Hierarchical Multi-label Object Detection of Rare Classes for Autonomous Driving

Bachelor Thesis

for the

Bachelor of Engineering

from the Course of Studies IT-Automotive

at the Cooperative State University Baden-Württemberg Stuttgart

by

Phillip Lippe

20. August 2018

Time of Project	11. June 2018 to 20. August 2018
Student ID, Course	9045534, IT-Automotive 2015
Training Company	Daimler AG, Stuttgart
Supervisor of Training Company	Jonas Uhrig
Reviewer of Cooperative University	Andreas Baisch

Author's declaration

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema: *Hierarchical Multi-label Object Detection of Rare Classes for Autonomous Driving* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, 20. August 2018

Phillip Lippe

Abstract

An autonomously driving vehicle has to understand complex urban street scenes to navigate through traffic safely. Modern techniques of Deep Learning enable to accurately detect objects like other vehicles or pedestrians in an image. Usually, to select a category for an object, an artificial neural network predicts a probability distribution for a predefined class set choosing the category with the highest value. However, within the class set, relationships and similarities are neglected although the objects can be ordered in a hierarchical manner offering several benefits. Thus, the following thesis deals with introducing a hierarchical class structure for analyzing traffic scenarios with a particular interest in optimizing the accuracy of rare classes.

The hierarchy is implemented as a post-processing step so that the network still predicts a single class vector. During inference, the correct class is determined by selecting the node with the highest score on the first level of the hierarchy, and recursively continuing the search with its child classes. The prediction can stop at an inner node if the score is lower than a predefined threshold. Object attributes are added to the class structure as an independent class set which is evaluated after the class search has finished. Besides other techniques, the network is optimized regarding rare classes by using a novel metric loss function based on a hierarchical adaption of the Intersection over Union (IoU). The approaches are tested on the task of semantic segmentation. In experiments, the hierarchical classifiers significantly outperform networks trained on a flat class set by up to 3.5% of the mean IoU on the Cityscapes validation dataset. Even a hierarchy of about 70 classes achieved a similar accuracy than the flat version with less than half the number of categories while generating a much more detailed classification of the traffic scene.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Proposed Approach . . . 3
- 1.3 Outline . . . 4

2 Foundations **5**
- 2.1 Machine Learning . . . 5
- 2.2 Artificial Neural Networks . . . 10
- 2.3 Convolutional Neural Networks . . . 14
- 2.4 Image understanding with CNNs . . . 21
- 2.5 Related work . . . 29

3 Hierarchical Object Detection **34**
- 3.1 Hierarchical classification . . . 34
- 3.2 Optimization on rare classes . . . 43
- 3.3 Hierarchical metric . . . 49
- 3.4 Metric Loss Function . . . 59

4 Experiments **69**
- 4.1 Experimental setup . . . 69
- 4.2 Hierarchical classifier for Semantic Segmentation . . . 70
- 4.3 Testing scalability of hierarchical classification . . . 76
- 4.4 Threshold adaptation . . . 80

5 Outlook **86**

6 Conclusion **90**

A Dataset distribution **100**
- A.1 Training Datasets . . . 100
- A.2 Validation Datasets . . . 108

B Label Hierarchies **113**
- B.1 Small-scale hierarchy . . . 113
- B.2 Standard hierarchy . . . 114
- B.3 Large-scale hierarchy . . . 115

List of Figures

1 Typical urban street scene [12] . . . 1
2 Hierarchical classification . . . 3
3 Relations of T, P and E . . . 5
4 Supervised learning examples [75, 12] . . . 8
5 Reinforcement learning [88] . . . 9
6 Artificial neuron architecture [15] . . . 11
7 Comparison of activation functions . . . 11
8 Network structures [15] . . . 13
9 Unrolled RNN [66] . . . 14
10 Convolution operation [41] . . . 15
11 Pooling layer [3] . . . 16
12 AlexNet [46] . . . 18
13 Inception modules [89] . . . 19
14 Residual block [32] . . . 20
15 ResNet Explanation [95] . . . 21
16 Example of semantic segmentation [12] . . . 22
17 Semantic segmentation with fully convolution neural networks [65, 12] . . . 23
18 R-CNN architecture [24] . . . 24
19 YOLO model [70] . . . 25
20 Single Shot Multibox Detector [56] . . . 26
21 KITTI [22] . . . 27
22 Cityscapes [12] . . . 28
23 Mapillary [62] . . . 29
24 Label hierarchy based on WordNet [68] . . . 30
25 Hierarchical classification on WordNet [68] . . . 30
26 Label hierarchy of Cityscapes, Mapillary Vistas and GTSDB [58] . . . 32
27 Multi-stage semantic hierarchy [58] . . . 33
28 Hierarchy compared to flat classification . . . 35
29 Comparison of tree and DAG taxonomies . . . 36
30 Variations of hierarchical classification [78] . . . 38
31 Handling object attributes in hierarchies . . . 41
32 Class distribution of Cityscapes by pixels [12] . . . 44
33 Loss function for different focusing parameters [55] . . . 47
34 Weighting classes by moving average filters . . . 49
35 Label-dependent moving average filters . . . 50
36 Example hierarchy . . . 55
37 Depth-dependent distance metric . . . 56
38 Comparison of distance-based approaches . . . 74
39 Prediction for object attributes . . . 75
40 Remaining problems for object attributes . . . 76
41 Predictions of different hierarchies . . . 77
42 Prediction distribution of two examples . . . 80

43 Threshold adaption 82
44 Class correlations 83
45 Correlation between classifiers 85
46 Novel objects 89
47 Small-scale hierarchy 113
48 Standard hierarchy 114
49 Large-scale hierarchy 115
50 Larger figures of large-scale hierarchy 116
51 Larger figures of large-scale hierarchy (infrastructure) 117

List of Equations

2.1 Cross Entropy [27] . . . 7
2.2 Cross Entropy for classification [27] . . . 7
2.3 Binary Cross Entropy (BCE) [27] . . . 7
2.4 Joint distribution in unsupervised learning [27] . . . 9
2.5 Converting supervised, conditional distributions to joint distributions [27] . . . 9
2.6 Mathematical representation of an artificial neuron [15] . . . 10
2.7 Softmax [27] . . . 16
2.8 Log-likelihood softmax [27] . . . 17
2.9 Hierarchical, conditional class probabilities [68] . . . 31
3.1 Focal loss for binary cross entropy [55] . . . 47
3.2 Precision measure [14, 82] . . . 52
3.3 Recall measure [14, 82] . . . 52
3.4 F-score [14, 82] . . . 53
3.5 Accuracy rate [14, 82] . . . 53
3.6 Intersection over Union (IoU) score [67] . . . 53
3.7 Hierarchical precision measure [14] . . . 54
3.8 Hierarchical recall measure [14] . . . 54
3.9 Distance contribution [14, 87] . . . 55
3.10 Precision and recall measure with distance contribution [14, 87] . . . 56
3.11 Adapted distance contribution . . . 58
3.12 Hierarchical confusion matrix's parameters . . . 58
3.13 Hierarchical measures . . . 58
3.14 Approximated confusion matrix parameters . . . 60
3.15 Approximated IoU score . . . 60
3.16 Gradients of approximated IoU score . . . 61
3.17 Class imbalance invariance for subclasses . . . 62
3.18 Conditional IoU loss along the path from the root node to a class . . . 63
3.19 Weighting IoU loss by leafs . . . 63
3.20 Propagating false positives throughout the hierarchy . . . 63
3.21 Example for propagating false positives . . . 64
3.22 Definition for binary MAF weights . . . 65
3.23 MAF weights for IoU loss without subclasses . . . 65
3.24 MAF weights for IoU loss with descendants . . . 65
3.25 Adjusted MAF weights for IoU loss without descendants . . . 66
3.26 Label-dependent weight factors for false positives . . . 66
3.27 Minimum number of labels for true positives . . . 66
3.28 Label number adaptation for false positives . . . 67
3.29 Distance weighting for false positives . . . 67
3.30 Distance weighting for true positives . . . 68

List of abbreviations

BCE	Binary Cross Entropy
CNN	Convolutional Neural Network
DAG	Directed Acyclic Graph
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit [10]
IoU	Intersection over Union
LSTM	Long Short-Term Memory [35]
MSCOCO	Microsoft Common Objects in Context [54]
ReLU	Rectified Linear Unit [61]
RGB	Red-Green-Blue (color model)
RNN	Recurrent Neural Network

List of Tables

2 Confusion matrix [14] 51
3 Example of confusion matrix for multi-class classification 52
4 Performances of various experiment settings 72
5 Overview of the cityscapes training dataset 102
6 Overview of the cityscapes special training dataset 104
7 Overview of the mapillary training dataset 107
8 Overview of the cityscapes validation dataset 109
9 Overview of the mapillary large-scale validation dataset 112

1. Introduction

1.1. Motivation

The vision of autonomous driving is becoming reality step by step. Since 1987 a research project called PROMETHEUS [98] has successfully led to the first autonomous vehicle, many companies started to further develop this technology and make it ready for series production. For example, Mercedes Benz presented the prototype *S500 Intelligent Drive* in 2013 [101]. This system was able to drive the approximately 100km route between Mannheim and Pforzheim autonomously equipped by only sensors out of series production.

