820HQ laptop CPU @ 2.70 GHz is ~ 32ms. If execution is limited to only two threads, resembling a small embedded processor, this value rises to ~ 71ms.

Timing measurements are often not published alongside high-performing publications. In exceptional cases, they are reported on powerful GPUs. To get a rough estimate about the runtime of state-of-the-art networks, the inference of one point cloud from *SemanticKITTI* with a network closely comparable to *PointPillars* (Lang et al., 2019) is timed. Doing so took ~ 1.86s, which is nearly 60 times slower than the presented network, on the same CPU.

5.5.5 Conclusion

The presented holistic approach of combining fast clustering on two-dimensional mappings of lidar sensor data with a dedicated meaningful data representation for segmented object instances and a fast classification network architecture was evaluated extensively in the preceding sections.

While facilitating real-time computation on CPU, it is capable of providing good results in detecting and classifying relevant road user classes on many metrics. Through evaluation on public data it was shown, that it achieves good performances on automotive lidar semantic segmentation and object detection tasks, while being orders of magnitude faster to compute than current state-of-the-art approaches. The component-wise decomposed normal vector image, instance masks and selected statistics make for a efficient data representation to be exchanged between the clustering and classification algorithm.

Through the combined use of instance segmentation and classification of these separated objects, the output can further be regarded as a panoptic segmentation of street scenes. There is only little previous work on this topic in general and to the author's knowledge, this is the first method which can accomplish this task in real time on CPU.

Finally, figure 5.10 shows the visualisation of a lidar point cloud recorded with a 40-channel *Hesai Pandora* sensor on top of a vehicle rotating 360° in a busy urban environment. This representation makes it possible to clearly see how many object instances are clustered in such a scene and how the classification manages to suppress all irrelevant classes and highlight the important road users.

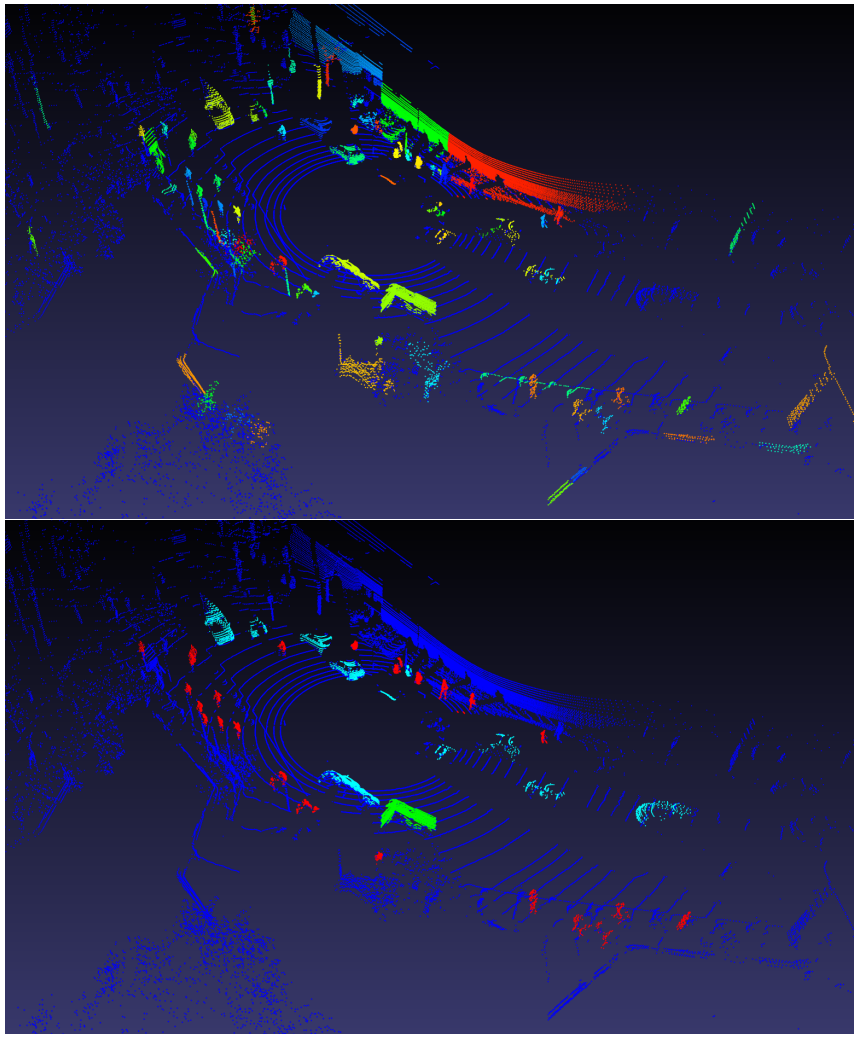


Fig. 5.10.: Oblique view of the Lidar point cloud of a street scene with coloured highlighting of the clustered object instances *(top)* and their classification *(bottom)*. Please note that the clustering output is agnostic to object classes and colours are randomised and used to differentiate instances. In the classification, pedestrians are shown in red, cars in light blue and trucks in light green.

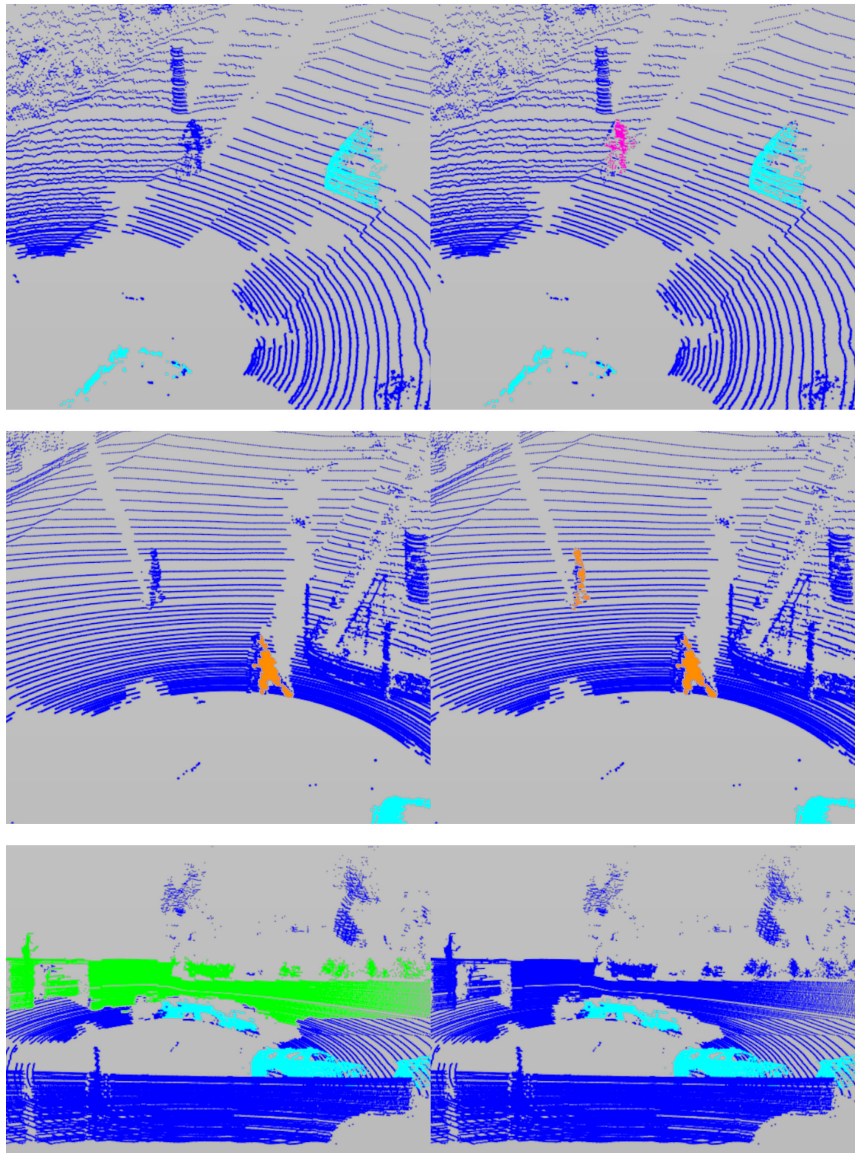


Fig. 5.11.: Influence of the additional statistic vector described in 5.3. The *top left* and *middle left* figure show examples for false negatives and the *bottom left* figure for false positives respectively. The additional statistic vector prevents some of these errors *(right side)*.

6 Aspects of Pedestrian Feature Extraction

The mere detection of pedestrians is an insufficient level of information for the development of models and systems towards autonomous driving. While the ability to locate people in an image, e.g. by specifying a bounding box, is still a fundamental necessity, more features need to be extracted to gain a real understanding. Will the person cross the road, is he or she walking or standing still and possibly making a gesture directed at the oncoming ego-vehicle? These are the questions, which eventually need to be answered to facilitate path planning or de- and acceleration scheduling.
This chapter highlights two features which can play an important role for the development of such algorithms. Section 6.1 is concerned with the detection of pedestrian body keypoints. A sequence of steps is presented for processing camera and lidar data to generate a representation as input for an efficient machine learning network. A few remarks to optimise a straightforward method for pedestrian awareness state detection are given in section 6.2.

6.1 A New Approach for Pedestrian Body Keypoint Detection

For the object-specific analysis for the understanding of weaker road users like pedestrians, the detection of body keypoints and the subsequent derivation of a skeleton model is an important building block. It allows for several subsequent analyses, such as alternating movements of the legs indicating walking, raising of an arm to make a gesture or the direction of gaze.

Like with many other fields of machine learning, there is an nearly unmanageable number of existing publications[1]. At this stage, algorithms for keypoint detection can be divided into two main categories; those that first detect individual instances of

[1] The interested reader is invited to take a look at the very sophisticated keypoint detection literature survey of (Zheng et al., 2020), in which they cite not less than 299 references.

people and then estimate a set of keypoints in this predefined framework/bounding box (called *top-down* methods), and those that determine presumed keypoints on the entire image and then combine these into individual skeletons in a second step. The latter are often referred to as *bottom-up* approaches and popular works include those of (Cao et al., 2021; Fang et al., 2017) and (He et al., 2017). These generally show a very good performance and can learn a potentially deeper understanding of important points in context, but involve more computational effort on average. This is due to the fact that much deeper neural networks are required, not only because of the larger input size, but due to the complexer task of assigning detected keypoint to individual persons. While the results generated with these algorithms can be impressive on benchmark data using high-resolution photographs and video material in mostly well-lit environments, they do not necessarily translate all that well to automotive camera sensor data. Here, cameras often have a lower resolution, where a more distant pedestrian might only be represented by a few dozen pixels, sometimes don't even provide colour information and captured scenes will have diverse lighting conditions.

Not only because of this, but also from a systemic point of view, a top-down approach based on already detected object boxes is better suited for an application in the vehicle in an automotive context. Object detection, including pedestrians, can be considered available as it is a necessity for many driver assistance systems in a modern car and often performed with several different sensors and made robust through fusion and tracking. Therefore, letting the network first learn the task of detecting the person can be regarded as an overhead and needlessly added load for the scarce computation resources in the vehicle. This approach to keypoint estimation also allows for easier and faster later adaptations and a more flexible design of the overall system in terms of the order and frequency of execution of individual algorithmic building blocks.

