Using Convolutional Neural Networks to distinguish vehicle pose and vehicle class

Erkennung von Fahrzeugpose und Fahrzeugklasse mit Convolutional Neural Networks
Master-Thesis von Christoph Münker aus Schlüchtern
Tag der Einreichung:

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Hien Dang
3. Gutachten: Stefan Luthardt

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

Using Convolutional Neural Networks to distinguish vehicle pose and vehicle class
Erkennung von Fahrzeugpose und Fahrzeugklasse mit Convolutional Neural Networks

Vorgelegte Master-Thesis von Christoph Münker aus Schlüchtern

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Hien Dang
3. Gutachten: Stefan Luthardt

Tag der Einreichung:

Technische Universität Darmstadt
Fachbereich Informatik

Fachgebiet Knowledge Engineering Group
Prof. Dr. Johannes Fürnkranz

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 28. November 2016

(Christoph Münker)

Abstract

Convolution Neural Networks (ConvNets) are specialized Neural Networks, which are well suited for the processing of images and had enormously success on many machine learning problems in the last years. Therefore, we are interested in the performance of ConvNets for the classification of vehicle pose and vehicle class on images of vehicles. To apply ConvNets on these two tasks, we have developed our own ConvNet architecture, and compared this to three state-of-the-art ConvNet architectures. Furthermore, we study the capability of the cooperation between these two tasks, because the appearance of the vehicle class is dependent on the pose of the vehicle. Therefore, we investigate two approaches. The first approach is the multi-task learning, which simultaneously learns the two tasks on the same model. As the second approach, we use an expert learner. It uses the detected pose on the image to chose a specially trained model on this pose for the classification of the vehicle class.

As data basis is used "The Comprehensive Cars" dataset from Yang et al. [57]. From this dataset, we use 97,269 images with 967 different vehicles to train and test the ConvNet. The labels for the vehicle pose are manually expanded and improved. For the vehicle class, we define a new categorization and could semi-automatically assign the labels. At the end the vehicle pose is described by eight orientations and the vehicle class by eight classes.

The results show, that the learning of the vehicle pose is not a problem. The single-task model with default settings get an error rate of 1.12%. The vehicle class is not so good and get an error rate of 28.89% with the default settings. But the multi-task learning increases the performance of the vehicle class, so that we reach an error rate of 26.04% with our best single model. The ensemble learning improves the error to 24.87%.

Zusammenfassung

Convolution Neural Networks (ConvNets) sind spezielle Neuronale Netze, die gut geeignet sind für das verarbeiten von Bildern und haben in den letzten Jahren ernormen Erfolg auf vielen Problemen des maschinellen lernens. Daher sind wir an der Leistung von ConvNets zur Klassifizierung von Fahrzeugpose und Fahrzeugklasse auf Fahrzeugbildern interessiert. Um ConvNets auf den zwei Aufgaben anzuwenden, haben wir eine eigene ConvNet Architektur entwickelt und diese gegen drei moderne ConvNet Architekturen verglichen. Des Weiteren, untersuchen wir die Fähigkeit der Zusammenarbeit zwischen den beiden Aufgaben, da das Erscheinungsbild der Fahrzeugklasse abhängig von der Fahrzeugpose ist. Daher untersuchen wir zwei Ansätze. Der erste Ansatz ist Multi-Task Lernen, das auf einem Model simultan beide Aufgaben lernt. Als zweiten Ansatz, verwenden wir ein Expertenlerner. Es nutzt die erkannte Pose auf einem Bild, um ein speziell gelerntes Model auf dieser Pose für die Klassifikation der Fahrzeugklasse auszuwählen.

Als Datengrundlage dient der "Comprehensive Cars" Datensatz von Yang et al. [57]. Von dem Datensatz verwenden wir 97.269 Bilder mit 967 verschiedenen Fahrzeugen um das ConvNet zu trainieren und zu testen. Die Labels für die Fahrzeugklasse sind händisch erweitert und verbessert worden. Für die Fahrzeugklasse definierten wir eine neue Einteilung und konnten die Labels halbautomatisch zuweisen. Am Ende ist die Fahrzeugpose mit acht Orientierungen und die Fahrzeugklasse mit acht Klassen beschrieben.

Die Ergebnisse zeigen, dass das Lernen der Fahrzeugpose kein Problem ist. Das Model für nur Fahrzeugpose hat mit den Standardeinstellungen eine Fehlerrate von 1,12%. Die Fahrzeugklasse ist nicht so gut und hat mit den Standardeinstellungen eine Fehlerrate von 28,89%. Das Multi-Task Lernen verbessert die Leistung der Fahrzeugklasse, so dass wir mit unserem besten einzel Model eine Fehlerrate von 26.04% erreichen. Das Ensemble Lernen verbessert die Fehlerrate auf 24,87%.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Goal of this Thesis . . . 2
- 1.3 Thesis Structure . . . 3

2 Basics of Machine Learning, Neural Networks and Convolutional Neural Networks **5**
- 2.1 Basics in Machine Learning . . . 5
  - 2.1.1 What is Machine Learning? . . . 5
  - 2.1.2 Learning Problems . . . 6
  - 2.1.3 Under- and Overfitting . . . 7
  - 2.1.4 Hyperparameter and Model Selection . . . 8
- 2.2 Artificial Neural Networks . . . 8
  - 2.2.1 Perceptron . . . 9
  - 2.2.2 Feedforward Neural Networks . . . 10
  - 2.2.3 The Training . . . 11
  - 2.2.4 Activation Functions . . . 16
  - 2.2.5 Regularization . . . 18
- 2.3 Convolutional Neural Networks . . . 20
  - 2.3.1 Convolution Layer . . . 20
  - 2.3.2 Pooling Layer . . . 22

3 Related Work **25**
- 3.1 Vehicle Pose . . . 25
- 3.2 Vehicle Class . . . 25
- 3.3 Vehicle Pose and Vehicle Class . . . 25

4 The Dataset **27**
- 4.1 The Comprehensive Cars (CompCars) Dataset . . . 27
- 4.2 Preparation of the Dataset . . . 29
  - 4.2.1 The Pose Label . . . 30
  - 4.2.2 The Vehicle Class Label . . . 32
  - 4.2.3 Splitting of the Dataset . . . 34
- 4.3 The Pose Crowd Labeling . . . 35
  - 4.3.1 The User-Interface . . . 35
  - 4.3.2 Quality Assurance . . . 36
  - 4.3.3 Acknowledgment . . . 36

5 Preprocessing and Augmentation of the Images **39**
- 5.1 Overview of the Preprocessing Pipeline . . . 39
- 5.2 Centering of the Vehicle . . . 40
- 5.3 Histogram Equalization . . . 41
- 5.4 Augmentation . . . 42
- 5.5 Scale the Image . . . 43
- 5.6 Colorspace Transformation . . . 44
- 5.7 Feature scaling . . . 45

6 Methodology **47**
6.1 Transfer Learning Models . . . 47
6.1.1 Inception-v3 . . . 49
6.1.2 ResNet-50 . . . 50
6.1.3 VGG16 . . . 51
6.2 Our Model Architecture . . . 51
6.3 Multi-Task Learning . . . 53
6.4 Expert Learning . . . 54
6.5 The Training . . . 55
6.5.1 Further Regularizations . . . 55
6.5.2 Default Setup . . . 55
6.6 Testing . . . 57

7 Evaluation **59**
7.1 Evaluation Metrics . . . 59
7.2 General Evaluation . . . 61
7.3 Single-Task Learning . . . 64
7.3.1 Vehicle Pose . . . 64
7.3.2 Vehicle Class . . . 65
7.3.3 The Problem of the Vehicle Class . . . 67
7.3.4 Compare with Transfer Learning Models . . . 68
7.4 Multi-Task Learning . . . 69
7.4.1 Baseline Multi-Task Learning . . . 69
7.4.2 Compare with Transfer Learning Models . . . 70
7.4.3 Change of Branching Layer . . . 70
7.4.4 More Tasks for the Multi-Task Learning . . . 71
7.4.5 More Data for the Training . . . 72
7.4.6 Change of Hyperparameters . . . 72
7.4.7 Ensemble Learning . . . 75
7.5 The Expert Learning Approach . . . 76
7.6 Deeper Look . . . 77
7.7 Discussion . . . 78

8 Conclusion and Future Work **81**
8.1 Conclusion . . . 81
8.2 Future Work . . . 81

Bibliography **83**

List of Figures

2.1 Workflow of a Machine Learning Problem . . . 6
2.2 Diagram for Optimal Capacity . . . 7
2.3 Example for Under- and Overfitting of a Function. . . . 8
2.4 Diagram of one Perceptron . . . 10
2.7 Momentum and Nesterov Momentum Update . . . 16
2.8 Diagram for the Activation Functions of ReLU and ELU . . . 17
2.9 Example for Convolution Layer . . . 21
2.10 Example for Max-Pooling and Average-Pooling . . . 22

4.1 Example Images of the CompCars Web-Nature Dataset . . . 28
4.2 Representation of the Vehicle Pose Labels . . . 31
4.3 Distribution of the Vehicle Poses . . . 31
4.4 Distribution of the Vehicle Class . . . 34
4.5 The User Interfaces of the Web-Application CROWD LABELING . . . 37

5.1 Flowchart of the Preprocessing Pipeline . . . 39
5.2 Example of the Centering of a Vehicle in an Image . . . 40
5.3 Distribution of the Bounding Box Aspect Ratio . . . 41
5.4 Example of Histogram Equalization . . . 42
5.5 Examples of Augmentation . . . 43

6.1 Example of an Inception Module in Inception-v3 . . . 49
6.2 Example of a Residual Block in ResNet-50 . . . 50
6.3 Architecture Comparison of VGG16 and VPVC-Net . . . 52
6.4 Comparison of Different Resolutions . . . 53
6.5 Extended Last Layers of VPVC-Net for Multi-Task Learning . . . 54

7.1 Runtime Comparison for the Training of One Epoch . . . 61
7.2 Progress of Training/Validation Loss and Error Rate for Vehicle Pose and Vehicle Class . . . 63
7.3 Wrong Predicted Example Images for Vehicle Pose . . . 65
7.4 Vehicle Class Error for every Vehicle Pose . . . 66
7.5 Wrong Predicted Example Images for Vehicle Class . . . 67
7.6 Comparison of Transfer Learning Models and VPVC-Net on Single-Task Learning . . . 68
7.7 Comparison of Transfer Learning Models and VPVC-Net on Multi-Task Learning . . . 70
7.8 Comparison of the Change of the Branching Layer on Multi-Task Learning . . . 71
7.9 Comparison of More Tasks on Multi-Task Learning . . . 72
7.10 Comparison of More Data on Multi-Task Learning . . . 73
7.11 Comparison of Preprocessing Hyperparameters on Multi-Task Learning . . . 74
7.12 Comparison of Different Augmentation Levels on Multi-Task Learning . . . 75
7.13 Comparison of Different Dropout Levels on Multi-Task Learning . . . 75
7.14 Comparison of Different Ensemble Sets on Multi-Task Learning . . . 76
7.15 Vehicle Class Error of the Expert Learner Approaches for every Vehicle Pose . . . 77
7.16 Histogram for the Number of Correct Predictions for an Image on 66 Different Models . . . 78
7.17 Occlusion Sensitivity of Selected Images . . . 80

List of Tables

4.1 Overview for the Number of Images for the three different Types in the CompCars Dataset 27
4.2 Compare of KBA-Class and the new defined Vehicle Class 33
4.3 Number of Images and different Vehicles for Trainings-, Validation- and Test-Set 34

6.1 Comparison of Error Rates and Parameter of the Transfer Learning Models on ImageNet Challenge 48
6.2 Hyperspace for the Hyperparameter Search for the Transfer Learning 48
6.3 Default Hyperparameter for Training and Testing 56

7.1 Example Confusion Matrix for Multi-Class Classification 59
7.2 Prediction Runtime for Batch-Size 1 and 64 on two different GPUs 62
7.3 Five Results of the Single-Task Learning on the VPVC-Net for each Vehicle Pose and Vehicle Class 63
7.4 Confusion Matrix for Single-Task Learning of Vehicle Pose 64
7.5 Confusion Matrix for Single-Task Learning of Vehicle Class 66
7.6 Confusion Matrix for Multi-Task Learning of Vehicle Class 69
7.7 Comparison of Multi-Task Learning with the Results of Single-Task Learning 70
7.8 Comparison of More Data on Multi-Task Learning 73
7.9 The Settings of the Augmentation for the Testing of Different augmentation Levels 74
7.10 The System Performance of the Expert Learning Approach 77

1 Introduction

Vehicles! Everywhere where we go, we see vehicles on the streets. Either they are parking or they are driving on the streets. When we walk through the streets, we rarely see two optically identical vehicles. They appear in different colors and different shapes. From small to large, from sports car up to a pickup. For everybody exists the right vehicle, which fulfills their individual needs. One person likes a large vehicle, another needs a practical vehicle, like a pickup. The family needs a minivan and the single living in the city might prefer a small vehicle. But there is not a standard vehicle for every class. Every automobile manufacture has their own ideas regarding vehicle design. This makes a huge variation of different vehicle types. But in the most designs a common structure exists. So it is easy for us to recognize the pose of a vehicle. If we see a vehicle from the side, then it is easy to determine if it is the left or right side, because the most vehicles have a characteristic design, to reduce the air resistance. Front and back is also easy, because it is common, that red lights are used for back, and white lights for front. Or if we can see the exhaust, we know we see the backside of the vehicle.

But not only the recognition of the pose is easy for us. We can also easily classify the vehicles into categories. There is also a common structure for the vehicle classes. We know the difference between a minivan and a minibus or a compact and a large vehicle. In case of the latter, the difference lies in the size of the vehicles, and sometimes in a different type of the body shape (hatchback vs. sedan).

It is easy for us to classify vehicle poses and classes, even if we see the vehicle only on an image. We can give a person an image of a vehicle, and the person can tell us the pose and the vehicle class of the vehicle. But for machines, it is a hard problem. Since decades, there is research to teach machines to see and understand images. A promising machine learning method to solve this problem are the *Convolution Neural Networks* (ConvNets). They have in the last years enormous success on image classification tasks [8, 16, 18, 19, 26, 47, 52, 53]. We see a huge potential of the ConvNets to solve the two tasks. Therefore, we want to study the capability of ConvNets to classifying the vehicle pose and the vehicle class on images. Furthermore, we want to explore the capability, that one of the tasks can support the other task to increase its performance.

But what is the vehicle pose and vehicle class useful for? For example, it could be helpful to improve the driver assistance systems in vehicles. The vehicle pose and vehicle class are added as attributes to all visible vehicles. This information is then available for all subsystems, which can use it to make a better decision. To add the vehicle pose to standing vehicles, can be very needful, because there is no information from other systems about their possible movement direction. A standing vehicle could be a parallel parked vehicle on the road or a standing vehicle on the driveway. The vehicle on the driveway is a more potential hazard, because it could be anytime pull out. So a subsystem could use these informations and can plan a strategy, for the case, that the vehicle is suddenly moving. The vehicle class could be used by another subsystem to anticipate the other driver's behavior. For example, a sports vehicle behind us means a higher chance of being outpaced than by a small vehicle.

Another application could be for traffic surveillance. It could be used to create statistics of different vehicle classes, or used for a vehicle class based toll. Also it could be used for improvement of tracking. If the tracking was lost, the search area could be reduced, because it exists only two directions, in which the vehicle could have moved in a short time.

Additionally, the classification of vehicle pose and vehicle class on images could be used for image search tools to tag the images with these attributes. There are much more possible applications. But to generalize, the vehicle pose and vehicle class are two useful attributes for intelligent processing of vehicle images.

1.1 Motivation

Convolution Neural Networks (ConvNets) are specialized Neural Networks, which are well suited for the processing of images. The ConvNets are not a new idea, for example in 1989 have LeCun et al. [27] a ConvNet used to recognize handwritten digits in mails, but the hype around ConvNets has started 2012 after the incredible winning of Krizhevsky et al. [26] on the ImageNet challenge [42] for classification of images into 1,000 categories. The ConvNet solution of Krizhevsky et al. had a top-5 error-rate of 15.3% compared to the second place with 26.17%. Since this, the error rate is further decreased with other ConvNet designs and further research on the training of Neural Networks. ConvNets have not only reached good results on ImageNet. They are also successfully applied on different tasks on images, like the recognition of numbers in street view images [16] or in traffic sign classification [8]. But also there are ConvNets for non image tasks. For example, ConvNets can be also applied in natural language processing [10].

ConvNets or rather Neural Networks (NN) are very interesting machine learning methods. They have a layered structure, and every layer could learn an abstract representation of the input. This ability is called *Feature Learning* or known as *Representation Learning* [29]. Feature Learning is a method, that allows a machine to learn from raw data. The machine takes the raw data as input and learns automatically the needed features to solve the machine learning problem. For example, in image classification is the raw data an image, which is represented by an array of pixels. These arrays are fed to the ConvNet, and the ConvNet learns useful features from these images to solve the machine learning problem. In the first layer, the ConvNet could be learn to detect edges, in the next layer the arrangements of the edges and so on [29]. More generally formulated, in the lower layers are basic concepts learned, and with every further layer, die concepts are combined to higher concepts.

In traditional image classification, there must be extracted useful features from the image, which are then used on machine learning algorithms like an SVM [11]. These handcrafted features must be carefully chosen, otherwise, the classification has a bad performance.

This problem of carefully chosen features is not present in NN, but they have other difficulties. One of the difficulties is the right choice of hyperparameters and of the architecture. These have direct influence on the performance.

1.2 Goal of this Thesis

In this thesis, we use *Convolution Neural Networks* (ConvNets) to predict the vehicle pose and the vehicle class on images. For every task, we use a classifier with eight different classes. We make the assumption, that the images shows only one vehicle, and that the vehicle is depicted in the center of the image. In real applications, this will be never the case. Therefore, it could be used another ConvNet to detect vehicles on images. The detected vehicles can then applied to the ConvNet of this thesis to produce the two attributes for these vehicles. But this is not part of this thesis.

There is not only one ConvNet architecture, which works well on all problems. Therefore, a ConvNet must be developed for a problem. For this thesis, we develop our own ConvNet architecture, which can be applied to the two tasks. We had the goal, that the ConvNet works on small images and the training should be fast, because we have only a limited resource of time and computation power. To compare the performance of our developed architecture, we compare the performance on three further ConvNets architectures, which are Inception-v3 [52], ResNet-50 [18] and VGG16 [47]. To use the three architectures, we are using a transfer-learning approach, to train the architectures for this two tasks.

Furthermore, we are interested in the existence of cooperation between the two tasks, because the appearance of the vehicle class is strongly depended on the pose of the vehicle. Therefore, the question arises, can one task help the other task to increase their performance? To answer this question, we are testing two different approaches. The first approach is the multi-task learning [4], which means, that the two tasks are simultaneously trained on the same model.

The second approach, is the concept of an expert learner [44]. The prediction of the two tasks are applied sequential. The prediction of the first task is used, to chose an expert model for the prediction of the second task. The expert model is a specially trained model on the prediction of the first task.

The further goal of this thesis was to find a suitable dataset for the ConvNet. ConvNets needs for a good classification much data. Therefore, the dataset should have enough images with labels for the two tasks. A well suited dataset for this thesis is the “Comprehensive Cars” dataset [57]. It has a huge amount of images and labels, but the labels are not perfectly labeled for our purpose. Therefore, we expanded and relabeled the labels.

1.3 Thesis Structure

In the next chapter is introduced the basis of machine learning. Furthermore, Aritifical Neural Networks and Convolution Neural Networks will be explained. In Chapter 3 are presented existing approaches for the recognition of vehicle pose and vehicle class. After the related work chapter, the “Comprehensive Cars” dataset is presented, and how the dataset and labels are prepared for this thesis. After this, in Chapter 5 is explained the preprocessing pipeline for the preparation of the image, before it is applied to the ConvNet. The preprocessing pipeline contains also the augmentation of the images. In Chapter 6 we present our ConvNet architecture, and also the ConvNet architectures for the comparison. Furthermore, we describe the approaches of multi-task learning and of the expert learner. The evaluation of the developed ConvNet is carried out in Chapter 7. Last but not least, in the last chapter is given a conclusion of this thesis and the outlook for the future work.

2 Basics of Machine Learning, Neural Networks and Convolutional Neural Networks

Artificial neural networks and especially convolutional neural networks are the main topic of this thesis. Therefore, it is important to get a good introduction into this topic. It is not possible to go deep into the matter. There are complete books which fill this topic, like the "Deep Learning" book from Goodfellow et al. [17]. So this chapter can only provides a shallow introduction to theses topics.

We start in this chapter with a short introduction to the basics of machine learning and continue with an introduction of artificial neural networks and convolutional neural networks.

2.1 Basics in Machine Learning

This Section provides a short introduction into the basics of machine learning. It introduces the main concepts of machine learning and describes shortly some typical vocabulary in machine learning. This Section is addressed to the readers, which has no experience with machine learning.

2.1.1 What is Machine Learning?

Mitchell [34] define machine learning with this definition:

> *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 the experience E.*

This is a little bit confusing definition, but after we introduce some examples for task T, performance measure P and experience E, it will be make more sense.

The *task T* describes the task which should be fulfilled with the computer program. This could be a classification, regression, clustering or some more complicated task, like driving a vehicle. In this thesis the task T is a classification problem for both tasks. We have the task to classify the pose into eight directions and to classify the vehicle class.

The *performance measure P* is a measurement of the quality of the learned task. The performance measure P is depending on the task T. For classification it could be used the accuracy, which describes the ratio of correct classified examples with respect to all examples. In regression could it be the squared distance to the correct value, and in the task of driving a vehicle, it could be the average traveled distance until occurred the first failure. Sometimes could be used a measurement which penalize some mistakes not so hard like others.

With this measurement, we could evaluate the learned program or the model. Usually, we are interested in the performance of the model on unseen data. But therefore, the model must be evaluated on an independent *test set*. This test set is separated from the training data and is never used for the training. Then we could do an estimation of the performance on unseen data.

The *experience E* describes source of the data which is used to learn. In a classification and regression task, it is a data set with input data and the desired output label to the input label. In case of the driving a vehicle it are a recorded sequence of images and steering commands. In this thesis, is the experience a huge amount of images, which shows one vehicle, and to every image exists two labels, which describe the pose and the vehicle class of the vehicle in the image.

A practical view of a machine learning system is depicted in Figure 2.1. The process is split into two phases. In the first phase, the machine learning algorithm is used to learn from the training data, and the second phase is the prediction. The training data could be labeled images of vehicles with the task to

predict the pose. The machine learning algorithm learns a model on these data and this model can then used to predict unseen images of the same task. This prediction is happen in the second phase and apply only the learned model. In our example, the learned model get images of a vehicle and must predict the pose.

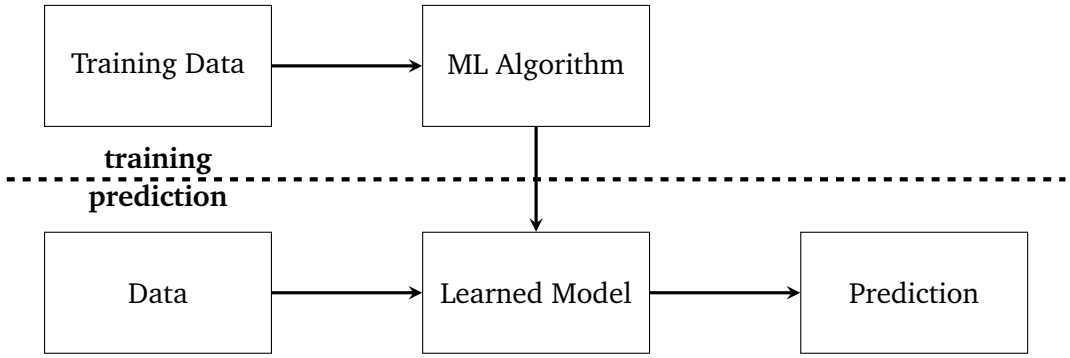


Figure 2.1: Workflow of a machine learning problem. The data will be used to train a model with a machine learning algorithm. Then, in the prediction phase, the learned model can be used to generate a prediction.

2.1.2 Learning Problems

In machine learning it can be distinguished between different learning problems. The main learning problems are:

Supervised Learning: In supervised learning, the machine learning algorithm gets a *labeled* training set. The training set consists of several examples, and every example is a pair of input data and a labeled output data. The goal is to find a general function or rule that maps the input to the desired output label. Furthermore, the mapping should be general so that unseen data is also correct mapped.

The supervised learning could be further splitted in classification and in regressions learning problems. *Classification* means, that the input of an example is divided into two or more classes. And the goal is to find a mapping of the input to this classes, even if the the input is an unknown example. For example could be used the spam-classification of emails. The classes are “spam” or “not spam”. And the machine learning algorithm should be learn from examples with the given label a function or rule, which can classify an email to these two labels.

Where as *regression* means a learning of a continuous value. Or rather, it should be learned a function, which represents the label value in dependency of the input. This could be for example the price of a house, and the input is the size of the house and the size of the property.

Unsupervised Learning: In unsupervised Learning, the machine learning gets an *unlabeld* training set. The training set has only examples, but no label. The machine learning algorithm should find a structure in the data. This could be done with clustering methods. This means, that examples should be grouped, which have similar properties.

Reinforcement Learning: In reinforcement learning the machine learning algorithm interact with an environment and must reach a certain goal. This could be to learn to playing a game, or to driving a vehicle in a simulation. The algorithm gets only information how good or bad he has interact with the environment. For example in learning to play a game, could this information the winning or losing of a game.

The classification of the supervised learning can be further distinguished into

Multi-Class: Multi-class describes only, that it exists more then two classes for one label. For example, the label weather could be sunny, rainy or cloudy, which are three classes for the label weather.

Multi-Label: Multi-label describes the fact, that one label could have multiple classes for one example. For example, a document has the classes “politics”, “sports” and “science”. But sometimes, a document can be classified to two of the classes like politics and sports.

The learning problem of this thesis is associated to the supervised learning and more precisely to a multi-class problem. Our dataset consists of serveral images, and to every images exists labels for two tasks. One task is the pose and the other task is the vehicle class. Both tasks have eight different classes into the image can be classified. The dataset will be described in Chapter 4.

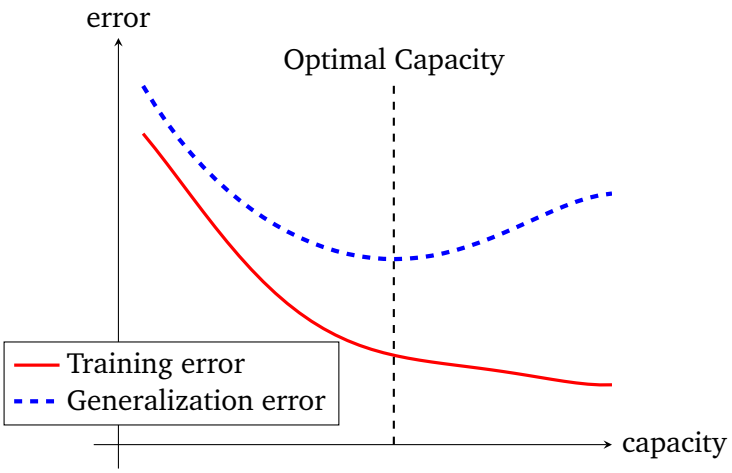


Figure 2.2: Shows typical curves for training and generalization error in dependency of the capacity of the model. Optimal capacity is reached at the minimal generalization error. Left of the optimal capacity is the model underfitting. On the right side of the optimal capacity is the model overfitting. The generalization error has typical a U-shaped curve.

2.1.3 Under- and Overfitting

A machine learning algorithm must perform well on unseen data. The ability to perform well on unseen data is called *generalization*. The generalization error is measured on a test set. If a model perform not well on unseen data, then there are two reasons. Either the model has not enough capacity and *underfit* the underlying function, or it has too much capacity and *overfit* the underlying function. If a model is underfitting, then will be the training error and also the test error high. If the model is overfitting, then is the training error low and the test error high. The goal is to find a model, which has a low generalization error. The typical curves for training and generalization error are depicted in Figure 2.2.

In Figure 2.3 are three diagrams depicted, which shows the same noisy sampling of a sinus function. The samples are used learn a model, which describes the underlying function. The left diagram learned a model with a low capacity. So the training error is high and a test set would be also produce a high test error. In the center is depicted a model with a higher capacity. This shows a good approximation of the underlying function. Nevertheless the model has a training error, because we sampled the sinus with noise. But this model will have on a test set the best generalization error compared to the other two models. The third diagram shows a model with a high capacity. The training error will be low, but the learned model is not a good approximation of the sinus function, which would result on a test set with a high error.

It is also important to use the right capacity of a model for the problem. And if we add right regularization, then it is possible, that we can use a higher capacity [17]. Because the regularization damping the capacity of the model.

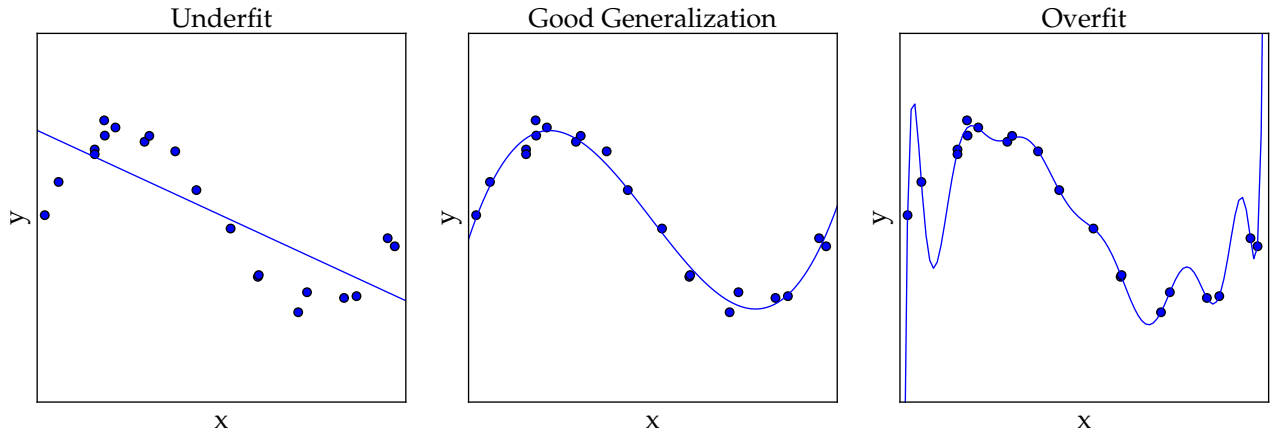


Figure 2.3: Theses three diagrams show three different models, which tries to fit the sampled points. The sampled points are sampled from a noisy sinus function. The models are described by a polynom of degree {1,4,15}. The left model underfits the underlying function. The model in the center is a good approximation of the unerlying function. The right model overfits the underlying function. It has the smallest error to fit the sampled points, but on unseen samples, it will be provide a bad error. Idea for this figure is from [17].

2.1.4 Hyperparameter and Model Selection

Hyperparamters are parameters which belongs to the machine learning algorithm. With these parameters, the behavior can be changed of the algorithm, to be precise with some of the parameters can be regulate the capacity of the model. These parameters will be not learned during the training, but will be set by the user at start of the learning process. In case of neural networks, this could be the learning rate, or the architecture of the model (number of layers or width of layers). In other words, hyperparameters are parameters, which are not learned during the learning process.

Normally, we test several models with different hyperparameters and chose the model with the lowest error. If we do this test only on the training set, then is the lowest error reached with the model, which has the highest capacity. This is not useful, how we saw in the section about overfitting. We want the model with the lowest generalization error. So the training set will be further split in a validation set. So that we have three data sets, a training, validation and a test set. The validation set is only used to measure the generalization error, and is not used for the training. The model with the smallest generalization error will then be used as the best model. But the predicted performance on the validation set has a bias, because we chose the model, which maximize the validation set. Therefore, the performance should be not measured on the validation set. That is why we have a third dataset, the test set. On this test set could we now predict the performance of the model, and this is a good approximation, how good the performance will be on unseen data.

If the data set is small, then is normally used a cross validation, instead of a hard split in training and validation set. But in this thesis, we have a huge data set and this is not needed.