However, autonomous driving remains complex and challenging because there are so many possible scenarios that a car has to solve. Such a system has to tackle three main tasks: detection of its surrounding, understanding of the environment and selecting its action and trajectory. Multiple different sensors like camera, radar and lidar observe all objects around the ego vehicle. Because the outputs of these sensors only consist of pixels or point clouds, an interpretation system processes the raw data to extract objects like pedestrians and vehicles, and other relevant information. As the last step, the car has to plan a safe and comfortable trajectory where it will drive. A typical urban street scene that illustrates the challenges for autonomous driving is shown in Figure 1.

Figure 1: A typical urban street scene contains several objects like pedestrians, vehicles and traffic signs. Also, different kind of surfaces need to be distinguished determining the space where the car can drive. The detection of objects is complicated by shadows, occlusion and more visual effects [12].

Considering the human as a driver in a car, it is noticeable that he only needs a single *sensor* for these steps: the eye. Without any distance sensors or radars, a person manages to assess and react to the situation. Besides, most signals in traffic are based on visuals, such as traffic lights and signs. Consequently, the camera is one of the most critical sensors to realize autonomous driving.

Although the interpretation of an image is not an issue for a human, it remains to be a tough challenge for a computer [39]. Images only represent a two-dimensional reflection of a three-dimensional environment. Thus, to understand and analyze the illustrated scene, the interpretation system needs to reconstruct the three-dimensional context. A human daily interacts with such an environment whereas a computer only processes pixels in two dimensions. Furthermore, objects in a scene need to be detected regardless of its perspective, size, brightness, color, background and several more properties. Besides, occlusion and blurriness complicates the task.

In the last decade, the field of computer vision experienced a significant change due to modern techniques like machine learning. Since 2012, when the artificial neural network called AlexNet [46] outperformed humans in the ImageNet challenge [75] where images are classified in one of 1,000 possible categories, machine learning became a popular research topic and application for intelligent systems. Such algorithms *learn* from huge datasets and optimize their internal parameters to approximate the correct output. Today, deep neural networks are applied for the task of image understanding in the context of autonomous driving and are able to recognize various object types accurately [12, 22, 39, 62].

A conventional method to express the analyzed information of an RGB image is semantic segmentation where every pixel is assigned to a class out of a predefined set. In case of the example image in Figure 1, a neural network would classify the pixels into pedestrians, passenger-cars, traffic lights and many more. The classification is performed by determining a probability distribution over the possible classes and selecting the class with the highest value. However, not every pixel can always be certainly assigned to one of the set. For example, in Figure 1, even for a human, the vehicles and infrastructures at the end of the road are hard to categorize due to the blurry pixels. Also, no class is specified for the bulky waste on the left sidewalk next to the pharmacy.

Still, a typical neural network would select a class of the predefined set for both kind of pixels which would lead to many false predictions. If a human would have to assign a class to every pixel, he probably chooses a different method similar to the structure of the natural language: a hierarchical classification. Multiple classes can be grouped to a more general category, like bicycles and motorcycles to ridable vehicles, until only the root node is left. A prediction consists of a path from the root node to the final class. An example is visualized in Figure 2. In this hierarchical structure, the bulky waste can be certainly categorized as objects that the car should not run over. Furthermore, the blurry objects in the background can be assigned to general classes like vehicle and infrastructure instead of unnecessarily selecting a more specific class that is probably wrong.

A hierarchical classification has even more benefits. Datasets like Cityscapes [12] contain labels for about 25 different classes including eight categories of vehicles. However,

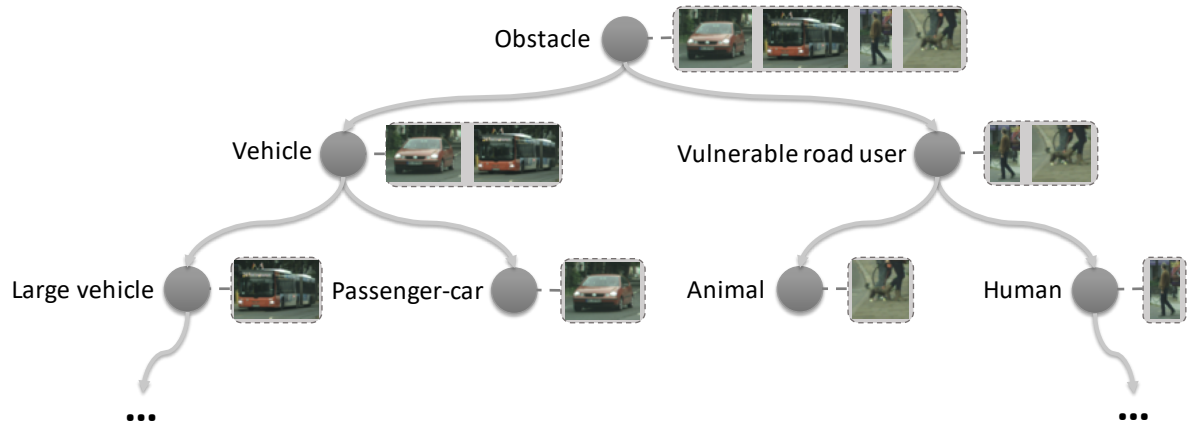


Figure 2: Objects are classified in a hierarchical structure so that the root node, for example *obstacle*, is assigned to almost all objects. Every level of the hierarchy splits up the set of objects, until the leaf nodes contain the most specific classes. In the Figure, example objects extracted from Cityscapes [12] are placed on the right to every node.

to obey all traffic rules, a much more distinct disposition is needed as an autonomously driving vehicle has to react to police cars and fire trucks. Within the dataset, these duties are ignored so that a police car is labeled as a general passenger car. Even if the dataset would distinguish between both vehicles, the police cars occur less frequently and are therefore harder to detect for a network, although it shares some significant features with other cars. A hierarchical class structure can express such relationships so that the duties can be learned in parallel to the vehicle category. Also, if multiple, heterogeneous datasets are used that contain different detailed labels, a hierarchical classification can be simultaneously trained on all sets whereas the most general classes could only be used for a standard classifier.

As these benefits are crucial for understanding urban street scenes, this thesis deals with the implementation of a hierarchical class structure for the task of semantic segmentation with deep neural networks.

1.2. Proposed Approach

To adapt the concept of hierarchical classification for computer vision, several tasks need to be tackled This thesis concentrates on the following three:

Hierarchical class structure Hierarchical classifiers are already applied across different application domains including text classification [17, 87] and gene categorization [8, 19]. The architectures of the classifiers alternate in some significant properties. For example, the local classifier architecture applied an own classifier for each node independently, or a global classifier determines the probabilities for every

class in a single evaluation. However, most approaches share the same procedure to determine the predicted class during inference. Starting at the first level of the hierarchy, the probabilities of each node are compared. The node with the highest probability is selected, and the search is recursively continued until either the prediction ends in a leaf, or the probability of the selected class is lower than a specified threshold. The second criterium enables predictions of an inner node if no descendant can be certainly chosen.
To select the best fitting approach for the task of semantic segmentation, this thesis discusses different architectures of hierarchical classifiers and sets those into context with deep neural networks.

Optimizing on rare classes A common challenge for machine learning algorithms is to handle imbalanced data [30, 31, 45]. A hierarchical class structure can improve the accuracy on such classes by sharing features between nodes having the same ancestor. Furthermore, the loss function regarding which the network tries to optimize its internal parameters, can be shifted. Moving average filters are applied to measure the detection performance of the network during training and weighing the loss to focus on poorly classified categories. Another aspect that is discussed in this thesis is a metric loss function that is invariant to the number of pixels per class. The function is further adapted to the hierarchical structure taking the class relationships into account.

Hierarchical metric Usually, deep neural networks for semantic segmentation are evaluated by metrics comparing the prediction to human-labeled images. However, standard measurements do not take the class relationships into account so that a misclassification of a bus as a truck is considered similarly as if the network would have predicted the class road. Also, inner nodes that constitute a more general classification cannot be evaluated with flat metrics. To fairly compare the performances of different hierarchical classifiers, a new metric needs to be developed that reflects the semantic distance of the prediction and correct class in the hierarchy.

1.3. Outline

Besides the introduction, the following thesis is divided into five parts. Section 2 introduces the area of machine learning and artificial neural networks. Furthermore, it summarizes related works that are taken into account. The methods developed for the application of hierarchical classification for semantic segmentation are described in Section 3. Experiments that are implemented for testing the proposed approaches and their results are discussed in Section 4. At the end of this thesis, an outlook of further research and application is discussed followed by an overall conclusion.