6.1.1 Network Architecture

Assuming an image patch representing the area of a bounding box containing a pedestrian in a camera frame as input, a neural network to generate a set of keypoints could take a structure as depicted in figure 6.1. Such an encoder-decoder shape, often referred to as *autoencoder* (Hinton and Salakhutdinov, 2006; Rumelhart et al., 1986b), offers two helpful properties in this context. Firstly, it automatically encodes a representation of a given input, hence the name. Secondly, from this internal encoding it then creates an output which has the same width and height as the given input. While this can be achieved with different operations, transposed

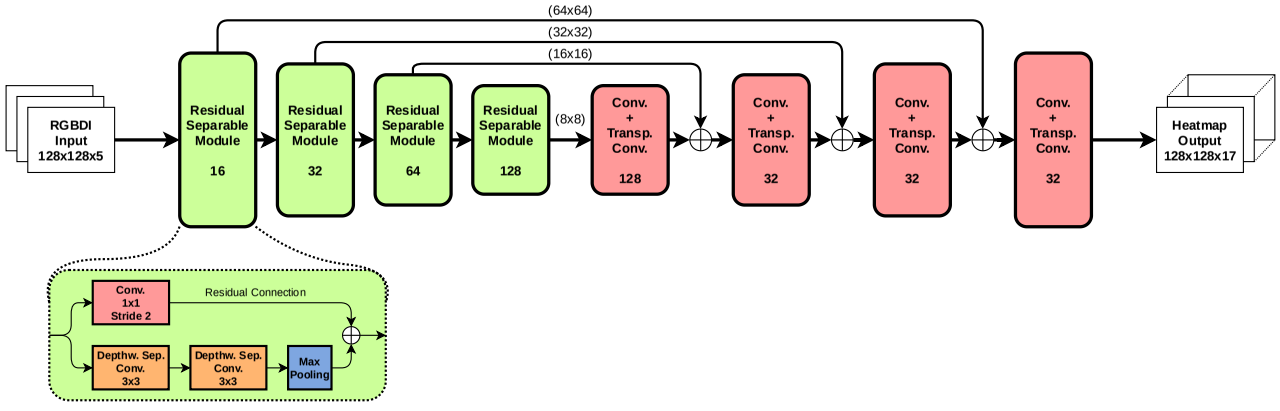


Fig. 6.1.: Encoder-decoder CNN structure to generate heatmaps which estimate keypoints on image patches.

convolutions, sometimes incorrectly dubbed *deconvolutions*, are used here. This layer applies the convolution operation to a dilated and padded version of its input to generate an upsampled output (a graphical visualisation of this concept can be seen in appendix A.1).

The architecture also makes use of *skip connections* to concatenate feature maps from the contracting path with feature maps of the same resolution of the expanding path (Huang et al., 2017). They provide an alternative path for the gradient to mitigate a possible "vanishing" gradient problem in deep layers due to a long chain of multiplications with values smaller than one (cf. section 5.4). Especially with this form of adding "skipped" information by concatenation, it also gives the network the opportunity to reuse the features from the downwards path.

6.1.2 Data Generation

As the objective for this development was to offer a fast yet robust pedestrian keypoint detection, it is sensible to leverage the possibilities of the sensor equipment of a modern vehicle beyond just a camera image. Lidar data is a very useful option for that. In this context it could offer helpful assistance in two ways. First, by cutting out the points actually reflected from the respective pedestrian's body, it can act as a contour mask making it easier for the neural network to separate it from the background. The depth information can also convey a sense of three-dimensional perspective, which could mitigate problems with uncertainty of whether a certain limb is in the fore- or background.

Using a database of 668 logs of street scenes recorded with a *Hesai Pandora* (Hesai, 2018) mounted to the roof of a car, a data set is generated for training. This sensor offers a 360° rotating 40-channel lidar and different cameras, of which a front-facing RGB-camera with an horizontal angle of view of 52° is used. Figure 6.2 shows an example frame with the respective lidar points overlayed.
In order to generate training samples of individual pedestrians for keypoint detection from this data, the following steps are carried out:

1. All camera frames are processed with the state of the art detection and classification algorithm *YOLOv3* (Redmon and Farhadi, 2018), which at this point is not further specified (cf. section 7.1.2). Every detection that assigned with the class "person" is considered, but only included for further steps if it meets certain filter criteria. These include the confidence of the detection algorithm, where a score of 0.9 or higher must be reached, and a threshold for

the size of the bounding box in pixels. A valid detection must be at least 40 pixels wide and 80 pixels tall.

2. Based on the rectangle describing the detection in the camera image, a frustum can be described in the three-dimensional point cloud if the transformation to the lidar sensor is known. This cone-shaped section of the lidar space already correctly delimits the width and height of the detected person, but the depth must be delimited in relation to the sensor origin, as this otherwise encompasses the entire range and background behind the person would be included, rendering the idea of masking out its shape unsuccessful. For this purpose, a simple statistical histogram of the number of points along the depth can be used in conjunction with a "non-maximum suppression". In case of later integration into an overall system, the available information from the lidar object clustering (cf. section 5.2) can be used. Exemplary results of the isolation of person instances in the lidar space can be seen in figure 6.3.

3. The by the person detection delimited point cloud is transferred into two two-dimensional image representations, one for the depth and one for the reflection intensity information. Both are normalised, as especially the depth values would otherwise cover only a very small value range. In addition, a classic image processing technique, dilation, is applied in vertical direction to condense the sparse lidar points, especially in areas where the vertical angle between the lidar channels is larger, into a closed form (see 6.3 right).

4. As a manual annotation would have exceeded a reasonable time frame, the found person instances are automatically annotated by the very deep state of the art algorithm *AlphaPose* (Fang et al., 2017). Confidence filters are also applied here to sort out samples for which the network is unsure about how to label them. The structure of the annotated keypoints follows the format proposed by (Lin et al., 2014). It denotes the classes the following order: “nose”, “left eye”, “right eye”, “left ear”, “right ear”, “left shoulder”, “right shoulder”, “left elbow”, “right elbow”, “left wrist”, “right wrist”, “left hip”, “right hip”, “left knee”, “right knee”, “left ankle”, “right ankle”.

5. Finally, the data is prepared for the training of the neural network. A five-channel input image is composed of the camera’s colour image, as well as the lidar depth and intensity images. This combination is scaled to a size of 128×128 pixels while maintaining the original aspect ratio and, if necessary, padded with noise. The keypoints labels, given in image coordinates, are transformed into heatmaps, one for every point. For each of them a single-channel image is created with the same size as the input. For the pixel-position

of the respective point, it has a value of 1 and the adjacent pixels take values according to a two-dimensional normal with a very small variance. This later helps the learning process, as the network is not directly "penalised" by the loss function for an estimate that is off by only one or two pixels. All other heatmap pixels are set to zero. An example of the input and the heatmap labels can be seen in figure 6.5a

In this way, a total of approximately $250\,000$ samples are generated. They are divided into a training set and a validation set in a ratio of 80:20.

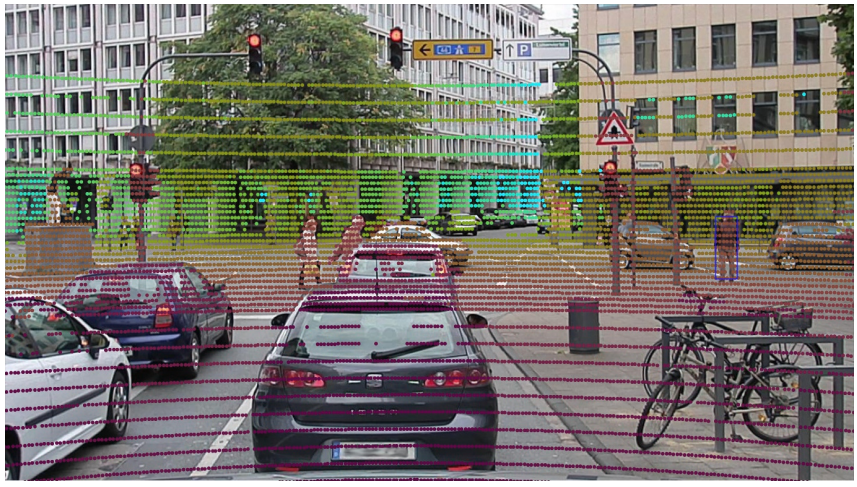


Fig. 6.2.: Camera frame overlayed with a projection of the respective lidar point cloud section.

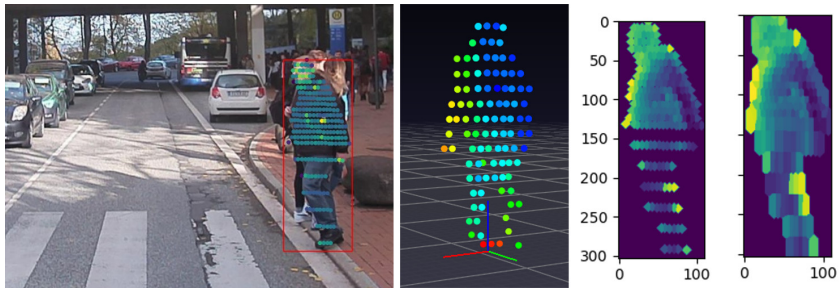


Fig. 6.3.: *Left*: Person detected in the camera image with body limited lidar points projected on top. *Middle*: Section of a point cloud limited to a body instance. *Right*: Body depth image before and after applying dilation in vertical direction.

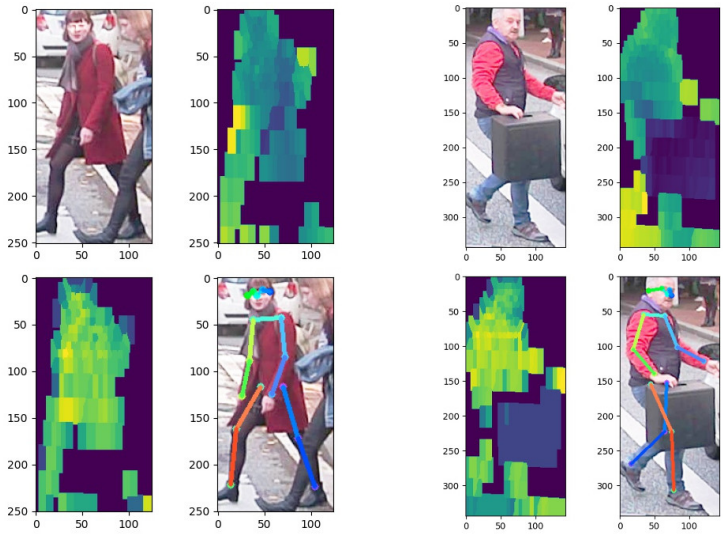
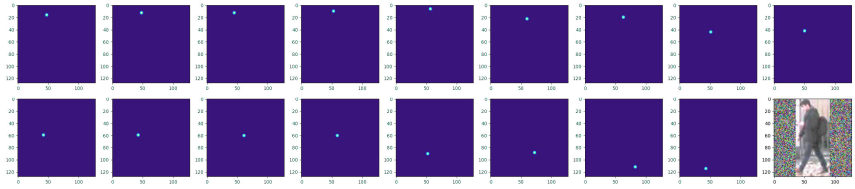
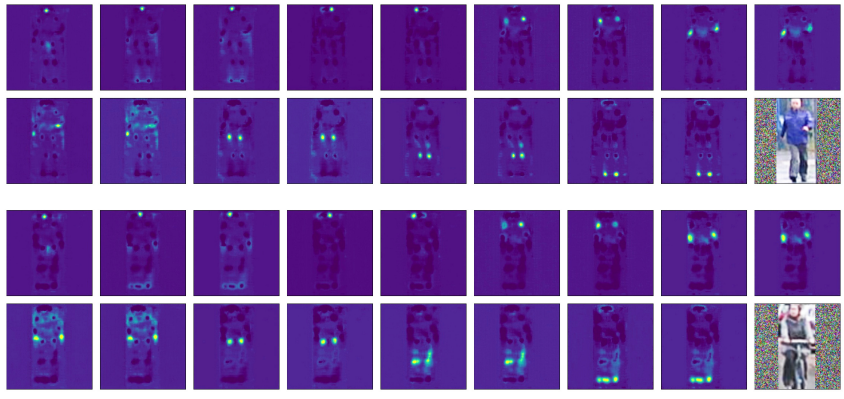


Fig. 6.4.: Two examples for a pedestrian instance represented through a camera image (*top left*), lidar depth (*top right*) and intensity (*bottom left*) images. A skeleton created from connecting predicted body keypoints, is shown in the *bottom right*.