2.2 Artificial Neural Networks

Artifical Neural Networks (ANN) – or short only *Neural Networks* (NN) – are inspired by neuroscience. It is the attempt of mimicking the *biological neural network* (BNN) of a brain, to solve problems in the same way. The research of ANNs are not new. They have a long history, and starts in the 1940s. The breakthrough of ANNs occurs with the introduction of *perceptrons* by Rosenblatt [], which is the basis of ANNs to this day. For a good historical survey of NNs, we recommend the paper of Schmidhuber [45]

Over the time, the ANNs had different periods of popularity and was further development. The last period is today, under the name of *Deep Learning*. After the winning of different competitions with ANNs, the popularity of ANNs are increased in the research community. The main characteristic of deep learning are the larger networks – deeper and wider – which are used to solve a problem. Goodfellow et al.[17] see two reasons for the success of deep learning:

1. More computationally power
2. Larger datasets

Today, the computers have more computationally power to train a bigger network in shorter time. So it is possible to test different structures and different hyperparameters in shorter time. Furthermore, the larger datasets helps to find a better generalization.

We would like to add a third reason for the success of deep learning. The simplicity of applying deep learning on a problem. It is not more needed to have expert knowledge on the problem domain, to extract useful features. In the most cases, the raw data need only a little preprocessing step, before the data is used to train an ANN. The ANN learns the needed features from the raw data to fulfill the task. This simplicity and the good results on different areas, make the ANNs popular in our opinion.

How the name of an ANN suggests, an ANN contains a network of neural units. Typical the ANN is structured in layers. Every layer contains a different amount of neural units, which will be shortly called *units*. The units are an abstract model of biological neurons. Every unit is with other units connected. There are a differentiation of the network types, depending on the type if interconnection. If the connection goes only in one direction to the output layer, then the network is called *feedforward neural network*. If it exists connections to the same layer or back, then the network is called *recurrent neural network*. For this thesis, we consider only feedforward neural networks.

In the following subsections, we introduce the basis of ANNs. This includes the perceptron, the feedforward neural networks and the training of them. Furthermore, we introduce modern techniques to improve ANNs, as well as typical regularizations, which are used in this thesis.

This Section is only a short introduction to Neural Networks, and the information are taken from the chapters for Neural Networks from the books of Adamy [3], Goodfellow et al. [17], and Russell and Norvig [43], if not other cited.

2.2.1 Perceptron

How in the introduction mentioned, the ANNs are inspired by biological neural networks. The perceptron is an abstract model of a single neuron and was introduced by Rosenblatt [].

In Figure 2.4 is depicted a perceptron. The perceptron has multiple inputs $(x_1, x_2, \ldots, x_n)$, and only one output y. Furthermore, the perceptron has a bias input, which is called x_0, and has the typical value of -1. For every input is derived a linear combination with the weights $(w_0, w_1, \ldots, w_n)$:

$$z = \sum_{i=0}^{n} x_i \cdot w_i \tag{2.1}$$

During the training, the weights will be learned. So it is possible to reinforce or to weaken the input and get another output.

After the linear combination is computed, the value z is inserted into the activation function $\sigma(z)$. This activation function activates the output, if the threshold is fulfilled. There are different activation functions with different properties, but for the simplicity, we using the heavyside-function:

$$\sigma(z) = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{if } z \geq 0 \end{cases} \tag{2.2}$$

With this activation function, the perceptron has the ability to linear split the hyperspace. This means, that the perceptron can only learn problems, which are linear separable. For many tasks, this is not sufficient. But with multilayer perceptrons, it is possible to solve nonlinear problems, which is as next presented.

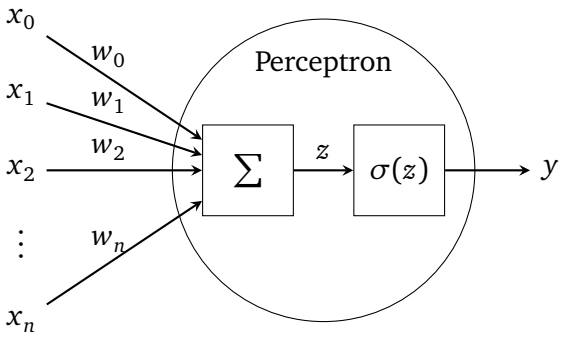


Figure 2.4: One perceptron with n inputs and one output. The circle describes the perceptron, in which is applied the linear combination and the activation function.

2.2.2 Feedforward Neural Networks

The feedforward neural networks or known under the name multilayer perceptron, are a multilayered networks of perceptrons. This means, that several perceptrons are connected in series. In Figure 2.5 is an example feedforward neural network depicted. A feedforward neural network consists of multiple *layers*, which have specific names. The first layer is called the *input layer*. The last layer is called the *output layer*. The other layers are the *hidden layers*. The hidden and output layer consists of several perceptrons, which are called *units*. The input layer contains only the values for the input. The *depth* of a network describes the number of layers, and the *width* of a layer describes the number of units. The depicted example network in Figure 2.5 has the depth of three. The width of the hidden layers are four and the width of the input and output layers are two. The bias units are depicted with the +1, and will be not count to the width of the layer.

Between every layer, the units are *fully connected* (FC). This means that every unit of layer $l+1$ has the output of all units from layer l as input. The number of links and consequently the number of weights, will fast increase, if the width of the layers are increased. If layer l has a units and layer $l-1$ has b units, then the number of weights $|W_l|$ between this layer are $|W_l| = a \cdot (b+1)$. The $+1$ comes from the bias input. So it is not uncommon that modern NNs have millions of parameter, which must be learned.

Through this concatenation of several layers, the feedforward neural networks gets the capability to solve non-linear problems, if the activation function is not linear. Would be the activation function a linear function, then has the network only the capability to solve linear problems, because the composition of linear function produce again a linear function. So the activation function should be a non-linear function. In Section 2.2.4 we show some useful activation functions.

Furthermore, a feedforward neural network has a very powerful property. With only one hidden layer with sufficient units and the right activation function, like the sigmoid function, the network can approximate arbitrarily closely every continuous functions on a closed and bounded subset of $\mathbb{R}^n$. This means that a network with a single layer has the ability to represent any function. But it is not guaranteed, that the training algorithm will find this function. It is possible, that the training algorithm is not able to find the right value for the parameters. Or the training algorithm choose the wrong function due to overfitting. Furthermore, the single layer may be infeasible large and it exists no heuristic how large the layer should be.

Nevertheless, deeper networks are empirical better and have a lower generalization error as shallow networks with one hidden layer. Goodfellow et al. [16] has showed empirical, that deeper networks

are better on the recognition of house numbers in street view imagery. Nielson [35] see the reason for the better performance of deeper networks in the ability to learn a complex hierarchy of concepts. For example in the recognition of objects in images, the deeper networks learns in the lower layers edges and corners, and compose this information to more complex structures. In other words, in lower layers will be learned simple representations, which then are composed to more complex representations, up to an approximation of the searched function.

For the designing of NNs, it is practical to use a layer as a uniquely building block. So the layers of this section, will be called *fully connected* (FC), because all units of the layer are connected with the input of the layer. Later we will present other types of layers, like convolution, pooling or dropout layer. Commonly, a layer has only one specific task. Furthermore, the layers could be seen as a function $f^{(i)}(\mathbf{x}) = \mathbf{y}$, which manipulates the input $\mathbf{x}$ to output $\mathbf{y}$. So a feedforward neural network can be described as a composition of functions with

$$f(\mathbf{x}) = f^{(n)}(\ldots(f^{(2)}(f^{(1)}(\mathbf{x})))) \tag{2.3}$$

where $f(\mathbf{x})$ describes the complete network.

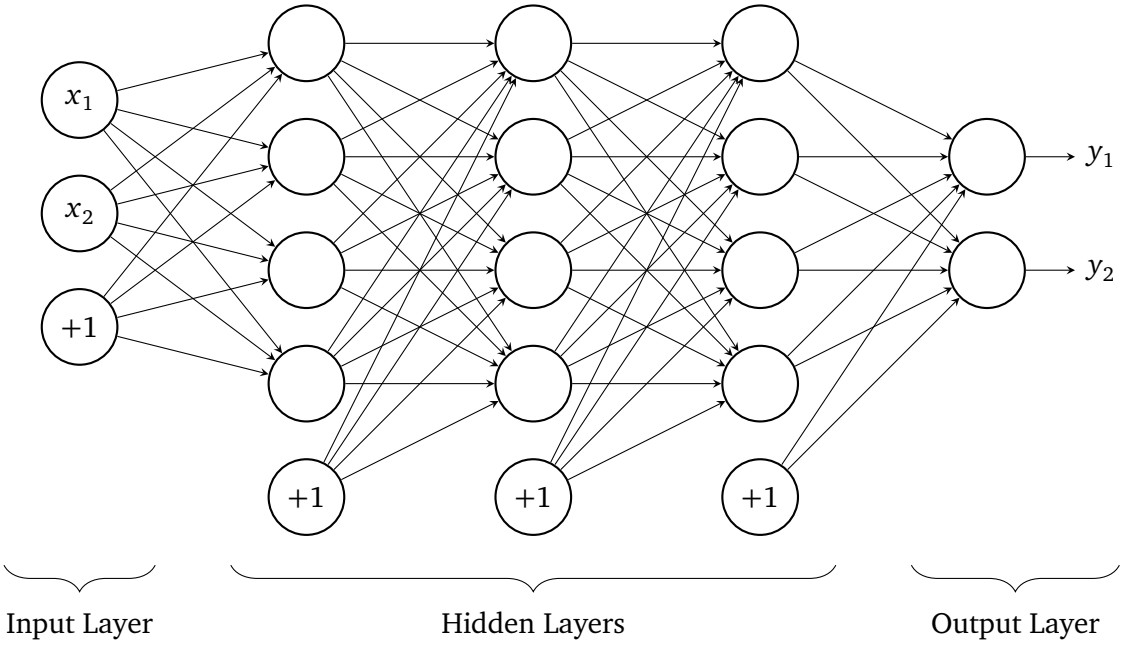


Figure 2.5

2.2.3 The Training

So far, we have not described how the learning work for ANNs. In this section, we consider only the supervised training. This means, that we have a training set of input vectors $\{\widetilde{\mathbf{x}}_n\}$, where $n = 1, \ldots, N$, with a corresponding set of output vectors $\{\widetilde{\mathbf{y}}_n\}$. The goal is for the training the function $f(\mathbf{x}; \Theta) = \mathbf{y}$ of the ANN so to adjust, that the function approximates the training data. Θ describes the parameter, which are learnable during the training. In our case, this are the weights on the links between the units. Therefore, it will be used a *loss function*. A simple loss function $J(\Theta)$ is the *mean square error* (MSE), which compute the squared difference between the $f(\widetilde{\mathbf{x}}; \Theta)$ and $\widetilde{\mathbf{y}}$. Furthermore, the loss function computes the average loss over the complete training set

$$J(\Theta) = \frac{1}{2N} \sum_{n=1}^{N} \| f(\widetilde{\mathbf{x}}_n; \Theta) - \widetilde{\mathbf{y}}_n \|^2 \tag{2.4}$$

The loss function $J(\Theta)$ is used to minimize the error between the training data and the ANN with respect to Θ. For the minimization of the loss function, we will use a *gradient descent* method, which needs the gradient of the loss function with respect to Θ. To calculate the gradient, we using the *back-propagation* algorithm, which is later in this section introduced.

The gradient descent method is an iterative optimization algorithm, which updates the weights with the help of the gradient. In the simplest version, it has the following form:

$$\Theta \leftarrow \Theta - \eta \cdot \frac{\partial J(\Theta)}{\partial \Theta} \tag{2.5}$$

where η describes the learning rate. There are some modifications of the gradient descent, which will be later introduced. The gradient descent methods find not necessarily the global minimum. The gradient descent can find a local minimum or in the worst case it can stuck in a saddle point. Therefore, another initialization of the parameter Θ can results in a different solution.

The initialization of Θ and thus from the weights of the network, is important. Glorot and Bengio [14] showed, that a randomly initialization works not so good for deep networks, and they introduced a new heuristic. But before we show the heuristic, we define a notation for the weights in Θ. Every weight in Θ will be described by $w_{j,k}^{(l)}$, where l is the layer, j describes the unit in the layer and k describes the unit of the previously layer. For a better understanding, this is in Figure 2.6 depicted for the unit j in layer l. Let's go back to the heuristic of Glorot and Bengio. The heuristic uses the layer size n_{l-1} of the previously layer and the size n_l of the actual layer l. To initialize the weights of layer l, they use the following initialization:

$$w_{j,k}^{(l)} = U(-\frac{6}{\sqrt{n_{l-1}+n_l}}, \frac{6}{\sqrt{n_{l-1}+n_l}}) \tag{2.6}$$

where $U[-a,a]$ describes a uniform distribution between (-a,a), j and k describes the weights in the layer l.

As next, we show the algorithm for the backpropagation. After this we introduce a loss function for the classification problem, and then we describe the modification of gradient descent for a faster convergence.

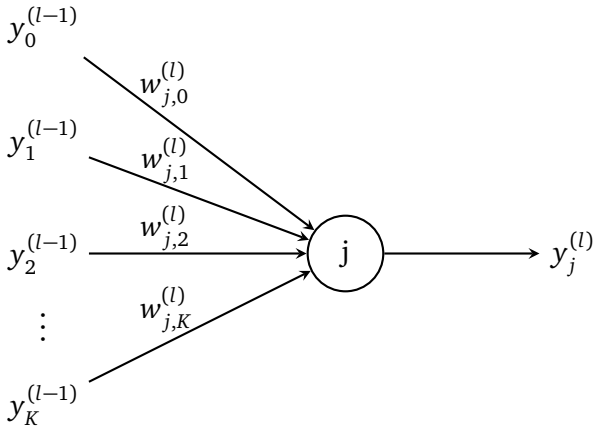


Figure 2.6: Shows one unit of the layer l. Image is adapted from [3].

Backpropagation

The backpropagation is an algorithm to propagate back the error from the loss function through the network to compute the gradient $\frac{\partial J(\Theta)}{\partial w_{j,k}^{(l)}}$. Backpropagation is not the learning algorithm of the ANN, but

rather an efficient method to compute the gradient in respect to the weights. The learning algorithm is the gradient descent, which needs the gradient of the loss function, to update the weights.

To derive for every weight the gradient, the backpropagation algorithm has the following recursively equation:

$$\frac{\partial J(\Theta)}{\partial w_{j,k}^{(l)}} = \delta_j^{(l)} \cdot y_k^{(l-1)} \tag{2.7}$$

with

$$\delta_j^{(l)} = \begin{cases} \frac{\partial J(\Theta)}{\partial y_j^{(l)}} \cdot \sigma_l'(z_j^{(l)}), & \text{if } l \text{ is the output layer} \\ \left(\sum_{i=1}^{q} \delta_j^{(l+1)} \cdot w_{i,j}^{(l+1)}\right) \cdot \sigma_l'(z_j^{(l)}), & \text{if } l \text{ is a hidden layer} \end{cases} \tag{2.8}$$

Where $y_j^{(l)}$ describes the output of unit j in layer l and q is the size of units in layer $l+1$. The function σ_l is the activation function of layer l and $z_j^{(l)} = \sum_{i=0}^{K} w_{j,i}^{(l)} \cdot y_i^{(l-1)}$ is the activation value from unit j in layer l. In Figure 2.6 is for a better understanding of the values for one unit j in layer l the values depicted.

Before the backpropagation process can be started, the forward propagation must be computed, to generate the activation values $z_j^{(l)}$ and the output of $y_j^{(l)}$. These values are needed for the backpropagation.

After all gradients are computed, the gradient descent step can be applied, with

$$w_{j,k}^{(l)} \leftarrow w_{j,k}^{(l)} - \eta \cdot \frac{\partial J(\Theta)}{\partial w_{j,k}^{(l)}} \tag{2.9}$$

And then the next iteration can be started. This is repeated, until an abort criterion is reached. This could be a maximal number of iterations, or that the loss is smaller than a predefined value.

Loss Function

In the Equation 2.4, we introduced the MSE loss function. This is one of many loss functions, which can be used. A loss function maps values of one or more variables to a single value of $\mathbb{R}$, and this value represents the loss. The loss describes the discrepancy between the function $f(\widetilde{x},\Theta)$ and the target value $\widetilde{y}$. The learning of the neural network is the reducing of the loss, which are described by the loss function. Therefore, the loss function $J(\Theta)$ is minimized with respect to Θ, which are the parameters of the neural network.

It exists different loss functions and the chose of the loss function depends on the learning problem. In regression problems, the MSE loss function is a good choice. But for the problem of classification it is not so practicable. Therefore, it exists a common loss function for multi-class problems. The name is cross-entropy loss or also known as log loss. It calculates a loss between two distributions, with

$$L(\mathbf{p},\mathbf{q}) = -\sum_{i=0}^{n} p_i \cdot \ln q_i \tag{2.10}$$

where $\mathbf{p}$ is the target distribution and $\mathbf{q}$ is distribution which is computed by the neural network.

To understand why the output of the network is now a distribution, we do an excursion to the output encoding for classification. For classification, the output is so encoded, that every class has its own output, which is represented in a vector of $\mathbf{q}$. The target label is also encoded as a vector $\mathbf{p}$ with the

property, that only one entry is 1 and all other are 0. This is called a one-hot vector. The classes are enumerates, so that every class represents an entry in the vector. This means, that the class i is represented in the vector $\mathbf{p}$ at index i. The vectors $\mathbf{p}$ and $\mathbf{q}$ are then a distribution, if the sum of all the entries are 1, and all entries are ≥ 0. For the one-hot vector $\mathbf{p}$, it is easy to see, that this property is fulfilled. But the computed vector of the neural network has not necessary this property. Therefore, it is used the *softmax activation function* at the output layer to get a distribution for the output. It changes the output so, that the required properties are fulfilled. We introduce the softmax activation function later in Section 2.2.4. The learning goal is now to minimize the loss between this two distributions. The nice side effect is, that the distribution $\mathbf{q}$ gives us the percentage how sure the neural network is, that the prediction is this classes.

After we introduced the output encoding, we would like add a note to the Equation 2.10. This is not the final loss function $J(\Theta)$, because this describes only the loss of one example in the training set. How in Equation 2.4 for the MSE, the average is computed of the cross entropy loss over all training examples.

Gradient Descent

We mentioned, that the Equation 2.5 for the gradient descent is a simple variant of gradient descent. For a better overview, we show the equation again:

$$\Theta \leftarrow \Theta - \eta \cdot \frac{\partial J(\Theta)}{\partial \Theta}$$

The gradient descent algorithm is an iterative optimization algorithm to find a local minimum of a function. It can be illustrated with the following scenario. We are on a mountain and want back to the valley. On the mountain is the visibility extremely low, because it is very foggy. Therefore, we see not the path to the valley and have only local information to find a path down to the valley. So we search the steepest descent at the current position. We go a step of size η in the direction of the steepest descent and repeat the procedure, until we reached no improvements.

In analogy, the valley is the searched minima, but the path to the valley is unknown. The search of the steepest descent is the computation of the gradient. And the update of the parameter, is the step which we are going down with the step size η. It is not ensured, that we reached the global minimum. The valley could be only a local minimum.

The choice of the learning rate η is important, because if we use a to small step size, the algorithm needs very long and it could stuck in a local minima. If η is to big, it could happen, that we over jump the valley with the best solution. So it is one of the main hyperparameter of a neural network, and should be carefully The starting point is also important, because another starting point results in a different path, which could find a better (local) minima.

Stochastic Gradient Descent

For the gradient descent exists some interesting extensions. The first and important extension is the *stochastic gradient descent* (SGD) with *mini-batches*. It does the gradient descent iteration with a small subset of examples from the training set, instead the complete training set. The normal gradient descent is very unpractical regarding the computation time and update speed. For one update, it processes the complete training set, which can have millions of examples. This means, it must compute millions of forward propagations, before it can compute one backpropagation step for the update of the weights. With SGD with mini-batches, it does with a small size of examples a weight update, but with this huge amount of updates it will be find the right direction, which minimize the loss. Some of the updates will be go in the wrong direction, but if the majority goes in the right direction, then the loss will be minimized. The size of the mini-batches ranging from one to a few hundred. With a low mini-batch size, the training time is increased enormously, because for every example is computed a forward propagation and a backpropagation, and the learning rate η must be also small, because the gradient has a high

variance. So the mini-batch is mostly set to a higher value with a power of two. But it is very interesting, that the best results are often reached with a mini-batch size of one [17]. And that a huge mini-batch size (over 1.000) result's in a (bad) sharp minima [23]. It is common, that the mini-batch size is only called *batch size*.

At this point, it is appropriated to introduce two common terms in the training of neural networks:

Iteration: One iteration describes one update with the gradient descent. In other words, it is one learning step.

Epoch: One epoch describes one pass through the complete training set. Normally, one epoch consists of several iterations, because the size of the training set is much bigger than the batch size.

Momentum Update and Nesterov Momentum

A further extension of the gradient descent is the momentum update. The learning with normal gradient descent is sometimes slow, but can accelerate with the momentum method. The path of the normal gradient descent goes mostly in zigzag lines to the minimum. With the help of the momentum method is tried so solve this problem, by using a decaying moving average of the past gradients. This works good for gradients with noise or for gradient with small values which have a consistent direction. The formula for the gradient descent with momentum update is the following

$$\begin{aligned} v &\leftarrow \mu v - \eta \cdot \frac{\partial J(\Theta)}{\partial \Theta}, \\ \Theta &\leftarrow \Theta + v. \end{aligned} \tag{2.11}$$

It introduces a further hyperparameter $\mu \in [0; 1)$, which describes the decaying rate of the momentum. A higher value slows down the decaying rate. Commonly values for μ are 0.5, 0.9 and 0.99 [17]. The hyperparameter μ is after Sutskever et al. [50] a further critical parameter, which should be tuned.

The momentum helps to stay in the direction of the previously gradient updates. In analogy to the mountain, it is better to go further in the direction of the previously directions, as to jump from on direction to another. If the direction has changed, then will be changed the momentum to, but slowly. Because the directions of the previously directions are added to the momentum.

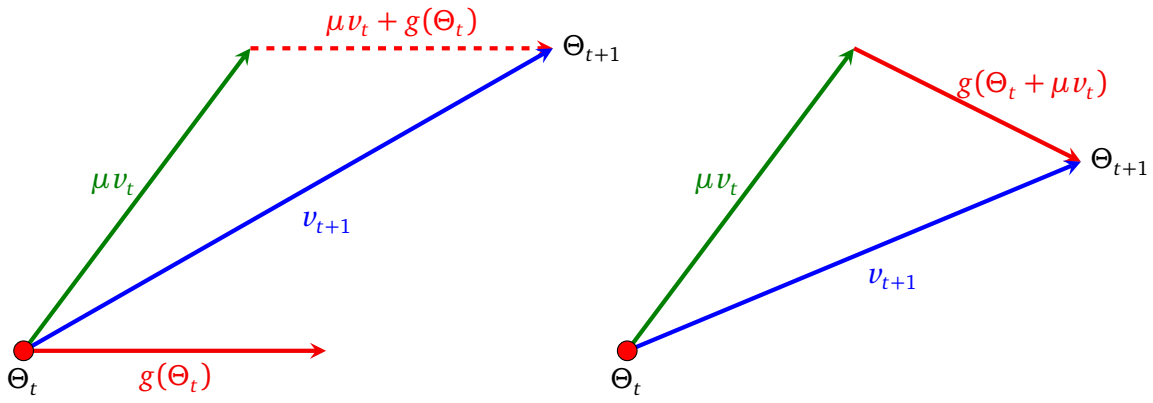
The momentum update is depicted for a 2D-space problem in Figure 2.7a. The momentum μv_t at update time t, has the direction of the green arrow, which are the directions of the previously gradient descent updates. The gradient $-\eta \frac{\partial J(\Theta_t)}{\partial \Theta}$ has the direction of the red arrow. Without the momentum update, the update of Θ_{t+1} would cause at the end of the red arrow. But with the moment update, the update of Θ_{t+1} results in the blue arrow.

A further extension of the momentum update is the *Nesterov momentum*. It uses the knowledge, that we will go in the direction of the momentum μv, and therefore it computes the gradient at the position $\Theta + \mu v$ and not at the actually position of Θ. This results in the following equations

$$\begin{aligned} v &\leftarrow \mu v - \eta \cdot \frac{\partial J(\Theta + \mu v)}{\partial \Theta}, \\ \Theta &\leftarrow \Theta + v. \end{aligned} \tag{2.12}$$

Please take care, that the only difference between this Equation and the Equation 2.11 is the position of evaluation of the gradient.

The Nesterov momentum could be seen as a correction factor of the momentum. If the update of the momentum will be to wide, then it will be correct this update. In Figure 2.7 are both momentum updates depicted for the comparison.



(a) Momentum update

(b) Nesterov momentum update

Figure 2.7: It shows the difference between momentum update and Nesterov momentum update. In momentum update, the gradient will be evaluated at the current position (red circle). Instead, the nesterov momentum does the evaluation of the gradient at the tip of the green arrow. The gradient of $J(\Theta)$ with respect to Θ at position Θ_t is represented with the function $g(\Theta_t) = -\eta \frac{\partial J(\Theta_t)}{\partial \Theta}$. The index t and $t+1$ describes the variables before and after the update. Image is adapted from [50]

Adaptive Gradient Descent

For completeness, we want mention, that it exists adaptive gradient descent methods, which can also used as a SGD. Adaptive means, that they have not a fixed learning rate η. Every parameter has its own learning rate, which is updated during updates of the gradient. The learning rate for every parameter depends on the value of the gradient in the update. The most commonly used adaptive algorithms are *AdaGrad*, *RMSProp* and *Adam*.

2.2.4 Activation Functions

The activation function $\sigma(z)$ is the main part of a neural network, which gives the network the power to solve non-linear problems. If the units have no activation function or only a linear activation function, then the neural network could only solve linear problems. No matter how deep the neural network is. So the activation function should have a non-linear characteristic. A further property for the activation function is, that it must be continously differentiable. Otherwise, the backpropagation algorithm will not work, because it cannot compute the gradient.

In the beginnings of the research of neural networks, there were used saturated activation functions. This means that the activation was limited to (0,1) or (-1,1). As activation functions were used the *sigmoid* or *tanh* function. Both have the problem, that by small or big values of z the value of the gradient goes to zero, and the convergence of the learning decreases. In other words, the training of the neural network needs much more time. Another problem is the vanishing or exploding of the gradient, if the network is depth. The vanishing gradient occurs in earlier layers, because through the backpropagation will often be multiplied with small values. Many multiplications with small values between 0 and 1 tends toward zero. The exploding gradient, can happen, if the weights are big, and the activation is near 0. At this point the derivative of sigmoid and tanh has reached their maximum.

With the time, new activation functions are developed, like the ReLU or ELU. In this section, we want to show three activation functions, which are used in this thesis. These are ReLU, ELU and Softmax.

ReLU - Rectifiere Linear Unit

Actually, the *Rectifiere Linear Unit* (ReLU) is the most used activation function for neural networks. The formula for this function is simple

$$\sigma(z) = \max(0,z) = \begin{cases} 0, & \text{if } z < 0 \\ z, & \text{if } z \geq 0 \end{cases}. \tag{2.13}$$

This means, that the function is 0, if z is negative otherwise the value is z. The ReLU activation is in Figure 2.8 depicted. The interesting fact is, that the function is for positive values linear and not saturated. A further property, which makes ReLU so popular is the fast computation of the derivative. The derivative is either 0 (at negative values) or 1 (at positive values). Through this, the vanishing of the gradient is not more a problem, because a multiplication with 1 doesn't change the error.

Glorot et al. [15] showed, that ReLU has without pre-training better results as tanh or softplus. Furthermore, Krizhevsky et al. [26] have applied ReLU in combination with Convolutional Networks and have a 6 time faster convergence as with the same network with the activation function tanh.

Batch normalization with ReLU is heavily used in modern network architectures [18, 19, 52, 53]. We introduce batch normalization in the next section, but ReLU with batch normalization increases the learning speed of the network. This comes from the normalization of the batch normalization, so that the mean is zero. It is known, that the input with zero mean increases the convergence [20].

ReLU has a problem with dying units. This is the case, if no example in the training set can activate the unit. If this happens, then is the unit dead and cannot more activated, because the gradient is forever 0 and the weights are not more changeable for this unit. To counteract the dying units, it is possible to use leaky ReLU, which change the behavior for negative activation. It adds for the negative z a linear component, with the slope $0 \leq \alpha \leq 1$. So the gradient is never zero.

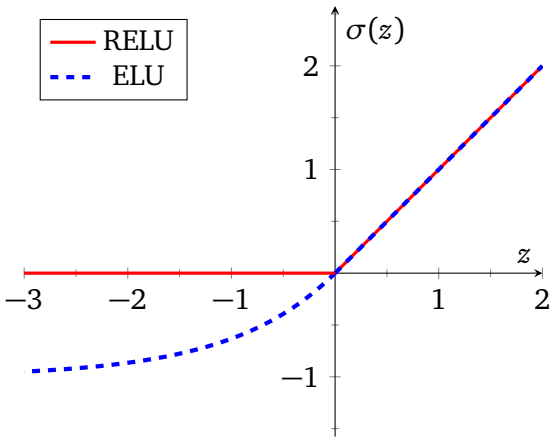


Figure 2.8: Diagram for the activation functions of ReLU and ELU with $\alpha = 1$.

ELU - Exponential Linear Unit

Another way to counteract the dying units is with the *Exponential Linear Unit* (ELU) [9]. The formula for this activation function is

$$\sigma(z) = \begin{cases} \alpha \cdot (e^z - 1), & \text{if } z < 0 \\ z, & \text{if } z \geq 0 \end{cases} \tag{2.14}$$

where $\alpha > 0$ is a hyperparameter, which is normally set to 1.

How ReLU, it has a linear function for positive values. But for negative values, it has an exponential function, which converged for $z \to -\infty$ against $-\alpha$ (see Figure 2.8). This helps, that the gradient will be

never zero and counteract the dying units, which can appear in ReLU. Furthermore, through the negative activation, the mean activation is near zero, and this helps for a faster convergence of the network, because the input for the next layer has a zero mean. The authors of ELU [9] had in experiments a significantly better outcome as with other activation functions, like ReLU with batch normalization. Furthermore, they found no difference between ELU with or without batch normalization.

The drawback of ELU is the computation of the exponential function. This is not so cheap like in ReLU with only applying the max function. For the derivative of ELU, the exponential function must be also computed

$$\sigma'(z) = \begin{cases} \sigma(z) + \alpha, & \text{if } z < 0 \\ 1, & \text{if } z \geq 0. \end{cases} \tag{2.15}$$

Softmax

In the section about the loss function, we have used the *softmax* activation for the encoding of the output. This is the main application of the softmax activation function, to represent a probability distribution over K classes, and therefore it is only used in the output layer. The softmax is applied on the complete output layer and needed so all linear combination $\mathbf{z} = [z_1, z_2, \ldots, z_K]^T$ of all units. The formula for softmax is the following

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad \text{for } j = 1, \ldots, K. \tag{2.16}$$

It scales every output of $\sigma(\mathbf{z})_j$ to a range between 0 and 1, and the sum of $\sum_{j=1}^{K} \sigma(\mathbf{z})_j = 1$. This represents a probability distribution. With the exponential function, the largest value in $\mathbf{z}$ will be highlighted and the other values will be suppressed. For example, let $\mathbf{z} = [1, 2, 3, 2, 1]^T$, then is the activation $\sigma(\mathbf{z}) = [0.07, 0.18, 0.50, 0.18, 0.07]^T$. The value 3 get much more of the available space, as the other values. If we would only scale the vector $\mathbf{z}$ to one, with a division of the length of the vector $\mathbf{z}$, we would get $\frac{\mathbf{z}}{|\mathbf{z}|} = [0.11, 0.22, 0.33, 0.22, 0.11]^T$. Therefore the softmax gives the largest value in $\mathbf{z}$ a bigger reward.

2.2.5 Regularization

Regularization are methods to control the overfitting or rather to improve the generalization error. There are different methods, how to apply regularization. Some of the methods are common approaches in machine learning, and other are only usable for neural networks. Goodfellow et al. [17] argument, that large models, that has been appropriately regularized, find the best fitting model. At this point, we show four regularization methods, which are used in this thesis.