2. Foundations

This section gives a brief introduction to the field of machine learning and its application in computer vision. The basics of general machine learning are explained in the first subsection, while the following subsection reviews artificial neural networks as a part of learning algorithms. Subsequent sections give a deeper insight into Convolutional Neural Networks and their application in the field of computer vision. As the last part of this section, related works are presented and summarized.

2.1. Machine Learning

The field of machine learning summarizes algorithms that become more accurate in predicting outcomes without being explicitly programmed [48]. The basic concept is using statistical analysis of received input data and its corresponding correct output. But what *learning* actually means in the context of algorithms is succinctly defined by Mitchell [59]:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measures P, if its performance at tasks in T, as measured by P, improves with experience E." [59, p. 2]

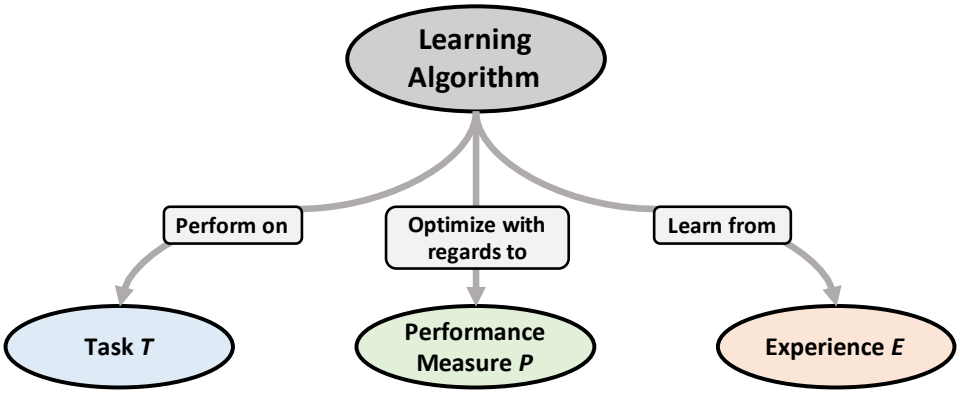


Figure 3: Overview of the relations between the learning algorithm and its task T, performance measure P and experience E. While T specifies the process, E defines how the datasets look like and P the way algorithms learn from the data.

The interaction between task, performance and experience is shown in Figure 3 and described in the following sections. The layout is based on [27, 59] to give a brief introduction to the huge area of machine learning.

2.1.1. Task *T*

A learning algorithm is designed to solve a specific task. This task defines how input should be processed and what the result of this algorithm looks like. As an example, if a robot should be able to walk, the task it tries to solve is walking. The input could be

an RGB image of its surrounding while the output is the leg's movement.

When the algorithm is learning on a task, the input it gets is called an *example*. An example $\boldsymbol{x}$ is a collection of features that have been measured as a sample input of the algorithm's system. Mathematically, the example is represented as a vector $\boldsymbol{x} \in \mathbb{R}^n$ which elements are features. For an image, this means that the pixel values are the features of the example.

After defining what a task is and how an input looks like they could be distinguished in different kinds. Due to the huge variety of possible tasks the following ones represent a small subset of the most common machine learning tasks. For a more detailed list see [27].

Classification One of the basic tasks of learning algorithms is specifying the input's category among multiple possibilities. This means that the algorithm represents a function $f : \mathbb{R}^n \to \{1, ..., k\}$ that classifies the input example with its features to one of k categories. The output for choosing a category is usually defined by a numerical value $y = f(\boldsymbol{x})$ or a one-hot vector $y \in [0, 1]^k$ for which f calculates a probability distribution over classes. Typical use cases of this kind of tasks are object and handwriting recognition on images [50, 75].

Regression Some tasks require continuous numerical values instead of discrete classes as output. In comparison to classification, the represented function of the algorithm is defined by $f : \mathbb{R}^n \to \mathbb{R}$. Regression is used for various motor control tasks [64] and object localization in images [24, 56, 70].

Synthesis and sampling A different kind of task is when the algorithm is asked to generate new data examples that are similar to those in the training dataset. The input of this task is flexible and depends on the specific application. For predicting the next frame of a video sequence like proposed in [20, 97], the algorithm's function is represented by $f : \mathbb{R}^{n \times t} \to \mathbb{R}$ with t as number of input frames. Recent approaches for this task often include Generative Adversarial Networks [28] that reach performance to fool a human distinguishing between realistic and synthesized data [29].

2.1.2. Performance Measure *P*

In order an algorithm can learn, it has to get a feedback of how good its predicted output was. So a quantitative measurement of its performance must be designed, usually specific to the belonging task T. Besides, different algorithms can easily be compared using the same performance measurement. In the learning process, the goal of the algorithm is to optimize its parameter towards performance gain.

A common way to measure the performance on tasks like classification is to define the accuracy of a model. Accuracy is the proportion of examples for which the model

predicts the correct output. The opposite of it would be the error rate, the proportion of examples for which the model predicts an incorrect output. During training, the performance is rated more precisely to optimize the internal parameters, so that the output is close to the labels. The difference between prediction and the desired output is measured by a loss function, or also called objective function, which the learning algorithm tries to minimize. For classification, a popular loss function is cross entropy [27]. It quantifies the difference between two probability distributions p and q by adding the Kullback-Leibniz divergence $D_{KL}(p||q)$ to the entropy of p. Over a set of values $y \in \Omega$, the cross entropy is defined as:

$$H(p,q) = H(p) + D_{KL}(p||q) = H(p) + \int_{y\in\Omega} p(y) \cdot \log \frac{p(y)}{q(y)} \tag{2.1}$$

However, if both distributions are discrete and only defined for certain values, the formula can be simplified by replacing the integral with a sum and eliminating the entropy of p. In the case of classification, p is the ground truth distribution and q the prediction while both depend on an input vector x. The value y is one of the output categories and therefore discrete. The average cross entropy over a set of N examples and M classes can be calculated as:

$$H(p,q) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{M} p(y_k|x_i)\log q(y_k|x_i) \tag{2.2}$$

The performance of a learning algorithm would be ideal if $p = q$ where the Kullback-Leibniz divergence is minimal at a value of 0. As the entropy $H(p)$ cannot be affected by q, the cross entropy also has a minimum of $p = q$. This is why learning algorithms are often optimized to minimize the cross entropy of its prediction and the ground truth.

If only labels of 1/100% or 0/0% are used, the ground truth distribution is limited to $p(0|x_i) = z_i$ and $p(1|x_i) = 1 - z_i$ for the label z, and the prediction $\hat{z}$ equally to $q(0|x_i) = \hat{z}_i$, and $q(1|x_i) = 1 - \hat{z}_i$. Therefore, the loss function can be simplified as follows [27]:

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}(z_i \log \hat{z}_i + (1 - z_i)\log(1 - \hat{z}_i)) \tag{2.3}$$

This simplification of cross entropy is called Binary Cross Entropy (BCE), as it uses binary labels. However, the loss function often depends on the application of the system and has to be adjusted for every task.

2.1.3. Experience *E*

As the last part of the machine learning basics, the source from which the algorithm learns should be explained. The basic component of experience is a dataset consisting of a collection of examples. Mostly it is split up into a training and testing part. The

training dataset is used for the algorithm to learn from, while on the testing dataset the performance is measured regarding new images which the algorithm has not seen for training. Evaluating on unknown examples checks whether the system has learned general concepts of the dataset and can transfer them to the test dataset (called *generalization*), or learned details and noise in the training data that solely fit on this specific dataset (called *overfitting*).

Three different common kinds of datasets could be distinguished depending on the way the algorithm gains experience of the data:

Supervised learning The easiest way for an algorithm to learn how the estimated function looks like is having for every example the correct output, called label or ground truth, and optimize it towards that. The structure of the ground truth depends on the task which should be learned. Figure 4b shows an example for classification, for which the label is only one value/class, and as a comparison, one for semantic segmentation, where the algorithm tries to classify every pixel by its own. Well known datasets for supervised learning are ImageNet [75] for image classification and Cityscapes [12] for semantic understanding of urban street scenes. More datasets will be discussed in detail in section 2.4.3.

(a) Classification label (b) Semantic label

Figure 4: (a) The input is an RGB image like the one shown on the left. The label belonging to it is given by the class "Firetruck" [75]. (b) For semantic segmentation, the label is not only one class. It is an image of the same resolution as the input with a class label for every pixel. For visualization every class label is represented by a different color [12].

Looking from a statistical perspective, supervised learning has a random vector $\boldsymbol{x}$ as input and learns to predict the correct label $\boldsymbol{y}$ by estimating $p(\boldsymbol{y}|\boldsymbol{x})$. The ground truth has to be generated by an instructor that is mostly human. While creating classification labels is quick, producing a semantic label for a high-resolution image takes a long time. This is why datasets like Cityscapes [12] only have about 5,000 images whereas ImageNet [75] contains up to 14,000,000. Another application field for supervised learning is regression tasks like bounding box estimation on images (see section 2.4.2) [18, 24, 54, 56, 70].