(a) Labels



(b) Network output generated by training with the loss function presented in equation 6.1.

Fig. 6.5.: Visualisation of heatmaps for 17 keypoints with the respective input image patch.

6.1.3 Training and Evaluation

To train the autoencoder network introduced in section 6.1.1, several aspects to improve performance are considered. During training, the input batch data is augmented by optionally vertically flipping the image and/or shifting it by a certain amount of pixel to the left or right and/or rotating it by up to $\pm 15°$. This should enable the algorithm to learn translation invariant features and possibly develop a better understanding of keypoint regions.
Beyond this, the training process uses a learning rate scheduling, where it is halved every time the validation loss was not improved for 10 consecutive epochs.

Calculating the loss during training is in itself straightforward thanks to the heatmap label structure. The label and output prediction matrices have the same shape and can therefore be subtracted from each other. Applying a mean squared error (MSE) would be an obvious choice. However, initial training attempts showed that the network in this way is hardly incentivised to make any useful predictions. Generating noise with a very low amplitude / small random numbers is as good a solution as predicting an actual keypoint, since the relevant area only accounts for a very few of the $128 \cdot 128 = 16384$ pixels per heatmap.

In order to remedy this situation, for each label heatmap Y_i a mask M_i is created. It contains small values, e.g. 10^{-3}, in every cell, except for those where Y_i has values larger than 0.01. There, it is set to 1. This helps to focus the loss calculation on the relevant areas and suppresses other regions by giving them a lower weighting. With labels Y_i and output estimates $\hat{Y}_i$, the loss function can thus be stated as

$$L(Y, \hat{Y}) = \sum_{i=0}^{N} (Y_i - \hat{Y}_i)^2 \cdot M_i \qquad \text{with } M_i(n,m) = \begin{cases} 1, \text{ if } Y_i(n,m) > 10^{-2}, \\ 10^{-3}, \text{ else} \end{cases} \tag{6.1}$$

for $N = 17$ heatmaps. Experiments showed that there is no detectable difference between defining the loss as a residual sum of squares (RSS), as has been done here, or as MSE.

As the examples in figure 6.5b demonstrate, this enables the network to generate output heatmaps which show a clear understanding of body keypoints in the input image. Parts of the head show very distinctive detections, whereas, especially in the case with the cyclist, leg regions seem to evoke a bit more uncertainty. Generally, pairs of body parts show a certain level of ambiguity, as the network unsurprisingly has a difficult time to distinguish between right and left limbs. This is also a problem for a lot of other published approaches.

While a large evaluation is not possible at this stage, if only because, to the author's knowledge, there is no data set comparable to the presented data generation with camera and lidar inputs, training on these samples was analysed to identify certain behaviour characteristics. By tendency, performance profits from a larger input batch size, reducing the validation loss by around ten percent when increased from 8 to 64 ceteris paribus. This is to expected, as a larger number of samples in a badge will stochastically include varied body poses. Conversely, there is hardly any influence if the lidar intensity values are omitted from the input, as the network does not seem to extract any useful information from them. The lidar depth image as such, in contrast, seems to be an important addition improving validation performance by up to fifteen percent.

6.2 A Note on Pedestrian Awareness Detection

An attentive human driver can decide within a fraction of a second, whether a pedestrian in the vicinity of the vehicle will behave in such a way that his intervention is required. He or she is able to subconsciously analyse relevant explicit and implicit behavioural patterns of the person and to draw conclusions for risk assessment. When it comes to transferring these skills to a computer, it is not immediately clear which features are particularly important. However, one aspect that should be included without question is the recognition of the pedestrian's awareness level.

In order to implement this, an algorithm is to classify image data of pedestrians into one of the following four classes:

Aware The pedestrian is facing the vehicle, looking directly at it or its immediate surroundings.

Partially Aware The pedestrian is looking sideways from the vehicles perspective but the face is still visible.

Unaware The pedestrian is looking away from the vehicle in more or less the opposite direction and the face is not visible.

Distracted The pedestrian at a mobile phone or a similar object in his/her hand.

Examples for these classes can be viewed in figure 6.6.

To classify these samples, a small and conventional CNN architecture can be used. It is build from three 3×3 convolution layers interconnected with maximum pooling. The intermediate feature maps are fed into a global average pooling layer to reduce

Fig. 6.6.: Examples for the four awareness state classes. *From left to right*: aware, partially aware, unaware and distracted.

dimensionality and eventually processed by two fully connected layers, the last being the class output logits.

6.2.1 Data and Results

As source for data an internal database containing a variety of logs recorded by a vehicle with a multi-sensor setup is considered. Object annotations, including pedestrians, are available as three-dimensional bounding boxes which can be projected into a camera image. From this a number of image patch samples can be cropped. These are then manually labelled into the four categories described above and a histogram equalisation is applied to mitigate the effect of changing lighting conditions in the recorded scenes a correct exposure.
Eventually, approximately $6\,200$ samples are found for the *aware* class, $10\,000$ for *partially aware* and *unaware* and $7\,000$ for *distracted*. Per class, $1\,500$ samples are kept for validation, while the rest is used for training.

The class-wise training results can be seen in table 6.1 for three different metrics. If *precision* is defined as the ration of true positive (TP) classifications and the sum of TP and false positives (FP) and *recall* is the ratio of TP to TP plus false negative (FN) classifications, then the F_1 score is derived as the harmonic mean of both and can be expressed as

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \,.$$

In contrast to the general *accuracy*, which is usually defined as the ratio of all positive classification to the total number of samples, it highlights false classifications and

		Precision	Recall	F_1 Score
Class	Aware	0.83	0.85	0.75
	Partially Aware	0.82	0.86	0.84
	Unaware	0.94	0.88	0.91
	Distracted	0.90	0.87	0.88

Tab. 6.1.: Class-wise metrics for pedestrian awareness classification.

can be helpful in situations where the data set is not balanced. For this case the accuracy for all classes is 0.862, but the F_1 scores for the *aware* and *partially aware* class show a slightly impaired performance. When a confusion matrix is created for the validation set classification results, it shows that several samples of those two classes were mutually misclassified.
If this problem could be remedied by adding additional features, this straightforward approach could otherwise provide valuable pedestrian information to subsequent algorithms to be used in tasks like pedestrian path prediction or action classification.

7 Applications

While nearly all of the methods and systems presented in the preceding chapters where to some extent developed to support endeavours in the field of machine learning-aided autonomous driving in urban areas, this chapter should emphasise their practical usability and give examples of applications.
The combination of some aforementioned functions with additional building blocks for pedestrian localisation and movement prediction into a new and complete system is explained in section 7.1. A framework is presented, describing information flow from raw sensor data through various algorithm blocks towards generating an output useful to subsequent vehicle path planning and risk assessment systems.
Section 7.1.3 describes some additional information on integrating the algorithmic parts into a real time vehicle environment.
Finally, a short overview of additional existing and possible future applications for parts of this work is given in section 7.2.

7.1 Designing a New System for Pedestrian Localisation and Path Prediction

This section describes a new system for localisation, perception and path prediction of pedestrians. Using raw automotive sensor data and contextual maps as input, the system comprises several algorithmic building blocks, which incorporate a number of aspects of methods presented in the previous chapters of this work. As an output, the system can generate information about the current and possible future positions of detected pedestrians in the surroundings of the vehicle. This output can then be used by subsequent systems dealing with tasks like path planning for automated vehicles or risk assessment for an emergency brake assistant.

7.1.1 Context and Requirements

As initially described in chapter 1, various portions of this work were developed against the background of the research project “@City”. It is the current autonomous

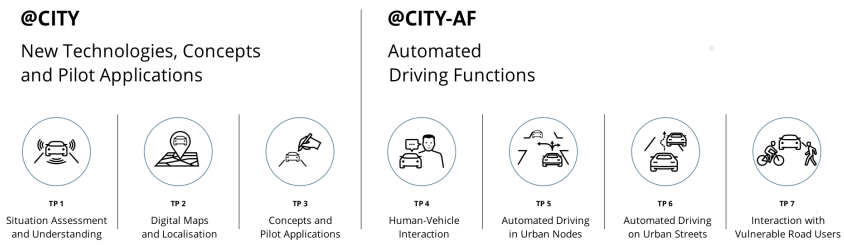


Fig. 7.1.: Overview of the @City project structure.

driving research project funded by the German Federal Ministry for Economic Affairs and Energy based on a decision by the German Bundestag. With representatives from most of the relevant German car manufacturers, top-level suppliers and additional companies and participants from universities as project partners, it is split into the two main categories “@City” and “@City-AF” (for “automatisierte Fahrfunktionen” or automated driving functions). These are then further subdivided into a total of seven subprojects, as shown in figure 7.1, representing a field of concrete functions and methods to be developed.
For this work, the context of subproject “TP7 - Interaction with Vulnerable Road Users” is specifically relevant.

With the automation of vehicles, technical systems are entering as actors in road traffic, which was previously characterised solely by human action. Automated vehicles are faced with the challenge of safely mastering diverse and complex situations with other road users. An essential prerequisite for this is that automated vehicles are able to recognise and understand the behaviour and intentions of other road users. This is particularly true in situations with vulnerable road users, which communicate in road traffic mainly by means of poses and gestures. In order to anticipate the behaviour of vulnerable road users, automated vehicles must be able to reliably recognise and correctly interpret their actions. These goals of the subproject TP7 are further subdivided into four work packages:

The first work package and starting point for a more advanced behavioural analysis is the safe and robust recognition of weaker road users in traffic. Camera sensors and possibly laser scanners and high-resolution radar sensors should be used for this. Through sensor fusion, the information from the different sensor sources is then combined into a consistent representation and tracking should be used to be able to continuously associate detected instances with their respective behaviour and movement record. The detection of weaker road users is to be realised in particular

also by means of learning methods, such as deep neural networks, and should be able to robustly detect weaker road users even under difficult conditions, such as partial occlusion.

One focus of the sub-project is on the detailed analysis of weaker road users. Relevant features for intention and gesture recognition should be defined and methods should be developed and used to extract them from identified object instances. Among those, features like head and body pose, gaze direction or leg position might be used, as well as specific gestures such as hand signals of a cyclist for example.