L2-Regularization

L2-Regularization is a regularization methods, which is added to the loss function $J(\Theta)$. It penalizes the weights $\mathbf{w}$ from Θ, but not the bias weights. The regularization term $\Omega(\Theta) = \lambda \frac{1}{2} \|\mathbf{w}\|^2$ is added to the loss function. This results in a new loss function

$$\widetilde{J}(\Theta) = J(\Theta) + \Omega(\Theta) \tag{2.17}$$

which is used to minimize the loss of the neural network. Furthermore, it is added a further hyperparameter λ, which represents the strength of the regularization.

But what does this regularization? Through the adding of the quadratic weight to the loss function, are weights with a high value penalized, because they increases the loss. So, weights with a high value are bad for the loss. In other words, the L2-regularization brings the weights closer to the zero.

Early Stopping

On training of large models, normally the training and validation error decreases over the time, but at one point the validation error starts to increase. At this point the model is starting to overfit, and learn specific properties of the training set. To stop at this point, it is applied the method of *Early stopping*. It returns the model, which has the lowest validation error. Therefore, the training needs a validation set, to evaluate periodically the validation error. The normal period is after every epoch.

But how could we ensure, that this increasing was not caused by noise? The training is not stopped after the first increasing of the validation error. The network is further trained until a threshold of "number of epochs without improvements" is reached. Through the evaluation of further epochs, we get the trend of the validation error for more training. For example, if 10 times in a row, the validation error has no improvements compared to the best validation error, then is the training stopped, and the model with the best validation error is returned.

Batch Normalization

The training of a network changes the weights on every layer. This change has the effect, that the input distribution changes during updates of previously layers. The authors of *batch normalization* [20] called this effect *internal covariate shift*. To counteract the change of the distribution during the learning, they introduced the batch normalization. Every mini-batch, which is inserted in the network, will be on every layer normalized on the input of the previously layer. The formula for the normalization is the following

$$\mu \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.18}$$

$$\sigma^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu)^2 \tag{2.19}$$

$$\widetilde{x}_i \leftarrow \frac{x_i-\mu}{\sqrt{\sigma^2+\epsilon}} \tag{2.20}$$

where μ and σ describe the mean and the standard deviation, $\widetilde{x}_i$ is the normalized value of the input. The value m describe the size of the mini-batch. The value $\widetilde{\mathbf{x}}$ has then a mean of 0 and a standard deviation of 1. The batch normalization will be applied after the linear combination of a unit.

To not lost the representation power of the units, the value $\widetilde{x}_i$ will be scaled and shifted

$$y_i \leftarrow \delta\widetilde{x}_i + \beta \tag{2.21}$$

where δ and β are learned parameter during the training. So the value y_i could have any mean and standard deviation. This is useful in the case of using sigmoid as the activation function. Without the scaling and shifting, the activation function will be act as an almost linear function, because through the normalization, the input is scaled to a range, on which the sigmoid function is almost linear.

At test time, μ and σ are replaced with the average, which is collected during the training. So it is possible to predict a single example, without to have a minibatch.

Through the applying of batch normalization, the learning rate can be increased, and this results in a faster training. Furthermore, the accuracy is increasing compared to the same network without batch normalization [20].

The batch normalization helps the activation function ReLU to learn faster and have a better performance, but on the activation function ELU there are no difference between with or without batch normalization [9].

Dropout

Dropout was developed by Srivastava et al. [49] and is a regularization method for neural networks. The key idea is, that units will be randomly dropped for every iteration of the training. This means,

that the dropped units and their connection are zero. This should prevent the co-adapting of the units. During the testing the dropout is not available, and all units are used for the predicting.

Srivastava et al. [49] tested dropout on different datasets and they had on all an improvement of performance. The drawback of dropout is the increasing training time, because not all weights a trained in one iteration. If one unit is dropped, then are all depending gradients are zero and therefore the corresponding weight will be not updated.

Dropout could be seen as a training of a subset of the model. Every iteration builds a different version of the model, and every weight is updated with another set of weights. This prevents the co-adapting of the units, because a unit cannot more rely that another unit is available.

A similar dropout is the *Spatial Dropout*, which is introduced by Tompson et al. [55]. It is used in convolution neural networks to drop complete feature maps and not only single units. This brings us to the next section, where we introduce convolution neural networks.

2.3 Convolutional Neural Networks

Convolution Neural Networks (ConvNets) are specialized Artificial Neural Networks. ConvNets are well suited for the processing of images, but the same concepts are adaptable for other fields, like audio or video. In this section, we describe the ConvNets for the processing of images.

A ConvNet consists of multiple convolution and pooling layers. At the end follows normally a fully connected layer. A pooling layer is applied after one or multiple convolution layers. The convolution layers have the task to extract useful features from the input, which results in multiple feature maps. The pooling layer reduces the spatial size of these feature maps.

In image processing is often applied convolution on images. It manipulates an image or extracts features from an image. Hence, there are different filters with different kernel sizes. For example, the Sobel filter, which has a kernel size of 3×3 size, will be applied with convolution on an image. The result is an image, which shows the edges of the original image. This idea of filters for feature extraction is translated to ConvNets. But instead of using hard coded filters, the ConvNet learns these filters. But to apply this on neural networks, it must be changed some structures on the neural networks.

In the next sections, we will describe the convolution and the pooling layer. The information for the ConvNets are taken from the book of Goodfellow et al. [17] and the Stanford Lecture CS231n [22].

2.3.1 Convolution Layer

A convolution layer consists of units as well as a normal neural network, but with a different arrangement and a different connectionism of the units. The main differences to the neural networks are:

- Three dimensional arrangement of the units, instead of 1 dimension
- Weight sharing
- Local connectivity

The *three dimensional arrangement* comes from the image. A colored image has normally three channels (red, green, blue), and every channel is described by a two dimensional matrix. Therefore, the input of the ConvNet is a three dimensional matrix. The output of a convolution layer is again a three dimensional matrix, with feature maps of two dimensions and this x times for the number of filters for this layer. Every filter produces a feature map.

Weight sharing means, that the same weights are used for multiple output units. Through this, the ConvNet gets the property, that the features are invariant against translation. This means, that a feature can be found on the complete input. To compare this with the Sobel filter, the weights are the filter, and this filter will be used on the complete input to generate the output.

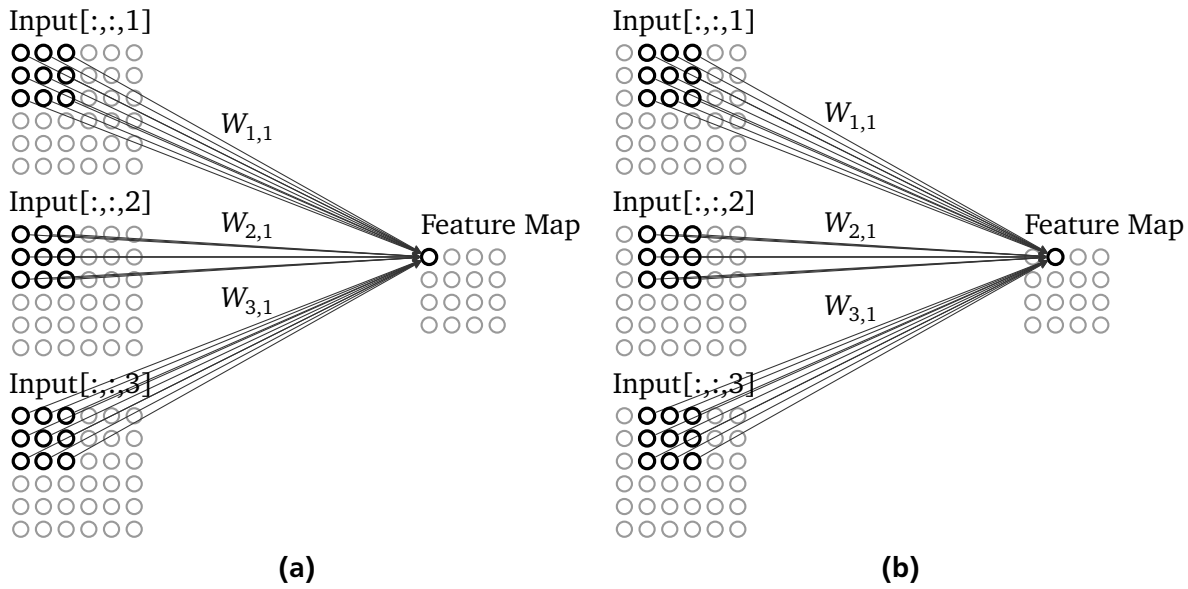


Figure 2.9: The two figures show the connection between the input and two output units of the convolution layer. Furthermore, the convolution layer has only one feature map for the output. The activation function and the bias are omitted for a better overview. The local connectivity is depicted in the two figures, with a kernel size of 3×3. (a) and (b) shares the same weights for the computation of the feature map. For another feature map, it would be used new weights. $W_{i,o}$, describes a 3×3 weight matrix, which is used between the input channel i and the output channel o. The output channel is called feature map. The idea for this representation is from [35].

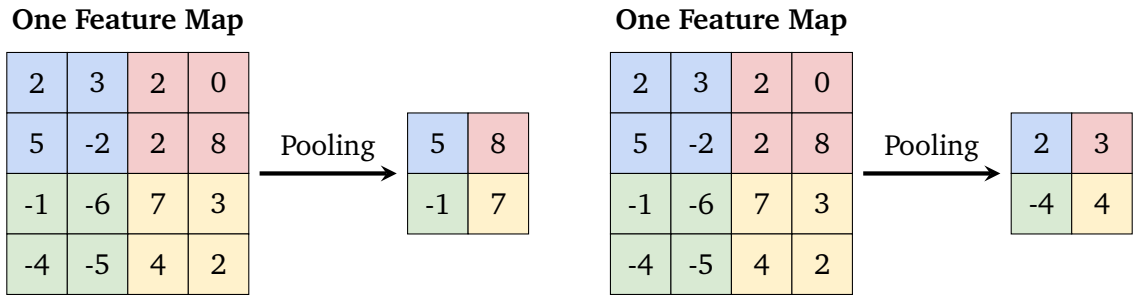
Local connectivity means, that not all units of the input are connected with the output unit. The size of the local connectivity is described by the kernel size.

To get all this together, in Figure 2.9 is provided an example. The two figures show the connection between the input and two output units of the feature map. The input is showed with three channels. This could be the color channels R,G,B of an image, or the feature maps of a previously layer. The input is described by a three dimensional matrix. As output is depicted one feature map. For more feature maps is the behavior similar. The activation function and the bias are omitted for this example, but would be also applied on the units. The bias is also shared, which means, that one bias weight is used for all units in one feature map. The weights $W_{i,o}$ are a 3×3 matrix, which describe one kernel, between $Input[:,:,i]$ and the $featureMap[:,:,o]$. The weights $W_{:,o}$, describe one filter and are shared for one feature map. This means, that the same weights are used for the computation of all units in one feature map.

The spatial size of the feature maps are smaller than the spatial size of the input channels. In our example is the input spatial size 6×6, and the feature map only 4×4. This comes from the kernel size, because the kernel could be only applied on 16 different positions in the input channel. To avoid this effect and get the same spatial size for the feature map, the input must be extended with a zero-padding.

The number of units for the output, depends on the spatial size of the input and the number of filters. The number of filters describe how much feature maps should be generated. But the spatial size of the feature map and so the number of units, depends on the input spatial size.

Through the weight sharing and local connectivity, the number of weights decreases exponentially compared to a fully connected layer with similar number of units. Let K the width and height of a kernel, I the number of input channels and F the number of filters, which is equivalent to the number of feature maps. Then has the layer only $K^2 \cdot I \cdot F$ number of weights. In our example of Figure 2.9, is $K = 3$, $I = 3$ and $F = 1$. Hence, the number of weights are $3^2 \cdot 3 \cdot 1 = 27$. In a fully connected network, the number of weights will be $6 \cdot 6 \cdot 3 \cdot 4 \cdot 4 = 1728$, which is much more!



(a) Max-Pooling **(b)** Average-Pooling

Figure 2.10: Example for the max-pooling and the average-pooling with a filter size of 2×2 and a stride of 2×2. Adapted from the Stanford Lecture CS231n [22].

Another parameter for a convolution layer, besides the number of filters and kernel size, is the stride. The stride is the size of sliding over the input channel. In our example, we have a stride of (1,1), and the local connectivity is slide one unit to right or down, for the next output unit. If we increase the stride to (2,2), then we would slide the local connectivity two units to the right or down, to calculate the next output unit. This has the effect, that the spatial dimension is reduced for the feature map.

But where is happened the *convolution*? From the previously point of view, it is not obvious where the convolution is applied. But mathematically, the convolution is applied between Input$[:,:,i]$ and the weights $W_{i,o}$. Let us formulate, that Input$[:,:,i] = V_i$ and V_i is a two dimensional matrix, which describes the i-th channel from the input. And Z_o is also a two dimensional matrix, which describes the o-th output channel or called feature map. $W_{i,o}$ describes a kernel with the size of $k \times k$. Then is the o-th feature map calculated with

$$Z_o = \sigma\left(\sum_{i=0} V_i * W_{i,o} + b_o \mathbf{1}\right) \tag{2.22}$$

where the asterisk $*$ denotes the convolution, σ the activation function and b_o the shared bias for the feature map, which is described by a scalar value. The above formula is not a scalar! Behind V_i, $W_{i,o}$ and $\mathbf{1}$ are 2D-matrices.

The discrete 2D-convolution is defined for two matrices I,K at point (i,j) with

$$(K * I)_{i,j} = \sum_m \sum_n I_{i-m,j-n} K_{m,n} \,. \tag{2.23}$$

The convolution in a convolution layer are multiple convolutions, which are added up, which we can see in Equation 2.22. Without the addition of the convolutions of the input channels, we would not get a combination of features, which are essential for a ConvNet.

2.3.2 Pooling Layer

The pooling layer is normally applied after some convolution layers. It has the function to reduce the spatial size of the feature maps. This reduce the amount of parameters and the computation time, because the next layer get a smaller version of the feature maps. In other words, it down samples the feature maps. With one pooling with the stride size of 2×2, the spatial dimension of the feature maps is reduced of 75%. The pooling works independently on every feature map and has no activation function or weights to learn. Furthermore, the pooling layer helps to be invariant to small translations of the input, because a small translation has mostly no change on the values of the outputs of the pooling layer.

The most common pooling layers are the max- and average-pooling. Both pooling layers take as hyperparameters the filter size and the stride. The max-pooling computes the maximum, which lies in the receptive field of the filter, and the average-pooling computes the average. An example for both poolings are depicted in Figure 2.10. As recommend values for the filter size and the stride are usually for both the size of 2×2. A stride of 1×1 has no down sampling character, and therefore it is not common. A higher size for the stride has normally a negative effect, because the reduction is to much (the filter size has at the minimum the same size like the stride). A filter size of 3×3 and a stride of 2×2 has sometimes a performance gain [26].

If the feature map is not divisible by the stride without remainder, then it will be cut off the last row or column, and we lost information. In this case, it is better to add a padding. Another approach to avoid this problem is to use an input size of power of two for the network. And add to all convolutions a padding. So will be all feature maps divisible by the stride size, if the stride was 2×2.

There are further approaches for the pooling, like to use a pooling with max+average. Another approach is to use no pooling, and instead to use a convolution with a stride [48]. This reduce also the size of the feature maps, and has the benefit, that a further non-linearity is used.

3 Related Work

We are not the first, who tries to predict the vehicle pose and the vehicle class. Therefore, this chapter gives an overview of other works in this field. This chapter is divided in three parts. The first two parts describes related works, which have tried to predicted either the vehicle pose or the vehicle class. The third part reports related work, which have tried to predict both classes in one model.

3.1 Vehicle Pose

The most related works for the prediction of the vehicle pose, have further the task to detect the vehicle on the image. Firstly will be detect the vehicle, and then will be tried to predict the pose on these images. The difference lies only in the used approach.

In [37], they used a three step approach for the detection and prediction of vehicle on an image. The first step is to estimate a bounding box. To this bounding box will be predicted the pose with a Naive Bayes classifier. They use spatial pyramid histograms as features. They categorize the pose into 16 different classes. The last step is to predict if a specific object is within the bounding box. They tested their approach against a simple sliding window approach, and have showed, that their approach is better. For the pose estimation, they have reached good results. They have the problem, that some poses are estimated with the opposite side, because there are a similar vehicle appearance.

Missing content because of deadline of this thesis. Tip to other students, write your related work as early as possible!

3.2 Vehicle Class

The most related work, has a low number of classes for the vehicle class. Mostly, they distinguished between passenger cars, SUVs, vans, trucks and motorcycles. Furthermore, they the classification was applied on surveillance cameras images, which shows the vehicle only from one pose. With this setting, it is easy to distinguish different vehicle types.

In [21], they used partial gabor filter banks to distinguish the vehicle class. As vehicle classes, they used sedan, hatchback, van, bus and truck. The images shows the vehicle only from the side. With this setting, they reached a performance of around 95%.

The researcher Yang et al. [57], which provides the dataset for this thesis, have on their dataset tested the prediction of the body class. The body class contains 12 classes, but the classes are imbalanced. This means that there are classes with very few samples, and other classes which have much more. In the dataset are the classes SUV and sedan overrepresented. Both together counts over 55% of all samples. They trained for every pose a ConvNet to predict the body class. Unfortunately, they have never tested to train one ConvNet on all viewpoints. The accuracy of the predictions lies between 54% and 62%. The prediction of the body class from the viewpoint front was the worst. But the prediction from a side viewpoint worked better.

Missing content because of deadline of this thesis. Tip to other students, write your related work as early as possible!

3.3 Vehicle Pose and Vehicle Class

There are only a few works, which have tried to predict the vehicle pose and a vehicle class. In [61] is described a framework for detection of vehicles and adding of attributes to these vehicles. The attributes

are the vehicle pose, the vehicle body class and the color of the vehicle. They used two ConvNets. One was for the detection of the vehicle, and the other for the classification of the three attributes. They tested single- and multi-task learning for the three attributes and have observed, that multi-task learning has a benefit. They used the CompCar Dataset [57], which is the same like we used for this thesis. But they use the default labels for pose and for body class. For the color label, they labeld 10.000 images. Furthermore, they added for the training 200,000 negative examples, and for the positive examples. For the body class they discarded the half classes, to improve the accuracy. So they used only the biggest six classes for the body class. For the testing, they split randomly the dataset. We saw in a default random splitting a problem, which we state in Chapter 4.2.3. The results show, that the pose can be good classified, if the resolution is high. The accuracy lies around 98%. The body class reach for twelve classes only 69%, but reduced to six classes, the accuracy goes up to 95%. They used for the accuracy the overall accuracy, and it is questionable, whether the results are so good, because the three biggest classes in body class, makes around three quarters of all samples. A showing of the confusion matrix, or the evaluation with the macro F1-Score would be helpful.

In [40] is presented a 3D shape reconstruction of vehicles, which only use 2D images. They used one ConvNet to segment the vehicle and to another ConvNet for the prediction of the pose. They used a fine granular division of the pose in 24 and 72 bins. Furthermore, they calculated a latent space from CAD models, which is then used with the segmentation and the pose to optimize the 3D shape. They reached for the pose a median error of around 7 degrees for approach with 72 bins. But during the optimization of the shape, they could further improve the pose get a median error of around 4 degrees. The vehicle class is not direct calculated, but through the good optimization of the 3D shape (from 1459 optimizations are only 5 diverged), we think, it should be possible to find a CAD model, which has the smallest distance to the 3D Shape. And this CAD model represents one of the vehicle class. Therefore, we had added this work to both classes.

4 The Dataset

One important step for this work, is the right choice of the dataset. The dataset should contain images of different vehicles and show the vehicles from different poses. For the classification task it is needful to have the labels for the vehicle pose and for the vehicle class. Furthermore, the dataset should have extensive images of different vehicles to have enough samples for the training of a convolutional neural network.

We found many different datasets, but the most provide only one label for one of the two tasks. Either the pose is available [12, 13, 31, 37] or only a simple body classification [25]. We found two datasets which provides both. One is the "Pascal3D" [56] dataset and the other is the "The Comprehensive Cars" [57] dataset. The Pascal3D has a huge amount of annotations per sample, like the orientation in degree, elevation (angle of the camera position), the distance of the camera and more useful information to describe the object in a 3D environment. Furthermore, it has a vehicle class label for every vehicle. But the disadvantage of the dataset is, that it is missing information about same vehicle-models. This information could be helpful for the splitting in train- and test-sets, to test the generalization of unknown vehicles. In Section 4.2.3, we go deeper into this problem and why this missing information could be a problem. A further disadvantage is the amount of samples, so we decided against this dataset.

Therefore, we are using "The Comprehensive Cars" (CompCars) Dataset from Yang et al. [57] for this thesis. In the following sections, we represent this dataset and how we prepare this dataset for our usage. The dataset fulfills almost the needed requirements, but the labels have some missing information. The pose labels does not make a difference between left and right and the vehicle class is a mix between vehicle class and body class. But the labels could be easily expanded to our desires. This process will be described in the second part of this chapter.

4.1 The Comprehensive Cars (CompCars) Dataset

In this thesis, the Comprehensive Cars (CompCars) Dataset from Yang et al. [57] is used. It is one of the biggest Datasets for categorization of vehicles. It contains 214,345 images of vehicles and vehicle parts from 1,687 different vehicle models. Furthermore, the dataset has a rich amount of labels, like a simple pose label and a body class label. With this amount of images and the provided labels, make this dataset useful for this work.

Type of Image	Number of Images
Web-Nature	136,727
Vehicle Parts	27,618
Surveillance	50.000

Table 4.1: Overview for the number of images for the three different types in the CompCars Dataset.

The CompCars dataset consists of three different types of images. The first type are images, which shows mostly only one vehicle, and is collected from the web. This means that the data are collected from automotive forums, public websites and search engines. The second type are images of vehicle parts. The vehicle parts shows detailed images of the exterior (headlight, taillight, fog light and air intake) and for the interior (console, steering wheel, dashboard and gear lever). The images are also collected from the web. The third type of images are recorded surveillance images and shows the vehicle only from the front. The surveillance images have a big variance of light conditions, because the image

are recorded to every daytime. The number of images for the individual types of images are depicted in the Table 4.1. For this work, we only use the first type of images, the web-nature type. In Figure 4.1 are depicted example images of the web-nature dataset, to get a feeling, how the vehicles' appearance on the images.

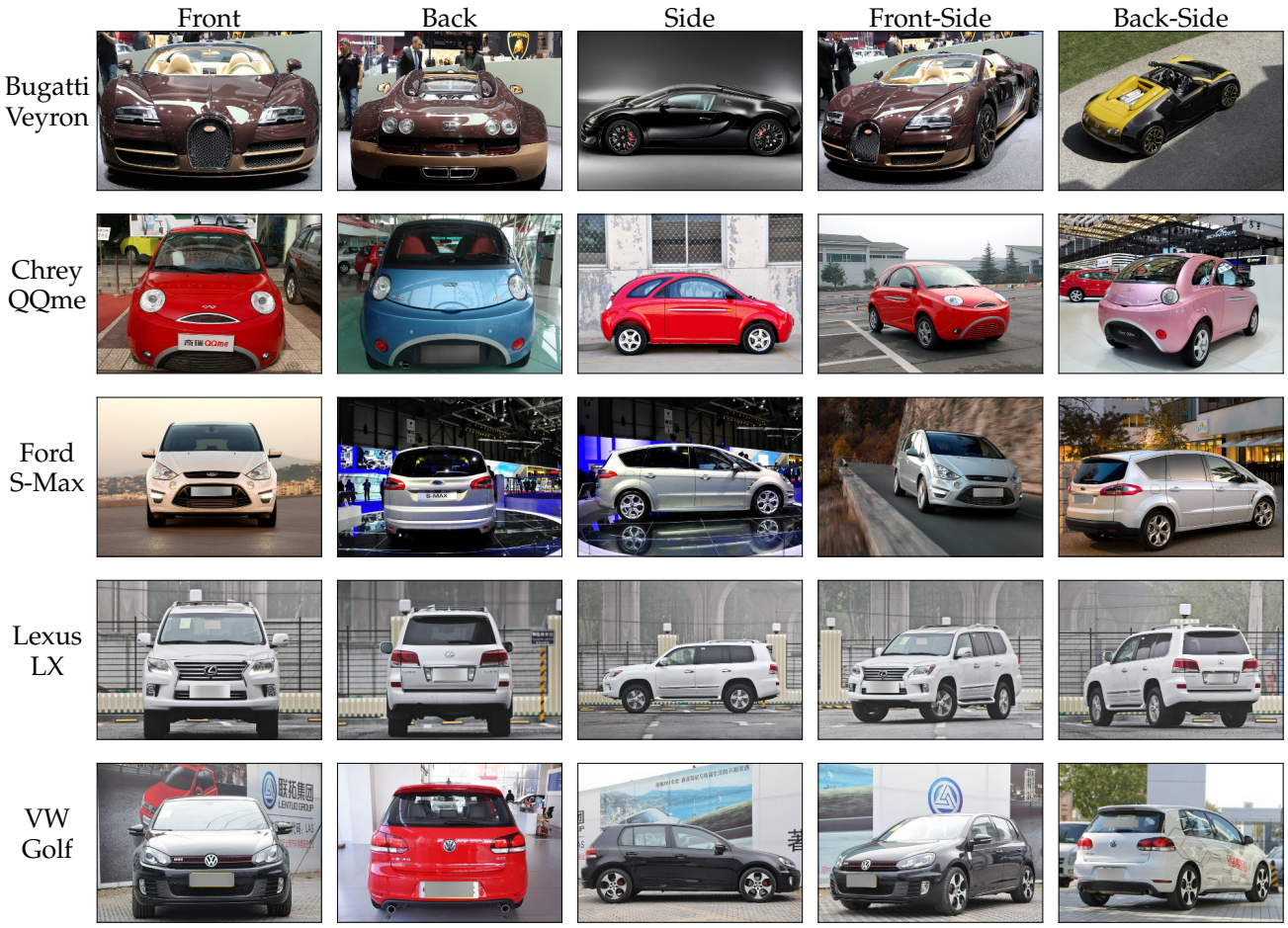


Figure 4.1: Example images of the CompCars web-nature dataset.

The dataset consists not only of images, rather it has a rich amount of labels. For all images exists the following labels:

- vehicle make
- vehicle model
- year of manufacture
- pose
- bounding box

Furthermore, for the images of the surveillance type, it exists the vehicle color as a label. For the label pose, it exists only five different classes, which are "Front (F)", "Back (B)", "Side (S)", "Front-Side (FS)" and "Back-Side (BS)". The information of a left or right pose is regrettably missing. The bounding box describes a frame around the vehicle in the image, and thus mark the vehicle to which the label belongs. This is helpful, because some images shows several vehicles.

For some of the vehicle models exists additional labels, which describe the attributes of a vehicle model. The following list shows these attributes:

- maximum speed
- engine displacement
- number of doors
- number of seats
- body class

The first two attributes are represented by continuous values. The number of doors and seats are discrete values. The body class consists of twelve label values, which are MPV, minibus, SUV, crossover, sedan, estate, hatchback, fastback, pickup, sports, convertible and hardtop convertible.

The resolution and aspect ratio differs from image to image. The average resolution of the images are ($x = 858.04; y = 589.45$) and the minimal ($x = 400; y = 267$). The resolution of the images are sufficient for our purpose. The bounding box, which frame a vehicle in the image, takes in average 45.65% of the image. So, the vehicle takes almost half of the image and therefore, the vehicle is in the most cases the largest object in the image.

The authors of this dataset used the dataset to classify the vehicle model and to predict the five attributes. Further they used the dataset for a vehicle verification. For the three tasks, they used a ConvNet model. They found out, that for the classification of the vehicle models, it is better to use all poses. They concluded, that the ConvNet can learn a discriminative representation across different poses to learn a classifier for vehicle-models.

For the attribute prediction, they used a train-test set splitting on the vehicle models, so that only one vehicle model is in one of the train-test sets. Unhappily, they trained and tested only on different poses, and left out a test on all poses. It is striking, that the prediction on the side pose reach the best results for the body class attribute with an accuracy of 62.7%.

In this section we described the CompCars dataset and showed that the dataset is a large-scaled dataset. For this work we limited us to the web-nature images without the vehicle parts. Further this subset of images was reduced to images which provides the attribute “body class”. So that at the end only 97.269 images will be used in this thesis. This subset of images gives us a great diversity of different vehicles and the needed labels.

For our work we must improve the pose labels and we modify the body class labels to our needs. This is the topic of the next section.

4.2 Preparation of the Dataset

Not the complete CompCars Dataset was used for this work. As in the previously section mentioned, the web-nature images is used. Furthermore, some of the images have no pose-labels and was also removed from the dataset. The most images was removed, because they have a missing body-class label (38,884 images have no body-class label). This was done preventative, in case that this label will be used in a later scenario. Furthermore, images with wrong labels, like the bounding box, was also removed. There are images with a negative bounding box and images with a bounding box of smaller than 10 pixels, which are not useful. While the preparation of the pose labels are removed further 222 images, because they showed the vehicle either from the top, showed a sketch of a vehicle or has multiple vehicles on the images, where it was not clearly to which the label belongs. After all of the filtering, the dataset has still 97,269 images from 967 different vehicle models.

This 97,269 images are all the images, which we will use for this work. In the next subsections, we will describe the preparation of the labels. We start with the pose labels, where we add the left- and right-side information. After this, we describe the change of the body-class to a vehicle class. For that, we adapted the classification from the “Kraftfahrt-Bundesamt” (KBA), which is in a second step reduced

to our used vehicle class. We call this reduced class for the rest of this thesis only “vehicle class” or sometimes “vClass”.

After the preparation of the labels, we describe the splitting in train-, validation- and test-set, and discussion the splitting method.

4.2.1 The Pose Label

The dataset provides a pose label, but with only five different label-values. As a remainder, the five label values were “Front (F)”, “Back (B)”, “Side (S)”, “Front-Side (FS)” and “Back-Side (BS)”. The values *-Side describes the left- and right-Side, but only as one label-value. For this work, the differentiation between left and right is needful. So the label-values which describe a side, will be split in left and right. So that at the end the pose label has the following values:

- Front (F)
- Back (B)
- Left (L)
- Right (R)
- Front-Left (FL)
- Front-Right (FR)
- Back-Left (BL)
- Back-Right (BR)

In Figure 4.2 are the poses from the top view depicted. The eight areas around the vehicle are the viewpoint to the vehicle and describe the resulting pose of the vehicle. The viewpoint areas are not equally distributed around the vehicle, because the poses Front, Left, Right and Back allow only a little mix of both sides. This results in really monotonous images for Front, Left, Right and Back poses, which only shows one of this sides. The mixed poses FL, FR, BL and BR have a bigger variation of different views. For example images, we refer to Figure 4.1, which shows images for different poses.

To improve the pose labels, we implemented a crowd based solution to get from different people results. We called the crowd based solution Crowd Labeling. It is a web application and will in Section 4.3 explained.

The Crowd Labeling was firstly only for the labeling of left and right for the *-Side images, but we found several images which was totally wrong classified. This mean, that we found images with the label “Side”, which shows vehicles from the front or from the back. So we mistrust the statement in the paper [57] of the dataset, that the pose labels are labeled from “professional annotators”. We go one time through all images and marked the images, which has a wrong label or are not useful, like images which shows the vehicle from the top. Furthermore, we marked images, which maybe need a second opinion, because they were marginal between two labels. These marked images are then processed in a second task. For the second task, the Crowd Labeling was changed, so that the user could label these images to the eight poses.

The first labeling in left and right goes very well. For every “*-Side” image, we get three opinions. One opinion was from us, because we make one complete run and labeled these images too. From 69,075 images, only 889 images were different labeled from the crowd. And only 10 images of them were majority wrong labeled. We know this, because we go through all different labeled images and had the last word for these images.