Unsupervised learning In unsupervised learning, the ground truth is missing. The algorithm has to learn useful properties of the dataset structure without explicitly showing it. Usually, for deep learning, the aim is to capture the entire probability

distribution either explicitly, as in density estimation, or implicitly, for generating new images at the task of synthesis and sampling. Another possible application after capturing the distribution is clustering the dataset in different categories by self-explored structures. So it learns a joint distribution of all features in the input vector $\boldsymbol{x} \in \mathbb{R}^n$ [27]:

$$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1}) \tag{2.4}$$

Although the distribution estimation and the way of learning looks different to supervised learning, both could not be clearly separated. Given the ground truth, every unsupervised task could be converted to supervised, and on the other hand the conditional probability of the supervised way can be solved by learning the joint distribution of $p(\boldsymbol{x}, y)$ by unsupervised methods [27]:

$$p(y|\boldsymbol{x}) = \frac{p(\boldsymbol{x}, y)}{\sum_{y'} p(\boldsymbol{x}, y')} \tag{2.5}$$

A learning paradigm taking some of supervised and unsupervised is the semi-supervised learning where for example only some examples have labels but not all of them. Unsupervised learning mostly takes longer until the algorithm has fully learned the distribution because it has to explore the structure by itself. However, as a huge advantage no instructor is needed, and so much more data is easily available. Applications of unsupervised learning include speech recognition/generation [76, 96] and video prediction [20, 97].

Reinforcement learning A common human way to learn new things is by "trial and error" [36, 88]. Simulating this behavior, reinforcement learning is a paradigm in which the algorithm interacts with its environment and learns from feedback or reward that is returned. Therefore, the system selects actions in an environment to maximize the expected reward. Figure 5 illustrates this framework.

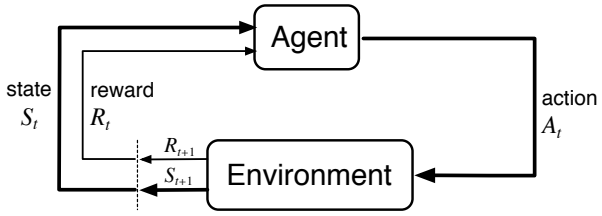


Figure 5: The learning algorithm is represented by an agent that takes an action A_t and receives a reward R_{t+1} from the environment. Its decision is based on the current state S_t and reward R_t. In games the state usually is an RGB image of the game the users sees and reward is given by the score [88].

An intuitive application of such learning problems is playing games [40, 60]. In classical computer games like Atari [60], there are a limited number of possible actions the player can take. The algorithm has to learn the expected reward of an action based on the input that is mostly the pixel values of the game. Approaches like AlphaGo [79] have successfully shown the possibilities of reinforcement learning beating humans performance.

While games always have a certain reward signal (the simplest one is winning or losing), some applications are missing that out. As an alternative, inverse reinforcement learning (IRL) is used where an expert demonstrates the task that the algorithm should learn [2]. Reinforcement learning applies for developing intelligent game agents [60, 79], motion planning [81] and much more.

2.2. Artificial Neural Networks

Artificial neural networks are graph-based models that are inspired by the human cognition [15]. A single neuron represents the smallest unit or node of such a network interlinked by many interconnections with other neurons using its activation to communicate. This behavior is simulated by simplified mathematical models for which the neurons are represented as graph nodes and the connections are weighted directed edges. To understand the mechanism of artificial neural networks, the following section is divided into three parts. The first paragraph deepens the mathematical model of a neuron while the second handles the basic architecture of a network and the third introduces recurrent connections and their application.

2.2.1. Artificial neurons

Figure 6 shows the mathematical model of a single neuron. It has multiple external inputs $\{x_1, x_2, ..., x_n\}$ which are usually the output of other connected neurons. All of them are weighted by a corresponding parameter of $\{w_1, w_2, ..., w_n\}$ defining the relevance of an input for this neuron. Finally, the weighted inputs are summed up to u for which an additional parameter b is taken into account. This variable is called "bias" and shifts the evaluated sum with a constant value. The neuron's output y is calculated by taking u as input for an activation function $g(u)$ that defines the output's mapping to the input. During the learning process, the neuron adjusts its parameter including input weights and bias towards the expected output.

Summarizing an artificial neuron is represented by the following equation [15]:

$$y = g(b + \sum_i x_i \cdot w_i) \tag{2.6}$$

The activation function's goal is reducing the output range of values because the weighted sum u could theoretically be anything between $-\inf$ and $+\inf$. If many neu-

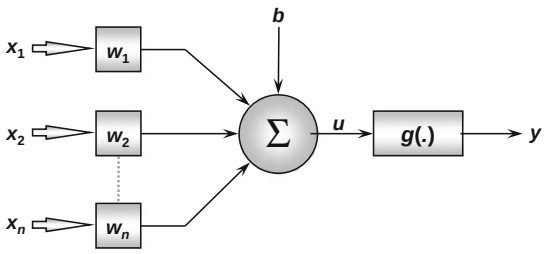
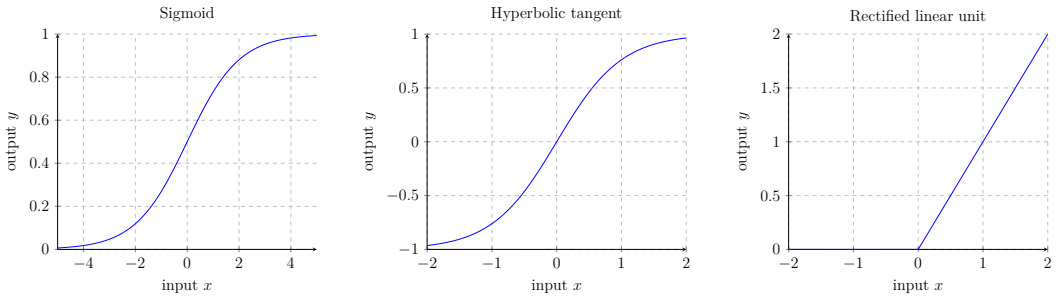


Figure 6: The mathematical model of a neuron consists of a weighted sum of its inputs that is shifted by a constant bias. The sum is given as input to an activation function to get the final output [15].

rons are cascaded, the output value could quickly blow up. Furthermore, from a biological point of view, the output of a neuron is the activation that is also fixed to specific borders. Without an activation function, the weights and bias would simply do a linear transformation of the input limiting the capacity to solve complex problems. So, it is also used to introduce non-linearity to the artificial neuron. There are several activation functions which can be used, but only three should be described further here: sigmoid, tanh and ReLU.

The sigmoid function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$ and plotted in Figure 7a. It is fully differentiable and nonlinear while having a range of values between 0 and 1. As the gradients of the sigmoid function have a maximum for the output value of 0.5, it tends to push all values to either 0 or 1. So it is a common activation function for the classification task.



(a) $\sigma(x)$ (b) $\tanh(x)$ (c) $\mathrm{ReLU}(x)$

Figure 7: (a) The sigmoid function maps the inputs to a range of 0 to 1 while having high gradients near to $y = 0$ to bring the output more to either 0 or 1. (b) The hyperbolic tangent is similar to the sigmoid function but has a output range of -1 to 1. (c) A rectified linear unit (ReLU) is 0 for all input lower than 0. All other values are processed linearly so that they do not change.

Figure 7b shows the hyperbolic tangent that is defined as $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. It can be written by using the sigmoid function as $\tanh(x) = 2 \cdot \sigma(2x) - 1$ and shares most of the properties with the sigmoid function. As the output range of values is between -1

and 1 it is often used as output activation function for tasks like image generation.

Rectified linear unit (ReLU) is another more simpler activation function that is plotted in Figure 7c. It is defined by the equation $\text{ReLU}(x) = \max(0, x)$ and so 0 for every input lower than 0. The output range is $[0, \text{inf})$ what can still blow up. However, one big advantage of this function is that it is less computationally expensive and the activations are sparse what makes the network more efficient and faster to learn. ReLU is everywhere differentiable except at $x = 0$ but has no gradients for $x < 0$.

Overall there is not the one activation function that can be used for every network. Some activation functions have been shown to be better suited for specific tasks as sigmoid is often used for binary classification output [70], tanh for synthesizing [20] and ReLU for hidden layers inside the network [25, 46].

2.2.2. Deep Feedforward Networks

After analyzing the concept of a single neuron this paragraph deals with the interconnection and combination of multiple neurons to a network. The quintessential artificial neural architectures are deep feedforward networks, or multilayer perceptrons [27]. These models are called feedforward because information flows through the whole network without any connections in which the outputs of the model are fed back into itself. Networks with such connections are called Recurrent Neural Networks (RNN) and are described in Section 2.2.3.