Often, the context in which weaker road users act is important for the exact interpretation of behaviour or intention. Therefore, in this third focus subject, relevant elements of the traffic environment (e.g. objects, occupancy / open space, kerbs, zebra crossings) must also be recognised as contextual information and interpreted in combination with the gesture features. On the basis of the previously recognised poses and gestures as well as the relevant contextual information, intention recognition and behaviour modelling for weaker road users can then take place.

In the end, the fourth and last work package is concerned with integrating all of the features into a vehicle and optimising their performance for real-time capability. Scenarios for evaluation should be defined and tested both online and offline.

7.1.2 System Overview

The sensor inputs, algorithmic blocks and the output of the proposed system are described below. From object detections in both the camera and lidar domain, which are hereinafter subject to low-level fusion and tracking, pedestrians are filtered. Algorithms for behaviour assessment then extract distinct features. Together with the tracked object detections and a preprocessed representation of map data from the vicinity of the vehicle, these become input to the path prediction in the following step. An overview of the structure is shown in figure 7.2.

Inputs

The first type of sensor required for this system is a camera sensor. While a front facing grey scale camera with a medium wide-angle lens fulfils the minimum requirements, RGB information and a 360° field of view could be beneficial for

a more robust performance, extended perception and larger context information. The output from the camera module should be an rasterised 8-bit image.

Lidar is the other important sensor technology for this system. In theory a wide variety of solid state and rotating lidars could be used. The higher the number of scan lines (or channels) and the wider the area covered, the more detailed the representation of the environment becomes. As an output, the lidar should provide a list containing the position, in Cartesian or cylindrical coordinates, and intensity of all reflected points per channel.

The proposed system further makes use of equipment for position determination. A receiver to obtain information from a type of global navigation satellite system (GNSS), like GPS (global positioning system) for example, would be the preferred choice. It is recommended to use methods like differential GPS (dGPS) and/or ego-motion compensation through data from an inertial measurement unit (IMU), to increase the accuracy of the localisation. A precise position determination is key for this kind of application, as half a metre can make an important difference between locating the vehicle or detected surrounding objects on the sidewalk or the road. Position information should be provided in a spherical geographic coordinate system. The world geodetic system (WGS) in its latest revision as WGS84 can be regarded as the standard (Decker, 1986).

The fourth and last required input is a map providing accurate position description of the road and surrounding objects like poles, vegetation and fences, as well as infrastructure including pedestrian crossings, traffic lights and sidewalks. The format and level of detail of this map can vary greatly and both offline maps and representations generated during operation could be considered. Distinguishing between driving lanes and areas that are safe for pedestrians is the central aspect of this function.

Algorithmic Blocks

In a first step, object detection is performed on camera and lidar sensor inputs. Relevant objects (e.g. pedestrians, cars, trucks, bikes) are detected independently in the image and lidar domain.

For image-based detection an implementation of YOLOv3 (Redmon and Farhadi, 2018) is used. This detector-classifier neural network falls into the category of so called one-stage detectors. In contrast to popular two-stage approaches like Faster R-CNN (Ren et al., 2015), the "you only look once" method produces bounding boxes

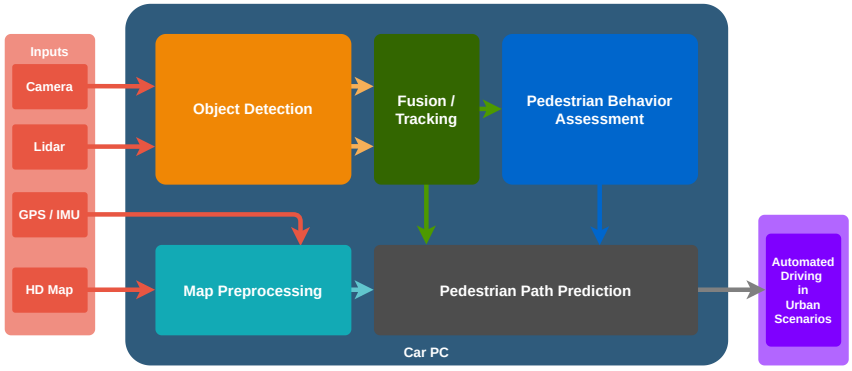


Fig. 7.2.: General overview of the proposed system architecture for pedestrian localisation and path prediction. Inputs shown in red are fed into the software framework running on a computer in the vehicle. The output can then be forwarded to path planning algorithms for automated driving on urban streets.

and assigns class probabilities in a single pass of the image through the network. Simply put, this is achieved by the network dividing the input image into a fixed grid with a cell size of $n \times n$ pixels. Each of those cell then predicts a fixed number of possible object bounding boxes b along with a corresponding confidence score representing the probability that the bounding box contains an object. If the network does not recognise an object in the cell, this score will be 0. Furthermore, each of the grid cells simultaneously predicts a class probability, in this way generating something that could be regarded as a coarse segmentation map of the input image. To combine bounding box proposals and classifications, the conditional probability $P(P_{Class}|P_{Box})$ is calculated and non-maximum suppression (NMS) is applied to threshold duplicate detections.

Following this methodology allows the network to perform object detection tasks in a rapid fashion, since the required prediction can be expressed as one single tensor of size $n \times n \times (b \cdot 5 + c))$m with c being the number of trained object classes and the 5 representing the parameters x_{centre}, y_{centre}, width, height and confidence score for each bounding box proposal. This leads to a short computation time compared with leading two stage approaches all the while retaining a high level of accuracy.

In the context of this system, the YOLOv3 detector is used to create the following output for each object instance in every camera image input to be passed on to subsequent algorithm blocks: (1) a bounding box rectangle in two-dimensional image pixel coordinates $[x_1, y_1, x_2, y_2]$, (2) an object class ID [a defined integer, e.g. 1 for car and 2 for pedestrian etc.], (3) a classification confidence score [a float value

between 0.0 and 1.0] and (4) the corresponding image frame index [an integer or another identifier].

To perform object detection on a lidar point cloud input, the new approach for panoptic segmentation, described in detail in chapter 5, is applied. As it was designed for fast execution, it is a sensible choice for usage in a real-time system. Apart from being an independent and redundant source for object detections, the use of an lidar algorithm has additional benefits for the proposed system. Objects like pedestrians detected in the lidar domain can be easily located in the three-dimensional environment of the vehicle. This is crucial in order to easily determine the distance to the driving lane for example. The dimensions of the respective objects are thus also directly known. A panoptic segmentation algorithm, like the one proposed, also offers the advantage of providing classification information for each single voxel. If transformed to the camera image space, this can be used as an object segmentation mask to outline the object's shape just like an image segmentation would.

To further optimise the runtime of the clustering/classification algorithm when sharing computation resources with other programs, all clusters that have less than 100 points are rejected as described before, but additional restrictions tailored to pedestrian detection are added. In this way all clusters that are wider than two metres, larger than three or smaller than a half metre are roughly filtered out. Finally, the processed point cloud itself is restricted to 60° in front of the vehicle as depicted in figure 7.3.

The format of the output used here to describe detections generated by this method conforms to the following definition: (1) a three-dimensional object bounding box $[c_x, c_y, c_z, l_x, l_y, l_z, r_z]$, described by a centre point c, the expansion l representing width, depth and height and the rotation r around the z-axis, in the vehicle coordinate system (VCS). This is a right-handed coordinate system with x pointing in driving direction (forward), y pointing rightward and z pointing to the ground. The origin of the VCS is defined as the lateral centre of the vehicle's rear axle. The other parts of the output formatting are identical to the one used for camera image detections, being (2) a class ID, (3) a confidence score and (4) a point cloud frame index.

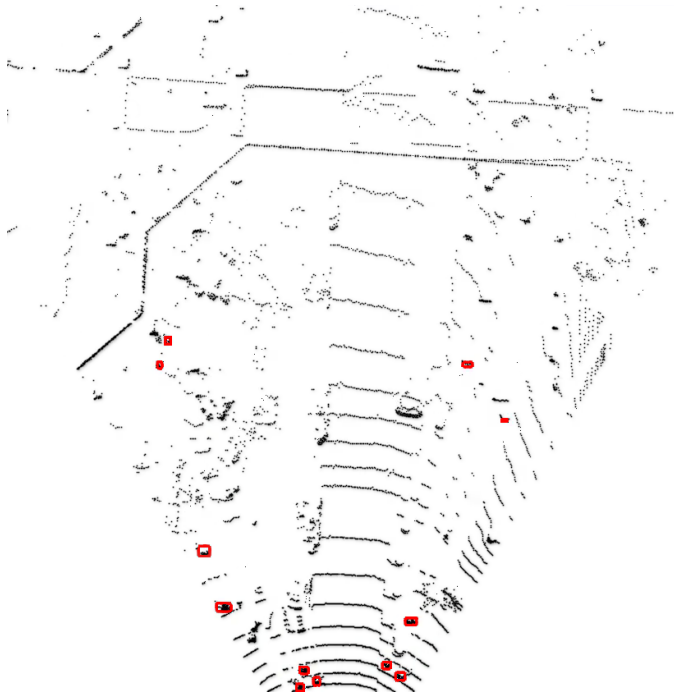


Fig. 7.3.: Two-dimensional bird's eye view of a lidar point cloud restricted to an angle of view of 60° in driving direction. Detected pedestrians are indicated through red rectangles.

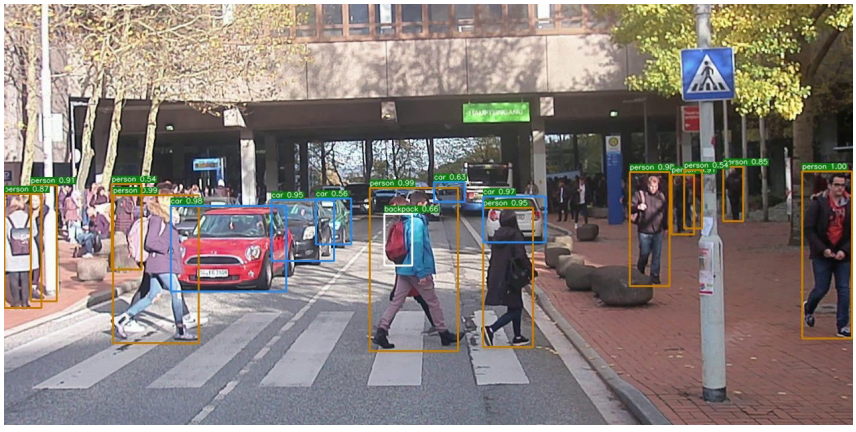


Fig. 7.4.: Example for the output of the YOLOv3 algorithm on a camera image recorded with the @City vehicle. Text on green ground denotes the class prediction and confidence score for a respective bounding box.