The second task of the Crowd Labeling was harder, because it includes the images which are between two labels. We had 3,715 images, which shows the vehicle from all poses. These images have we marked

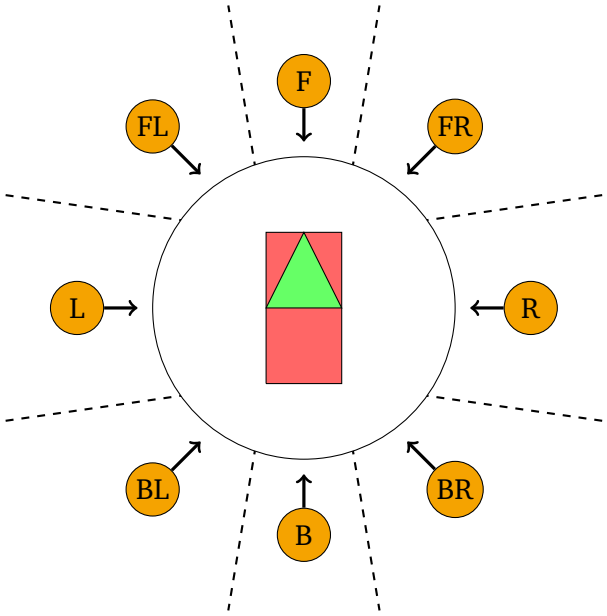


Figure 4.2: Shows the eight label-values for the vehicle pose labels from the top view. A vehicle is depicted in the center. The front of the vehicle is marked with the green triangle. Around the vehicle are eight areas, which describe the viewpoint to the vehicle, which results in the corresponding pose. The viewpoint areas are not equally distributed around the vehicle.

in the run through all images, which was either totally wrong labeled or was between two poses and we want further opinions. We collected nine opinions per image, and two of them were from us. We go two times through these images and labeled these, to give our opinion a stronger weight. Only 1,494 images were clearly labeled and we can assume that these labels are correct. For the other 2,221 images, we go through these images and take a look into the distribution of the opinions for every image, to avoid miss labeling. Most of them had only one or two failures and could be easily take over. Images between two poses got opinions for both poses. Most of them has one opinion more and the majority label is in the most cases used. In some cases we decided against the majority, because for a consistent labeling. There are some images, on which the viewpoint is not changed, but the vehicle changed his state, like the convertible roof was closed, but on these images the majority has different labeled the pose. In this case we decided against the majority, to get a consistent labeling.

Another case is, that one opinion was totally wrong, and so exists a tie for the both poses. In these cases, we decided the label for these images. At the end, we must intervene for 172 images.

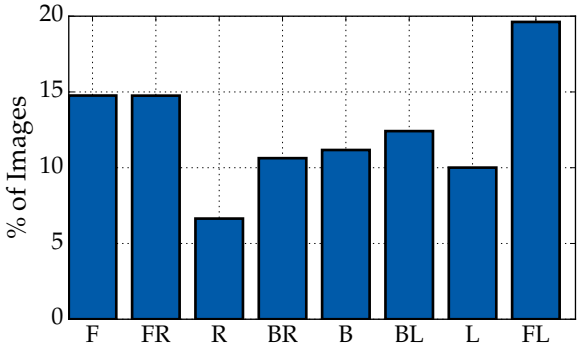


Figure 4.3: The distribution of the vehicle poses for the prepared dataset

In Figure 4.3 is the distribution of the pose labels depicted. The most images exist for the pose Front-Left, which shows the vehicle from the driver's side. But the other two poses, which shows the Front,

are popular too. Images from the side are not so popular, but side images means really side images, with only a little bit of front and back. And this constraint limit the amount of different images for the side. Noticeable is, that images of the right side are very unpopular.

4.2.2 The Vehicle Class Label

The body-class of the CompCar dataset is not the vehicle-class what we wanted. Just a reminder, the body-class has the following twelve classes: MPV, minibus, SUV, crossover, sedan, estate, hatchback, fastback, pickup, sports, convertible and hardtop convertible. It is possible, to use it as a vehicle-class, and we will also transfer some labels of it, but we are missing the size of the vehicles for a good vehicle-class. With vehicle size, we mean a differentiation of passenger vehicles into small and large vehicles, like “Smart” vs. “VW Passat”.

For example, the size of a passenger vehicle could be used in a driver assistant system, to decide the possible behavior of both drivers. A small vehicle will be not so good accelerate on an acceleration lane, like large vehicles. So the notice of a small vehicle on the acceleration lane, could the driver induce to change the lane. A large vehicle would mostly have a good acceleration and can easily integrate into the traffic, and so the probability is low, that the driver have to change the lane to make space for the vehicle. The body-class has no information gain. There are small vehicles and large vehicles with the same shape of the trunk.

Another argument against the body-class lies in the insufficient labeling of the dataset. The dataset labels the body-class for every vehicle-model, and not for every image. We found some vehicle-models, which contains different body-classes. For example, for the vehicle-model “Opel Insignia”, we found images for a sedan and images for an estate, but the dataset labeled the vehicle-model as an estate. This makes the generalization for a body-class hard, if it is inconsistent labeled.

Therefore, we decided to adapt a common vehicle classification from Germany. The “Kraftfahrt-Bundesamt”[1] (KBA) has defined 13 labels[2] for vehicles [24]. This classification could be divided into two sections, the first describes the size of passenger vehicles, and the second describes the vehicles for a special purpose. For clarity, one vehicle can only in one of the classes. In Table 4.2 are twelve of the 13 classes depicted. We omitted the class “Recreational Vehicle”, because this class is not needed by us.

For the classification of the vehicles after KBA, the KBA has no formula and the KBA is using the optical appearance and other characteristics [24]:

- Dimension of the vehicle
- Weight
- Motorization
- Performance
- Size of the trunk
- Number of seats
- Seat height (front)
- Shape of the back
- Basic price

So the classification is not clearly defined and depends on the decision of humans. Some of these characteristics are hidden variables, because we have only images of the vehicles and thus we have only

[1] www.kba.de

[2] The label SUV is added in January 2013. Before this date, SUVs was labeled as Off-Road Vehicles

	KBA-Class	vehicle class
Size	Mini	Small
	Subcompact	
	Compact	Medium
	Mid-size	Large
	Mid-size Luxury	
	Luxury	
Special Purpose	SUV	SUV
	Off-Road	
	Sports	Sports
	Mini-Van	Van
	Large-Van	
	Utilities	Minibus
		Pickup

Table 4.2: Shows the classes of the KBA-Class and the new defined vehicle class for comparison. The KBA-Class shows only twelve classes, because the class *Recreational Vehicle* not needed by us. Note, that the class *Utilities* is splitting in two classes for the new vehicle class and not follow the trend of reducing.

the optical appearance as a decision. Additionally, there are vehicles for multiple purposes, which makes difficult to classify to one of the labels.

The KBA is not the only institution/market segment, that split the vehicles in size and special purpose. For an overview of different vehicle classification, we recommend the article in the english Wikipedia[3]. There is a nice table, which compare the different vehicle classes with each other.

For our desires, we use the KBA-Class as basis and modify it. For the section "Size", we define only three classes: *Small*, *Medium* and *Large*. In the section "Special Purpose", the *SUV* and *Off-Road* classes are combined to one class with the name SUV. Furthermore, *Mini-Van* and *Large-Van* is combined to *Van*. The class *Utilities* is not reduced and is split to *Pickup* and *Minibus*. These two classes are so different, that a combination make no sense. Furthermore, there are enough images for these two classes, that this is not a problem. In Table 4.2, we show the KBA-Class and the new defined vehicle class side-by-side.

After we defined the vehicle class, it arises the question, where we get the labels for this? The body-class from CompCar dataset provides for some vehicle-models this information. So for, Pickup, Minibus and Van (=MPV), we could easily take over the labels. For the class SUV, we combine the SUV and the Crossover label from the body-class. Crossover are vehicles between SUV and a normal passenger vehicle, but show pronounced characteristics of an SUV. For the Sports label, we combine the Sports, Convertible and Hardtop-Convertible labels. Furthermore, some vehicles with a coupe style are added to the sports-class too, because they looked similar to the vehicles of this class. For example, the vehicle-models "BMW M6 Coupe" and "Audi S5 Coupe" are added to this class.

The remaining vehicle-models are labeled with the help of the German Wikipedia[4]. The Wikipedia provides for the most vehicle-model the KBA-Class. So these informations are extracted for the most vehicle-models. Some vehicle-models are unknown in the Wikipedia or the information was not available. For these vehicle-models, we tried our best and labeled this to the best of one's knowledge.

In Figure 4.4 is depicted the distribution of the vehicle class. The labels Large and SUV are the biggest labels and make up together almost 50% of the vehicle class. Whereas the Pickup label is only

[3] "https://en.wikipedia.org/wiki/Car_classification" Version of the 16.10.2016

[4] de.wikipedia.org

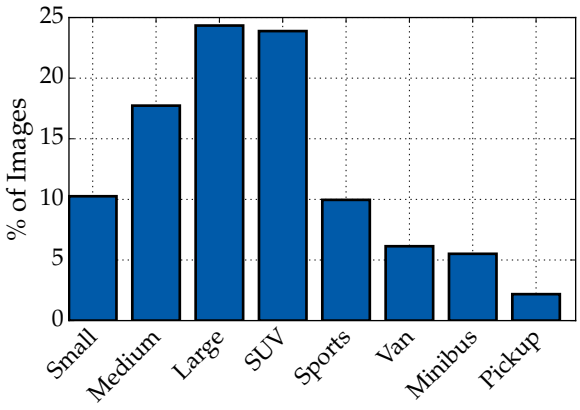


Figure 4.4: The distribution of the vehicle class for the prepared dataset

represented by 2.2% of all images. This is a huge difference, but there are enough images for this class, to not become a problem. That the label sports is only good represented, because the people take more pictures of desirable vehicles, as of other vehicles. So, we can assume, that this is not uncommonly for a web-nature dataset. Furthermore, the distribution of the vehicle class does **not** represent the reality in Germany. The KBA publish yearly statistics to registered vehicles in Germany.[5] It shows, that the distribution in Germany differs from the distribution of the images. For example the label SUVs are overrepresented in the CompCar dataset. In January 2016, in Germany are only 8.1% of the vehicles SUV and Off-road vehicles[5]. For the other classes exists this discrepancy too.

4.2.3 Splitting of the Dataset

For this work, we decided to use a holdout-method with a validation-set for the splitting of this dataset, because the amount of data is huge. For that the dataset is split in three datasets, which are called training-, validation- and test-set. The split ratio is 60-20-20 percent and in Table 4.3 The biggest set is the trainings-set and will be only used for the training. The validation-set is the set, which is used to optimize the model. This mean, that on this set, will be decided when we stop the training to avoiding overfitting. Also, this dataset will be used to chose the best model. The drawback of the validation-set is, that the computed performance is biased, because we choose the model, which maximize the performance on this validation-set. Therefore, the performance on the validation-set is not a good estimation for the performance of unseen data. But there are a third set, the test-set. This set will be only used on the final model, to get an approximation of the trained model for unseen data.

Set	Number of Images	Number of different Vehicles
Training	58,344	579
Validation	19,391	194
Test	19,534	194

Table 4.3: Number of images and different vehicles for trainings-, validation- and test-set.

The dataset is not simply randomly split in the three sets. We randomly splitting after the vehicle model, so that every images of the same vehicle model are in one of the three sets. Further we add similar vehicle models to the same set. Similar vehicle models will be found through similar vehicle

[5] "http://www.kba.de/SharedDocs/Publikationen/DE/Statistik/Fahrzeuge/FZ/2016/fz12_2016_pdf.pdf", Last visit at 02.10.2016

model names. The dataset makes a difference between some vehicle models, if for a vehicle model exists different body classes or a powerful versions. For example, the vehicles “Ford Fiesta” and “Ford Fiesta ST” or “Audi A3 hatchback” and “Audi A3 sedan” are similar vehicles and will be added to the same set.

Furthermore we tried to reach a similar distribution on the three sets for the vehicle-class label, so that a small class is not under- or overrepresented in one of the sets. For that, we tried different random seeds until the right splitting is found.

Why we do this complicated splitting? Normally, the validation- and test-set should contain only unseen images, but the same vehicle from a little different pose or a changed color, is not really an unseen image. The trained model could learned a manifold for the pose or color, and so the images of a vehicle with a different pose or different color are not more unfamiliar. So the images of one vehicle model will have a dependency, which can influence the test for the generalization of the trained model. This issue has Schenkel [44] observed in his Bachelorthesis, with only 20 different vehicle models. He tested his used dataset with a random split and with a split on the vehicle models. The results for the split of the vehicle models are worser then with the random split. He concluded that the good results on the test-set for the random split comes from known vehicle models from the trainings-set. Therefore it is important to know dependencies of the data and split the data so, that the data of the validation- and test-sets are independent of the training-set.

We see a similar dependency in similar vehicle models, because many of them will have only a small variation in the design of the vehicle. Like already mentioned before, the “Ford Fiesta” and “Ford Fiesta ST” are two similar vehicles, which cause in small design differences.

A second point for this splitting by vehicle model is, that the validation-set can then used to avoid the overfitting. For that the validation-set measure a metric, like the loss, and can then stop the training, if the metric does not more improve. This regularization method is called Early-Stopping It is possible, that the trained model is overfitting and it learn the vehicle class on other features without a generalization. Without the split by vehicle models, you would not see this overfitting, because the dependency of the data. So, if the dataset is not split after the vehicle models, the validation- and test-set will not show the overfitting of the trained model. The sets will even show good results, because they have the images of the same vehicle models, on which the learned model is trained. But in reality, the results will be bad, if unknown vehicle models are predicted on this learned model.

4.3 The Pose Crowd Labeling

The Crowd Labeling was implemented during this thesis, with the goal to improve the pose labels of the CompCars Dataset [57] with the help of the crowd. The Crowd Labeling is a web-application, which can used in a browser and through the adaptive front-end it was possible to use it on mobile browsers, too.

4.3.1 The User-Interface

The web-application was used for two task. The first task was used to label the images in left and right. The second task was more complex. The user must label the image into eight poses.

For the first task, the user get displayed 78 images. If the user was on a device with a big screen, the images was displayed in a table with three columns, otherwise all images was displayed in one column. The task of the user was to click these images, which shows the vehicle from left. If he click on the image, the image get an orange border and in the top left edge, the label was displayed. With a further click, the user could reverse his decision. For simplicity, all displayed images was either from Front-Side, Side or Back-Side. This makes the task easier, because the user must only remember one pose for this round. For a better understanding of the user-interface, it is for the first task depicted in Figure 4.5a.

Further the user had the possibility to report images. For that, the user must only click right on the image and could report the image. This was useful to report images, which do not belong to one of the two labels.

The second task, the user get only one image displayed and had next to the image a wheel selector, on which the pose could be selected. For better understanding, this is depicted on Figure 4.5b. The wheel selector has eight directions and if the user choice one direction, then he could click on the OK-Button in the center of this wheel-selector. After the click on the OK-Button, the choice is saved and a new image is displayed. It was possible, that the user could go one image back, if he thought he has labeled the previously image wrong and could change his decision.

Further, for every labeled image, the user get one point, which is then displayed in the Top10. This was a little incentive to the user, to label more images to get a higher score or to be better then an other user.

4.3.2 Quality Assurance

To ensure the quality of the labeling, we used different mechanisms. This is needed, because there are people, which maybe not understand the task, or try to break the system.

The first mechanism is the registration of users. The users must create an account. For that, they must only choose a name and a password. With this information, it is possible to filter out users, which maybe not useful. As a little notice, we did not need to filter out users. All users have done a great work.

For the first task, there are added eight images, from which the poses are known. From this eight images, four images had to be clicked by the user, so that all eight images are right labeled. This means, that four images are in one class and the other four images are in the other class. So, if the user labeled one of the eight images wrong, the user get displayed a warning and has the opportunity to improve his solution, after he tried to submit his solution. With this step, we ensure that the user has carefully labeled the images and not randomly labeled. To submit a randomly labeled solution, the user must have very much luck. If he forget to label one of the known images, then the solution will rejected. If he click on all images, it will be rejected, because he had labeled the four images of the other class. So it is very hard to submit a randomly labeled solution.

For the second task, there exists no mechanism to avoid randomly submissions. But we increased the number of solutions for every image to nine. At the end, to every image in the second task exists nine solutions.

So that a wrong solution of users are not weight to much. Therefore, for this task was collected nine different solutions per image.

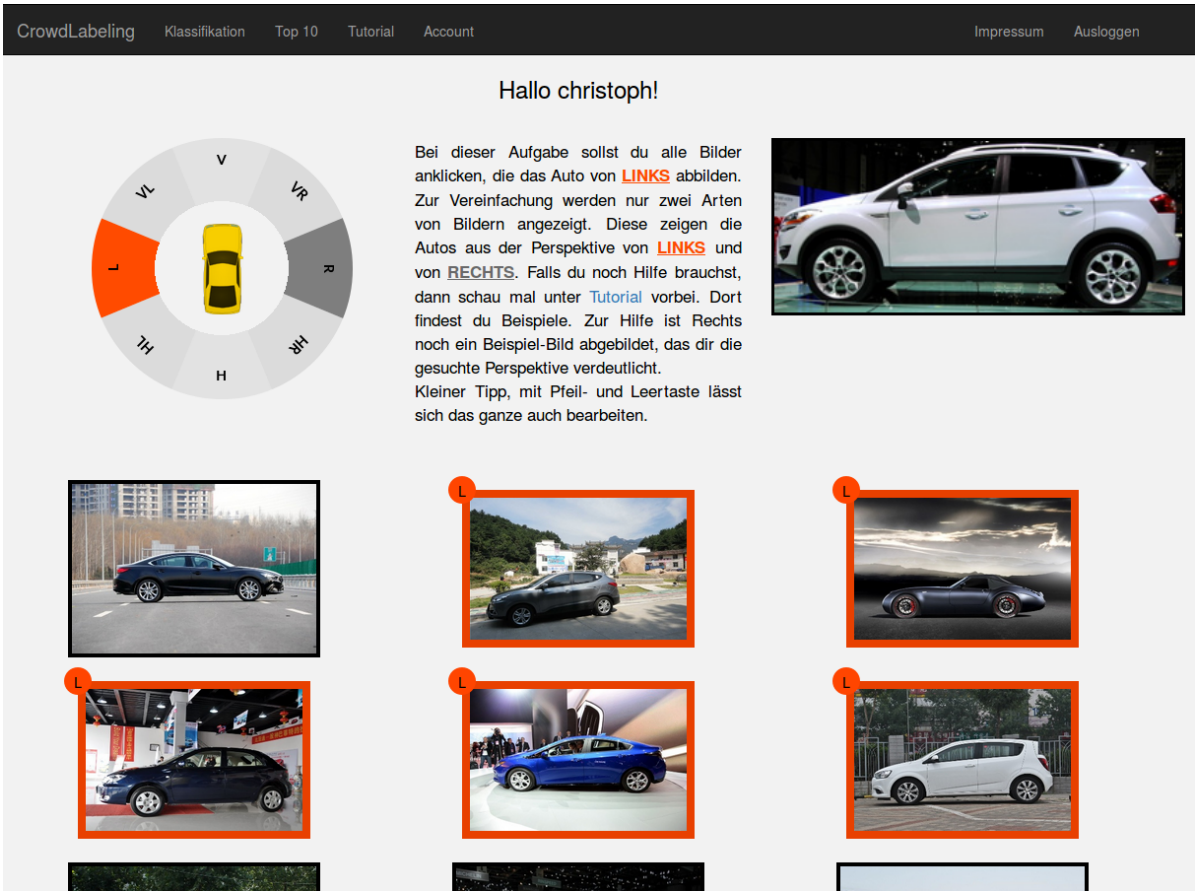
The last quality assurance was the check of these images, which has different solutions. These images will be manually checked, and if the majority is not totally wrong, will the majority solution accepted. So it is ensured, that these images will be not get only our opinion, because there exists images between two poses and these images are hard to decide which label are adequate.

4.3.3 Acknowledgment

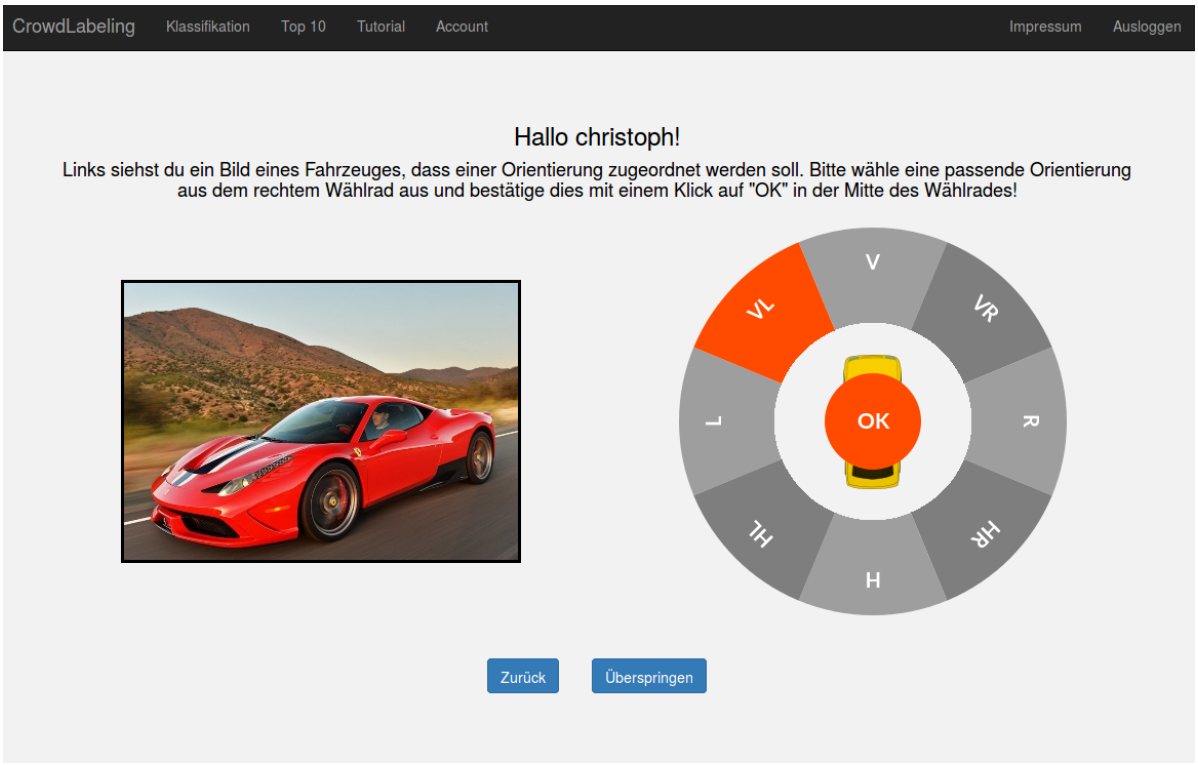
At this point, we will acknowledgment to all participants of the Crowd Labeling. Without your commitment, we had not get so fast and accurate results for the pose.

This acknowledgment goes to the following users (ordered after the number of submitted solutions):

FelixDerAchte, Andrea, Paul, Patrick, ShadowWalker, fiona, tianyi, StefanLuthardt, Hanno, Thomas, kiki, volker, hiendang, calamondine



(a) The user interface for the first task. The user should mark all images, on which the vehicle is depicted from left. As help, an example image is depicted on the right side of the task-description.



(b) The user interface for the second task. The user should choose the pose for the left depicted vehicle.

Figure 4.5: The user interfaces of the web-application CROWD LABELING.

5 Preprocessing and Augmentation of the Images

Before the images are fed into the ConvNet, the images should be preprocessed and augmented. The preprocessing steps consist of the centering of the vehicle, enhance the contrast, scale the image, change the colorspace and scale the channels of the image. Furthermore, this chapter handles the augmentation of the images, because the augmentation is a part of the preprocessing in the training.

This chapter starts with an overview of the preprocessing pipeline. After this, every preprocessing step will be presented. The order of the sections follows the order of the preprocessing pipeline.

5.1 Overview of the Preprocessing Pipeline

Before the image is feed into the ConvNet, the image must pass through the preprocessing pipeline. Our preprocessing pipeline consists of six different steps, which are depicted in Figure 5.1. The image will be loaded as an RGB image and goes then into the *Centering*. In the *Centering*, the image will be transformed so that the vehicle is in the center. This is possible, because the position of the vehicle is known through the bounding box. After the centering, the image goes to the *Histogram Equalization*, which enhances the contrast of the image. After this, the *Augmentation* is applied. The augmentation does different transformation and modification on the image, so that is created a slightly different version of the original image. Then, the image is scaled to the corresponding size of the ConvNet input-size. After the *Scaling*, the color space can be transformed. Until here, the preprocessing was applied to the image in RGB, if another color space is desired, then it is applied here. The last preprocessing step is the *Feature Scaling* of the channel-values, which rescale the channel-values, so that all channels of the image have the same scale. After the processing through the preprocessing pipeline, the image is ready to feed in to the ConvNet.

The preprocessing pipeline is designed so that not all steps must be used. Five of this six steps are optionally. The exception is the scaling of the image, because the image-size must be fit to the input-size of the ConvNet.

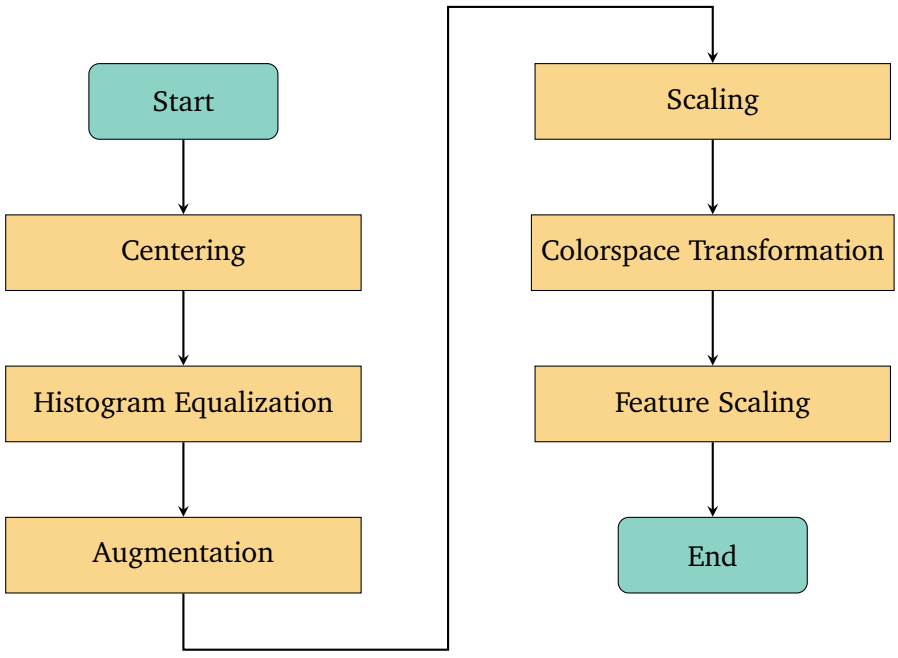
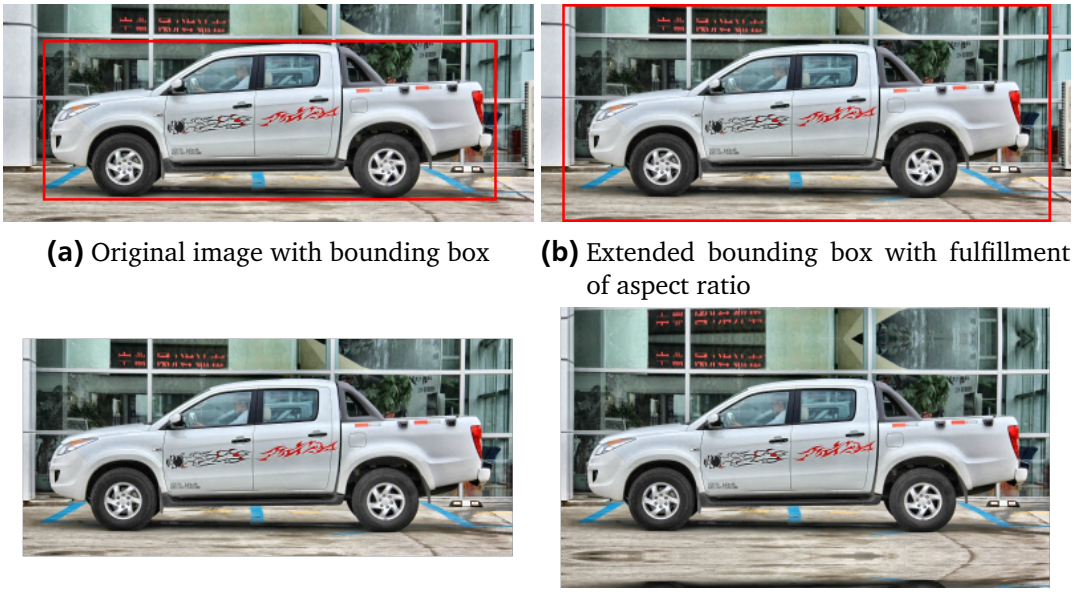


Figure 5.1: Flowchart of the Preprocessing Pipeline.

5.2 Centering of the Vehicle

For this preprocessing step, we make the assumption, that the trained ConvNet will be only used to predict the labels for an image, which shows one vehicle and has a relative position to the center of the image. This could be reached, if another algorithm found a vehicle on an image and crop out the vehicle. This cropped image is then inserted into our trained ConvNet, to predict the labels.

This assumption is valid for the most images of the dataset, but some of the images shows multiple vehicles and it is not clear, to which the labels belong, or shows vehicles with much background. Therefore, the Bounding Box will be used to get the vehicle into the center. Further the Bounding Box will be adjust to fulfill the aspect ratio (width divided by height) of the input-size for the ConvNet. The Bounding Box should be only extended, because otherwise the vehicle will be truncated.



(a) Original image with bounding box

(b) Extended bounding box with fulfillment of aspect ratio

(c) Image trimmed

(d) Image with mirrored edges (top and bottom)

Figure 5.2: Example of the centering of a vehicle in an image. In **(a)**, the original image is depicted with the bounding box. In **(b)**, the bounding box is extended and it is tried to fulfill the aspect ratio of 1.83, but the image is to small to reach the desired aspect ratio, so the complete height is used of the image. In **(c)**, the image is trimmed with the size of the bounding box. In **(d)**, the image is extended to fulfill the aspect ratio. Hence, the image is mirrored at the top and bottom edges, to reach the aspect ratio.

The extending of the bounding box is a little bit complicated, but it can be divided into three phases (for a better understanding, the phases are illustrated in Figure 5.2):

Extending Bounding Box: In the first phase, the Bounding Box will be extended, to get a little margin to the vehicle. For that the sides of the Bounding Box will be extended on all four sides with an factor of 3.75%. If the image does not allow this extension, then will be tried to add the remaining size to the other side.

Fulfill Ratio until Image is Saturated: After the Bounding Box is extended, we want to reach the aspect ratio of the input-size for the ConvNet. So the Bounding Box will be further extended to reach the desired aspect ratio. If the aspect ratio of the Bounding Box is smaller than the desired aspect ratio, then the X-axes will be extended in both directions. Otherwise the Y-axes will be extended. How in the previously extension, it will be ensured, that the constraint of the image is fulfilled, and that on both sides the same amount is extended.

After this extension, the Bounding Box will be cut out of the image, and is now our new image.

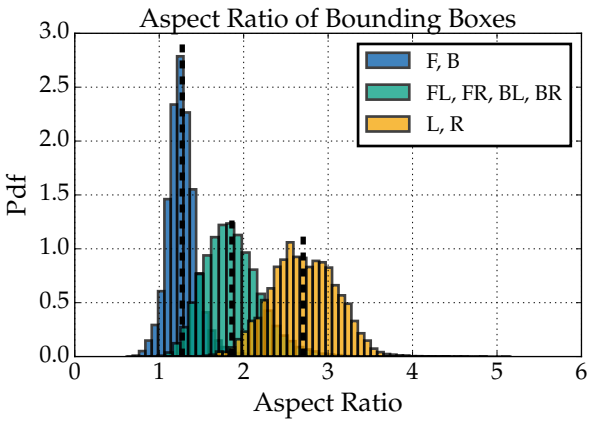


Figure 5.3: The distribution of the Bounding Box Aspect Ratios for different poses. For a better readable, some of the poses are combined. (Best viewed in color)

Fulfill Ratio with Mirroring: It is possible, that the aspect ratio is still not fulfilled. If the new image's aspect ratio lies not in a range of 0.1 around the desired aspect ratio, then the new image must further extended. This could happen, if the image was not big enough to extend. Therefore, there is no information, as it could be further extended. But it is possible to mirroring on the edge and use the existing information of the image. In most cases it must only a little bit extended, to come in the range of the 0.1 around the desired aspect ratio.