In general, an artificial neural network consists of multiple layers that can be divided into three main parts:

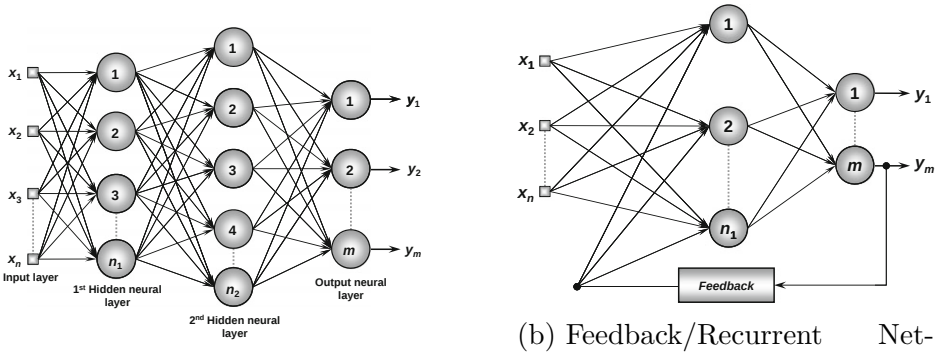
Input layer The first layer of a network receives information (data) from the external environment. Frequently, these inputs are normalized between -1 and 1 for better numerical precision for the mathematical operations and to keep the network's weight within a certain range [15, 27].

Hidden layers The main computing layers of a neural network are called *hidden layers* because they are invisible for the external environment. The structure consists of multiple, stacked neurons that are responsible for extracting patterns associated with the input. Neurons are only connected between consecutive layers but not within a single layer.

Output layer The last layer of a network is composed of neurons that are responsible for predicting the final output based on the previous hidden layers. Similar to the input of the network, the output is usually normalized as well.

The structure of a simple feedforward network with two hidden layers is shown in Figure 8a. The input is processed straight forward through the network, and there is no recurrent connection. In comparison, the model in Figure 8b has a feedback from the

last output layer back to the network. With this, the next calculated output depends on the result of the previous step and can learn from a time component (for details see Section 2.2.3).



(a) Feedforward Network

(b) Feedback/Recurrent Network

Figure 8: (a) This figure illustrates fully connected feedforward network with two hidden layers. All neurons of one layer are connected to every neuron of the next layer with no feedback connection [15]. (b) In comparison to feedforward network a RNN has a feedback connection from one later neuron back to one in a earlier layer [15].

In the example of Figure 8a, all input nodes are fully connected with all neurons of the first hidden layer. Every neuron of this layer has, therefore, n inputs and n corresponding weights next to one bias that has to be adjusted by learning. With n_1 neurons, the first hidden layer contains $(n+1) \cdot n_1$ learnable parameters. Layer two has therefore $(n_1+1) \cdot n_2$ and the output layer $(n_2+1) \cdot m$ parameters. With an increasing number of neurons, the amount of parameters blows up which becomes computational expensive and hard to learn. Sparser layers are for example convolutions that operate on a grid of features and are therefore mostly applied on images. As this thesis deals with the task of image understanding, the convolutional neural networks are described in detail in Section 2.3.

2.2.3. Recurrent Neural Networks

Recurrent Neural Networks (RNN) [74] are focused on processing a sequence of values $x_0, ..., x_n$. Of course, such a sequence could also just be stacked together to one big input to a multilayer network, but this would not be parameter efficient. For example, consider a model for language understanding that should extract a year out of a sentence. Two samples are "I went to Nepal in 2009" and "In 2009, I went to Nepal" [27]. The corresponding word appears in the first sentence at sixth and in the other at the second position. Still, the parameters could be shared between these two-time steps.

Recurrent Neural Networks have, as in Section 2.2.2 described, feedback connections from a neuron's output back into the network. The values of these connections are held

over time steps, so that processing input x_t depends on computation results of x_{t-1}.

A better understanding of a recurrent network is unfolding it over multiple time steps, as illustrated in Figure 9. Network A gets x_t as input to generate its output h_t. In addition, there is a connection over time steps, so that x_{t-1} influences the prediction. Therefore, x_{t-1} is influenced by x_{t-2} and so on, h_t depends on the whole previous sequence $x_0, ..., x_t$.

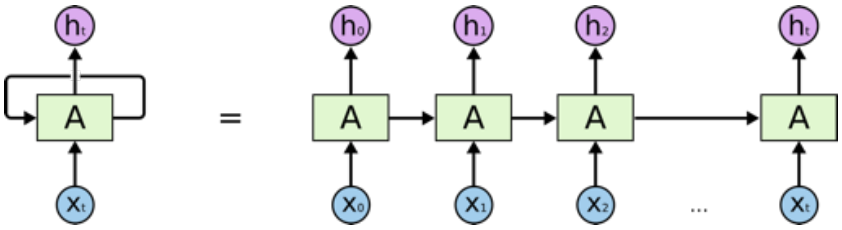


Figure 9: A Recurrent Neural Network can be seen as a chain of multiple time steps with feedback connections between them. Due to it is the same network, the same weights are used for processing the input. Moreover, the recurrent output of the previous step influences the final prediction [66].

The design and position of the feedback connection are based on the task. Some examples are:

- RNNs that produce an output h_t at every time step and have recurrent connections between hidden units as shown in Figure 9.
- RNNs that produce an output h_t at every time step that is taken as input for the next time step. A common application for this design is video prediction [20, 97].
- RNNs that only produce a final output after reading a whole sequence. For this, hidden units have recurrent connections in between to process and extract useful information out of every single input. This variation is mostly applied to language understanding [26, 83].

The detailed structure inside the network is not specified and can be adjusted to the particular task. However, popular recurrent architectures include Long Short-Term Memory networks [35] that solve the gradient vanishing problem for large input sequences.

2.3. Convolutional Neural Networks

A special kind of networks for processing data that has a known, grid-like topology like images, are Convolutional Neural Networks (CNN) [49]. They are based on the primary visual cortex of the brain and strongly inspired by neuroscientific research. CNNs usually consist of two important layers: the convolutional layer that is a variation of the

mathematical operation called *convolution*, and the pooling layer that reduces the inner width and height dimension of the data.

2.3.1. Architecture design

An example to motivate the usage of convolutions is having an RGB image of 64×64 pixels as input to a network. If the first hidden layer consisting of a fully connected layer with 1024 neurons, the needed amount of parameters would be the multiplication of the number of pixels, color channels and neurons: $64 \cdot 64 \cdot 3 \cdot 1024 = 12,582,912$. One way to reduce this number is by considering the geometry of the image. A pixel correlates much more with its close neighbors than with far-off pixels [3]. Since the correlation of inputs is represented in the weights of a neuron, the connections could supposedly be reduced to only a local region, called kernel, of the image, like 5×5 pixels (see Figure 10). The stride of such an operation defines the step size over which the kernel is applied on the image. This causes the structure of the hidden layer to be a similar grid as the input. The amount of weights for every neuron is reduced to the kernel size for each channel ($5 \cdot 5 \cdot 3 = 75$) resulting in $75 \times 1024 = 76,800$ weights for all neurons.

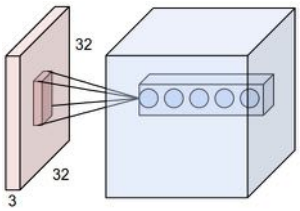


Figure 10: A neuron in a convolutional layer is only connected to a local region of the input. To apply different weight setting on the same kernel, multiple neurons are stacked behind each other which do not share the weights. However all neurons in one channel have the same to reduce the number of parameters [41].

Convolutional layers are going even further by sharing the weights between all neurons. So that one layer does not only consist of one weight set, multiple neuron grids are stacked on each other which are called channels. For example, applying a standard convolution layer with a 5×5 kernel and 64 channels on an input image of the size 64×64 pixels, the first hidden layer would consist of $64 \cdot 64 \cdot 64 = 262,144$ neurons but only have $5 \cdot 5 \cdot 3 \cdot 64 = 4,800$ weight parameters.

The different kernels of a convolutional layer could be seen as filters like edge filter which are applied on the input. The output is often referred as a feature map because the different filters scan the image for special local features like a horizontal or vertical line in an image. Moreover, if multiple convolutions are concatenated, the network learns a hierarchy of features, whereas the low level features recognize small structures close

to the RGB image, and the high level features combine those to detect objects that can take the whole image [99].

However, after applying a convolution, the resulting output has the same size as the input, but may not be so rich of information anymore [48]. To concentrate on the significant features, a pooling layer summarizes a small neighborhood to a statistic like selecting, for example, the maximum or mean value on 2×2 patches. Thereby, the height and width dimension can be down-sampled by using a stride reducing the computational effort and improving statistical efficiency [27]. It is worth mentioning that pooling is done on each channel separately. Figure 11 illustrates the mechanism of a max pooling layer on an image.