Before the individual object detections from the camera and lidar domain can be tracked, they are subject to a low-level fusion. A point w in the world/vehicle coordinate system can be transformed into the camera image plane using affine transformation. Given an intrinsic camera calibration matrix I

$$I = \begin{pmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.1}$$

with a focal length f and a principle point offset o, as well as an extrinsic calibration matrix E

$$E = \left(\begin{array}{c|c} R & \boldsymbol{t} \\ \hline \boldsymbol{0} & 1 \end{array}\right) = \left(\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{7.2}$$

consisting of a rotation matrix R

$$\begin{aligned} R(\alpha,\beta,\gamma) &= \underbrace{\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{R_{yaw}(\alpha)} \underbrace{\begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}}_{R_{pitch}(\beta)} \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}}_{R_{roll}(\gamma)} \\ &= \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix}, \end{aligned} \tag{7.3}$$

describing a roll by γ, followed by a pitch by β and a yaw by α, and a translation by t, a point c in the camera space can be calculated as

$$\begin{pmatrix} c_u \\ c_v \\ c_w \\ 1 \end{pmatrix} = I \cdot E \cdot \begin{pmatrix} w_x \\ w_y \\ w_z \\ 1 \end{pmatrix} \tag{7.4}$$

in homogenous coordinates. To obtain pixel values for a point i on a two-dimensional image plane, the u- and w-component have to be divided by the w-component, which is therefore also referred to as perspective divide:

$$i_u = \frac{c_u}{c_w}\,, \quad i_v = \frac{c_v}{c_w}\,. \tag{7.5}$$

Object tracking is not a subject of this work. Therefore, the following paragraph will only roughly outline the techniques used by other contributors.
Tracking is performed using an interacting multiple model (IMM) algorithm (Bar-Shalom et al., 1989) with two Kalman filters, one with a constant velocity and one with a constant acceleration assumption, and object instances are tracked in global GPS coordinates. The IMM algorithm allows combining state hypotheses from multiple filter models to get a better state estimate of targets with changing dynamics. This can be helpful for the sometimes irregular and spontaneous movement of pedestrians. A likelihood for each filter is computed and together with prior model probabilities and an a priori defined state switching matrix are then used to update the model probabilities. The estimates from each filter model are combined as a weighted sum using the updated model probabilities.
For associating new measurements with existing tracks, a joint probabilistic data association (JPDA) filter (Bar-Shalom et al., 2009) is applied. This *joint* version of the single-target PDA filter is a natural multi-target extension. Rather than making hard assignment decisions, like a nearest neighbour approach would, at each time-step, the probability that each measurement should be assigned to a particular target is used.

Although not yet fully defined at the time of writing, the algorithm block for pedestrian behaviour assessment contains a method for body key point detection (as described in section 6.1) and awareness recognition (see section 6.2), as well as action recognition in the future.
As previously described, body key point detection can benefit from the addition of a segmented lidar image representation to the input. In this system, such a mask can directly be derived from the instances provided by the lidar clustering step. The camera image patch showing the pedestrian is cropped from the bounding box detection through the YOLO algorithm. This same patch is also used as input to the awareness classification.

Both of these distinct features, body key points and awareness state, should in the future be a basis of data for a subsequent action classification. In the context of autonomous driving, especially in crossing scenarios, understanding pedestrian

behaviour becomes essential. The intentions of pedestrians can be ascertained based on either explicit socially understood gestures or on implicit changes in their body language. Some strong implicit indicators are movement of the head in the direction of the driver and movement of a leg onto the road. Traditional approaches like tracking algorithms which use only pedestrian dynamics will not be able to understand some pedestrians' intentions such as wanting to cross the road in time. Hence, models which are based on recognition of implicit gestures as the first step are necessary in order for autonomous vehicles to be able to accurately understand human behaviour and make time-constrained decisions. Understanding human behaviour usually means recognising actions being performed. When looking at pedestrians, this can either mean recognising coarser actions, such as crossing the road, or fine-grained actions such as the deceleration or stopping motion.

A first simplified approach could be the classification of pedestrian perception time series data into walking and standing. Although this is unquestionably already evident from the tracked detections in relation to the ego-vehicle in the larger context of the overall system, in this way a suitable method and optimised input feature selection could be identified. In a straightforward manner, one could create an $n \times 2k$ input vector, representing n time steps and xy-coordinates of k key points. This could be fed into a recurrent neural network like the LSTM (long short-term memory) algorithm (Hochreiter and Schmidhuber, 1997) to then predict.

A more sophisticated concept would be to introduce physical constraints into a neural network. While this is classically done by introducing a constraining term into the loss function, there are better performing approaches for body key points. (Yan et al., 2018a) represent the input as a graph consisting of edges between key points in both spatial and temporal direction. The graph is then partitioned into subsets based on the neighbouring nodes location relative to the centre of the graph and a minimum path distance. In a similar way, set division can be reached in temporal direction.

For references in how to handle possible training data, public data collections like the "Joint Attention in Autonomous Driving" (JAAD) dataset (Rasouli et al., 2017) offer traffic video sequences and annotate pedestrian behaviour on a timeline. These include actions (like crossing, looking, signalling etc.) and movement state (e.g. moving fast/slow, stopping, speeding up etc.).

The algorithmic blocks for map preprocessing and pedestrian path prediction are not part of this work and will therefore only be briefly touched upon to give a description of the contributions made by others.

High-resolution map data provided in the XML-based "OpenDRIVE" format (Dupuis et al., 2015). It includes information about the roadway, individual driving lanes and markings, as well as the sidewalk. Furthermore, a variety of road and road-side objects like pedestrian crossings, traffic lights, fences, trees, lampposts etc. are described. This map file is interpreted and transformed into a segmented image representation, reducing the information quantity to those relevant for pedestrian movement and given indication of walkable areas and risk classification. The relevant surrounding area can be identified by self-localisation through data from the GPS/IMU input.

Together with the tracked object instances and pedestrian behaviour features, the segmented map is then passed on to the path prediction algorithm. There are a variety of methods to solve this task which should not be further expanded here. In general, a plain approach could use the past trajectory from the tracking data and employ a simple neural network to learn a regression task to predict future positions. Information from the behaviour assessment inputs could be added for a better understanding of non-linear movement in cases of certain actions performed by the pedestrian. More recent and elaborated methods like (Cui et al., 2019) predict multiple possible trajectories of actors while also estimating their probabilities using very deep convolutional networks to automatically derive relevant features for the task.

Output

The output of the described system can be adapted according to the needs of subsequent algorithms (e.g. for path planning). A sensible option might be to provide a list of 3D pedestrian bounding boxes in global coordinates, representing their current positions, and combine them with individual estimates for the predicted position in the next seconds. As explained in the description of the pedestrian path prediction block, such predicted positions could be represented as discrete single trajectories or as a multimodal probability distribution of future positions. While the first option provides a clearer decision boundary, the latter enables a more complete representation of the overall situation. This could help decisions made by following path planning software to be aware of uncertainties in the prediction, especially in difficult cases.
An example visualisation of the system's output can be seen in figure 7.5.

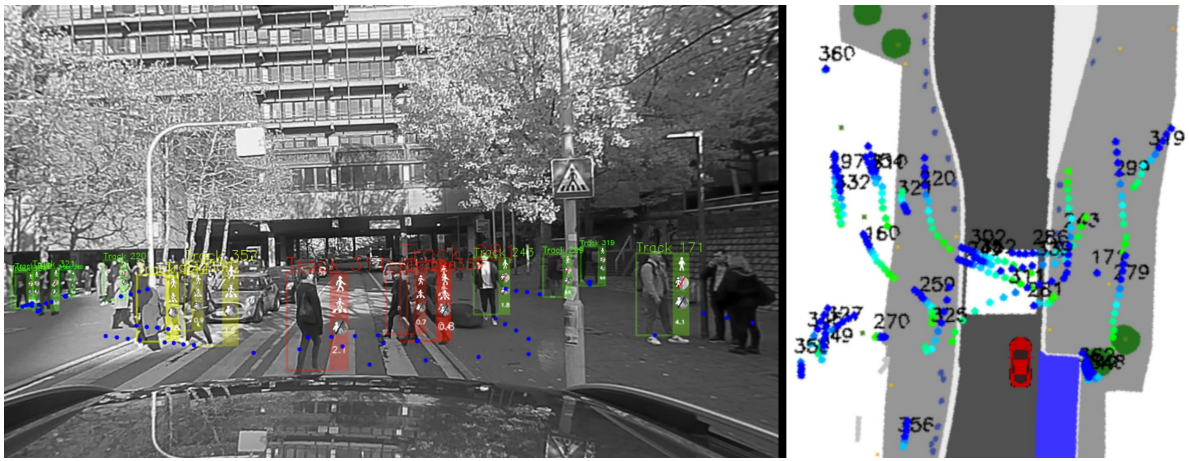


Fig. 7.5.: Output of the proposed system for pedestrian localisation and path prediction. Bounding box colours in the camera image on the left represent tracked pedestrians directly in or about to enter the lane of the ego vehicle *(red)*, other lanes on the roadway *(yellow)*, or outside the road and not about to enter soon *(green)*. On the right, the segmented map is shown including the past *(green dots)* and predicted trajectory *(blue dots)* of all detected pedestrians.

7.1.3 Vehicle Integration

At the time of writing, activities to integrate the system described above into a test vehicle were planned and started. As the @City project is set to conclude in summer of 2022 and a possible extension could further delay this date, the given descriptions of integration, although complete, are preliminary. The base vehicle used for this project is a BMW 540i as depicted in figure 7.6. All integration measures were carried out in collaboration with Aptiv staff.

Hardware Infrastructure

The main sensor used for this implementation is a *Hesai Pandora* (Hesai, 2018). This roof-mounted unit combines a mechanical rotating lidar with five camera sensors. The lidar consists of 40 channels, covering 360° in horizontal and 23° (-16°...+7°) in vertical direction. Vertical resolution is 1° for the upper five and lower ten channels and 0.33° in between. Horizontal resolution is 0.2° at a recording frequency of 10Hz. Four of the five cameras are arranged in a 90° orientation to each other and produce grey scale images with a horizontal opening angle of 129°. Additionally, a front-facing RGB camera with a field of view of 52° is included. All five image sensors offer a resolution of 1280×720 pixels.

A *Trimble Applanix Pos LV* dGPS/IMU unit is used to precisely locate the vehicle with a claimed theoretical accuracy down to ten centimetres.
For integration of the given algorithms, a computer is installed in the back of the car. It contains an Intel Core i7-8700T processor and an Nvidia GeForce GTX 1080 graphics card for fast parallelised execution of neural networks. Data between the roof-mounted sensor unit and the PC is transmitted over UDP/IP through an Ethernet switch. Information from the dGPS/IMU unit is send over a Controller Area Network (CAN bus) and translated to TCP/IP over a gateway interface.

Software Integration

To make the required algorithms useable in the vehicle, a dedicated runtime environment is employed. It handles data input from sensors, output to the car's internal network and offers the possibility to visualise raw sensor data and algorithm results conforming with a certain format. Furthermore, fundamental operations like camera image rectification and data stream synchronisation based on GPS timestamps are performed here.

Fig. 7.6.: The car used for activities in the @City project. The main sensor, Hesai Pandora, can be seen mounted on the roof.

With all of the proposed algorithms developed in Python (Rossum, 1995) to preserve experimental flexibility, it is important to facilitate an efficient way to call these. The runtime environment itself is written in Rust (Matsakis and Klock, 2014) but offers a Python "wrapper". This has the advantage of being able to instantiate the Python interpreter one time and handle all dependencies, libraries and algorithm classes, while executing the compiled Rust code from there. Calling the Python interpreter from inside Rust code would introduce more overhead in managing these algorithm classes and their inputs and outputs.

To enable parallelisation of several algorithms in Python, the library "Ray" (Moritz et al., 2018) is used. It allows for distributed execution of functions and class methods on both the CPU and graphic cards. How the algorithms will be arranged eventually is not finalised at this point. In general, methods based on machine learning strongly benefit from an execution on GPU, due to the high level of parallelisation. Tasks like data format conversion between algorithms blocks, object tracking, map data handling and the optimised lidar clustering will be performed on the CPU.