This complicated centering is needed to get the aspect ratio of the input-size for the ConvNet and not to truncate the vehicle at a random point to fulfill the aspect ratio. The truncation of the vehicle will be happen during the Augmentation.

The aspect ratio of the bounding box, and thus also of the vehicle, are very different. This reaches from 0.62 up to 5.15. For a better illustration of the Bounding Box distribution after the poses, it is in Figure 5.3 depicted. It shows three distribution for combined poses. It is nice to see, that some poses have their own aspect ratio. However, the figure shows, that the aspect ratio for the bounding box is very different. So one aspect ratio for all images doesn't exist. Therefore we use the average bounding box aspect ratio of 1.83 for the centering and for the input-size of the ConvNet.

5.3 Histogram Equalization

To compensate different contrasts in the images, we apply a Contrast Limited Adaptive Histogram Equalization (CLAHE) [39]. It enhance the contrast and like the normal histogram equalization as well. But the normal histogram equalization has the problem, when in the image exists regions with huge brightness differences, like black foreground and a white background, then the contrast can be to strong. This is in Figure 5.4b depicted. So, we use the CLAHE, which is an adaptive histogram equalization (AHE) with contrast limitation. The AHE works on local tiles, to improve the local contrast, to avoid the problem of the normal histogram equalization. The contrast limitation tries to prevent amplifying of noise, which can happen in homogeneous regions of an image. For example, the image shows a black region with some noisy pixel. The noisy pixel are in the not enhanced image not visible, but the enhancement can shift the intensity of this pixels, and the pixels are visible.

Typical the CLAHE will be applied on gray-scaled image. But the actual image is in RGB. So the image is in YCbCr colorspace transformed and later back to RGB. An another colorspace, which has a gray-scaled channel is possible as well. The YCbCr has in the Y-channel the gray-scaled image. So, only this channel is used to enhance the contrast. This is sufficient, because in the Y-channel is stored the luminance of the image.

(a) Original **(b)** Histogram Equalization **(c)** CLAHE

Figure 5.4: Example images for the enhancement of contrast. The normal histogram equalization is overexposed. The floor has lost of contour. Much better is the CLAHE. It has improved the contrast on the complete image. Best viewed in color.

5.4 Augmentation

Augmentation is not really a preprocessing step. It is a regularization method, which is used during the training. However, it is in the preprocessing pipeline, because it manipulates the image, and we want it to apply on the unscaled image. The unscaled image has more information, which can be used for a better approximation of the transformation. In an image, the pixels are mapped to a discrete position and a transformation of the image, cause in a new position for the pixels, which are not necessarily discrete. To make the mapping discrete, the value of the pixels will be approximated. This error is not so visible, if the resolution of the image is higher. Therefore, the augmentation will be applied before the image is scaled.

Augmentation is widely used in ConvNets to improve the results [8, 26, 46]. It increases the size of the trainings-set, so that the probability is very low, that the same image is used twice for the training. This results in a reducing of overfitting and helps to increase the performance of the ConvNet. Transformation like rotation and zooming, helps the ConvNet to be invariant toward rotation and different scales.

Augmentation transforms the image, so that a new image is produced. This could be reached with simple affine transformation like translation, zooming or rotation. It is also possible to do color perturbation or add noise. Another popular augmentation is random cropping on the image.

We implemented different augmentation transformation for the preprocessing pipeline, which can be independently used. The following list describes the used augmentations:

Rotate: Rotate randomly the image in a defined range. The rotation axes is in the center of the image. It helps the ConvNet to be invariant for rotations. The range is defined by the user in degrees.

Translate: The image is randomly translated to the x- and y-axes. The range is defined by the user in percent of the image size.

Zoom: The image will be zoomed in or zoomed out. The amount is randomly selected from the defined range. At zooming out, the missing pixels at the border are filled with nearest pixel of the image. It helps the ConvNet to be invariant for different scales of the vehicle. The range is defined by the user.

Color Jittering: Add or subtract a random value of a channel in the image. This means, that the intensity of the RGB-channels are shifted. This results in a slightly color change. The maximal value is defined by the user.

Random Cropping: Cut out a random part of the image. It is modified that the random cropping is applied only at the left or right side of the image. This means, that either the random part starts on the left side or the random part ends on the right side. So it is ensured, that the result shows a

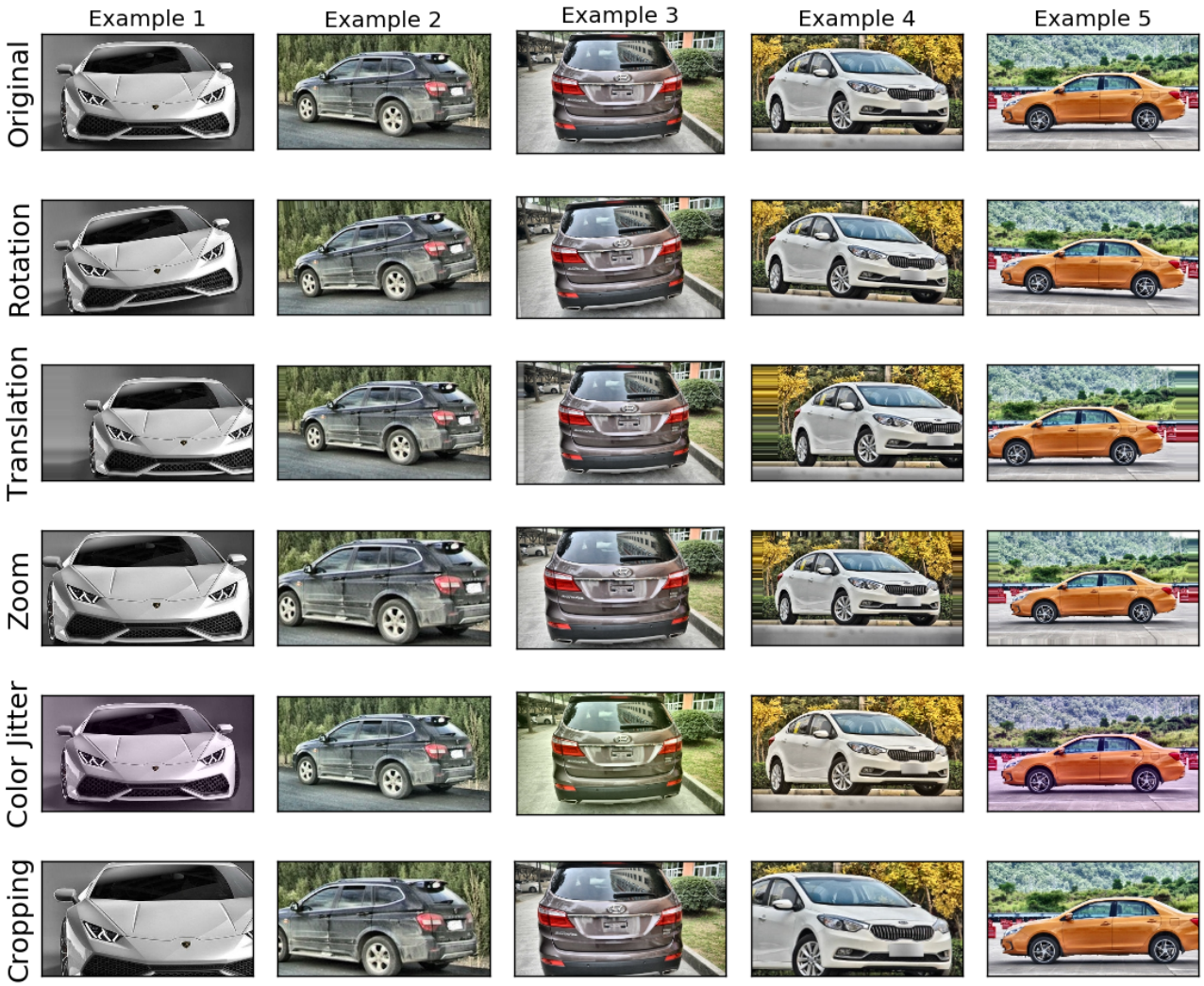


Figure 5.5: Shows example for different augmentation

vehicle, which is truncated only on one side. Furthermore, the aspect ratio for the input-size of the ConvNet is respected during the cropping. The cropping using a random size, between 0.7 and 0.9 of the original image. The probability of applying the cropping is defined by the user.

The order of the list, corresponds to the order of applying of the augmentation. Because, the order is important for rotation, translation and zoom. A rotation and then the translation of an image results in a different image, as first translate and then rotate. To demonstrate the resulting images after the augmentation, in Figure 5.5 are depicted example augmentations.

5.5 Scale the Image

After the augmentation, the image will be scaled to the input-size for the ConvNet. During the centering (see Section 5.2) we tried to fulfill the aspect ratio for the input-size. But we accepted images with a small difference to the desired aspect ratio. This difference must be now handled during the scaling. Or if the centering was not used, then should be also respected the aspect ratio.

But why should we not resizing the image without to respect the aspect ratio? This would cause in a stretching of the image in one direction. Therefore, the stretching would destroy information of the vehicle, like the proportion of the vehicle. For example, a medium sized vehicle could look like an SUV, because it looks higher. Therefore, the aspect ratio should be respected.

To scale the image and don't destroy the aspect ratio, we do a center crop after the scaling. This cut off only a little bit of the sides, but this reducing is harmless, because we had during the centering a margin added. But before the center crop is applied, the scaling is executed.

The target size is known for the center crop. But for the scaling, it must be calculated the scaling size, because one dimension must be a bit longer. This depends on the aspect ratio of the target size AR_{target} and the aspect ratio of the unscaled image AR_{image}. If AR_{image} is smaller than AR_{target}, then the width is the same to the width of the target size and the height must be calculated with

$$h_{\text{scaled}} = \frac{w_{\text{target}}}{AR_{\text{image}}}.$$

The h_{scaled} is the height for the scaling, and w_{target} describes the width of the target size. With the calculated height, it is now possible to scale the image and then to do a center crop. Because the scaled width is the same as the target width, hence only the height is truncated during the center crop. Therefore, it will be calculated how much on both sides must be cut off, and this amount is then cut off of the image, and the image has now the right size for the ConvNet.

For the other case, that AR_{image} is bigger than AR_{target}, it could be proceed in a similar way. In this case, the height is fix and the width must be calculated. The calculation is a little bit different. Instead of dividing with the AR_{image}, it must be multiplied with AR_{image}.

5.6 Colorspace Transformation

Until this point the image was preprocessed as an RGB image. Sometimes we don't want an RGB image to insert into the ConvNet. Maybe has another colorspace more information, or less information and helps the ConvNet to extract better features. Therefore, this preprocessing step can convert the RGB into another colorspace.

In the following list, we describe the features of some colorspaces, to get an insight into their properties:

RGB: The RGB has three channels, which describe the colors Red, Green and Blue. Every channel represents the intensity of this color in the image. Through different intensities on the three channels, it is possible to create other colors. For example the color black is reached, if all three channels are zero, and the color white is reached, if all three channels have their maximum intensities.

The RGB colorspace is the typical used in ConvNets, if the source image is colored.

Gray: A grayscaled image has only one channel, and the channel shows the intensity of light for this image. Low intensity means black and high intensity means white. Values between them describes a shade of gray. If a colored image is transformed to this colorspace, then is lost the color information and a retransformation is not possible.

HSV: HSV has three channels and describes a colored image too. The channels describe Hue, Saturation and Value. *Hue* describes the color. *Saturation* describes the saturation of the color and *Value* describes the darkness of the color.

For the human is this representation of the color the most understandable, because changes of one of the values results in an interpretable result. If we want to change the color, then we must only change the Hue-channel. In RGB, we must change up to three channels to get the same color change.

YCbCr: YCbCr has three channels and describes a colored image as well. One channel describes the luminance (Y), which shows a grayscaled image. The Cb- and Cr-channel describes the color difference to the Y-channel. The Cb-channel describes the color from yellow to blue, and the Cr-channel describes the color from turquoise to red.

5.7 Feature scaling

A typical machine learning preprocessing step is the feature scaling of the data. This means, that the data should be transformed to a reasonable range. This could be a range of [0,1] or a normal distribution around 0, so that all features – in our case the channels of an image – have the same range.

The typical range for a value in a channel lies between 0 and 255. It is normally not practical to input large numbers in a neural network, because this cause mostly in larger training time. LeCun et al. [28] have showed, that a zero mean input has a faster convergence. Therefore, it exists different ideas, how to scale the channels. Our developed preprocessing pipeline provides three of them, but only one is used at the same time:

Rescaling: To rescale a channel into the range of [0,1], it is used the following formula:

$$X_c' = \frac{X_c - min_c}{max_c - min_c}$$

X_c describes the channel as a matrix and X_c' is the rescaled channel. The values min_c and max describes the minimal and maximal value over the complete *training-set* for one channel. This could be in our case reduced to a simple division with the value of 255.

Zero Mean: The mean of a channel will be centered around 0. Therefore, the mean is calculated over the complete training-set on every channel and is then subtracted for every value in the channel. For example, for the red channel will be calculated the mean over all images of the training-set. This calculated mean is then used to zero mean the red channel in all images.

Normalization: Normalization is similar to the *Zero Mean* approach. The mean μ_c is calculated over all channels of the same color in the training-set, but further it is calculated the standard deviation σ_c as well. This formula describes the normalization process:

$$X_c' = \frac{X_c - \mu_c}{\sigma_c}$$

where X_c describes the channel as a matrix and X_c' is the rescaled channel.

For all three feature scaling are values $(min, max, \mu_c, \sigma_c)$ calculate over the complete training-set. These values are used also to preprocess images from the validation- and test-set, as well for unknown images to predict them. It is important to do this calculation only over the training-set, otherwise it is added a bias to these values from the validation- and test-set.

After the Feature Scaling, the image is preprocessed and can inserted in to the ConvNet for the training/prediction. The default setting for the preprocessing pipeline will be discussed in the next Chapter.

6 Methodology

In this chapter, we will present the model architectures for the prediction of the two tasks, vehicle pose and vehicle class, which are used in the next chapter for the evaluation. This is on the one hand our Furthermore, we show the concept of multi-task learning to combine two tasks in one architecture. A second concept to use information from one task for the other task is the concept of expert learner. Which is presented after the multi-task learning. After this, the chapter shows the default setup, which is used to train the models.

At the beginning of the search for a good ConvNet for the two tasks, we tested three existing models, with the method of transfer learning. The transfer learning is the transfer of weights from a previously trained model. The ConvNets from which we transfer the weights, are trained for huge object classification problems. One of the huge classifications problems was the ImageNet [42] challenge. On this challenge has reached the three ConvNets very well results.

We fine tuned these models for our problem and the results helped us to develop our own model architecture. We defined two requirements for our architecture in respect to the models of the transfer learning:

- at least the same performance,
- and faster.

We want at least the same performance like the models of the transfer learning. Furthermore, the architecture should be faster as the models of the transfer learning, because we had a limited resource of time and computational power for this thesis, and we would like to do a couple of experiments with this model. So a fast model is very helpful in this situation.

6.1 Transfer Learning Models

Firstly, we would like explain, what transfer learning is, before we show the used transfer learning models. Transfer learning is the using of a pre-trained model, which is transferred to another task. Either the complete model is transferred or only some first layers. It is not meant, that only the architecture of the model is transferred, rather the weights of the pre-trained model are transferred.

It exists two common approaches for the transfer learning. The first is, that the pre-trained model is only used as a feature extractor [36, 41]. This means, that the pre-trained model is used up to a specific layer for the generation of features. These features could be then used as input in another machine learning algorithm to train the other task. It is also possible to build on the top of the transferred layers some further layers, which are then learned, but the transferred layers are kept frozen[1]. So the task of the pre-trained model is only to extract the features, and the features are then used to train the task.

The second approach is the fine-tuning [58]. The pre-trained model or only a part of the model is transferred and on the top will be added further layers. But now, the weights of the transferred layers will be also trained. It is possible, to kept some lower layers frozen of the transferred model.

What is the benefit of this transferred learning? The pre-trained models are mostly learned on big datasets with millions of images. This has the benefit, that the model have learned useful features in the lower layers. These features could be useful for similar tasks. A nice side effect is, that the training is faster, if the pre-trained model was applied on a similar tasks, because the weights are good initialized. Furthermore, it can help small datasets to improve their performance, because the weights are well initialized, to detected common features in images. Yosinski et al. [58] showed that transfer learning and fine tuning improve the generalization on their tested datasets.

	Top-1 Error	Top-5 Error	Depth	# Parameter
Inception-v3	19.47%	4.48%	42	24M
ResNet-50	22.85%	6.71%	50	26M
VGG16	24.4%	7.2%	16	138M

Table 6.1: Comparison of error rates and parameter of the transfer learning models on ImageNet challenge. The error rates are from experiments with a single model with multi-crop. Values are taken from [18, 47, 52].

For the transfer learning, we use three pre-trained models, which are Inception-v3 [52], ResNet-50 [18] and VGG16 [47]. All three have achieved fabulous results on the ImageNet challenge [42] (see Table 6.1). The ImageNet challenge is a challenge of classifying around 1.2 million images into 1,000 categories. One interesting fact of the dataset is, that some of the images shows vehicles. Furthermore, the vehicles are categorized, for example there are categories for “station wagon”, “taxi”, “minibus” “minvan” and “race cars”. So the ImageNet dataset makes a difference in different types of vehicles. It is to assume, that the pre-trained models have the ability to distinguish between different types of vehicles.

We tested both approaches of the transfer learning, but only the last will be used for the comparison. The first approach was not successfully. The procedure was for all three models the same. For the first approach is the feature extractor approach with building fully connected network on top. Therefore, we used the pre-trained model up to the first layer of a fully connected layer and replaced the fully connected layer with our own version of a fully connected network. Instead of using a fixed fully connected layer, we searched with a hyperoptimization approach the best fully connected network, which optimize the loss of the network. For the hyperoptimization, we used the python library Hyperopt [1]. Hyperopt using a bayesian optimization method, to search in the hyperspace after the optimum. The search space for the hyperparameters are defined in Table 6.2.

Hyperparameter	Hyperspace
Learning Rate	$e^{\text{uniform}(-11,0)}$
Hidden Layers	{1, 2}
Width	{128, 256, 512, 1024, 2048}
Dropout	uniform(0, 0.7)

Table 6.2: The hyperspace for the hyperparameter search for the transfer learning.

The hyperoptimized fully connected network has between every fully connected layer a dropout layer for regularization. As output layer, we use a softmax activation, because both tasks are a classification task. The hidden layers has the activation function ReLU, because the original activation function was also ReLU.

For the fine tuning approach, we used also the pre-trained models up to the first fully connected layer. But instead of using the hyperoptimization, we using the best found fully connected network of the feature extractor approach. Furthermore, we used the weights of the fully connected network as initialization. This has the benefit, that the gradient update will be moderately. If we use a random initialized network on top, then it is possible, that we could destroy the learned weights of the pre-trained model, because of large gradient updates. Therefore, it is better to train firstly the added layer

[1] The weights will be not updated of the transferred model.

with a frozen transferred model, and then start to train with the complete transferred model[2]. For the fine tuning, we used all layers and the learning rate is decreased to 0.0001. The learning rate should be very low for fine tuning.

6.1.1 Inception-v3

The Inception-v3 [52] model is the third version of the GoogleNet [51] model. The network has a relative depth of 42 layers. The goal was to have fewer parameters and a faster computation as the VGG-Nets [47]. And as we see in Table 6.1, that this was successful. All convolution layer uses the activation function ReLU, and apply the batch normalization. The Inception-v3 starts with default convolution and pooling layers, to reduce the dimensional size of a feature map. Then it starts with the main characteristics of the network, the inception modules. The inception modules are parallel running convolution layers with different kernel sizes. One type of inception block is depicted in Figure 6.1, but there are two further types of inception blocks, which follows the same principles, but with other kernel sizes. The idea behind the inception module is, that it can extract similar features on different scales. The 1×1 kernel sizes has the purpose of the reduction of the number of feature maps, before is applied the expensive 3×3 convolution layer. For a detailed description of the network, we recommend the paper of Szegedy et al. [52].

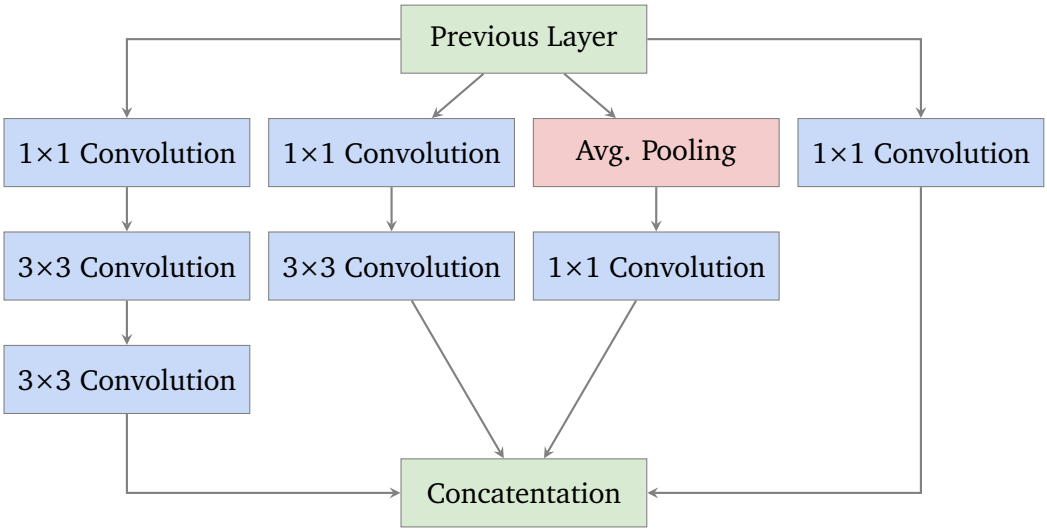


Figure 6.1: Example of an inception module in Inception-v3. The two successively 3×3 convolution layers have the same receptive field as a 5×5, but with the benefit, that they are faster to compute. The average pooling is executed with a stride of (1,1), so the pooling has no down sampling character. Furthermore, all layers applying padding, to produce the same spatial dimension of the feature maps, which are concatenated at the end of the inception module.

The network was designed for the ImageNet challenge and process images with the size of 299 × 299 pixels. We used the Inception-v3 with smaller images, and with another aspect ratio. Therefore, we must change the behavior of the first two pooling layers, so that they add a padding and not omit the information. Our used framework has the behavior to skip the last row/column, if the size of the feature map is not divisible by stride size. Otherwise, we had the problem, that the feature maps have a size of zero at the end. Furthermore, we change the last average pooling to a global average pooling [30]. It has the same purpose, like the original layer, but the global average pooling is independent of the spatial dimension of the feature maps. The last average pooling layer has in Inception-v3 the purpose to down

[2] This hint was found on a blog post of François Chollet. Source: "https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html", last visit at 03.11.2016.

sample every feature map to one value, but it depends on the spatial dimension of the feature maps, because the default pooling was used.

6.1.2 ResNet-50

ResNet-50 was developed by He et al. [18]. They observed, that deeper networks have a higher training error, as similar shallower networks. They had the idea, to add shortcut connections to the network, which they called identity mapping. The shortcut connection skips one or more layers and is then inserted again with an addition into the network. Figure 6.2 shows a residual block with this behavior. With this design of identity mapping, it is possible to build deeper models, with better performance. In [19], He et al. showed, that it is possible to build a network with 1000 layers with an improved approach.

As in Inception-v3, the input is firstly reduced with normal convolution and pooling layers. Some of the residual blocks have in the first convolution layer a stride of 2, to reduce further the spatial dimension of the feature maps. In these cases, the shortcut connection has also a convolution layer with stride 2. All convolution layers using batch normalization and ReLU activation function.

As in Inception-v3, the last average pooling layer has the task to bring every feature map to one value. Again, we replace this average pooling with global average pooling [30], to have the same functionality independent of the input size. Furthermore, we change the first pooling, so that it is using a padding. Otherwise, we had the same problem as in Inception-v3, that the feature maps could have a zero spatial dimension.

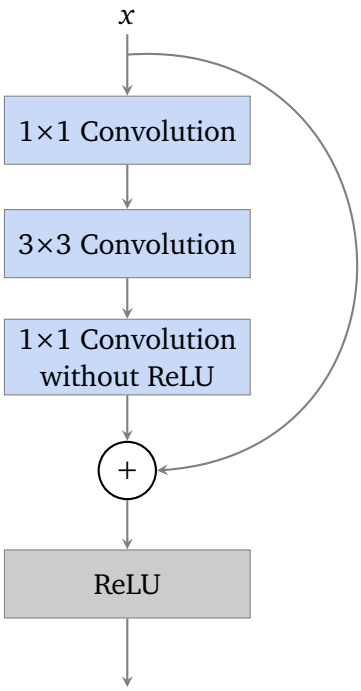


Figure 6.2: Example of a residual block in ResNet-50. All convolution layers uses batch normalization. The first 1×1 convolution layer reduce the size of the feature maps, and the second increases the size of the feature maps to the previous value. Please consider, that the last convolution layer has no activation function, the other convolution layers use ReLU as activation function.

6.1.3 VGG16

VGG16 was developed by Simonyan and Zisserman [47]. They used a simple sequential convolutional network, with 16 layers. The layers are convolution layers with the kernel size of 3×3, max pooling with size 2×2 and on top three fully connected layers. The network architecture is depicted in Figure 6.3a.

In the VGG16 are stacked two or three convolution layers with a kernel size of 3×3 to simulate a receptive field of 5×5 or 7×7. This has two benefits. The first, it will be applied more non-linearities. And the second, it reduces the size of parameters. Three stacked convolution layers with a kernel size of 3×3 needs only $3(3^2C^2) = 27C^2$ parameters, instead of one convolution layer with a kernel of the size 7×7, which needs $7^2C^2 = 49C^2$ parameters, where C describes a constant number of feature maps, which are going in and out on the convolution layer.

The simple network has his price. The VGG16 network is computationally expensive on huge input images. For the ImageNet challenge, the network gets images of the size 224×224. To train this network, Simonyan and Zisserman needed four NVIDIA Titan Black GPUs for 2-3 weeks.

The high computing time is due to the first layers of the network. Instead of reducing the spatial dimension of the feature maps in the first layers, as it is done by Inception-v3 and ResNet-50, it will be further applied on the same spatial dimension of the feature map. This means, that the first two convolution layers are applied on feature maps with the spatial dimension of 224×224. This results in a huge number of multiplications and additions, to compute the convolution.

A further problem is the transition of the convolution layer to the fully connected layer. At this point, the network has 512 feature maps with a dimension of 7×7. Every unit is now fully connected with the first fully connected layer with the width of 4,096. This results in $512 \cdot 7 \cdot 7 \cdot 4{,}096 = 102{,}760{,}448$ parameters. This transition makes about 74% of the parameters of the network.

For completeness, we have not applied any modification on this network, to use it for the transfer learning.

6.2 Our Model Architecture

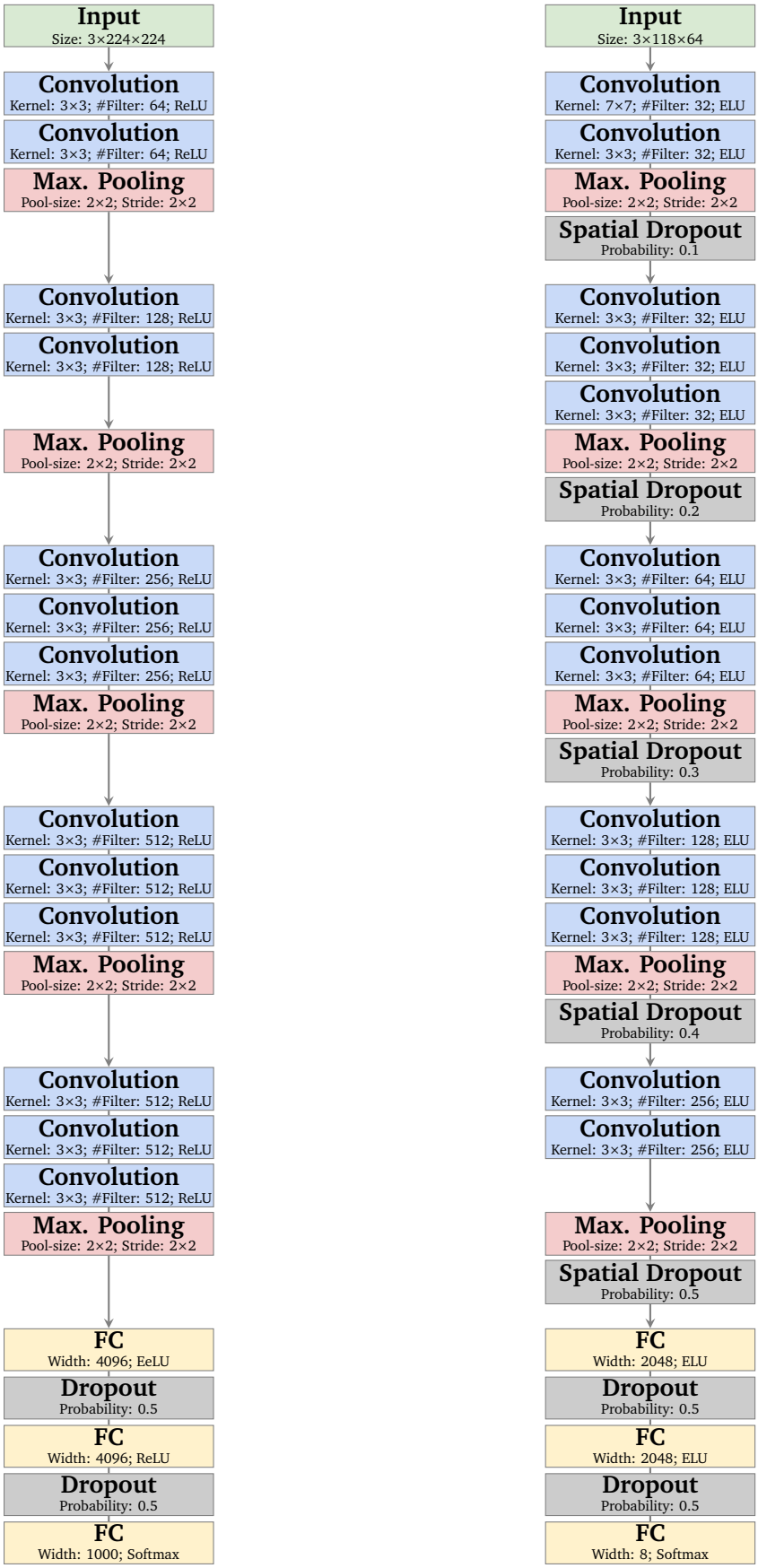
We had the goal, to develop a model, which has a similar performance, like the tested models in the transfer learning. Furthermore, our model should be faster than the models of the transfer learning. As basis for our model, we take the VGG16 model, because the VGG16 model reached the best performance on the transfer learning on small input images.

For an effective way to reduce the computation time, we decided to use a default input size of 118×64 pixels. On this resolution, humans can easily recognize the details of the vehicle, which is a good sign, that the resolution is not to tight for a recognition. In Figure 6.4 are depicted four images of the same vehicle with different resolutions, to get an idea, how much information contains in an image of different resolutions.

Furthermore, we reduced the number of filters in the convolution layers and reduced the width in the fully connected layers. This helps further to reduce the training time. Through the reduction of the input size and the reduction of the width of the fully connected layers, the transition between the convolution and fully connected layer has been extreme decreased. In our final architecture, the number of parameters between these layers are only 4,194,304 parameters and for the complete network, we have only 9,797,448 parameters to train.

We tested a huge amount of different modifications and other interesting techniques, but two techniques had much impact on the performance. The first is the change of the activation function from ReLU to ELU. It has a bit better performance than ReLU with batch normalization. And the second is the adding of spatial dropout after every convolution stack, or rather after every pooling. It has no influence whether before or after pooling.

The final model architecture is depicted in the Figure 6.3b. We call our architecture the VPVC-Net. For comparison, on the left side is depicted the original VGG16 architecture. We change the first convolution



(a) VGG16 **(b)** VPVC-Net

Figure 6.3: Architecture comparison of VGG16 and VPVC-Net.