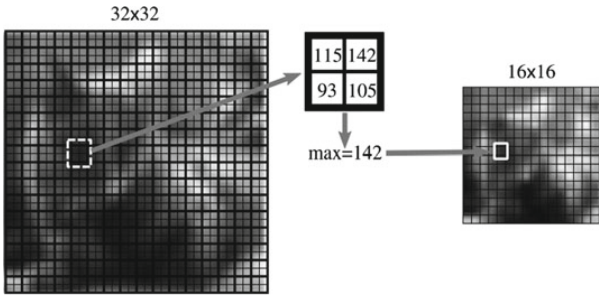


Figure 11: This figure illustrates the procedure of a max pooling layer with a kernel size 2 and stride 2. On every 2x2 pixel patch a max operation is applied to extract only the highest values. This reduces the dimensionality of the input by the factor of 2 and summarizes the local pixel values [3].

The max pooling operation is inspired by the biology where neurons that are strongly activated exceeds their neighbors and weaken their influence. Pooled feature maps are more likely to be invariant to noise or small translations, as a minor change of the input does not significantly affect the summarized output. This property can help rather learn whether some feature is present at all in a small area than the exact position of it, which is especially important for applications like image classification.

The overall architecture of Convolutional Neural Networks consists of multiple stacked convolutions, pooling operations and activation functions like ReLU. As the output structure depends on the task of the network, the final layers are adjusted to it. For example, the task of classification ends up in a 1D-vector, so that the final layer usually is fully connected. The corresponding predictions have to be constrained between 0 and 1 representing the probabilities distribution over all classes while summing up to one. A common way to ensure that is by applying the softmax function on the outputs z:

$$\text{softmax}\,(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.7}$$

One advantage of the softmax function is, that when applying loss functions like the Cross Entropy which maximizes the log-likelihood, the calculation can be simplified as the exp is reversed by log [27]:

$$\log \operatorname{softmax}(z)_i = z_i - \log \sum_j e^{z_j} \tag{2.8}$$

The softmax function can also be applied for semantic segmentation [57, 65]. However, the network architecture has to be adjusted to generate a label in the shape of the input image, which is further discussed in Section 2.4.1.

2.3.2. Example architectures

Convolutional Neural Networks mainly consists of convolutions, ReLU activation functions and pooling operations. Nevertheless, the concrete number, order and parameters of these operations significantly influence the network's performance. Therefore, this section gives a brief overview of three selected, well-known CNN architectures.

AlexNet In 2012, Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton proposed a new deep convolutional neural network, called the AlexNet [46], which was trained to classify over 1 million images in the ImageNet challenge [75] into 1000 different categories. Evaluating the model on a test dataset whether the correct class is within the top 5 predictions, the AlexNet achieved an error rate of only 15.3% significantly exceeding the previous state-of-the-art and other submissions. Therefore, this publication constitutes one of the first outstanding results by CNNs in the field of Computer Vision.

The input to the network is a $224 \times 224 \times 3$ RGB image, on which a convolutional layer with kernel size 11 and stride 4 is applied. This reduces the height and width dimension to 55 with a channel size of 96. The following two convolutions with subsequent max pooling further reduce the feature space. Next, three additional convolutions and a final max pooling are applied before reshaping the two-dimensional features into a linear vector. These values represent the input to three subsequent fully connected layers, each consisting of 4096 neurons. Finally, a softmax function is applied to the output of the last fully connected layer to determine the probabilities distribution over 1000 classes.

Figure 12 summarizes the network architecture. Overall, the model contains about 60 million parameters and 650,000 neurons exceeding the memory of a single GPU with 3GB RAM. Therefore, the network is split onto two GPUs running in parallel and communicating only in certain layers to reduce training time.

Next, to the general layer structure, the AlexNet uses further techniques to improve the performance. For example, after each convolutional and fully connected layer, a ReLU [61] is applied as an activation function to decrease the training time in comparison to saturating non-linear units like tanh. Although the ImageNet dataset contains over 1 million images, the network might struggle with overfitting

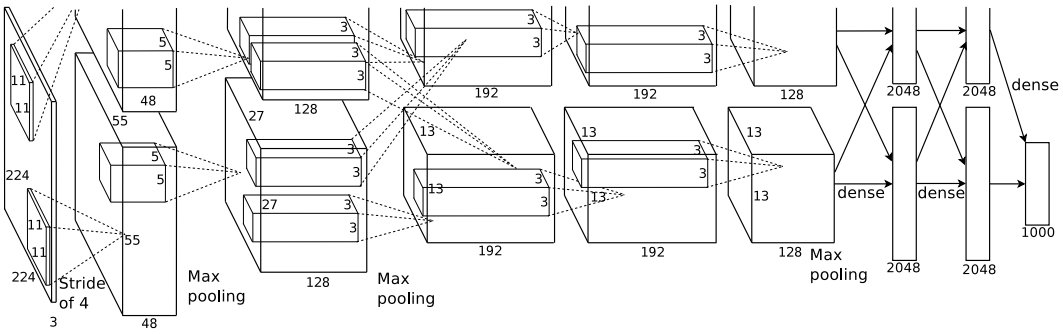


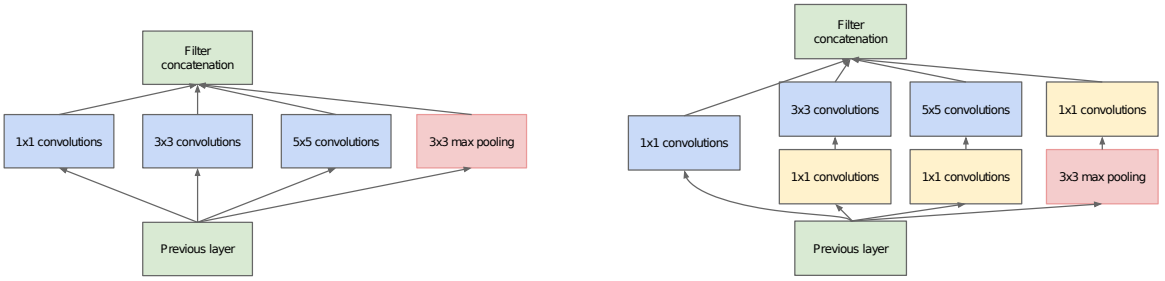
Figure 12: The architecture of the AlexNet consists of five convolutions, three max-pooling and three fully-connected layers. The network is trained on two GPUs on which the computations are divided as shown in the top- and bottom-row. Communications between both GPUs is constrained to certain layers [46].

its 60 million parameters on the training examples. To prevent this, the input images are augmented by cropping a random patch out of the original with the size of 256×256 pixels, and slightly alternating the values of the RGB channels. Within the network, a technique called *dropout* [84] is applied which randomly sets the output of a hidden neuron to zero removing it from the forward and backward pass. This approach reduces co-adaptions of neurons and forces the network to learn robust features [46].

From today's perspective, the AlexNet constitutes a small architecture with only a few layers. It can analyze small-scale images, but with increasing GPU speed and memory, more complex networks like the GoogLeNet [89] or ResNet [32] can be applied to gain higher accuracy. Still, various techniques introduced by the AlexNet, like using ReLU, dropout and data augmentations, can be found in other networks.

GoogLeNet Another famous network architecture is the GoogLeNet [89], which has won the ImageNet challenge with a top-5 error rate of only 6.67% in 2014. The main component of the model is the "Inception module" that is thought to represent the optimal local structure. To develop the perfect module, the input needs to be processed at various scales. On the one hand, features correlate within a small local region, especially on the first layers of the network, which can be covered by a 1×1 convolution. However, there also exists clusters that are spread out more spatially so that only a larger patch size can recognize these. To encounter all problems, an Inception module consists of a convolution with kernel size 1×1, 3×3, 5×5 and a max pooling which are applied in parallel on the same input (see Figure 13a). The output features are concatenated in the end and constitute the input to the next module.

The overall network consists of multiple, stacked Inception modules with increasing channel sizes. Also, the convolutions, especially with larger patch sizes, have a smaller number of channels to reduce the computational effort and following the



(a) Standard Inception module (b) Inception module with feature reduction

Figure 13: (a) The standard Inception module applies a convolution with kernel size 1×1, 3×3 and 5×5 on the input in parallel. Furthermore, a max pooling with a patch of 3×3 computes additional features. Finally, the outputs are concatenated over channels [89]. (b) To improve computational efficiency, 1×1 convolutions are applied before expensive operations like the 5×5 convolution. Note, that the feature reduction of the max pooling is processed after the operation, because it is not computationally expensive while the subsequent 1×1 convolution can enrich the pooled information [89].

hypothesis that there are less large-scale clusters than local ones [89]. Nevertheless, multiple, sequentially applied Inception modules generate a larger channel size that makes the convolutions expensive. A common way to reduce the feature dimension is by applying a 1×1 convolution right before the larger operation. Hence, the channels are compressed, but the layer also includes additional ReLU units introducing further non-linearity. This concept is visualized in Figure 13b.

The GoogLeNet architecture, submitted for the ImageNet challenge in 2014, contained nine Inception modules with feature reduction. However, the first few layers still consisted of simple convolutions with subsequent max pooling for efficiency. The output of the last Inception module (dimension $7 \times 7 \times 1024$) is averaged over height and width so that a 1024 large feature vector remains. On this, dropout, a fully-connected layer and the final softmax is applied.