7.2 Further Applications

Apart from the main use in the system developed for the @City project, methods presented in this work have also been used in other areas. Two examples are given in the following.

Using Active Learning Methods for Efficient Data Annotation

Aspects of the active learning methods presented in chapter 3 were integrated into a web-based data annotation tool. Here, a user could upload image data and a classification algorithm, like a neural network for example. He or she would then be asked to label a small (random) initial training set and provide or label a test set. Afterwards this would be used to automatically train the classifier. All of the remaining unlabelled data is then passed through the algorithm to create class predictions and one or several active learning methods are used to determine a fixed portion of samples which, when added to the training set, could potentially most beneficial to improve accuracy. The loop could then continue with the user labelling these samples and the labels predicted in the previous inference would be applied as presets to reduce this task to merely correcting false predictions, therefore saving time. The process could continue until either a specific performance score was achieved on the test set or a time or monetary budget is exhausted.

To evaluate the potential of this tool chain, tests were conducted with previously fully annotated datasets for hand gesture and traffic sign classification. In the first case the classification accuracy of the full set baseline was reached with labelling $132\,576$ of $673\,792$ samples ($\approx$19.6%). For the traffic sign data this threshold was reached at around 30% of the complete data.

Panoptic Lidar Segmentation for Automated "Ground Truthing"

While active learning methods can help to narrow down the choices to the most valuable samples for manual labelling, it is even more desirable to limit the user input required for creating a final annotation to a minimum. With the development of ever more powerful and well generalising machine learning algorithms, in connection with a larger quantity and higher quality of publicly available datasets, their application to label familiar but unseen new data samples became increasingly fruitful. Although it is strictly advisable not to blindly trust even the very best neural networks, manual interaction for data annotation can be mainly reduced to reviewing samples and correcting mistakes made by the algorithm, therefore saving the vast majority of working time.

Object instances of cars, bikes, trucks and pedestrians were to be labelled in new sequences of lidar point clouds. As this application was run offline in a server infrastructure and processing time played a very subordinate role, state-of-the-art network architectures could be employed in conjunction with non-causal

tracking/smoothing filters. Even though these computationally intensive, benchmark leading methods can clearly outperform the real-time approach presented in 5, the later proved to be a valuable addition in detecting objects far away from the sensor. Many datasets focus on instances in close and medium ranges leading to a reduced performance in the far field. The proposed algorithm combination can mitigate this shortcoming, as the clustering only depends on a small number of connected points to detect an instance and the optimised classification uses a fixed size image representation, which is more relying on shape and perspective than size and resolution. Overall, this collection of algorithms could achieve 96% precision and 97% recall for cars, 97%/89% for bikes, 95%/78% for pedestrians and 99%/92% for trucks, when compared to a manually annotated baseline on the same data. This could potentially reduce time required for labelling of data logs immensely.

8 Conclusion and Outlook

8.1 Conclusion

In the present work, new methods have been designed and evaluated that address both the fundamentals of machine learning and its different applications to sensor data, especially in the automotive field. Ultimately, this led to the development of the presented experimental system for pedestrian detection, feature extraction and behaviour prediction for autonomous driving in urban scenarios.

The two chapters 3 and 4, concerned with more elementary considerations of the field, aimed to provide new perspectives on how to design and train convolutional neural networks.
The former underlined how the concept of active learning and its application to supervised classification problems can be a very helpful tool to mitigate the cost and time requirements for data annotation, but has to be utilised with care and domain knowledge, as it was shown to not necessarily be unbiased. This was especially highlighted concerning the replacability of classifiers, which can have negative impacts on production programmes if the architecture is changed at a later point. The later chapter detailed the development of an approach for architecture selection of CNNs and proposed a fast and robust heuristic evaluation method for candidates. Together with Bayesian optimisation, it was combined into a complete search algorithm.

An end-to-end approach for real-time panoptic segmentation of lidar sensor data, was presented in chapter 5. It was explained, how a clustering algorithm, leveraging several processing steps to preserve three-dimensional information after reduction to a two-dimensional representation for fast computation, can create an object instance segmentation as foundation for a new classification architecture. Special attention was paid to the development of a meaningful data format that efficiently represents complex features. Detailed evaluations were carried out on public data, to show that a decent performance on panoptic segmentation and other task for automotive purposes is possible, even while achieving fast computation on restricted hardware.

Chapter 6 covered further considerations regarding pedestrian behavioural feature extraction. An new approach to combine camera and lidar sensor information into a body-specific feature collection for keypoint recognition was presented and the reader was made to understand the benefits of a top-down approach to this problem for in-vehicle use. This enabled the use of a network consisting of significantly fewer parameters than needed for very deep full-frame architectures. Additionally, remarks on the detection of pedestrian awareness were given.

Eventually, the complete framework of an holistic system for pedestrian localisation and movement prediction was presented. It described the information flow from raw data of selected sensors through various algorithmic blocks towards generating an output useful to subsequent vehicle path planning and risk assessment systems. Here, connections where made to the developments of previous chapters and a plan for integration was laid out.

8.2 Outlook

It would be very interesting to transfer the robustness considerations for active learning to other ML tasks and their respective sample query strategies. Particularly for the problem of semantic segmentation there are interesting new publications (Mackowiak et al., 2019; Colling et al., 2021). Here, the potential for saving annotation time is particularly great and this is a task of great importance for autonomous driving.
Beyond that, finding a way to select the query strategies themselves in an active fashion based on the data in use and learning algorithm parameters might be the next step forward for active learning.

For the proposed architecture search, it would be evident to expand the optimisation process to include more criteria. Especially integrating a trade-off for a candidate's computational cost would be helpful. Additionally, one could implement a method to simultaneously evaluate a network with several initialisations and determine a measure on how prone an architecture is to such influence.

The presented system for real-time panoptic segmentation of lidar sensor data is very much self-contained and possible changes must be carefully coordinated with each other. Yet again, there is room for different aspects of implementation optimisation and even more efficient data handling, to possibly bring this complex task to a true embedded platform. Furthermore, it would be interesting to include a fast method for causal object tracking, maybe even in the depth image space.

The statements of chapter 6 leave more room for further research. The awareness feature extraction is made to be directly integrated into the network for body keypoint detection, whose combination with clustered and classified lidar detections is already hinted at in chapter 7. Since for a complete system the number of individual networks can pose a computational overhead, those three functions and maybe even camera image detection could possibly be integrated into one multi-headed end-to-end network.

As for the superordinate task of pedestrian understanding for autonomous driving, no specific outlook for further research is given at this point. The topic is just to vast and discussing the possibilities would require at least another two chapters. Yet, to close this work with reference to its very first section; it will be exciting to see which findings will be presented, once this next chapter in the legacy of research projects towards assisted/autonomous driving comes to a close.

Bibliography

Abadi, Martín, Ashish Agarwal, Paul Barham, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. https://www.tensorflow.org/. 2015 (cit. on p. 79).

Abubaker, Ayman, Rami Qahwaji, Stan Ipson, and Mohmmad Saleh. "One scan connected component labeling technique". In: *IEEE International Conference on Signal Processing and Communications*. 2007, pp. 1283–1286 (cit. on p. 64).

Alsfasser, Martin, Jan Siegemund, Jittu Kurian, and Anton Kummert. "Exploiting polar grid structure and object shadows for fast object detection in point clouds". In: *International Conference on Machine Vision*. 2020, 114330G (cit. on p. 60).

Angluin, Dana. "Queries and Concept Learning". In: *Machine Learning*. Vol. 2. 4. 1988, pp. 219–342 (cit. on p. 26).

Ankerst, Mihael, Markus M. Breunig, Hans Peter Kriegel, and Jörg Sander. "OPTICS: Ordering Points to Identify the Clustering Structure". In: *SIGMOD Record (ACM Special Interest Group on Management of Data)*. Vol. 28. 2. 1999, pp. 49–60 (cit. on p. 61).

Arriaga, Octavio, Matias Valdenegro-Toro, and Paul G. Plöger. "Real-time Convolutional Neural Networks for emotion and gender classification". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 2019, pp. 221–226 (cit. on p. 70).

Bar-Shalom, Y., K. C. Chang, and H. A. P. Blom. "Tracking a maneuvering target using input estimation versus the interacting multiple model algorithm". In: *IEEE Transactions on Aerospace and Electronic Systems*. Vol. 25. 2. 1989, pp. 296–300 (cit. on p. 105).

Bar-Shalom, Y., F. Daum, and J. Huang. "The probabilistic data association filter". In: *IEEE Control Systems Magazine*. Vol. 29. 6. 2009, pp. 82–100 (cit. on p. 105).

Bayer, Bryce E. "Color imaging array". In: *US Patent 3,971,065*. 1976 (cit. on p. 22).

Behley, Jens and Andres Milioto et al. "SemanticKITTI API". In: *https://github.com/PRBonn/semantic-kitti-api*. 2019 (cit. on p. 77).

Behley, Jens, Martin Garbade, Andres Milioto, et al. "SemanticKITTI: A dataset for semantic scene understanding of lidar sequences". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 9297–9307 (cit. on p. 72).

Behley, Jens, Andres Milioto, and Cyrill Stachniss. "A Benchmark for LiDAR-based Panoptic Segmentation based on KITTI". In: *arXiv [cs.CV] 2003.02371*. 2020 (cit. on p. 78).

Behley, Jens and Cyrill Stachniss. "Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments." In: *Robotics: Science and Systems*. 2018 (cit. on p. 67).

Bergstra, James and Yoshua Bengio. “Random Search for Hyper-Parameter Optimization”. In: *Journal of Machine Learning Research*. Vol. 13. 10. 2012, pp. 281–305 (cit. on p. 55).

Bischl, Bernd, Jakob Richter, Jakob Bossek, et al. “mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions”. In: *arXiv [stat.ML] 1703.03373*. 2017 (cit. on pp. 55, 58).

Bischl, Bernd, Simon Wessing, Nadja Bauer, Klaus Friedrichs, and Claus Weihs. “MOI-MBO: Multiobjective Infill for Parallel Model-Based Optimization”. In: *International Conference on Learning and Intelligent Optimization*. 2014, pp. 173–186 (cit. on p. 56).

Bogoslavskyi, Igor and Cyrill Stachniss. “Fast range image-based segmentation of sparse 3D laser scans for online operation”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2016, pp. 163–169 (cit. on pp. 61, 73).

Campbell, M., A. J. Hoane, and F. Hsu. “Deep Blue”. In: *Artificial Intelligence*. Vol. 134. 2001, pp. 57–83 (cit. on p. 8).

Cao, Zhe, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. “OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 43. 1. 2021, pp. 172–186 (cit. on p. 86).

Chen, Qi, Lin Sun, Zhixin Wang, Kui Jia, and Alan Yuille. “Object as Hotspots: An Anchor-Free 3D Object Detection Approach via Firing of Hotspots”. In: *arXiv [cs.CV] 1912.12791*. 2019 (cit. on p. 60).

Chen, Xiaozhi, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. “Multi-view 3D Object Detection Network for Autonomous Driving”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6526–6534 (cit. on p. 60).

Chollet, François. “Xception: Deep learning with depthwise separable convolutions”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 1800–1807 (cit. on p. 70).