(58x32)

(118x64)

(176x96)

(234x128)

Figure 6.4: Comparison of different resolutions. The values under the images gives the number of pixels for the x- and y-axes.

layer to a kernel size of 7×7, and all other convolution layers has the size of 3×3. The 7×7 on the first layer is not expensive, because the number of input feature maps are only 3 for RGB images. With 7×7, we have a wide reception field, but to omit the second convolution layer decreasing the performance. So we don't reduce this stacking. Furthermore, we add to the second stack a further convolution layer, and removed a convolution layer on the last stack, compared to the VGG16. The number of filters for the convolution layers are for the first both stacks 32, and is then increased by a factor of 2 for every stack. The convolution- and the pooling-layers have a padding. So the convolution layers doesn't change the spatial feature map size, and the pooling-layer has no lost of information, if the spatial feature map size is not divisible by two. The first two fully connected layers has a width of 2,048 and the output layer – which is also a fully connected layer – has a width of 8, because we have for both tasks a multi-class learning problem of 8 different labels. So every unit represents one label. The output layer uses the softmax activation function. The dropout probability for the first dropout layer starts by 0.1 and is then increased with 0.1 for the further dropout layers, until it reached 0.5.

The dropout layers are only used during the training, and thus they belong actually not to the model. But we want highlight that the spatial dropout layers had a great influence to learn a better model and therefore we added the dropout layers to the model overview in Figure 6.3.

6.3 Multi-Task Learning

Multi-Task Learning [4] is the simultaneous learning of multiple tasks in the same model. The tasks should be related together, otherwise the performance of the tasks could be worse than in learning with a single task. The nice effect of multi-task learning is, that the performance could increase, because the tasks can learn a shared representation. This means, that they can strengthen the shared parameters. In

a further sense, the multi-task learning belongs to regularization, because it has a direct influence on the shared parameters on the trained network.

Caruana has showed in multiple works [4–6], that multi-task increases the performance on different application with neural networks. Other researcher have also applied multi-task learning with positive effects. For example Collobert and [10] applied it on Natural Language Processing and had success with this approach. Abdulnabi [2] used it to improve the recognition of attributes in images. And Zhang et al. [60] applied it on facial landmark detection, and they used for example the pose of the head as a further task.

In this thesis, we have two tasks, which are related together. Both are applied on the same image and should learn features for the vehicles to distinguish the pose or the vehicle-class. So the multi-task approach should be also applied. Therefore, the model from the previously section is changed, because it was only for single tasks. Hence, to the penultimate layer is added a further output layer. This is depicted in Figure 6.5.

One output layer is for the pose and the other for the vehicle class. The loss function for the training has changed. Instead of computing for one output the loss, it will be compute for two outputs the loss and add them up, to get the final loss, which is then used to train the network. So the cross-entropy function from Equation 2.10 is used two times in the loss function $J(\Theta)$:

$$J(\Theta) = L(\mathbf{p}_{\text{pose}}, \mathbf{q}_{\text{pose}}) + L(\mathbf{p}_{\text{vClass}}, \mathbf{q}_{\text{vClass}}) \tag{6.1}$$

where $\mathbf{p}_{\text{pose}}$ and $\mathbf{q}_{\text{pose}}$ are the target and prediction vectors for the pose, and $\mathbf{p}_{\text{vClass}}$ and $\mathbf{q}_{\text{vClass}}$ are the target and prediction vectors for the vehicle class.

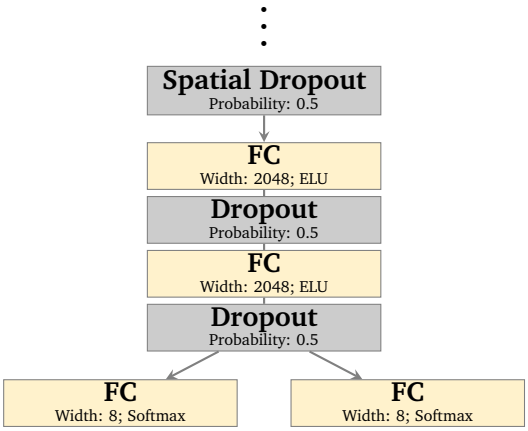


Figure 6.5: Extended Last Layers of VPVC-Net for Multi-Task Learning.

6.4 Expert Learning

Schenkel [44] has in his bachelor-thesis the idea, to use an *Expert Classificator* approach. Firstly, one of the task is applied and based on this result, it is applied a special trained classificator for the second task. For example, the first model classify the pose. Based on this pose, it is chosen an expert model to classify the vehicle-class. The expert model was optimized to classify the vehicle-class for this pose.

In our case, we would train nine different models. One model is for the classification into the pose. And the eight other models are for the classification of the vehicle-class dependent on the eight poses. The eight models for the vehicle-class are only trained on images for one pose, to get an expert for the vehicle-class from this pose. For example, all images which shows a vehicle from left, are used to learn an expert model for the vehicle-class. This model is then used to classify the vehicle-class, if the first classificator has classified the image into the left pose.

The benefit of this approach, is that the specialized models can become an expert for the second task, based on the assumption of the first task.

The drawback is, that the first model, which classify the first task, must be very well. Otherwise, the special trained models can only randomly guess the second task. Through the sequential use of the models, the error of the first model will cause with a high probability in an error of the second model. For example, if the first model has detected that the vehicle is viewed from left, but the true class is front, then can the special-model only randomly guess the vehicle-class. Therefore, the first task should be very well, to get a performance gain.

6.5 The Training

For the training of the VPVC-Net model, the model is implemented in Keras [7]. Keras is a high-level neural network framework, which run on top of Theano [54]. Theano is a framework to define, optimize, and evaluate mathematical expressions, and these could be done on GPUs. Keras abstracts the concepts of Theano, so that it is easy to model a layered network. Furthermore, it provides the possibility, to train the defined model.

We split this Section into two parts. The first part handles further regularizations or rather techniques, to improve the generalization error during the training. The second part defines the default values for the trained models, which are used for the evaluation.

6.5.1 Further Regularizations

We have provided one regularization method in the description of our model. This was the (spatial) dropout. But we use three further regularization methods, to improve the generalization. The first is the L2-Loss to regularize the weights. We use two different L2-Loss values, one is for the convolution weights, and the other is for the fully connected weights. Hence, the weights from the fully connected layer are more penalized than the weights of the convolution layer.

And the second regularization is an adaption of the learning rate. The adaption of the learning rate decreases the learning rate by a factor of 5, if no improvement occurred after 15 epochs. The third is the early stopping of the training. If no improvement occurred after 32 epochs, then it stops the training and the model from before 32 epochs is returned. Therefore, the model with the lowest loss is the model, which we have learned and using for the testing. The high number of epochs until we stop the training is owed by the learning rate decrease. We give a chance of two times to decrease the learning rate, before we stop the training. The loss on the validation-set is the measurement for the second and third regularization method.

The cross-entropy loss is a good performance measurement for the closeness of the prediction. It is possible, that the class is predicted correct, but the probability is low, but better than all other probabilities in the prediction vector. So the correct class is predicted, but the prediction for this example was not so good. In other words, the model is not clearly certain, which class it is, but it has a low trend to the right class. This closeness of the prediction to the target class, can express the cross-entropy loss. It is possible, that the loss is increasing, but the accuracy is stay on the same level. This shows the starting of overfitting, but the accuary does not show the effect of overfitting. Therefore, we use the loss for the early stopping.

6.5.2 Default Setup

The training of a model is very difficult, and has also influence on the performance of the model. If the learning rate is too high, then the model will be not converge. And if the learning rate is too low, then

the model needs much time and could stick in a poorly minimum. So the choice of the learning rate and the choice of other hyperparameter are essential for a good performance.

We tested for the main hyperparameter different settings, and we came to a setting, which is showed in Table 6.3. This is our default setup for the training. Furthermore, we add to the hyperparameter the possible settings of the preprocessing pipeline.

For the transfer learning, we use similar values. For the feature extractor approach, we used hyperoptimization to find the hyperparameter. So the replaced hyperparameter are depicted in Table 6.2 with their hyperspaces. Furthermore, we use no augmentation on the feature extractor approach.

For the fine tuning of the transfer learning models, the learning rate is changed 0.0001, the batch size to 32 and the early stopping stops after 7 epochs of no improvement.

The regularization for L2-Loss and reducing of the learning rate was not applied.

On all models, we use the cross-entropy loss function. As optimization method is used the stochastic gradient descent (SGD) with nesterov momentum, with a batch size of 64. Furthermore, we clipped the gradient to a norm of 4, if the norm of the gradient is higher than 4 [38]. The initialization of the weights are done with the Glorot initialization [14].

	Hyperparameter	Value
Model	Input Size	118×64
	Batch Size	64†
	Learning Rate	0.005†
	Momentum	0.90
	Epochs	200
	Gradient Clipping Threshold	4†
Regulariztion	L2-Convolution	0.0001‡
	L2-FC	0.001‡
	Reduce Learning Rate w/o improvements	15 Epochs ‡
	Reduce Learning Rate Factor	5‡
	Early Stopping	32 Epochs†
Preprocessing	Centering	Yes
	Histogram Equalization	Yes
	Colorspace	BGR
	Feature Scaling	Normalization
Augmentation	Rotation Range	[−15°,15°]
	X-Shift	15%
	Y-Shift	10%
	Zoom Range	[0.9,1.2]
	Color Jittering	10
	Probability of Cropping	25%

Table 6.3: Default hyperparameter for training and testing. † For transfer learning, the learning rate is 0.0001, batch size is reduced to 32 and the early stopping is applied after 7 epochs without improvement. ‡ This hyperparameters are not used for transfer learning.

6.6 Testing

For the testing of the learned models, the validation- and test-sets are used, which we have described in the Chapter 4. Before the images are fed into the model, the images are preprocessed in the same way, like the model was trained, but without augmentation. This means, that in general will be used the settings of Table 6.3, but without the settings of augmentation and regularization.

For every image and for every task, we get a vector for the prediction. This prediction contains the probability for every class, which describes the probability, that the model detects these classes. Therefore, the class with the maximal probability is used as the prediction for an image.

7 Evaluation

This chapter evaluates the performance of the VPVC-Net to predict the vehicle pose and the vehicle class. The VPVC-Net will be also compared to three ConvNets, which was presented in the previously chapter. Furthermore, it is evaluated the cooperation of the vehicle pose and vehicle class with multi-task learning and the expert learning approach. In addition, it will be tested different hyperparameter settings under the multi-task learning, because the multi-task learning showed good results. This has the benefit, that both tasks are tested with one model. After these results, it is tried to look deeper into the learned features of the basic multi-task model. At the end, the results are discussed.

The evaluation is applied on the test-set. This results represent the performance on unseen data, and are suitable for the evaluation. But before we start with the performance evaluation, we introduce the evaluation metrics, evaluate the training and prediction runtimes and show exemplary the progress of training/validation error rate and loss during the training.

7.1 Evaluation Metrics

For the evaluation of the performance of the vehicle pose and vehicle class will be used several metrics. The metrics quantifies the quality of the predictions. And therefore, they are usable for the comparison between different models.

For the evaluation is used as main metric the *error rate*. As further metrics, we use the *macro F1-score*, *top-2 error rate*, and took a look at the confusion matrix. We use the error rate metric, because it shows the ratio of misclassified samples, which is a good metric for us, because the goal is to reduce the error. Furthermore, the metric or the complementary (the accuracy), is used in the most related works, so it is useful for comparison, when we use it too. In addition, the value is easy to interpret. The macro F1-score is used to check for discrepancy in a single class. It would show a drop of the macro F1-score, if one class is in another model really bad classified. But before we start with the definition of these metrics, we introduce the confusion matrix, because from this could be derived the most metrics.

Confusion Matrix

The confusion matrix summarizes all useful information, to evaluate a model. But to compare different models, it is better to have metrics, which characterize the characteristics of this model. The most metrics could be derived from a confusion matrix.

		Predicted Class			
		A	B	C	Recall
True Class	A	$\mathbf{M_{AA}}$	M_{BA}	M_{CA}	R_A
	B	M_{AB}	$\mathbf{M_{BB}}$	M_{CB}	R_B
	C	M_{AC}	M_{BC}	$\mathbf{M_{CC}}$	R_C
	Precision	P_A	P_B	P_C	

Table 7.1: Example confusion matrix for multi-class classification

The default confusion matrix is defined for binary-class classification, but it is possible to extend the confusion matrix for multi-class classification. In Table 7.1 is depicted an example confusion matrix. In the entries of the confusion matrix are counted the predictions of the samples. So that the sum of all

entries is the number of all samples, let us call this N. The rows describe the true class, and the columns the predicted class. The samples are added to the entries in which they fall. For example, a sample which is predicted as B, but has a true class of A, would be added to the entry M_{BA}.

The diagonal of the confusion matrix describes the correct labeled samples. We added to the confusion matrix the recall and the precision as a further column and row. This is not common, but for the calculation of the F1score$_{\mathbf{M}}$, the values of the recall and precision are needed.

The recall defines the hit rate of a class. This is the ratio of the number of the correct predicted class divided by the number of all samples for this class, which is described by the sum of the row

$$R_X = \frac{M_{XX}}{\sum_Y M_{YX}} \tag{7.1}$$

where X and Y stands for every class in the confusion matrix.

The precision is a measure of the precision of the hit. It is the ratio of the number of the correct predicted class divided by the number of predictions for this class, which is described by the column

$$P_X = \frac{M_{XX}}{\sum_Y M_{XY}} \tag{7.2}$$

where X and Y stands for every class in the confusion matrix.

To compute the macro recall $Recall_{\mathbf{M}}$ and the macro precision $Precision_{\mathbf{M}}$, it will be calculated the average for R_X and P_X over all classes. This values will be need for the calculation of the macro F1-score.

Macro F1-Score

The macro F1-score is the harmonic mean of the macro recall and the macro precision.

$$\text{F1score}_{\mathbf{M}} = \frac{2 \cdot Precision_{\mathbf{M}} \cdot Recall_{\mathbf{M}}}{Precision_{\mathbf{M}} + Recall_{\mathbf{M}}}. \tag{7.3}$$

The macro recall and the macro precision have the same weight in the F1$_{\mathbf{M}}$. It combines the two measurements to one score, and the score reaches its best value at 1 and the worst at 0. The combination of macro recall and macro precision makes the F1score$_{\mathbf{M}}$ sensitive to outliers in single classes. This means, if a single or multiple classes are bad classified, then would this cause in a drop of the F1score$_{\mathbf{M}}$, if the error rate stays the same. But normally, if the error rate increases, the F1score$_{\mathbf{M}}$ decreases and the if the error rate decreases, the F1score$_{\mathbf{M}}$ increases.

Error Rate

The error rate could be also computed from the confusion matrix. Therefore, the accuracy is derived from the confusion matrix and then is computed the complementary to get the error. The accuracy is given by the diagonal of the confusion matrix divided by the number of samples N. Hence, the equation for the error is:

$$error = 1 - accuracy = 1 - \frac{1}{N} \sum_X M_{XX}. \tag{7.4}$$

The error rate metric could not represent classes, which are bad classified. Above all, if the data set has imbalanced classes. For example, we have two classes, one class has 90 samples, and the other 10. If the error rate shows, that 10% of the data is misclassified, then could this mean in the best case, that the small class has 10% error and the big class has only 10% error. But in the worst case, could this mean, that all samples are labeled as the big class, and the small class has 100% error.

The vehicle pose is relative well distributed (compare Figure 4.3), but for the vehicle class could be the class *Pickup* a problem (see Figure 4.4). But we will see later, that the Pickup class is well classified.

Sometimes, we look at the top-2 error, which describes the error, if the first and second guess was wrong. It is not possible to compute this metric from the confusion matrix.

McNemar-Test

If two models have nearly the same performance, then means this not necessarily, that the model with the higher performance is better. To decide if one model is better, it is possible to do a statistical test, with the null hypotheses, that both have the same distribution. McNemar-Test [32] is a simple test, to test this hypotheses.

For the calculation, it is only needed two frequencies B and C. If model 1 make the correct guess, but model 2 not, then it will increment the value B. If model 2 make the correct guess, but model 1 not, then it will increment the value C.

The two frequencies, will be now used in a χ^2-Test, with one degree of freedom and the probability of 0.95. This results in

$$\frac{(B-C)^2}{B+C} < 3.84 = \chi^2_{1;0.95} \tag{7.5}$$

and if this equation is not fulfilled, then exists a significant difference in the two models.

7.2 General Evaluation

In the previously chapter, we have mentioned, that the VPVC-Net should be faster to train, as the three ConvNets. To compare the training time of the VPVC-Net and the three ConvNets, it is measured the training time for one epoch on different input resolutions. One epoch contains the training of 58,344 images and a following validation of 19,391 images. The training was for the a single task. But for multi-task are the values similar. The times are measured on a Nvidia Geforce GTX 960 and are depicted in Figure 7.2. The VPVC-Net is of all tested ConvNets the fasted. With increasing resolution, the training time increase exponential. If we double one side on the image, it increases the image size by a factor of four. This has much influence on the ConvNets, because the feature maps are all increased by a factor of four. The steep rise of the runtime of the VGG16 lies in the transition of the convolution layer to the fully connected layer. This increases also with a factor of four.

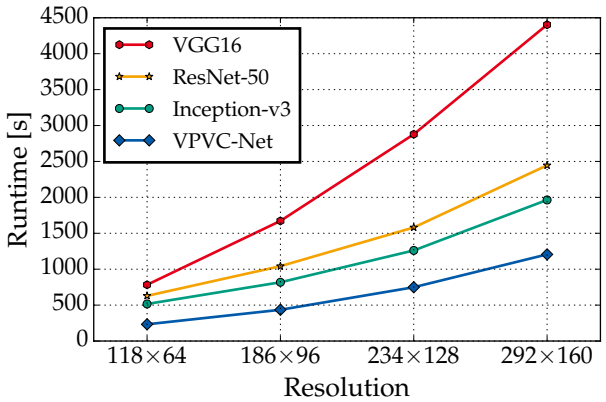


Figure 7.1: Runtime comparison for the training of one epoch. The training time was measured on an Nvidia Geforce GTX 960. One epoch contains the training of 58,344 images and a following validation of 19,391 images.

The interesting point is, that the VPVC-Net has with a resolution of 118×64, a better error, than the other three ConvNets with a higher resolution. For example, on the vehicle class is the VGG16 with the highest resolution the only network, which is significant better as the VPVC-Net.

We compared also the prediction time. We tested the batch-sizes 1 and 64 on two different GPUs. The measuring contains only the time for the prediction on the GPU, and no time for the preprocessing of the image. We collected 100 measuring points and averaged them. The results are depicted in the Table 7.2.

		Prediction runtime [s]							
	GPU	Nvidia NVS 5400M				Nvidia Geforce GTX 960			
	Resolution	118×64	186×96	234×128	292×160	118×64	186×96	234×128	292×160
Batch-Size 1	Inception-v3	0.278	0.303	0.343	0.362	0.015	0.017	0.019	0.020
	ResNet-50	0.157	0.239	0.240	0.274	0.011	0.013	0.015	0.018
	VGG16	0.180	0.293	0.412	0.606	0.0047	0.0088	0.014	0.018
	VPVC-Net	**0.029**	**0.042**	**0.055**	**0.076**	**0.0024**	**0.0035**	**0.0048**	**0.0066**
Batch-Size 64	Inception-v3	4.712	6.468	10.325	16.120	0.067	0.146	0.257	0.409
	ResNet-50	5.368	9.581	12.947	-	0.105	0.220	0.384	0.602
	VGG16	8.037	17.365	-	-	0.157	0.350	0.632	0.983
	VPVC-Net	**0.998**	**2.560**	**3.571**	**6.182**	**0.045**	**0.099**	**0.177**	**0.275**

Table 7.2: Prediction runtime for batch-size 1 and 64 on the GPUs Nvidia NVS 5400M and Nvidia GTX 960. The Nvidia NVS 5400M has problems with the cooling for the batch-size with 64 and lower the clocking, which cause in higher runtimes. Furthermore, for three experiments was not enough memory available. The values in black denotes the best value for this setting.

The Nvidia NVS 5400M is a notebook GPU from the year 2012, and has not much power. Furthermore, it has cooling problems and thus the clocking of the GPU is decreased during the experiments with batch-size 64. Therefore, the results are not really representable. But for the experiments with a batch-size of 1 is this problem not occurred. The results for the NVS 5400M show, that an old GPU has enough power to predict around 30 single images in the second with the VPVC-Net. The other ConvNets are not a so fast. But with a modern GPU, the performance are heavy increased for all ConvNets. With the VPVC-Net and the smallest resolution, the GTX 960 could around 400 single images per second process. But the other ConvNets are also not bad on the GTX 960. It is interesting, that VGG16 for batch-size 1 and a low resolution has a faster processing as the Inception-v3 or ResNet-50, but with batch-size 64, is the Inception-v3 much faster as the ResNet-50 and VGG16. But the VPVC-Net scales not so good for large batch-sizes like the Inception-v3 or the ResNet-50. The VPVC-Net has for example on the lowest resolution from batch-size 1 to batch-size 64 an increasing runtime factor of 19, but the Inception-v3 has only an increasing runtime factor of 4. We have also evaluated the runtime for the batch-size 128, but these are not depicted in the table. The runtime increases only with a factor of 2 for all ConvNets from batch-size 64 to 128. Therefore, the VPVC-Net will be stay faster than the Inception-v3 even if the batch-size is much more increased.

The comparison of the four ConvNets for the runtime of training and prediction shows, that the VPVC-Net is well designed for a fast processing. It is in both cases faster than the three other ConvNets, and we will later see, it has on smaller resolutions a better prediction performance than the other three ConvNets.

A further general evaluation, which we want to present, is the training progress over the epochs, because it is very similar for the most trained models. The difference lies mostly in a stretching or compression of the curves, so that it is reached slower or faster the convergence limit.

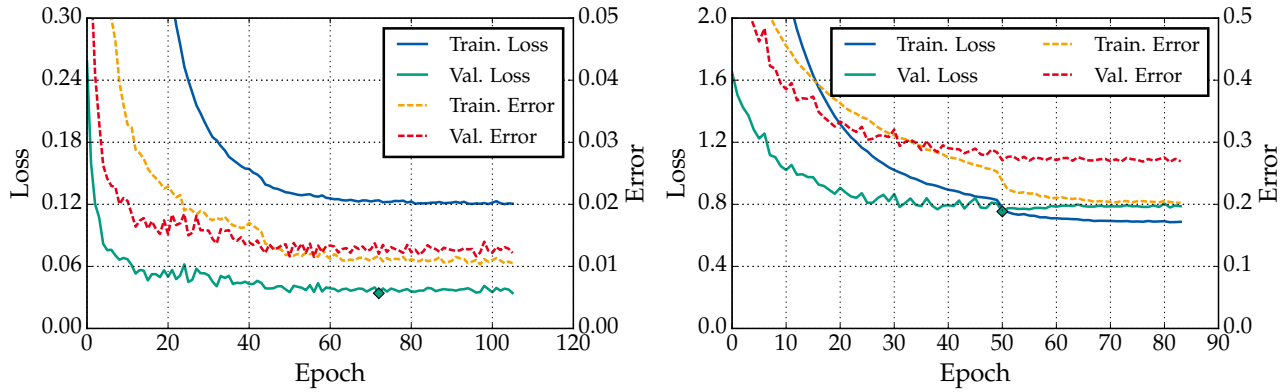
In Figure 7.2 is depicted a training progress for the two single-task models of vehicle pose and vehicle class. On every epoch is logged the loss and error for the training- and the validation-set. The loss for the training contains the L2-Loss of the regularization, and thus the training loss is higher than the validation loss. The diamond on the validation loss curve indicates the returned model from the early stopping. This means, that at this point was found the lowest validation loss.

It is interesting, that the validation error is in the first epochs much better as the training error. This effect happens, because we used dropout for the training. This means, that for the prediction during

the training is only used a subset of the network, but for the prediction for the validation is used the complete network.

Furthermore, we see on these plots the reducing of the learning rate. For vehicle pose, it is happened in epoch 44, which cause in a jump on the training error. The same effect could be seen in the plot for the vehicle class. Here is the first reduction happened in epoch 49. The further reductions are on both plots not visible. For the vehicle pose has the reduction of the learning rate a benefit, because it founds later a lower loss. For the vehicle class, the first reduction causes in the best minimum of the loss, and the further training does not improve the loss. The validation error has also no further improvement and stay on the same level. The training error and training loss, decreases after the learning rate reduction further, so the learning rate reduction was not to high.

Without the reduction of the learning rate, the gap between training error and validation error increases for the vehicle class. Furthermore, the loss starts to decrease, which imply that the model starts to overfit.



(a) Vehicle Pose **(b)** Vehicle Class

Figure 7.2: Progress of trainings loss and error, as well as validations loss and error for the vehicle pose and vehicle class on the single-task problem for the VPVC-Net. The diamond indicates the returned model from the early stopping. The trainings loss contains the L2-Loss from the regularization. Please note, that the axes have different ranges!

	Vehicle Pose			Vehicle Class		
	Error	Top-2 Error	$\text{F1score}_{\text{M}}$	Error	Top-2 Error	$\text{F1score}_{\text{M}}$
Run 1	0.0116	0.001	0.987	0.2963	0.1221	0.7141
Run 2	0.0117	0.001	0.9869	**0.2889**	0.1162	0.7205
Run 3	0.0117	0.0011	0.9868	0.2948	0.1174	0.7142
Run 4	**0.0112**	0.001	0.9876	0.3025	0.126	0.7107
Run 5	0.0126	0.0012	0.9862	0.2949	0.1285	0.7143
avg.	0.0118	0.0010	0.9869	0.2955	0.1220	0.7148
std.	0.0005	0.0001	0.0005	0.0043	0.0048	0.0032

Table 7.3: Five results for the single-task learning on the VPVC-Net for each vehicle pose and vehicle class. The last row shows the average and the standard deviation of the results. The best error is marked bold.

		Predicted Class								
		F	FR	R	BR	B	BL	L	FL	Recall
True Class	F	**2780**	4	0	1	12	0	0	14	0.9890
	FR	9	**2884**	19	1	0	10	1	0	0.9863
	R	0	21	**1251**	11	0	0	2	0	0.9735
	BR	0	0	11	**2087**	5	0	0	3	0.9910
	B	1	0	1	2	**2112**	2	0	0	0.9972
	BL	0	4	0	0	1	**2374**	13	0	0.9925
	L	0	0	2	0	0	17	**1896**	25	0.9773
	FL	11	0	0	8	0	0	8	**3931**	0.9932
Precision		0.9925	0.9900	0.9743	0.9891	0.9915	0.9879	0.9875	0.9894	

Table 7.4: Confusion matrix for single-task learning of vehicle pose.

7.3 Single-Task Learning

The single-task learning is the applying of every task on a unique model. We trained for the vehicle pose and vehicle class five models on the VPVC-Net to check the influence of different seeds. The seed has influence on the initialization of the weights, and this has influence to which local optimum the trained network converge. The best found models on the VPVC-Net for the vehicle pose and vehicle class are used as single-task baseline. Furthermore, the baseline of vehicle pose and vehicle class are compared with the three state-of-the-art models.

We describe the vehicle pose and vehicle class more detailed in this section, as in the other sections, because the most insights are the same. In addition, we go deeper into the problem of wrong labeled vehicle classes.

7.3.1 Vehicle Pose

The vehicle pose was trained five times, with different seeds. In Table 7.3 are depicted the results. The different seeds have only a little influence on the results. The standard deviation is very low. The McNemar-Test has only rejected the null-hypothese for Run 4 and Run 5 of the vehicle pose. The performance of the classification of the vehicle pose is in general very well. The error for the classification lies around 1.1% and the macro F1-score is also stable and near 1. The top-2 error lies around 0.1%. This means, that the first or second guess is mostly correct.

To look deeper into the distribution of the errors, it is in Table 7.4 the confusion matrix depicted for Run 4, because it was the best. The correct class is mostly hit. The errors happened either in the neighbor poses or in the opposite pose. For example, the errors of the neighbor poses are for **F**ront images in the poses of **F**ront-**L**eft or **F**ront-**R**ight. These errors come through images, which are lies between the two poses. Furthermore, the classification in **F**ront and **B**ack, hence the opposite side, is also a source of error. This is not unusually, because in [37, 44] was observed the same behavior. The appearance of the vehicles in these poses are very similar. They show the same shape, but some features are misinterpreted.

In Figure 7.3 are depicted example images of wrong predictions. We have found one image, which has a wrong label, but the prediction was correct. This image has a green border. The images with an orange and red border were predicted wrong, but with an orange border, the prediction was for a neighbor class. If we look at the images, we see, that the images could be also correct for this prediction. This shows, that the error rate should be lower than 1.1%. The really wrong classified images, are the

red images. They show the vehicle from the opposite side. Without the knowledge, that a big logo of the manufacture indicates the front, we would also have problems to label the first image of the “Benz SLS Class AMG”. A further problem are the doors, which are unusually open and are gull-wing doors. But the vehicle was in a second image also wrong labeled. We noticed on other wrong images, that vehicles with a long motor hood makes problems. The image for the “Ferrari California” is also wrong predicted. We think, that the red color of the vehicle removes the visible of the red back lights for the ConvNet. Furthermore, that the exhausts are detected as fog lights, because of that, the vehicle is labeled as front image.

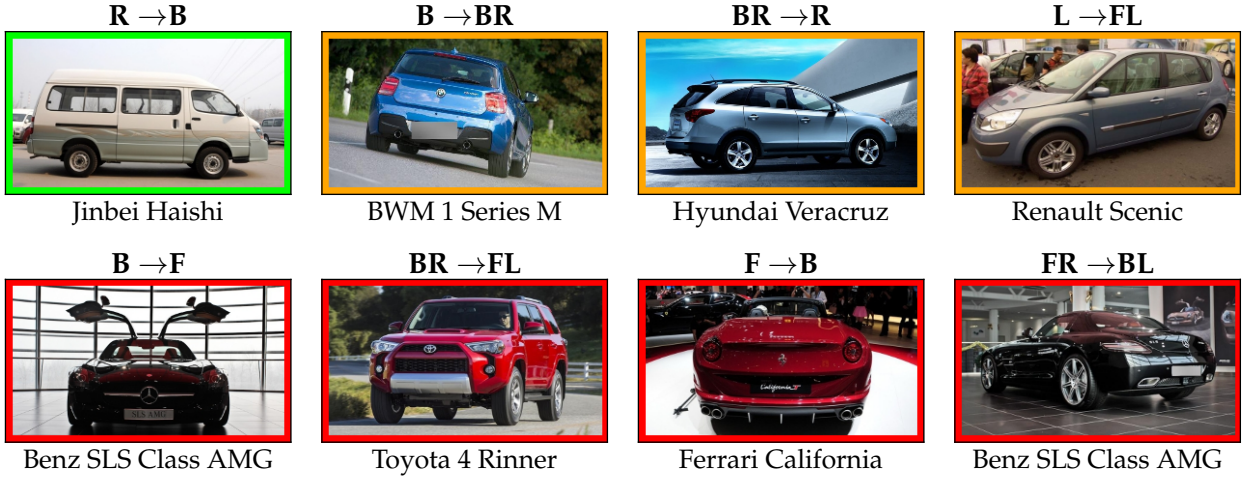


Figure 7.3: Wrong predicted example images for vehicle pose. The → indicates the predicted class to the true class. For example, R → B indicates, that **R**ight is the predicted class and **B**ack is the true class. The colors have also a meaning. Green, the prediction is correct, but the label was wrong. Orange, the prediction was for a neighbor class, but the image shows that the predicted class could be also correct. Red, the opposite was predicted.

7.3.2 Vehicle Class

As next, we look at the vehicle class. The results are not so well, like for the vehicle pose. It was also tested five different seeds. The results are depicted in the Table 7.3. The variation of the possible errors are much bigger. The difference between the worst and best run are 1.36%. This is a huge difference. The McNemar-Test has also showed, that the most null-hypotheses are rejected between the most runs. The macro F1-Score shows no anomalies. If the error decreases, the macro F1-score increases, a normal behavior. We have also looked into the confusion matrices of the unique runs, but we have not found any anomaly. The differences of the confusion matrices shows only the same trends and imbalances. There are only light differences, which cause in an overall difference of 1.36%.

It is interesting, if we look at the Top-2 error, that the error decrease to around 12%. This means, that the second guess is mostly correct, and the VPVC-Net has learned a good tendency to the right class, but another class has a better probability.