Containing 22 layers with learned parameters, the GoogLeNet is much deeper than the AlexNet having only 8 layers. Still, it is memory efficient containing just 5 million parameters. The advantage of the GoogLeNet is that its complexity can easily be adjusted to the specific task it is used for by introducing more Inception modules or varying the channel sizes. Moreover, there are multiple improvements and variations of the Inception architecture. For example, [90] proposed the Inception-v2 replacing large patch sizes with multiple small ones (two 3×3 instead of 5×5) and introducing asymmetric $1 \times n$ and $n \times 1$ convolutions that together constitute a $n \times n$ patch with significantly fewer parameters. This variant is more efficient regarding memory and runtime, and, hence, might be applied on devices with limited resources. A more complex architecture is proposed by [91] by combining the Inception module with residual connections of the ResNet [32], which is explained

in the following paragraph.

ResNet The deeper the networks become, the harder it is to train them [32]. Although problems like vanishing or exploding gradients [34] can be encountered by normalization [4, 38], experiments have shown that the accuracy significantly drops when using a large number of layers [32]. However, there exists a solution where the deeper network with more layers has the same performance as the shallower model by learning an identity mapping for the additional operations. Therefore, the decreasing accuracy indicates that deeper networks are harder to optimize.

One way to address this problem is by using residual connections which were introduced by the ResNet architecture [32] in 2015. The idea is to simplify the representations of identity mappings as the non-linearity within the network hampers such structures. Therefore, instead of plainly applying layers on each other, residual connections are added that bypass multiple layers. This concept is implemented by using stacked residual blocks visualized in Figure 14.

A residual block contains a few layers represented by a function $\mathcal{F}$ that process the input as it is done in standard CNNs. In [32], $\mathcal{F}$ consists of two layers each containing a convolution, mostly with kernel size 3×3, followed by a batch normalization [38] and a ReLU (excluding the last layer). Moreover, a skip connection bypasses the input of the block to the output of the last layer where an element-wise addition combines both feature maps. Hence, a residual block applied on the input x, generates the output $\sigma\left(\mathcal{F}\left(x\right) + x\right)$ with σ as the activation function (in the case of [32], ReLU is used as σ).

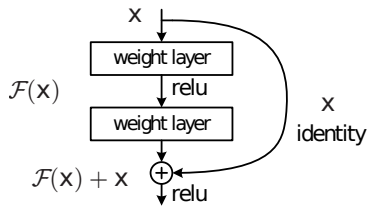


Figure 14: A residual block consists of two layers containing each a convolution and batch normalization, but also a residual connection bypassing both layers with an identity mapping. The output of the block is the sum of the features of the last layer and the input [32].

Residual blocks are easier to be optimized towards an identity mapping, because it is sufficient to output minimal values for $\mathcal{F}\left(x\right)$ and be independent to x instead of learning an identity mapping for those layers [32]. Even networks with more than 150 layers can be successfully trained with residual connections and exceed the performance of any shallower variation. The advantages of this architecture were shown in multiple competitions like winning the ImageNet challenge with a top-5 error rate of 3.57% in 2015. Hence, the ResNet constitutes one of the state-of-the-art Convolutional Neural Networks, especially for computer vision.

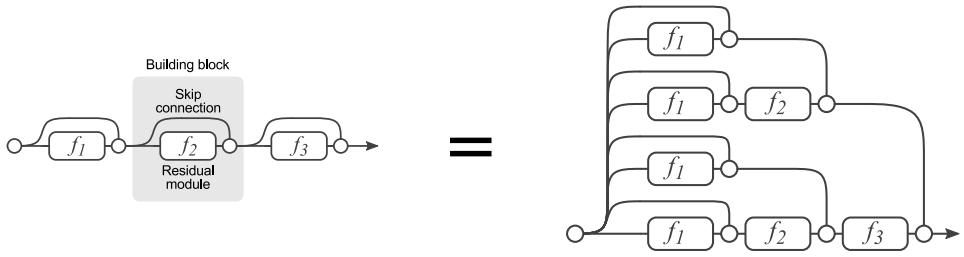


Figure 15: Residual connections are mostly interpreted as skip connections over blocks, as it is shown on the left side. However, visualizing all paths through the network shows an *unraveled view* where the number of paths exponentially increases with the number of blocks. The circular nodes represent element-wise additions [95].

Still, the concept of the ResNet can also be viewed from a different perspective. The skip connection and the layers within can be interpreted as two different paths, and stacking multiple residual blocks on each other generate even more paths through the network (see Figure 15). Analyzing the ResNet using this interpretation, it can be seen that the different paths do not strongly dependent on each other, so that network gets robust against deleting single layers [95]. Moreover, the ResNet further handles the vanishing gradient problem by supporting the gradient flow to the first layers by the skip connections enabling the training of very deep networks with more than 100 layers.

Besides the ResNet, there also exists variations or other architectures with which very deep networks can be trained [85, 91]. One is, for example, the DenseNet [37] that connects each layer within a block to every other layer in a feed-forward fashion. However, the ResNet was one of the first and most popular networks to reach a depth of more than 100.

2.4. Image understanding with CNNs

For the task of autonomous driving, it is essential to understand its environment by, i.e. analyzing camera images. The Convolutional Neural Network architectures presented in Section 2.3 focused on classifying the input image in only one of several categories. However, this is not sufficient for a street scenario, as there are multiple objects like other vehicles and pedestrians that need to be detected. Therefore, this section introduces two common techniques for detailed image understanding: semantic segmentation where the classification is done pixel-wise, and bounding box detection where each object is approximated with a rectangle.

2.4.1. Semantic Segmentation

The goal of semantic segmentation is not to classify the whole image into one single category, but to assign a class to every single pixel of the input image [39]. The result

provides a detailed understanding of the scene that can be used to analyze the traffic environment of an autonomous car. An example for semantic segmentation is shown in Figure 16.

Figure 16: An example street scenario with corresponding semantic segmentation label. Every class is visualized in a different color like cars in blue, pedestrians in red and traffic lights in orange. Image retrieved from [12].

To generate a high-resolution output by using a CNN, the network architecture is often alternated to a fully convolutional model [57] training it end-to-end on semantic labels. A fully convolutional network (FCN) extends the standard CNN architecture to process inputs of arbitrary-size interpreting fully connected layers as convolutions with a receptive field of the whole input. To upscale the compressed features to a full label image, coarse, high-level information from deep layers are combined with low-level features from shallower layers using backwards convolutions, also called deconvolutions. Instead of multiplying a weight matrix with a local region of the input and summing it up, an input pixel corresponds to a local region of the output. The deconvolution operation is shown in the bottom part of Figure 17.

A deconvolution can be used to implement a simple, bilinear upsampling filter. However, the weights of the layer can also be learned as it is just a reversed convolution. This concept was extended by [65] to learn a multilayer deconvolution network on top of the standard convolutional model. The first part of the network constitutes a feature extractor that analyzes the input image and compresses it into a multidimensional feature representation. The second part, the deconvolution network, tries to generate the object segmentation based on the extracted features by using learned deconvolutions, unpooling and ReLU activations. The unpooling operation is, similar to deconvolution, the reverse operation of pooling whereas the locations of the selected, maximum values in the convolution network are recorded and used to place the activations in the unpooling operations. Therefore, the deconvolution network is mostly a mirrored version of the convolutional part with convolutions replaced by deconvolutions and poolings by unpoolings. An example for such an architecture is visualized in Figure 17.

The convolution and deconvolution networks are not constrained to any certain structure and can be deployed with i.e. an AlexNet [46], GoogLeNet [89] or ResNet [32] (see

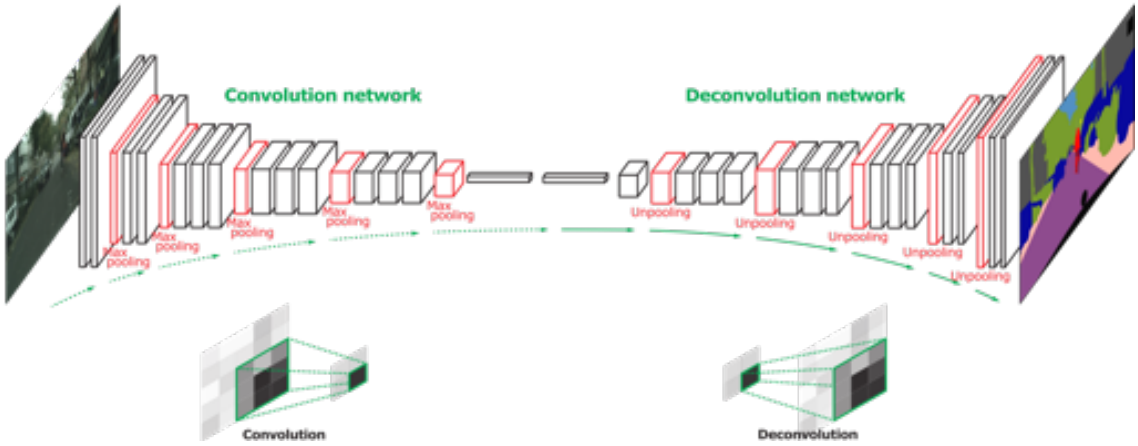


Figure 17: The overall architecture consists of a convolution network that extracts features from the input image and represents those in a low dimensional space, and a deconvolution network that generates a semantic segmentation output based on the extracted features. The deconvolution network mostly is a mirrored version of the convolution network and applies the corresponding, reversed operations like deconvolutions and unpooling. Figure adapted from [65] with example images of [12].