Colling, Pascal, Lutz Roese-Koerner, Hanno Gottschalk, and Matthias Rottmann. “MetaBox+: A New Region based Active Learning Method for Semantic Segmentation using Priority Maps”. In: *International Conference on Pattern Recognition Applications and Methods*. Vol. 1. 2021, pp. 51–62 (cit. on p. 114).

Comaniciu, Dorin and Peter Meer. “Mean shift: A robust approach toward feature space analysis”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 24. 5. 2002, pp. 603–619 (cit. on p. 61).

Cortes, C. and V. Vapnik. “Support-Vector Networks”. In: *Machine Learning*. Vol. 20. 1995, pp. 273–297 (cit. on p. 8).

Cui, H., V. Radosavljevic, F. Chou, et al. “Multimodal Trajectory Predictions for Autonomous Driving using Deep Convolutional Networks”. In: *International Conference on Robotics and Automation*. 2019, pp. 2090–2096 (cit. on p. 107).

Decker, B. Louis. “World Geodetic System 1984”. In: *Proceedings of the Fourth International Geodetic Symposium on Satellite Positioning*. 1986, pp. 69–92 (cit. on p. 100).

Dumoulin, Vincent and Francesco Visin. “A guide to convolution arithmetic for deep learning”. In: *arXiv [stat.ML] 1603.07285*. 2016 (cit. on p. 131).

Dupuis, Marius, Mohamman Bahram, Hans Grezlikowski, et al. “OpenDRIVE – Format Specification”. In: *Technical Report - VIRES Simulationstechnologies GmbH*. rev. 1.4. 2015 (cit. on p. 107).

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. “A Density-Based Algorithm for Discovering Clusters a Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *International Conference on Knowledge Discovery and Data Mining*. 1996, pp. 226–231 (cit. on pp. 61, 73).

EUREKA. *project website, archived state from 3rd April 2012, retrieved 4th March 2021*. https://web.archive.org/web/20120403075558/http://www.eurekanetwork.org/project/-/id/45. 2012 (cit. on p. 1).

Fang, Hao-Shu, Shuqin Xie, Yu-Wing Tai, and Cewu Lu. “RMPE: Regional Multi-person Pose Estimation”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2353–2362 (cit. on pp. 86, 89).

Fei-Fei, L., A. Karpathy, J. Johnson, and S. Yeung. *CS231n: Convolutional Neural Networks for Visual Recognition*. Stanford University Computer Science Class, http://cs231n.github.io/. 2017 (cit. on pp. 10, 19).

Freeman, Ido, Lutz Roese-Koerner, and Anton Kummert. “Effnet: An Efficient Structure for Convolutional Neural Networks”. In: *IEEE International Conference on Image Processing*. 2018, pp. 6–10 (cit. on p. 70).

Fukunaga, K. and L. Hostetler. “The estimation of the gradient of a density function, with applications in pattern recognition”. In: *IEEE Transactions on Information Theory*. Vol. 21. 1. 1975, pp. 32–40 (cit. on p. 61).

Fukushima, K. “Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position”. In: *Biological Cybernetics*. Vol. 36. 1980, pp. 193–202 (cit. on p. 16).

Geiger, Andreas, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? the kitti vision benchmark suite”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3354–3361 (cit. on p. 72).

Glorot, Xavier and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *13th International Conference on Artificial Intelligence and Statistics*. Vol. 9. 2010, pp. 249–256 (cit. on p. 49).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016, p. 332 (cit. on p. 17).

Hahnloser, R., R. Sarpeshkar, M. Mahowald, R. Douglas, and S. Seung. “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit”. In: *Nature*. Vol. 405. 2000, pp. 947–951 (cit. on p. 18).

He, Kaiming, 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. 86).

He, Kaiming, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778 (cit. on p. 32).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778 (cit. on p. 70).

Hesai. *Pandora All-in-One Sensing Solution for Autonomous Driving - User’s Manual*. 2018 (cit. on pp. 88, 109).

Hinton, Geoffrey E. and Sam Roweis. “Stochastic Neighbor Embedding”. In: *Advances in Neural Information Processing Systems*. Vol. 15. 2003 (cit. on p. 28).

Hinton, Geoffrey E. and R. R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks”. In: *Science*. Vol. 313. 5786. 2006, pp. 504–507 (cit. on p. 86).

Hinton, Geoffrey E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Improving Neural Networks by Preventing Co-Adaptions of Feature Detectors”. In: *arXiv [cs.NE] 1207.0580*. 2012 (cit. on p. 21).

Hochreiter, Sepp and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation*. Vol. 9. 1997, pp. 1735–1780 (cit. on p. 106).

Howard, Andrew G., Menglong Zhu, Bo Chen, et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *arXiv [cs.CV] 1704.04861*. 2017 (cit. on p. 70).

Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. “Densely Connected Convolutional Networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269 (cit. on p. 88).

Hubel, D. and T. Wiesel. “Receptive Fields And Functional Architecture Of Monkey Striate Cortex”. In: *The Journal of Physiologie*. Vol. 195. 1968, pp. 215–243 (cit. on p. 16).

Hull, J. J. “A database for handwritten text recognition research”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 16. 5. 1994, pp. 550–554 (cit. on p. 53).

Ioffe, S. and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37. 2015, pp. 448–456 (cit. on p. 22).

James, G., D. Witten, T. Hastie, and R. Tibshirani. *An Introduction To Statistical Learning*. Springer New York, 2013, p. 356 (cit. on p. 11).

Jarrett, K., K. Kavukcuoglu, M. Ranzato, and Y. LeCun. “What is the Best Multi-Stage Architecture for Object Recognition?” In: *IEEE International Conference on Computer Vision*. 2009, pp. 2146–2153 (cit. on p. 18).

Jarrett, K., K. Kavukcuoglu, Marc’Aurelio Ranzato, and Y. LeCun. “What is the best multi-stage architecture for object recognition?” In: *2009 IEEE 12th International Conference on Computer Vision* (2009), pp. 2146–2153 (cit. on p. 50).

Jones, Donald R., Matthias Schonlau, and William J. Welch. “Efficient Global Optimization of Expensive Black-Box Functions”. In: *Journal of Global Optimization*. Vol. 13. 4. 1998, pp. 455–492 (cit. on p. 55).

Kingma, Diederik P. and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations*. 2015 (cit. on p. 32).

Kirillov, Alexander, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollár. “Panoptic segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9404–9413 (cit. on p. 78).

Krige, Daniel G. “A statistical approach to some basic mine valuation problems on the Witwatersrand”. In: *Journal of the Southern African Institute of Mining and Metallurgy*. Vol. 52. 6. 1951, pp. 119–139 (cit. on p. 56).

Krizhevsky, Alex, Vinod Nair, and Geoffrey Hinton. “Learning Multiple Layers of Features from Tiny Images”. In: *Canadian Institute for Advanced Research*. 2009 (cit. on p. 32).

Lang, Alex H., Sourabh Vora, Holger Caesar, et al. “PointPillars: Fast Encoders for Object Detection From Point Clouds”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12697–12705 (cit. on pp. 60, 80).

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner. “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE*. Vol. 86. 11. 1998, pp. 2278–2324 (cit. on p. 16).

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on pp. 32, 53).

Lewis, David D. and William A. Gale. “A Sequential Algorithm for Training Text Classifiers”. In: *SIGIR ’94*. 1994, pp. 3–12 (cit. on p. 26).

Lewis, David D. and William A. Gale. “A Sequential Algorithm for Training Text Classifiers”. In: *SIGIR ’94*. Ed. by Bruce W. Croft and C. J. van Rijsbergen. London: Springer London, 1994, pp. 3–12 (cit. on p. 28).

Li, Q., S. Chen, C. Wang, et al. “LO-Net: Deep Real-Time Lidar Odometry”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8465–8474 (cit. on p. 67).

Lin, Tsung-Yi, Michael Maire, Serge Belongie, et al. “Microsoft COCO: Common objects in context”. In: *European Conference on Computer Vision*. 2014, pp. 740–755 (cit. on p. 89).

Mackowiak, Radek, Philip Lenz, Omair Ghori, et al. “CEREALS - Cost-Effective REgion-based Active Learning for Semantic Segmentation”. In: *British Machine Vision Conference*. 2019 (cit. on p. 114).

Matsakis, Nicholas D. and Felix S. Klock. “The Rust Language”. In: *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*. 2014, pp. 103–104 (cit. on p. 110).

Matsumoto, Makoto and Takuji Nishimura. “Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator”. In: *ACM Transactions on Modeling and Computer Simulation*. Vol. 8. 1. 1998, pp. 3–30 (cit. on p. 49).

McKay, M. D., R. J. Beckman, and W. J. Conover. “A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code”. In: *Technometrics* 21.2 (1979), pp. 239–245 (cit. on p. 56).

Minemura, Kazuki, Hengfui Liau, Abraham Monrroy, and Shinpei Kato. “LMNet: Real-time Multiclass Object Detection on CPU Using 3D LiDAR”. In: *Asia-Pacific Conference on Intelligent Robot Systems*. 2018, pp. 28–34 (cit. on p. 60).

Minsky, M. “A Neural-Analogue Calculator Based upon a Probability Model of Reinforcement”. In: *Harvard University Psychological Laboratories, Cambridge, Massachusetts*. 1952 (cit. on p. 7).

Minsky, M. and S. Papert. *Perceptrons : An Introduction to Computational Geometry*. MIT Press, Cambridge, 1969 (cit. on p. 8).

Mitchell, Tom M. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997, p. 2 (cit. on p. 9).

– *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997, p. 87 (cit. on p. 10).

Moosmann, Frank, Oliver Pink, and Christoph Stiller. “Segmentation of 3D lidar data in non-flat urban environments using a local convexity criterion”. In: *IEEE Intelligent Vehicles Symposium (IV)*. 2009, pp. 215–220 (cit. on p. 61).

Moosmann, Frank and Christoph Stiller. “Velodyne slam”. In: *IEEE Intelligent Vehicles Symposium*. 2011, pp. 393–398 (cit. on p. 67).

Moritz, Philipp, Robert Nishihara, Stephanie Wang, et al. “Ray: A Distributed Framework for Emerging AI Applications”. In: *13th USENIX Symposium on Operating Systems Design and Implementation*. 2018, pp. 561–577 (cit. on p. 110).

Nagi, J., F. Ducatelle, G. Caro, et al. “Max-Pooling Convolutional Neural Networks for Vision-based Hand Gesture Recognition”. In: *IEEE International Conference on Signal and Image Processing Applications*. 2011, pp. 342–347 (cit. on p. 19).

Netzer, Yuval, Tao Wang, Adam Coates, et al. “Reading Digits in Natural Images with Unsupervised Feature Learning”. In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. 2011 (cit. on pp. 32, 53).

Pearson, Karl. “On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*. Vol. 2. 11. 1901, pp. 559–572 (cit. on p. 28).

Peemen, M., M. Bart, and H. Corporaal. “Speed Sign Detection And Recognition By Convolutional Neural Networks”. In: *Proceedings of the 8th International Automotive Congress*. 2011, pp. 162–170 (cit. on p. 16).

Polyak, B. “Some Methods Of Speeding Up The Convergence Of Iteration Methods”. In: *USSR Computational Mathematics and Mathematical Physics*. Vol. 5. 1964, pp. 791–803 (cit. on p. 20).

Porzi, Lorenzo, Samuel Rota Bulo, Aleksander Colovic, and Peter Kontschieder. “Seamless scene segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8277–8286 (cit. on p. 79).