To look deeper into the classification of the vehicle class, we show the confusion matrix of the best run of the vehicle class. The confusion matrix is depicted in Table 7.5. All classes have true hits, so the VPVC-Net has learned features to distinguish the classes. Some classes are not so good distinguished, which cause in a low recall and low precision. The prediction for the minibus is very well. Pickup is also good predicted, but it will be often classified as an SUV, but SUVs will be rarely classified as an Pickup. From the confusion matrix, we could see the classes which are related to each other. For example, the classes Small, Medium and Large are related together. They make much error among each other. This

		Predicted Class								
		Small	Medium	Large	SUV	Pickup	Sports	Van	Minibus	Recall
True Class	Small	**1005**	624	64	178	2	20	103	2	0.5030
	Medium	328	**2071**	638	202	0	149	53	2	0.6015
	Large	11	830	**3343**	281	3	182	20	0	0.7158
	SUV	111	88	80	**4247**	9	7	139	10	0.9054
	Pickup	3	0	4	122	**318**	2	2	2	0.7020
	Sports	166	140	408	63	2	**1184**	1	2	0.6022
	Van	78	142	68	214	0	2	**747**	7	0.5938
	Minibus	6	0	1	20	4	0	48	**976**	0.9251
	Precision	0.5884	0.5317	0.7258	0.7973	0.9408	0.7658	0.6712	0.9750	

Table 7.5: Confusion matrix for single-task learning of vehicle class.

is not surprising, because the difference between these vehicles are the size. Furthermore, we will later see, that some vehicle types of Small, Medium and Large are wrong labeled.

The class Van shows also a relation to Small and Medium class. The appearance of some mini-vans could be related to these two classes.

Sports has also a strong correlation to the class Large. From the relabeling, we know, that many large vehicles have an appearance of a Sports class, or was labeled as a Sports class. For example, the vehicle types “BMW M4 coupe” or “Audi S5 convertible”, which are labeled as sports vehicle, but the normal vehicle types of these vehicles are labeled as large. The appearance of these vehicles is very similar. This cause in a confusion of the ConvNet, to learn features to distinguish these two classes. A multi-label approach, could be solve this problem.

The pose has influence on the appearance of the vehicle class. Therefore, we show in Figure 7.4 the error rate of the vehicle class are depends on the vehicle poses. It noticeable, that the error for the prediction of the vehicle class is strong correlate with the pose. A vehicle from the front or from the back is much harder to predict, than from another pose. The best error rate is reached from the side. From the side, the VPVC-Net could detect the shape of a vehicle, which is mostly unique for the most vehicle classes. A small vehicle is small, a large vehicle is large, but flat. An SUV is large and high, and the distance ground to the vehicle is high. A sports vehicle is mostly flat, and the distance to the ground is small.

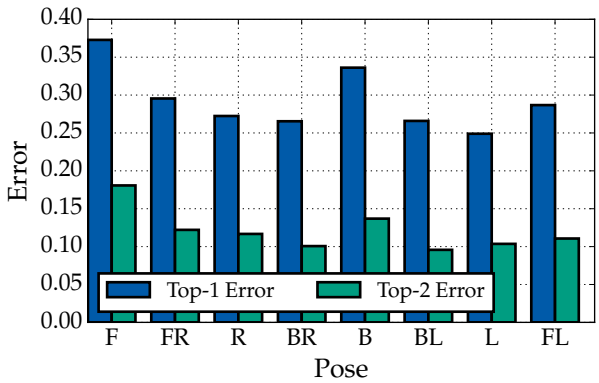


Figure 7.4: Vehicle class error and top-2 error for every vehicle pose. Lower values are better.

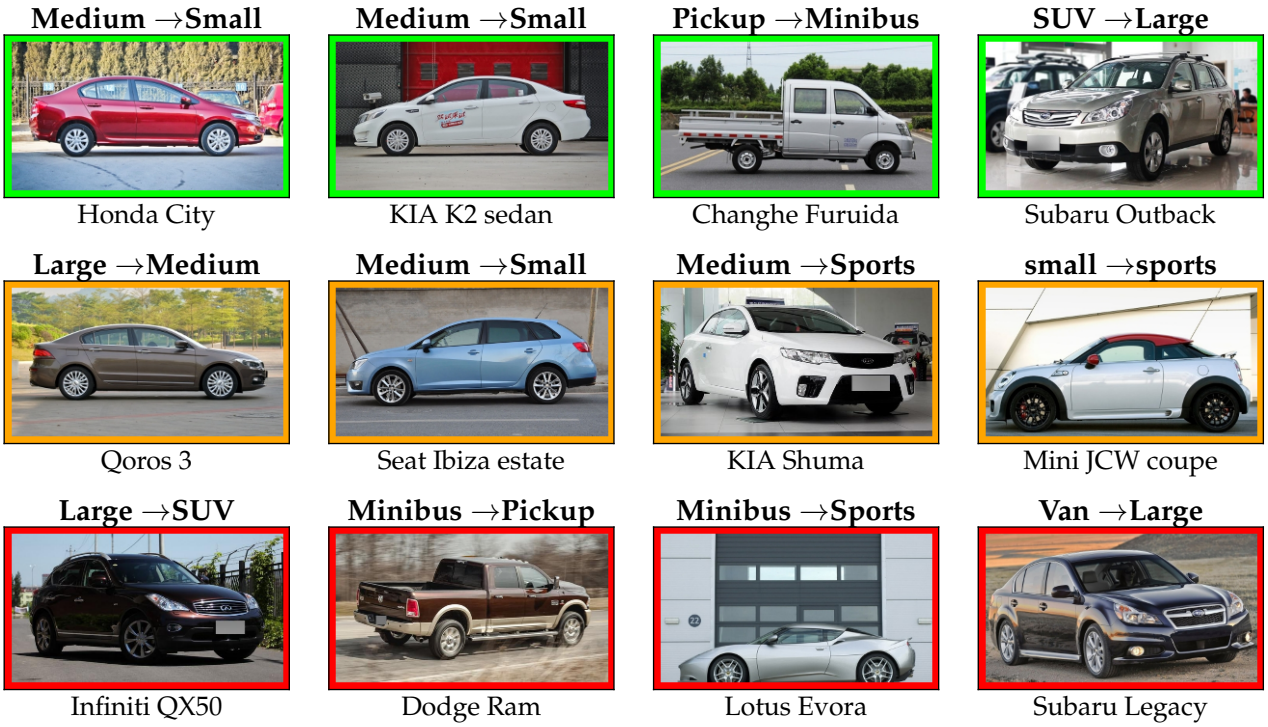


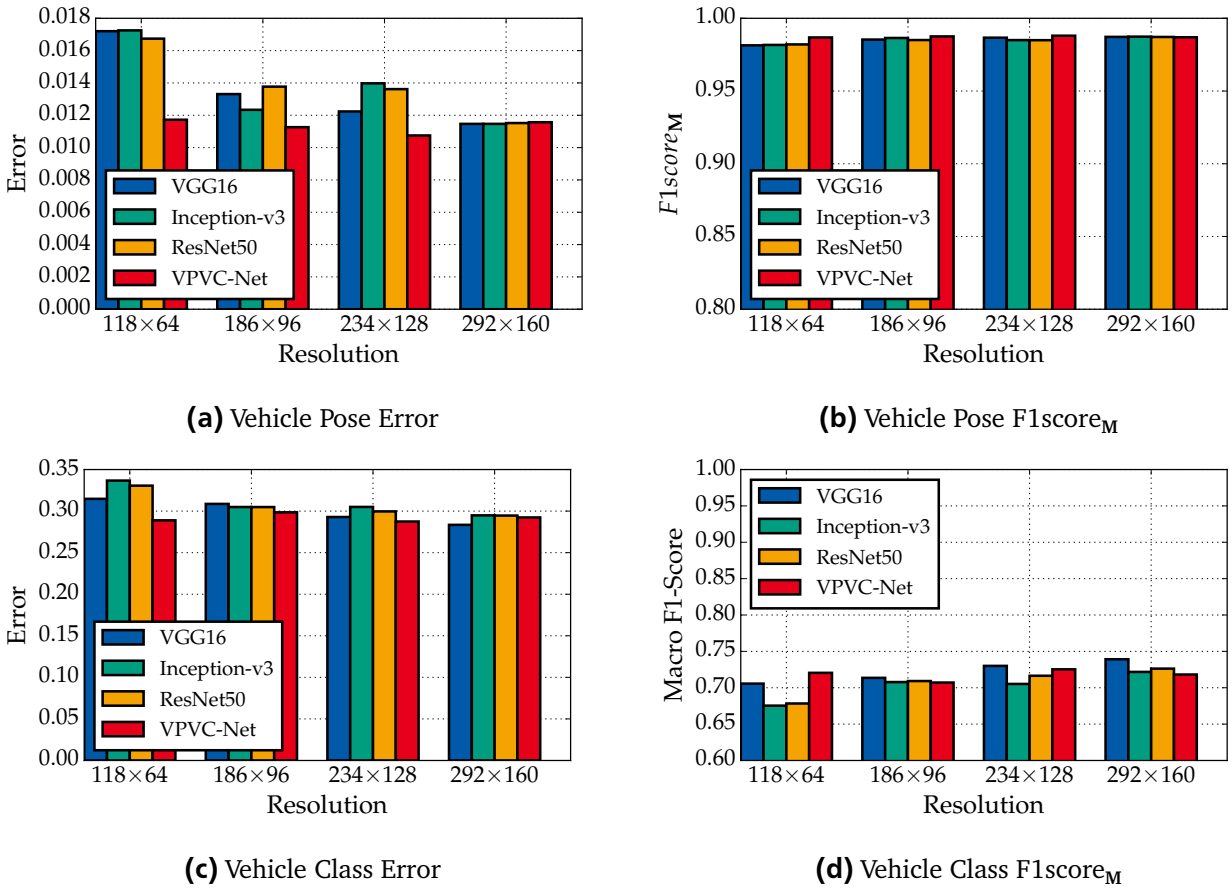
Figure 7.5: Wrong predicted example images for vehicle class. The → indicates the predicted class to the true class. For example, Large → Sports indicates, that Large is the predicted class and Sports is the true class. The colors have also a meaning. Green, the prediction is correct, but the label was wrong. Orange, the prediction was wrong, but the vehicle has a tendency to this class. Red, prediction was totally wrong.

Like for the vehicle pose, we also show images of vehicles, which are wrong predicted. These images are depicted in Figure 7.5. Some predictions were not wrong, but rather the labels are wrong. These are depicted in the first row. In the next section, we go deeper into this problem. The second row shows images, which are between two classes, or the prediction is not so wrong. For example, the "Seat Ibiza estate" is a small vehicle, but the appearance as an estate makes him larger, and without the knowledge that this is a Seat Ibiza, we would not say, that this vehicle looks like a small vehicle. The third row shows the totally wrong predicted vehicle classes. For the image with the "Lotus Evora" is the background the problem. The most minibuses have many windows, and this is availabe on the image. A segmentation of the vehicle from the background could help for such images.

In Figure 7.5 are the most images from the side. But there are also images from the front or back, which are wrong predicted. But for these images it is really hard to say to which class they belong.

7.3.3 The Problem of the Vehicle Class

We have in the previous section mentioned, that some labels could be in both classes, like in the Large and Sports class. This is one reason for a higher error. The other reason lies in the labeling of the images. In Chapter 4, we have described the relabeling of the images, and that we use the wikipedia as source to find the vehicle class label for a vehicle type. Unfortunately, the wikipedia provides only one vehicle class label for one vehicle type, but some vehicle types change their appearance with the change of the vehicle type generation. For example, the "Honda City", was in the first generation a small vehicle with a length of 3.3m. But in the seventh generation, it has changed his dimensions to 4.4m, and from the appearance, it could be a medium sized vehicle. The change of a vehicle class from the first generation to another vehicle class have we observed for some vehicle types in the test-set. And the remarkable fact is, that the VPVC-Net has predicted the most images for these vehicle types into the same class. For



(a) Vehicle Pose Error

(b) Vehicle Pose F1score$_{\mathbf{M}}$

(c) Vehicle Class Error

(d) Vehicle Class F1score$_{\mathbf{M}}$

Figure 7.6: Comparison of transfer learning models and VPVC-Net on single-task learning. The first two figures show the results for the vehicle pose, and the last two figures the results for the vehicle class. For Error, lower is better, and for F1score$_{\mathbf{M}}$, higher is better. Please take care, that the ranges of the Y-axes are different.

example, the "Honda City" from the Figure 7.5 has 120 images and from these images are 85 images predicted as the class Medium, and no image as Small.

For the test-set, we found in a short search 11 vehicle types which has a wrong vehicle class label. We found this problem also in the trainings-set. Mishkin et al. [33] has experimental showed, that a small clean dataset (without wrong labels) is better as a big dataset with noise. Therefore, the labels for the vehicle class should be reviewed and improved.

Unfortunately, we have detected this problem to late. Otherwise, we have it done for this thesis. But the time constraint does not allow it.

7.3.4 Compare with Transfer Learning Models

In this section, the VPVC-Net is compared to the three ConvNets, Inception-v3, ResNet-50 and VGG16, on the classification of the vehicle pose and vehicle class. The three ConvNets are using pretrained weights, which are provided by the Keras framework [7]. The models are then fine tuned, as we described in Chapter 6.1.

For a fair comparison, we increase the image size from 118×64 with a factor of 1.5 up to 292×160. The bigger images should help to improve the results, because the models are developed for bigger inputs as 118×64.

In Figure 7.6 are the results of the comparison depicted. The comparison includes the test of vehicle pose and vehicle class. As evaluation metric is used the error rate and the F1score$_{\mathbf{M}}$.

		Predicted Class								
		Small	Medium	Large	SUV	Pickup	Sports	Van	Minibus	Recall
True Class	Small	**1033**	656	54	170	1	19	61	4	0.5170
	Medium	289	**2106**	625	187	1	194	39	2	0.6117
	Large	16	782	**3407**	252	0	195	18	0	0.7296
	SUV	93	97	105	**4260**	5	5	113	13	0.9081
	Pickup	3	1	1	92	**354**	0	0	2	0.7815
	Sports	156	105	360	31	5	**1309**	0	0	0.6658
	Van	69	143	83	217	0	0	**729**	17	0.5795
	Minibus	8	0	0	10	4	0	35	**998**	0.9460
	Precision	0.6197	0.5414	0.7351	0.8162	0.9568	0.7602	0.7327	0.9633	

Table 7.6: Confusion matrix for multi-task learning of vehicle class

The three ConvNets could be profit from the higher resolution. With the highest resolution, the three ConvNets has for the vehicle pose a similar error as the VPVC-Net. The $\text{F1score}_{\text{M}}$ shows for the vehicle pose the same behavior.

For the vehicle class is the VPVC-Net also better on smaller resolutions. But higher resolutions for the VPVC-Net helped not the performance. The three ConvNets increases their performance, with higher resolution. With the highest resolution, the VGG16 is better as the VPVC-Net. This would be verified with the McNemar-Test. The $\text{F1score}_{\text{M}}$ shows similar behavior. If the error decreases, then increases the F1-Score.

It is interesting, that the VPVC-Net could not profit from the increasing resolution. But it has for the lower resolution already a well performance. With knowledge, that the training and testing speed is fast for the low resolution, and the good performance, the VPVC-Net is to prefer.

7.4 Multi-Task Learning

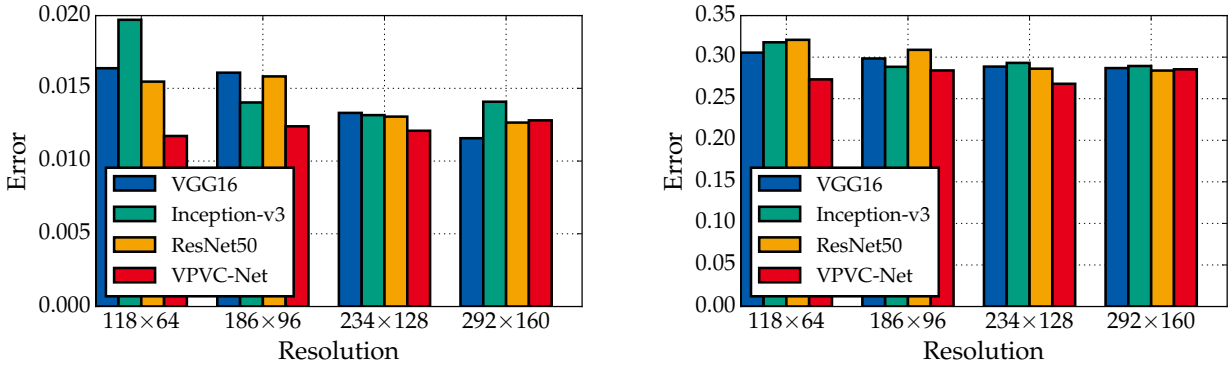
The multi-task learning is the simultaneously learning of the vehicle pose and vehicle class on one VPVC-Net. With multi-task learning was produced the best results for the vehicle class. Therefore, it was concentrated to test different settings on the multi-task learning, because this has the benefit, that we get with one model for both tasks results. The vehicle pose has sometimes a drop of the error rate, but compared to the gain of the vehicle class is the overall performance better. Furthermore, it will be concentrated to the evaluation metric of the error, because the $\text{F1score}_{\text{M}}$ shows similar results.

7.4.1 Baseline Multi-Task Learning

As first, we compare the multi-task learning with the results of the single-task learning, which is for the section of the multi-task learning the baseline. In Table 7.7 are depicted the results of the multi-task learning and the single-task learning. The multi-task learning improves clearly the vehicle class. But for the vehicle pose the single task reach a better error, but with McNemar-Test, there is no difference. The $\text{F1score}_{\text{M}}$ has a bigger difference as the error rate for the vehicle class. This indicates, that the internal values of the confusion matrix are getting better. In Table 7.6 is showed the confusion matrix for the vehicle class. If we compare the Recall and the Precision, then we see, that the most values have been increased. The diagonal entries have been also increased, except for the Van class. Here happened a little drop, but the precision for this class is go down. This means, that fewer misclassifications exists into this

	Vehicle Pose			Vehicle Class		
	Error	Top-2 Error	$F1score_M$	Error	Top-2 Error	$F1score_M$
Single-Task	**0.0112**	0.0010	0.9876	0.2889	0.1162	0.7205
Multi-Task	0.0117	0.0011	0.9871	**0.2733**	0.1077	0.7407

Table 7.7: Comparison of multi-task learning with the results of single-task learning. The results of *Multi-Task* are the baseline for the multi-task learning.



(a) Vehicle Pose Error

(b) Vehicle Class Error

Figure 7.7: Comparison of transfer learning models and VPVC-Net on multi-task learning.

class. Furthermore, the classification between SUV and Pickup has improved, which cause directly in a higher Recall for the Pickup class. All in all, the multi-class learning for the vehicle pose has increase the overall performance of the class. The increasing performance is not happened through an imbalanced increase of one class.

7.4.2 Compare with Transfer Learning Models

The multi-task learning was also applied on the transferred models. Therefore, was combined the top-layers of the models from the single-task learning comparison. The pre-trained weights was then loaded and the networks were fine-tuned.

The results of the error for vehicle pose and vehicle class are depicted in Figure 7.7. The VPVC-Net could not improve his performance with increasing resolution. For the vehicle pose, the performance decreases. For the three other ConvNets, the performance increases with the higher resolution. And at the end, the performance is for the vehicle pose better than for the VPVC-Net. But for the vehicle class, the performance is on the same level. If we compare the best performance of the three ConvNets, with the best performance of the VPVC-Net, then is the VPVC-Net much better.

It is interesting, that the other three ConvNets could not so much profit from the multi-task learning as the VPVC-Net.

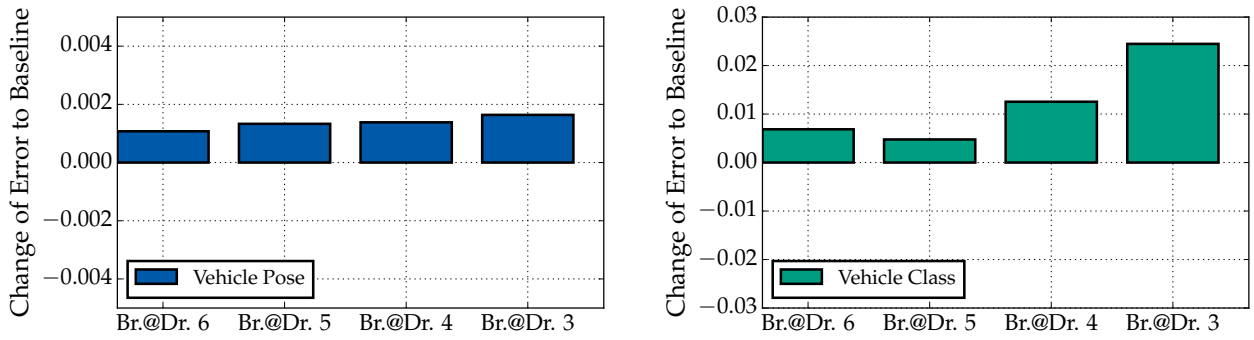
7.4.3 Change of Branching Layer

In Chapter 6.3 we have showed how the VPVC-Net was adapted to handle two tasks. Therefore, it was added a branch after the seventh dropout layer. In this section, it is tested the behavior of the system, if the branch layer is changed. Therefore, the position of the branch layer will be changed. The remaining layers will be doubled, so that for every output exists the VPVC-Net, but with shared layers until the branch.

We tested four models with a different position of the branching layer. The position of the branching layer is applied after a dropout layer. If we numerate from the dropout layers, then we get the number, where we apply the branching. For example, for the model “Br.@Dr.5”, we apply the branching after the fifth dropout layer.

In Figure 7.8 are the results for the change of the branching layer depicted. The figures show for vehicle pose and vehicle class the changes between the baseline error. Therefore, a positive value describes a higher error as the baseline.

With a shallower position of the branching, the error will be get worst. It is also better to split the network in a deeper layer.



(a) Vehicle Pose

(b) Vehicle Class

Figure 7.8: Comparison of the change of the branching layer on multi-task learning. The figures show the difference to the baseline of the multi-task learning, which can found in Table 7.7. A negative value describes a better error as the baseline.

7.4.4 More Tasks for the Multi-Task Learning

After Caruana [6], more extra tasks could help to improve the main task. Therefore, it is tested, whether more tasks could increase the performance. The CompCar dataset provides further labels, which can be used has extra tasks. The extra tasks are the *body class*, the *number of doors* and the *number of seats*. As a further task, it is added a binary task, which tries to predict the [small, medium, large] as size class, and [SUV, Pickup, Sports, Van, Minibus] as special purposes class. This label is named hierarchy.

We tested different combinations of these tasks. The vehicle pose and vehicle class are always used:

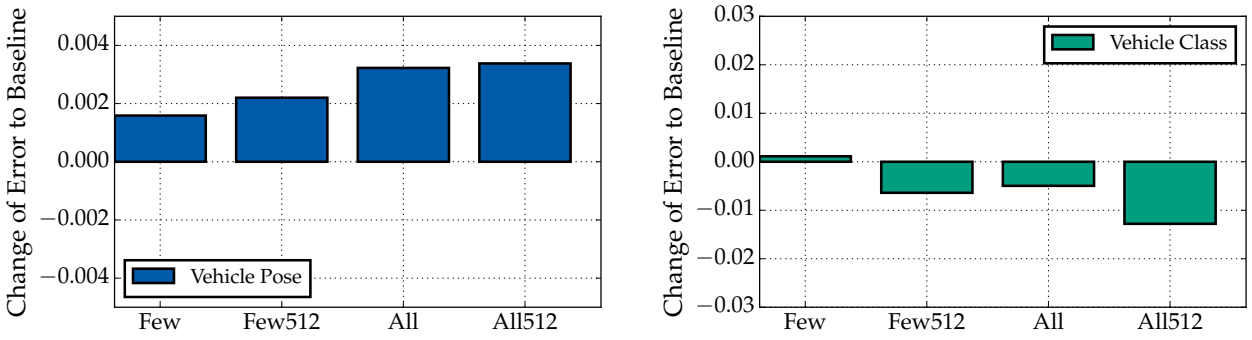
Few Add number of doors and number of seats as extra tasks

Few512 Same as *Few*, but to every output task is added previously a fully connected layer width the width of 512.

All Add number of doors, number of seats, body class and hierarchy as tasks.

All512 Same as *All*, but to every output task is added previously a fully connected layer width the width of 512.

In Figure 7.9 are depicted the results. The vehicle pose could not profit from more tasks. But the increasing of the error is moderately. Adding of doors and seats has no performance gain, but if we add further fully connected layers, it helps to increase the performance for the vehicle class. If we add the body class and the hierarchy as further tasks, it increases the performance significantly. The model for *All512* has with these changes an error of 0.2604 and is thus the best single model for VPVC-Net, which we have found.



(a) Vehicle Pose

(b) Vehicle Class

Figure 7.9: Comparison of more tasks on multi-task learning. The figures show the difference to the baseline of the multi-task learning, which can found in Table 7.7. A negative value describes a better error as the baseline.

7.4.5 More Data for the Training

More data for the training improves mostly the performance. Therefore, we use the validation-set also for the training. Furthermore, we relabeled the *Stanford Cars Dataset* [25] for the vehicle class. The vehicle pose is ignored for that dataset. This means, that the loss for the vehicle poss is zero for images of this dataset.

Through the adding of the validation-set to the training-set, we have lost an independent validation-set for the early stopping and the reducing of the learning rate. Therefore, we change the training method and use a linear learning rate reduction. The linear learning rate reduction reduce linear the learning rate in every iteration, so that at the end of the training the learning rate is zero. Furthermore, the model is trained on the complete 200 epochs without early stopping. The test-set stays untouched, and is only used to produce the results for these experiments.

We tested three different settings. The first (A) adds the validation-set to the trainings-set. The second (B) adds the validation-set and the stanford dataset to the trainings-set. The last (C) is the same like the second setting, but we apply the *Extreme512* from the previously section.

The results are depicted in Figure 7.10. The vehicle pose could profit from the more data, but with adding of more tasks, the performance decreases and is again so good like the baseline. The vehicle pose increases significantly with more data. The applying of more tasks increases further the performance. In Table 7.8 are depicted the exact values. It is interesting, that around 60.000 images (training-set size) are not enough for a good training of the vehicle class. With around 36.000 images more, the VPVC-Net could be significantly increase the performance. We could conclude, that more images is needed for a better classification!

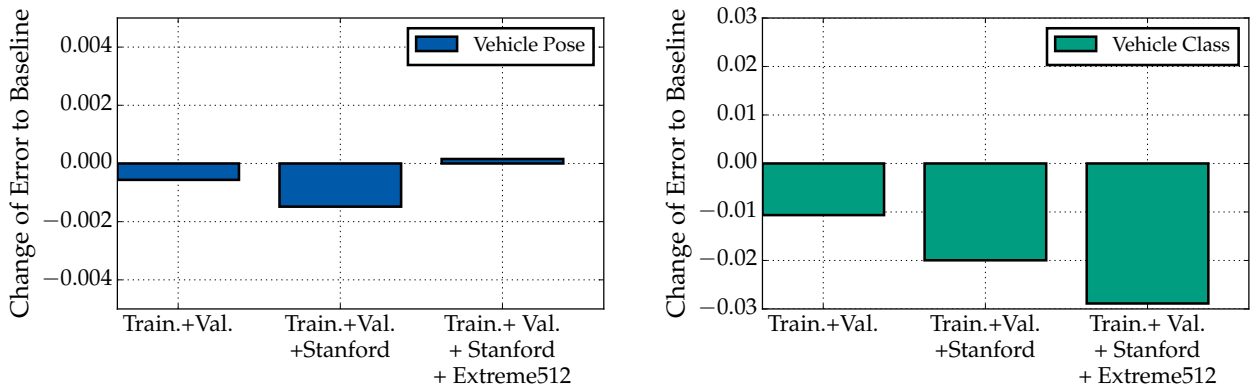
We do not count the results of this section to our best results. The adding of more examples increases in the most cases the performance.

7.4.6 Change of Hyperparameters

In this section, we investigate the influence of different Hyperparameters. We tested different preoprocessing settings and also different augmentation settings. At last, we test different dropout settings, to show the importance of dropout for the VPVC-Net.

	Vehicle Pose			Vehicle Class		
	Error	Top-2 Error	$F1score_M$	Error	Top-2 Error	$F1score_M$
Baseline	0.0117	0.0011	0.9871	0.2733	0.1077	0.7407
Setting A	0.0112	0.0009	0.9876	0.2626	0.0958	0.7511
Setting B	**0.0102**	0.0006	0.9886	0.2533	0.0860	0.7668
Setting C	0.0119	0.0011	0.9869	**0.2444**	0.0802	0.7753

Table 7.8: Comparison of more data on multi-task learning.



(a) Vehicle Pose

(b) Vehicle Class

Figure 7.10: Comparison of more data on multi-task learning. The figures show the difference to the baseline of the multi-task learning, which can found in Table 7.7. A negative value describes a better error as the baseline.

Preprocessing

In the Chapter 5 was introduced the preprocessing pipeline. In this section, we change some preprocessing parameters, and investigate the behavior of these settings.

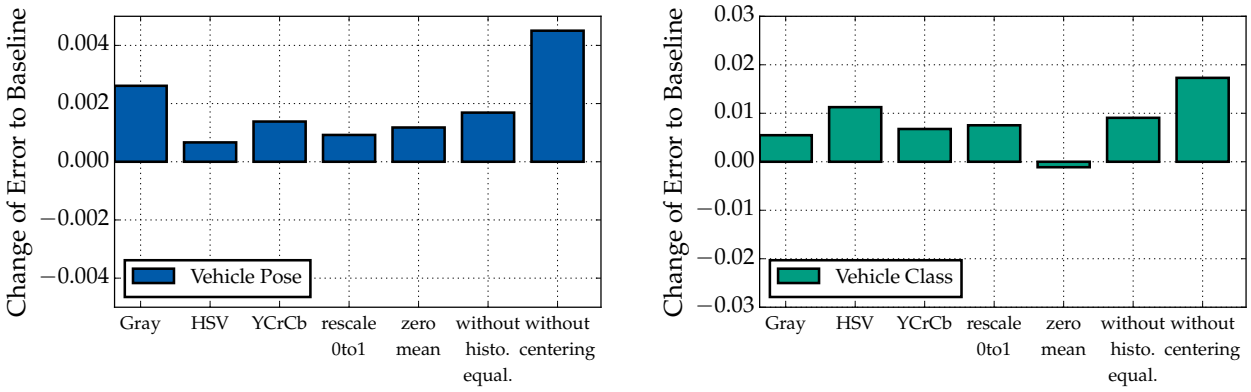
The settings include the change of colorspace (Gray, HSV, YCrCb), the change of feature scaling (rescale 0to1, zero mean), without histogram equalization and without centering of the vehicle.

The results are depicted in Figure 7.11. The color information is for the vehicle pose important, otherwise the gray colorspace has not a relative high drop. The change of these settings decrease the performance compared to the baseline. Only the feature scaling with zero mean has a light increase, but after the McNemar-test, the difference is not significantly. The highest drop has the setting *without centering*. The vehicle could be overall in the image, and with different sizes. Therefore, the VPVC-Net must be also learn the features on different scales. The settings for the preprocessing from the Table 6.3 for the default parameters are well choose.

Augmentation

For these experiments, we change the settings for the augmentation. The settings for different augmentation were, no augmentation, low augmentation, high augmentation, the settings for these three are depicted in Table 7.9.

The results are depicted in the Figure 7.12. No augmentation is a bad idea. It has the highest drop of all tested multi-task compared to the baseline. The other two settings are for vehicle pose on a similar level like the baseline. But for the vehicle class, the low augmentation is better. This means, that we



(a) Vehicle Pose

(b) Vehicle Class

Figure 7.11: Comparison of preprocessing hyperparameters on multi-task learning. The figures show the difference to the baseline of the multi-task learning, which can found in Table 7.7. A negative value describes a better error as the baseline.

	no	low	high
Rotation Range	[0°,0°]	[−10°,10°]	[−30°,30°]
X-Shift	0%	5%	20%
Y-Shift	0%	5%	20%
Zoom Range	[1,1]	[0.9,1.2]	[0.8,1.3]
Color Jittering	0	5	20
Probability of Cropping	0%	0%	50%

Table 7.9: The settings of the augmentation for the testing of different augmentation levels.

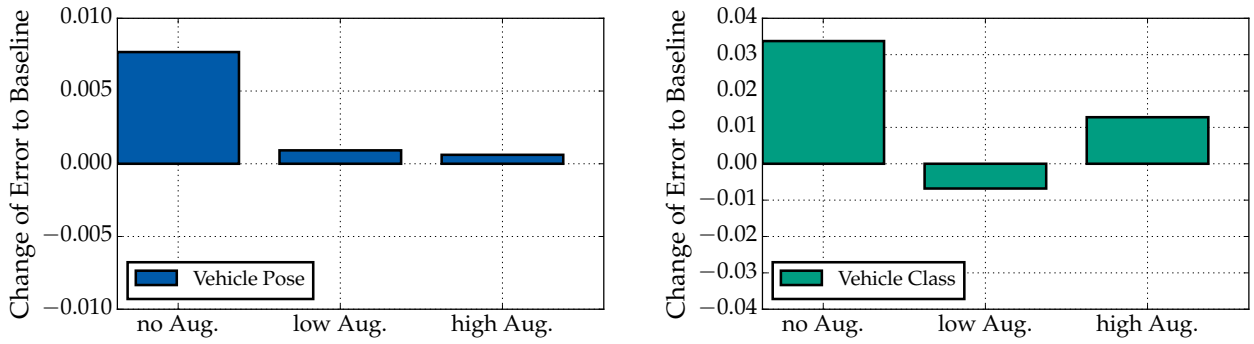
should change our default settings for the VPVC-Net to the low augmentation. The higher augmentation leads to decrease the performance.

Dropout

In this section, we test the behavior of different dropout settings. We tested no dropout, low dropout, and high dropout. For no dropout test, all the probability of all dropout layers was set to 0. For the low dropout test, the probability for dropout was for all dropout layers reduced with 0.1 outgoing the default setting. For the high dropout test, the probability for the dropout was for all dropout layers increased with 0.1 outgoing from the default setting.

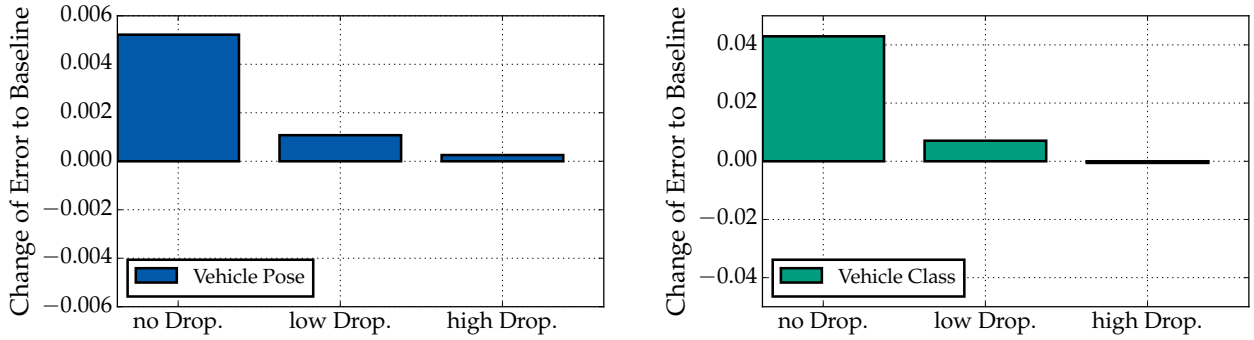
The results are depicted in the Figure 7.13. With no dropout, the VPVC-Net makes a higher error, as with dropout. The lower dropout could be to low. The higher dropout has no significant change for vehicle pose or vehicle class. But the drawback of higher dropout is the increasing runtime. It needs longer to converge to a local minimum.

To conclude the results: It is important to use the centering, augmentation and dropout. Without them, the error goes up, and this significantly. We have only tested the changes of one setting outgoing from the default settings.



(a) Vehicle Pose **(b)** Vehicle Class

Figure 7.12: Comparison of different augmentation levels on multi-task learning. The figures show the difference to the baseline of the multi-task learning, which can found in Table 7.7. A negative value describes a better error as the baseline.



(a) Vehicle Pose **(b)** Vehicle Class

Figure 7.13: Comparison of different dropout levels on multi-task learning. The figures show the difference to the baseline of the multi-task learning, which can found in Table 7.7. A negative value describes a better error as the baseline.

7.4.7 Ensemble Learning

We have trained many different models for the multi-task learning. These models could be used to increase the prediction accuracy, if we combine their predictions. A model provides for every sample a vector of probability. Therefore, from multiple models, we get for one sample multiple vectors, which are added up and afterwards normalized. This vector represent again a probability distribution, but an averaged over all models. The averaged vector is then used to decide the class for this sample.

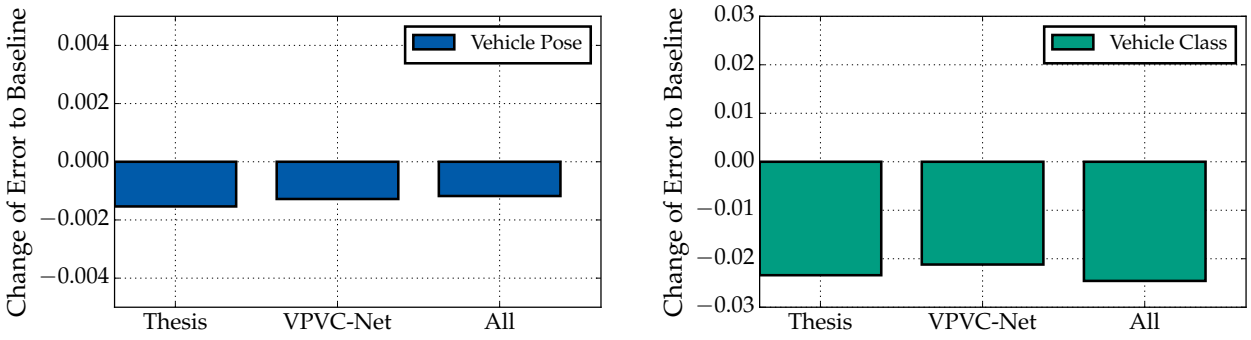
We tested this ensemble method with 3 different sets of models. The sets are following defined:

Thesis All VPVC-models which are used in this thes of the multi-task learning.

VPVC-Net We have much more models trained, as we in this thesis could present. Therefore, this set contains all VPVC-Nets of the multi-task learning.

All All models which were trained for multi-task learning. The difference to the previously set is, that in this set are added the transfer learned models.

The results for the ensemble learning are presented in Figure 7.14. All ensembles improve the error for vehicle pose and vehicle class. For vehicle pose, the *Thesis-Set* has the best error with 0.0102. The other ensembles have a similar error for the vehicle pose. For the vehicle class, the *All-Set* has the best performance with an error of 0.2487 and thus it is the best error, which we have reached on the test-set.



(a) Vehicle Pose

(b) Vehicle Class

Figure 7.14: Comparison of different ensemble sets on multi-task learning. The figures show the difference to the baseline of the multi-task learning, which can found in Table 7.7. A negative value describes a better error as the baseline.

7.5 The Expert Learning Approach

The expert learning uses the prediction of the vehicle pose, to decide, which specialized VPVC-Net should be used for the prediction of the vehicle class. We test four approaches:

Simple The simple approach trains eight unique VPVC-Nets, but instead of the complete trainings-set, the trainings-set is divided by the vehicle pose labels. So that every VPVC-Net trains the vehicle class for one pose.

Small The same like the *simple* approach, but the VPVC-Net is adapted. The number of filters is halved, as the width of the fully connected layer.

Transfer The transfer approach using a pre-trained model, to fine tune a model on the divided trainings-set. As pre-trained model is used the VPVC-Net for vehicle class from the Single-Task Learning. Also here again, it will be trained eight different models.

Transfer Extended It is like the *transfer* approach, but the training-set for each model is extended with the neighbor poses. And again here, it will be trained eight different models.

For the prediction of the vehicle pose is used the model from the Single-Task Learning for the vehicle pose. Depending on the prediction, it will be choose one of the eight models and the vehicle class will be predicted. The overall performance is in Table 7.10 depicted. The only approach, which is better as the baseline from the Single-Task Learning, is the transfer approach. The worst is the simple approach, but could be improved with the smaller VPVC-Net. The problem is the reduction of images. The training-set is divided after every pose, and this reduce the number of images for every trained model. But with adding of the neighbor images for the training to the transfer approach was not successful. The performance is gotten worse.

In Figure 7.15 is depicted the error rate for every VPVC-Net of the expert learner approaches after the vehicle pose. It is interesting, that the prediction of the vehicle class after the vehicle pose **L**eft and **R**ight are better, than the baseline of Single-Task. But on other vehicle poses is the baseline better.

To conclude the results of the expert learner approach: Only one approach is better as the single-task model for the vehicle class. But the baseline of the multi-task model is much better. Hence, the multi-task approach should be preferred.

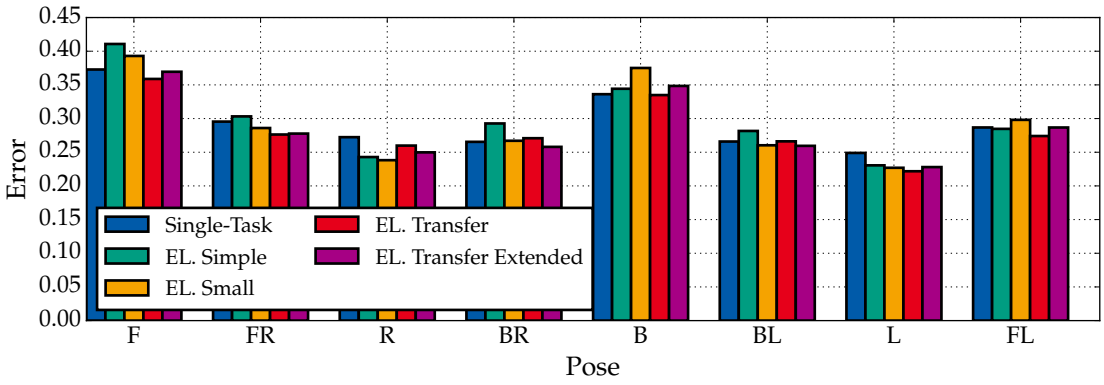


Figure 7.15: Vehicle class error of the expert learner approaches for every vehicle pose. The results of the single-task model for prediction of the vehicle class are added for comparison. Lower values are better.

	Vehicle Class		
	Error	Top-2 Error	F1score_M
Simple	0.3044	0.1240	0.7094
Small	0.2994	0.1227	0.7151
Transfer	**0.2858**	0.1113	0.7257
Transfer Extended	0.2893	0.1122	0.7221
Single-Task	0.2889	0.1162	0.7205
Multi-Task	0.2733	0.1077	0.7407

Table 7.10: The system performance of the expert learning approach. This means the performance of all eight specialized VPVC-Nets, combined in one system.

7.6 Deeper Look

We have tried to look deeper in the VPVC-Net and to understand which features were learned. Therefore, we have adapted the method *occlusion sensitivity* of Zeiler and Fergus [59], to find out, which parts of an image are sensitive for occlusion.

Over an image is a patch shifted, which occludes a part of the image. For every shift of the patch, it is computed a prediction. Therefore, for every pixel of the image exists a probability, how good the true class is predicted. These predictions could be then overlayed over the original image, and these images describes a heatmap. The heatmap shows the parts of an image, which are fragile to the occlusion. But only, if this feature was really important for this image. If on the image are multiple features, which are important, then could this method these features not detect.

Our setting was the following: We use a 13×13 large patch for the occlusion with the a gray color (RGB: [128,128,128]). Between the maximal and minimal probability of all predictions must be exist a difference of 20%, otherwise we are omitted the image.

In Figure 7.17 are depicted some interesting images, on which we can deduce some features, which are for the vehicle pose and for the vehicle class are important. The most images are depicted from the side, because these images were interpretable. The color blue indicates, that the occlusion decreases the probability of the true class. Red and orange indicates that the occlusion of this parts increases the performance of the true class.

For the prediction of the vehicle pose from the side, the side mirrors are important. This could be good seen in **b)**, **c)**, **d)**, **e)** and **g)**. Furthermore, the position of the wheel is also important. We have seen in

many heatmaps, which show the vehicle from the front-side or back-side, that the bigger wheel on the image was important (**a)** and **b)**). The bigger wheel means the wheel, which has a shorter distance to the viewpoint. For the prediction of the front (**f)**), it was for some images negative, if the logo or the radiator grille is occluded. But for the prediction of the vehicle class, it was advantageous, if the logo or the radiator grille is occluded.

For the some vehicle classes, it is important to detect the space between the vehicle and the ground. For **f)** is the a big gap important, but for the sports vehicle in **e)**, it is a low gap of importance.

Another feature for the vehicle class could be the wheelbase. On image the **a)**, **c)** and **g)** are both wheels important. But they are never to the same time occluded. So the missing of one wheel, drop the probability of these images. This could be infer to the feature, that the distance between these two wheels is important.

In addition, the occluding of some parts of the roof, decreases the probability too. In **e)** is the occlusion of the not available roof a criterion for the drop of the probability. **a)**, **b)**, **c)**, **d)** and **g)** has also some drop, if the rear shape of the vehicle is occluded. This could be infer, that the rear shape is important to differentiate the vehicle classes. For example, small vehicles have rarely a body shape of a sedan. They have mostly a hatchback.

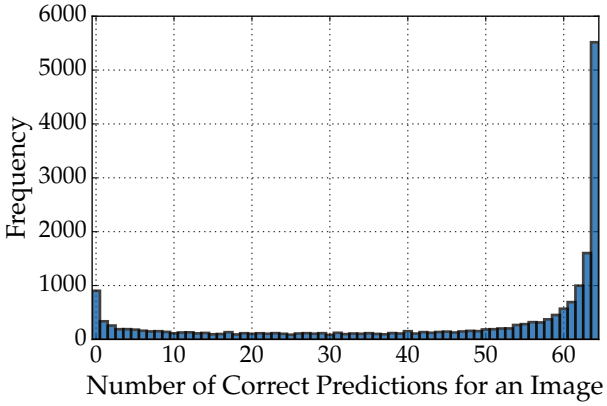


Figure 7.16: Histogram for the number of correct predictions for an image on 66 different models.

7.7 Discussion

In this chapter, we have investigated the training and predition times for the VPVC-Net and for three state-of-the-art ConvNets. We showed, that the VPVC-Net is the fastest, and in the following sections, we showed, that the prediction performance of the VPVC-Net is for small images better and on bigger images, the VPVC-Net has a similar performance like the state-of-the-art ConvNets.

But we think, that the three state-of-the-art ConvNets could also on small images produce good results. For example, we could remove on the first layer the stride of (2,2). This would cause in bigger feature maps for the deeper layers, and hence in increasing of computation time. Furthermore, we had never tested the three ConvNets with regularizations. This could also increase the performance of these networks.

Furthermore, we showed the single-task learning performance for the vehicle pose and vehicle class on the VPVC-Net. In addition, we showed a problem of the labels of the dataset, but unfortunately we have detected this problem to late. So it was not enough time to fix this problem, and to rerun the experiments. To show that this kis problem a real problem, we have evaluated for the vehicle class, how often the images were correct labeled on 66 models. The resulting distribution is depicted in Figure 7.16. There are nearly 1,000 images, where are never correct labeled on 66 different models. But on the positive side, there are over 5,500, which are on all 66 models correct predicted.

To evaluate the cooperation of the two tasks, we tested the multi-task learning and the expert learning approach. The multi-task learning has a good performance gain, compared to the single-task learning. Hence, we tested the most experiments as a multi-task model. The expert learner approach was a disappointment. Maybe it could with more data increase his performance, but with more data, the multi-task model could also increase his performance. Therefore, the multi-task learning is to prefer. A further benefit of the multi-task learning is, that both tasks could improve their performances. With the right setting, we could tune that both tasks increases their performances, for example, that both tasks get their own learning rate and own early stopping. Furthermore, the multi-task can predict two tasks on only one ConvNet with one additionally layer. For the expert learning, it must trained nine different networks, and on two networks must be predicted the prediction. This increases the prediction time.

The error of the vehicle pose is very well. In ensemle learning, we reach an error of around 1%. There is only one other work [61], which has also predicted the vehicle pose on so many images. They used for this the GoogLeNet ConvNet, which is the predecessor of the Inception-v3. They used also a multi-task learning approach to predict the vehicle pose. They reached only an error of 1.9%.

Unfortunately, the vehicle class is not comparable with other work, because we have defined our own classes. But other researchers [57, 61] have tried to predict the body class of a vehicle. The body class has twelve different classes and is also fine granular, how our categorization. This means, it is not only divided in passenger car, SUV, Van and Bus, rather in hatchback, sedan, estate and so on. They reached with this body class a best error of 30.14%. We are on the same level with our VPVC-Net, with 26.04%. Therefore, we could assume, that we are not so bad with our developed model for this task.

The most errors for the vehicle class are happened for front and back poses. This is not surprising, because there are little information, which could help to categorize the vehicle. We have also problems to classify the vehicle into the right class, if the vehicle is showed from the front or back.

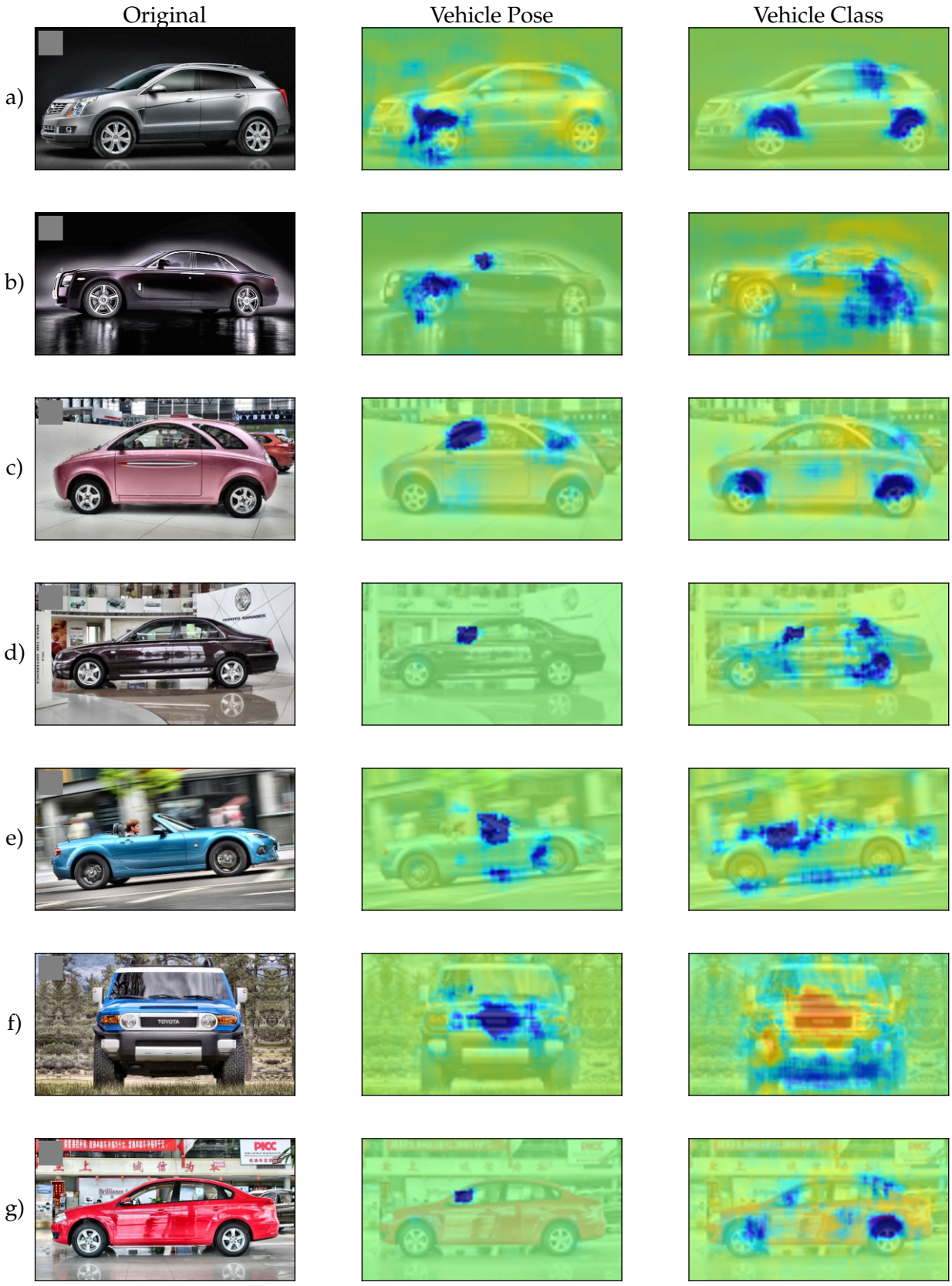


Figure 7.17: Occlusion sensitivity of selected images. The first column shows the original image with an example occlusion block in the left upper corner. The image in the middle, show the heatmap for the vehicle pose, and the image in the last column, show the heatmap for the vehicle class. The color blue shows the decrease of the probability. Red and orange shows an improvement of the probability.

8 Conclusion and Future Work

This chapter summarizes the approaches and results of this thesis and provides suggestions for further research.

8.1 Conclusion

In this thesis we used the dataset of Yang et al. [57], to classify the vehicle pose and vehicle class with *Convolution Neural Networks* (ConvNets). But before we could use this dataset, we have manually expanded and improved the vehicle pose labels, with the help of the web-application CrowdLabeling, which was developed during this thesis. In Addition, we defined a new classification for the vehicle class, based on the classification of the German traffic ministry (KBA). We could take over a bunch of labels from the CompCar dataset. But the missing labels are assigned with the help of the German Wikipedia or are labeled by us. We have seen in the evaluation, that the vehicle class has some wrong labels. Hence, the semi-automatically assignment of labels, was not perfect, and should be improved.

Furthermore, we showed our preprocessing pipeline for the preparing of an image, before it is applied on the ConvNet. In the evaluation, we have seen, that the centering of the vehicle has much influence on the performance. And that the histogram equalization helps to improve the results. The change of the colorspace, away from RGB, has not improved the performance. But it is interesting, that a grayscaled image, has almost the same performance for the vehicle class, as the baseline for multi-task learning.

We tested also the two classification tasks as a single problem, and to test the cooperation of these two tasks, we used multi-task learning and an expert learning approach. The multi-task learning, has a positive effect on the performance. We showed, that using of more than two tasks have a further positive effect for the vehicle class, but not for the vehicle pose. The expert learning approach was not successful. Only a transfer learning model from the single-task learning has provided a better error. But the performance was not so good as for the multi-task learning. In generally, the use of vehicle pose and vehicle class in multi-task learning is to prefer. It shows that these two tasks could cooperate, to improve their performances.

Furthermore, we showed a comparison with three state-of-the-art ConvNets. For a fair test, we fine tuned and tested the ConvNets on different image sizes, because the three ConvNets are designed for larger images. The three ConvNets could profit from the increasing image size, but the increasing image size was not so helpful for our ConvNet. Our ConvNet was for the single-task and for the multi-task on small resolutions the best model. But with increasing resolutions, the other three ConvNets could be increase their performance, to similar or light better performance. Interestingly, the three ConvNets have not got a performance gain from the multi-task learning, like our ConvNet.

Our developed ConvNet is for the two tasks well designed. The multi-task learning helped to improve the overall performance. But with too much multi-task, the error of the vehicle pose increases. We have explored, that the vehicle pose and the vehicle class could be used for a cooperative learning in a ConvNet with multi-task learning. Furthermore, we showed that more tasks for the multi-task learning can further improve the performance of the vehicle class.

8.2 Future Work

We have seen, that the vehicle class has some wrong labels. Hence, the vehicle class should be checked for all images, and not only for the name of the vehicle model. In this context, the label can be expanded to a multi-label classification problem. Because, some vehicles could be in multiple classes, like the “BMW M5”. It is a sport’s vehicle, but also belongs to the class “large”. Furthermore, the vehicle class could be

more fine granular, to distinguish more different types. For example could the SUV class distinguished in different sizes, like small, medium and large. Our developed web-application CrowdLabeling, could be expanded to do this job. If the CrowdLabeling is used, then could be collected further impressions of a vehicle. For example, is the vehicle fast, modified, for young or old people, and so on. This information could be used for multi-task training.

We have seen that multi-task learning, with more than two tasks, increases further the accuracy. The Wikipedia has for many vehicles the length, width, height and wheelbase. A simple mapping of the labels is not possible, because the year and version of the vehicle can have other data. For example, the "VW Polo" will be sell in Russia and India as a sedan with a 82mm longer wheelbase. Other tasks to learn are also possible, like the detection of vehicle parts.

Another approach could be the using of unsupervised learning algorithms. These could find another representation of the vehicle classes, by clustering the vehicles, which have a similar appearance. These clusters could then be used to define classes.

The pose of the vehicle could be also extended. Instead of using eight classes, it could be used a continues value to represent the angel. But the labels for the vehicle class, which are used in this thesis are not suitable for this job. But our model could be used as a transferred model to train a new task on a new dataset, like the Pascal3D [56] dataset. The dataset provides a fine granular label for the orientation of the vehicle. Our pretrained model could be a good starting point, to train the new task.

Bibliography

[1] James Bergstra and Dan Yamins and David D. Cox . Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms . In Stéfan van der Walt and Jarrod Millman and Katy Huff , editor, *Proceedings of the 12th Python in Science Conference* , pages 13 – 20, 2013.

[2] Abrar H. Abdulnabi, Gang Wang, Jiwen Lu, and Kui Jia. Multi-Task CNN Model for Attribute Prediction. *IEEE Trans. Multimedia*, 17(11):1949–1959, 2015.

[3] Jürgen Adamy. *Fuzzy Logik, Neuronale Netze und Evolutionäre Algorithmen*. Shaker Verlag, 2011. ISBN 978-3-8440-0397-0. 3., überarbeitete Ausgabe.

[4] Rich Caruana. Multitask Connectionist Learning. In *In Proceedings of the 1993 Connectionist Models Summer School*, pages 372–379, 1993.

[5] Rich Caruana. Algorithms and Applications for Multitask Learning. In *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996*, pages 87–95, 1996.

[6] Rich Caruana. A dozen tricks with multitask learning. In *Neural Networks: Tricks of the Trade - Second Edition*, Lecture Notes in Computer Science, pages 163–189. Springer, 2012.

[7] François Chollet. Keras - deep learning library for theano and tensorflow. `https://github.com/fchollet/keras`, 2015.

[8] Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

[9] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *CoRR*, abs/1511.07289, 2015. URL `http://arxiv.org/abs/1511.07289`.

[10] Ronan Collobert and Jason Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, pages 160–167, 2008.

[11] David A. Forsyth and Jean Ponce. *Computer Vision - A Modern Approach, Second Edition*. Pitman, 2012. ISBN 978-0-273-76414-4.

[12] Andreas Geiger, Christian Wojek, and Raquel Urtasun. Joint 3D Estimation of Objects and Scene Layout. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[13] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 9:249–256, 2010.

[15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *AISTATS*, volume 15 of *JMLR Proceedings*, pages 315–323. JMLR.org, 2011.

[16] Ian Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay D. Shet. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. *CoRR*, abs/1312.6082, 2013. URL http://arxiv.org/abs/1312.6082.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. Book in preparation for MIT Press, 2016. URL http://www.deeplearningbook.org.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. *CoRR*, abs/1603.05027, 2016. URL http://arxiv.org/abs/1603.05027.

[20] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In David Blei and Francis Bach, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, volume 37, pages 448–456. JMLR Workshop and Conference Proceedings, 2015.

[21] Peijin Ji, Lianwen Jin, and Xutao Li. Vision-based Vehicle Type Classification Using Partial Gabor Filter Bank. In *2007 IEEE International Conference on Automation and Logistics*, pages 1037–1040, 2007.

[22] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Lecture Notes to CS231n: Convolutional Neural Networks for Visual Recognition, 2015. URL https://cs231n.github.io. (visited: 03.11.2016).

[23] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *CoRR*, abs/1609.04836, 2016. URL http://arxiv.org/abs/1609.04836.

[24] Kraftfahrt-Bundesamt. Kraftfahrt-Bundesamt - Fachartikel: Marken und Modelle, Stand 15.05.2011, 2011. Online available at http://www.kba.de/SharedDocs/Publikationen/DE/Statistik/Fahrzeuge/FZ/Fachartikel/marken_modelle_20110515.html; last visit 02.10.2016.

[25] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D Object Representations for Fine-Grained Categorization. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, 2013.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[27] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

[28] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In *Neural Networks: Tricks of the Trade*, pages 9–50. Springer-Verlag, 1998. ISBN 3-540-65311-2.

[29] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, Lecun Y., Bengio Y., and Hinton G. Deep learning. *Nature*, 521(7553):436–444, 2015.

[30] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. *CoRR*, abs/1312.4400, 2013. URL http://arxiv.org/abs/1312.4400.

[31] Kevin Matzen and Noah Snavely. NYC3DCars: A Dataset of 3D Vehicles in Geographic Context. In *Proc. Int. Conf. on Computer Vision*, 2013.

[32] Quinn Mcnemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12:153–157, 1947.

[33] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of CNN advances on the ImageNet. *CoRR*, abs/1606.02228, 2016. URL `http://arxiv.org/abs/1606.02228`.

[34] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 9780070428072.

[35] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL `http://neuralnetworksanddeeplearning.com`. (visited: 28.10.2016).

[36] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1717–1724. IEEE Computer Society, 2014.

[37] Mustafa Ozuysal, Vincent Lepetit, and Pascal Fua. Pose Estimation for Category Specific Multiview Object Localization. In *Conference on Computer Vision and Pattern Recognition*, June 2009.

[38] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1310–1318. JMLR.org, 2013.

[39] Stephen M. Pizer, E. Philip Amburn, John D. Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B. Zimmerman, and Karel Zuiderveld. Adaptive Histogram Equalization and Its Variations. *Computer Vision, Graphics, and Image Processing*, 39(3):355–368, 1987.

[40] Qing Rao, Lars Kruger, and Klaus Dietmayer. Monocular 3D Shape Reconstruction using Deep Neural Networks. In *2016 IEEE Intelligent Vehicles Symposium, IV 2016, Gotenburg, Sweden, June 19-22, 2016*, pages 310–315, 2016.

[41] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 512–519, 2014.

[42] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3): 211–252, 2015.

[43] Stuart J. Russell and Peter Norvig. *Künstliche Intelligenz. Ein moderner Ansatz*. Pearson Studium, 2004. ISBN 9783827370891. 2., überarbeitete Ausgabe.

[44] Torben Schenkel. Erkennung von Fahrzeugtyp und Fahrzeugansicht anhand kooperativer Klassifikatoren. Bachelorthesis, TU-Darmstadt, 2016.

[45] Jürgen Schmidhuber. Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61: 85–117, 2015.

[46] Patrice Y. Simard, David Steinkraus, and John C. Platt. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *7th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3-6 August 2003, Edinburgh, Scotland, UK*, pages 958–962, 2003.

[47] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1556:1–10, 2014. URL http://arxiv.org/abs/1409.1556.

[48] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. In *ICLR (workshop track)*, 2015.

[49] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research (JMLR)*, 15:1929–1958, 2014.

[50] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David Mcallester, editor, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1139–1147. JMLR Workshop and Conference Proceedings, May 2013.

[51] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June: 1–9, 2015.

[52] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. *CoRR*, abs/1512.00567, 2015. URL http://arxiv.org/abs/1512.00567.

[53] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex A. Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *ICLR 2016 Workshop*, 2016. URL https://arxiv.org/abs/1602.07261.

[54] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL http://arxiv.org/abs/1605.02688.

[55] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann Lecun, and Christoph Bregler. Efficient object localization using Convolutional Networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 648–656, June 2015.

[56] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond PASCAL: A Benchmark for 3D Object Detection in the Wild. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 75–82, 2014.

[57] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. A Large-Scale Car Dataset for Fine-Grained Categorization and Verification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3973–3981, 2015.

[58] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems 27 (Proceedings of NIPS)*, 27: 1–9, 2014.

[59] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, pages 818–833, 2014.

[60] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial Landmark Detection by Deep Multi-task Learning. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI*, pages 94–108, 2014.

[61] Yi Zhou, Li Liu, Ling Shao, and Matt Mellor. DAVE: A Unified Framework for Fast Vehicle Detection and Annotation. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II*, pages 278–293, 2016.