Qi, C. R., W. Liu, C. Wu, H. Su, and L. J. Guibas. “Frustum PointNets for 3D Object Detection from RGB-D Data”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 918–927 (cit. on p. 60).

Qi, Charles R., Hao Su, Kaichun Mo, and Leonidas J. Guibas. “PointNet: Deep learning on point sets for 3D classification and segmentation”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 77–85 (cit. on p. 60).

Qi, Charles R., Li Yi, Hao Su, and Leonidas J. Guibas. “PointNet++: Deep hierarchical feature learning on point sets in a metric space”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5100–5109 (cit. on p. 59).

Rasouli, Amir, Iuliia Kotseruba, and John K Tsotsos. “Are they going to cross? A benchmark dataset and baseline for pedestrian crosswalk behavior”. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 206–213 (cit. on p. 106).

Redmon, Joseph and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *arXiv [cs.CV] 1804.02767*. 2018 (cit. on pp. 88, 100).

Ren, Shaoqing, 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*. Vol. 28. 2015 (cit. on p. 100).

Rojas, Raul. *Neural Networks*. Springer Berlin, 1996, p. 3 (cit. on p. 9).

Rosenblatt, F. “The Perceptron: A Probablistic Model For Information Storage and Origanization in the Brain”. In: *Psychological Review*. Vol. 65. 6. 1958 (cit. on p. 7).

Rossum, Guido van. “Python tutorial”. In: *CS-R9526 - Centrum voor Wiskunde en Informatica (CWI)*. 1995 (cit. on p. 110).

Rumelhart, D. E., G. Hinton, and R. Williams. “Learning Representations by Back-Propagating Errors”. In: *Nature*. Vol. 323. 1986, pp. 533–536 (cit. on p. 14).

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. 1986, pp. 318–362 (cit. on p. 86).

Samuel, Arthur. “Some Studies In Machine Learning Using the Game of Checkers”. In: *IBM Journal*. Vol. 3. 3. 1959 (cit. on p. 7).

Saxe, Andrew M., Pang Wei Koh, Zhenghao Chen, et al. “On Random Weights and Unsupervised Feature Learning”. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. 2011, pp. 1089–1096 (cit. on p. 50).

Schubert, Erich, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. “DBSCAN Revisited: Why and How You Should (Still) Use DBSCAN”. In: *ACM Transactions on Database Systems*. Vol. 42. 3. 2017, p. 21 (cit. on p. 61).

Sener, Ozan and Silvio Savarese. “Active Learning for Convolutional Neural Networks: A Core-Set Approach”. In: *International Conference on Learning Representations*. 2018, pp. 1–13 (cit. on p. 30).

Settles, Burr. “Active Learning Literature Survey”. In: *Computer Sciences Technical Report, University of Wisconsin-Madison*. Vol. 1648. 2009 (cit. on p. 25).

Shannon, C. E. “A mathematical theory of communication”. In: *The Bell System Technical Journal*. Vol. 27. 3. 1948, pp. 379–423 (cit. on pp. 30, 31).

Shi, S., X. Wang, and H. Li. “PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 770–779 (cit. on p. 60).

Shi, Shaoshuai, Chaoxu Guo, Li Jiang, et al. “PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection”. In: *arXiv [cs.CV] 1912.13192*. 2019 (cit. on p. 60).

Shin, Kiwoo, Youngwook Paul Kwon, and Masayoshi Tomizuka. “Roarnet: A robust 3d object detection based on region approximation refinement”. In: *IEEE Intelligent Vehicles Symposium*. 2019, pp. 2510–2515 (cit. on p. 60).

Silver, D., A. Huang, C. Maddison, and A. Guez et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature*. Vol. 529. 2016, pp. 484–489 (cit. on p. 8).

Simpson, Edward H. “Measurement of Diversity”. In: *Nature*. Vol. 163. 1949 (cit. on p. 31).

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *The Journal of Machine Learning Research*. Vol. 15. 2014, pp. 1929–1958 (cit. on p. 21).

Sutskever, I., J. Martens, G. Dahl, and G. Hinton. “On The Importance Of Initialization And Momentum In Deep Learning”. In: *Proceedings of Machine Learning Research*. Vol. 28. 2013, pp. 1139–1147 (cit. on p. 20).

Time Magazine. *Science: Radio Auto*. Vol. VI. 6. Aug. 1925 (cit. on p. 1).

Toffoli, Tommaso and Norman Margolus. *Cellular automata machines: a new environment for modeling*. MIT press, 1987, p. 60 (cit. on p. 64).

van der Maaten, Laurens and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research*. Vol. 9. 86. 2008, pp. 2579–2605 (cit. on p. 28).

Vapnik, V. and A. Chervonenkis. “A Class Of Algorithms For Pattern Recognition Learning”. In: *Avtomatika i Telemakhanika*. Vol. 25. 6. 1964, pp. 937–945 (cit. on p. 8).

Vapnik, V. and A. Lerner. “Pattern Recognition Using Generalized Portraits”. In: *Avtomatika i Telemakhanika*. Vol. 24. 6. 1963, pp. 774–780 (cit. on p. 8).

Vedaldi, A., K. Lenc, and A. Gupta. “MatConvNet - Convolutional Neural Networks for MATLAB”. In: *Manual* (2015) (cit. on p. 12).

Vial, Gregory. “Cyrillic oriented MNIST: A dataset of Latin and Cyrillic letter images”. In: *Website of Kaggle Inc. retrieved 22nd March 2021*. 2017 (cit. on pp. 32, 53).

Virtanen, Pauli, Ralf Gommers, Travis E Oliphant, et al. “SciPy 1.0: fundamental algorithms for scientific computing in Python”. In: *Nature Methods*. Vol. 17. 3. 2020, pp. 261–272 (cit. on p. 64).

Wang, Zhixin and Kui Jia. “Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2019, pp. 1742–1749 (cit. on p. 60).

Warde-Farley, D., I. Goodfellow, A Courville, and Y. Bengio. “An Empirical Analysis Of Dropout In Piecewise Linear Networks”. In: *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*. 2014 (cit. on p. 21).

Weyers, Patrick, Alexander Barth, and Anton Kummert. “Driver State Monitoring with Hierarchical Classification”. In: *21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pp. 3239–3244 (cit. on p. 43).

Xiao, Han, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *arXiv [cs.LG] 1708.07747*. 2017 (cit. on p. 32).

Yan, Sijie, Yuanjun Xiong, and Dahua Lin. “Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition”. In: *AAAI Conference on Artificial Intelligence*. 2018, pp. 7444–7452 (cit. on p. 106).

Yan, Yan, Yuxing Mao, and Bo Li. “Second: Sparsely embedded convolutional detection”. In: *Sensors*. Vol. 18. 10. 2018, p. 3337 (cit. on p. 60).

Yosinski, Jason, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. “Understanding Neural Networks Through Deep Visualization”. In: *Deep Learning Workshop, International Conference on Machine Learning (ICML)*. 2015 (cit. on p. 51).

Zhang, Z. “A flexible new technique for camera calibration”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 22. 11. 2000, pp. 1330–1334 (cit. on p. 22).

Zhao, Xin, Zhe Liu, Ruolan Hu, and Kaiqi Huang. “3D object detection using scale invariant and feature reweighting networks”. In: *AAAI Conference on Artificial Intelligence*. 2019, pp. 9267–9274 (cit. on p. 60).

Zheng, Ce, Wenhan Wu, Taojiannan Yang, et al. “Deep Learning-Based Human Pose Estimation: A Survey”. In: *arXiv [cs.CV] 2012.13392*. 2020 (cit. on p. 85).

Zhou, Y. and R. Chellappa. “Computation of optical flow using a neural network”. In: *IEEE International Conference on Neural Networks*. Vol. 2. 1988, pp. 71–78 (cit. on p. 19).

Zhou, Yin and Oncel Tuzel. “VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4490–4499 (cit. on p. 60).

Zhuang, F., Z. Qi, K. Duan, et al. “A Comprehensive Survey on Transfer Learning”. In: *Proceedings of the IEEE*. Vol. 109. 1. 2021, pp. 43–76 (cit. on p. 47).

Zöller, Marc-André and Marco F. Huber. “Benchmark and Survey of Automated Machine Learning Frameworks”. In: *Journal of Artificial Intelligence Research*. Vol. 70. 2021, pp. 409–472 (cit. on p. 48).

List of Figures

List of Tables

Appendix

A

	MNIST	USPS	CoMNIST	SVHN	AHC	OD
No. Samples	70 000	11 000	11 218	99 289	207 636	70 000
Image Size	28×28	16×16	32×32	32×32	13×13	46×30
No. Classes	10	10	26	10	3	2
No. Input Channels	1	1	1	3	1	1

Tab. A.1.: Description of the properties of the data sets used for evaluation in section 4.2.

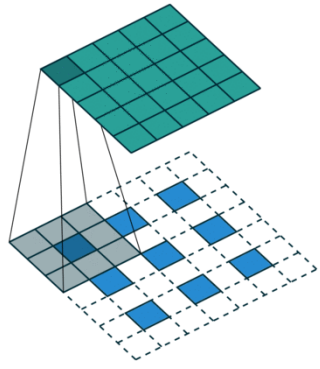


Fig. A.1.: Visualisation of a transposed convolution with a stride of two and padding (Dumoulin and Visin, 2016).

Acknowledgements

Finishing this thesis under the impression of a pandemic proved to be a challenge. Therefore, the following people deserve even more recognition and appreciation as they would have had for their lasting support anyway.

First of all, I would like to thank my supervisor Anton Kummert. Thank you for your guidance and providing me with the freedom to select many aspects of my research. Your reassuring but unobtrusive support enabled me and many of my peers to have a diverse and distinctively practical perspective on the promotion process. I would also like to thank Bela Gipp for taking over the responsibilities of the co-examiner.

In addition, I want to express my gratitude to Markus Bühren, Lutz Roese-Koerner and Dennis Müller, for their encouraging support, trust and guidance of nearly five years. I would also like to thank Christian Nunn for enabling this scholarship program and the industry-related research opportunity that comes with it.
Furthermore, I owe a great deal of appreciation and gratitude to the whole rest of the "AI & Computer Vision" team of Aptiv's Advanced Engineering department in Wuppertal. I always considered it to be an honest privilege, to be part of such an enjoyable, helpful and friendly group of like-minded engineers and researches within an otherwise sometimes strict and difficult industry.
I would like to specifically name Frederik Hasecke, Martin Alsfasser, Pascal Colling and Ido Freeman, who were not only valuable fellow scholars and researchers, co-authors and colleagues, but invaluable friends beyond that. Moreover, André Paus was the best colleague one could wish for to navigate the uncertainties of a public research project and lead endless discussions, both technical and non-technical.

My sincerest gratitude goes to Philipp "Wolle" and Anne Kaiser. Over many years you have proven to be more than friends and I cannot wait for more decades to come. I would also like to thank Johannes Staabs for being the best room mate I never had, fellow student and amateur cook, as well as nonsense movie night partner, one could ever ask for. I further want to thank my wonderful friends Timon, Fredo, Joscha, Kolja, Lars, Lisa, Toni, Ferdinand, Silke, Emi, Amrei and Yvonne, who make everything better all around.

Last but most definitely not least, I would love to thank my amazing parents, who never ceased to support me unconditionally and enabled me the start into a life full of opportunities.