Section 2.3.2). Furthermore, multiple improvements [9, 53, 100] have steadily increased the performance of networks for semantic segmentation [21, 39]. A common technique is to use skip connections from shallower layers of the convolution network to corresponding layers in the deconvolution network to gain a higher accuracy by using coarse- and fine-grained features [57, 72].

2.4.2. Bounding Box Detection

With semantic segmentation, an image can be analyzed pixel-wise and understood in detail. However, single object instances that have to be extracted for tasks like tracking cannot be obtained from semantic labels, especially when multiple objects overlap. Therefore, a new approach has to be developed for detecting object instances in images.

One conventional technique for this task is bounding box detection [24, 56, 70]. The goal of bounding box detection is to detect all objects by approximating their position with a rectangle that fits the object's shape best. For every box, the position and shape, mostly given by x and y coordinates of the center and width and height, as well as the object category has to be determined. To implement this concept with Convolutional Neural Networks, several challenges arise. First of all, every image contains an arbitrary number of objects. Hence, a flexible output size is required, but neural networks are optimized to fixed outputs [27, 70]. Another problem is that the objects significantly differ in their size. An object can cover the whole image, but it can also be only a few pixels wide. The last challenge that should be mentioned here is the detection of

multiple object categories in parallel. The network has to find and classify all objects in an image that is very challenging for a considerable number of possible categories.

The first approaches with machine learning algorithms on bounding box detection were based on a sliding window concept [24, 73, 77]. Therefore, patches of different sizes and positions are cropped from the image and processed by a classifier determining whether the sub-image contains an object, and eventually its category. This approach does not rely on any specific architecture of the classifier and is therefore easy to implement, but the resulting boxes are inaccurate in their size and require higher computational effort [70].

A more efficient, multi-stage approach was introduced by R-CNN (Regions with CNN features) [24] in 2014. The first step of the R-CNN is to propose multiple regions that probably contain an object. R-CNN does not rely on any specific algorithm to generate such region proposals, but in [24] selective search [93] was used for this task. The second module is a CNN, e.g. an AlexNet [46] that extracts feature vectors from the objects. The network is applied to the crop of the region that was proposed by the first module and is further resized to a fixed, squared input image. The computed features for each box are then classified by a Support Vector Machine, and a linear regressor tightens the box of the object if the box contains an object. Figure 18 summarizes this concept.

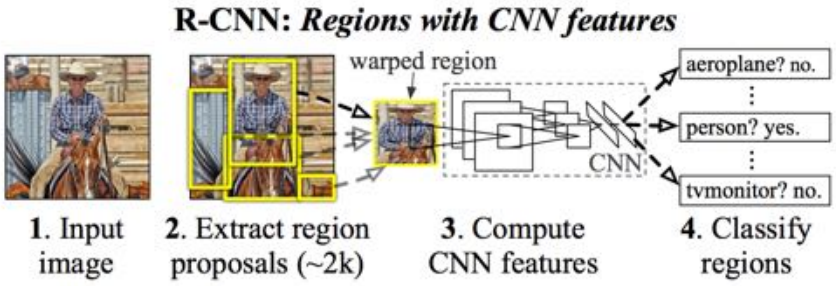


Figure 18: R-CNN operates in a multi-stage manner where the first step is to generate region proposals through an algorithm like selective search. These patches are cropped out of the image and rescaled to a fixed, squared size. A CNN extracts features that are subsequently used for classifying the objects [24].

The approach of R-CNN is intuitive, but also very slow. This is why an improved version, Fast R-CNN [23], was developed that alternated the original approach on two main aspects. Firstly, the feature extraction is performed only once on the whole image and the feature vectors for single objects are obtained by using *region of interest* pooling focusing on the features in the corresponding bounding box. Secondly, the Support Vector Machine is replaced by a softmax layer resulting in one single CNN to train. A further improvement is Faster R-CNN [71] that replaces the selective search algorithm by an additional neural network, called *region proposal network*. This network operates

on the last layer of the initial CNN in a sliding window manner reducing each 3×3 patch to a lower dimension like 256 features. For each of these representations, a set of boxes with various sizes, called default or prior boxes, are specified, and two fully-connected layers are applied to determine whether there is an object in the box or not, and the shape adjustments of the default bounding boxes.

However, all variants of R-CNN are not fast enough to apply them in real time. The first network achieving significant results on object detection with a performance of more than 45 frames per second was YOLO ("You Only Look Once") [68, 69, 70]. The name derives from the strategy that a single neural network predicts bounding boxes and object categories in one single evaluation only looking once at the image. To do this, the image is divided into a grid whereas each cell contains multiple default boxes like in Faster R-CNN and is responsible for detecting objects with their center within the cell. For each box, a confidence score reflecting the certainty of the network that the box contains an object, and the four shape parameters x, y, w and h are predicted. Only default boxes with a confidence score greater a certain threshold, i.e. 0.5, are selected as final predictions. The object class that is also determined by the network is shared among all boxes within a cell. The concept is visualized in Figure 19.

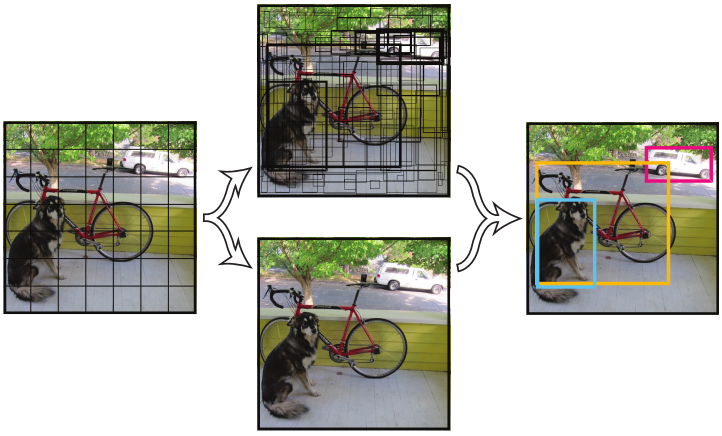


Figure 19: The approach of YOLO is to divide the input image into a grid. For each cell, the network predicts multiple boxes with a confidence score and an object class that is shared among all boxes (visualized by color encoding). Combining both predictions, the final output are those boxes with a confidence higher than a certain threshold [70].

The key idea of YOLO is how to calculate this grid efficiently. The network within YOLO is a CNN which architecture is like for the other approaches not set, but is often referred to as *base network*. However, instead of sampling the output down to a single feature vector, fully connected layers are applied on the last feature map to generate an output size of the grid. The parameters for the shape of the box, the confidence scores or the object classes are stacked over channels. Hence, if the grid has a size of 7×7 with two

default boxes and 20 classes, the output size is $7 \times 7 \times (2 \cdot 5 + 20) = 7 \times 7 \times 30$. Still, YOLO can make duplicate detection for the same objects. To prevent this, a non-maximum-suppression (NMS) is applied on the final predictions removing those predictions that have a large overlap with other predictions.

The disadvantages of YOLO are that it predicts fewer boxes than other approaches thus struggling with many small objects that are close to each other. Also, the regression error for the bounding boxes is treated in the same way for small and large boxes, although small boxes need to be more accurate. The second version of YOLO, called YOLO9000 [68], improves the regression by predicting offsets of hand-picked prior boxes instead of unbounded coordinates making it easier for the network to learn. Moreover, the fully connected layers are replaced by convolutions being able to train on multiple different resolutions. Also, the object class predictions are alternated to a hierarchical classifier which is discussed in more detail in Section 2.5.1.

Nevertheless, YOLO's predictions purely rely on coarse features as it solely performs on the last layer. Therefore, a multilayer approach was proposed for another, well-known architecture: Single Shot Multibox Detector (SSD) [56]. SSD has a similar architecture as YOLO but significantly differs in certain parts. First of all, the box predictions rely on prior boxes and their shape offsets like it is used for the second version of YOLO and Faster R-CNN. Also, in contrast to the confidence score of YOLO, SSD adds the class *background* to the softmax vector representing whether the box is empty or not. The most significant difference to all YOLO versions is that SSD predicts boxes on multiple grids of various sizes. Therefore, the last layers of the network consist of convolutions decreasing in size progressively. On the resulting feature maps, a final 3×3 convolution is applied to generate box predictions on the corresponding grid. As visualized in Figure 20, the original SSD approach used six different scales with overall 8732 box predictions.

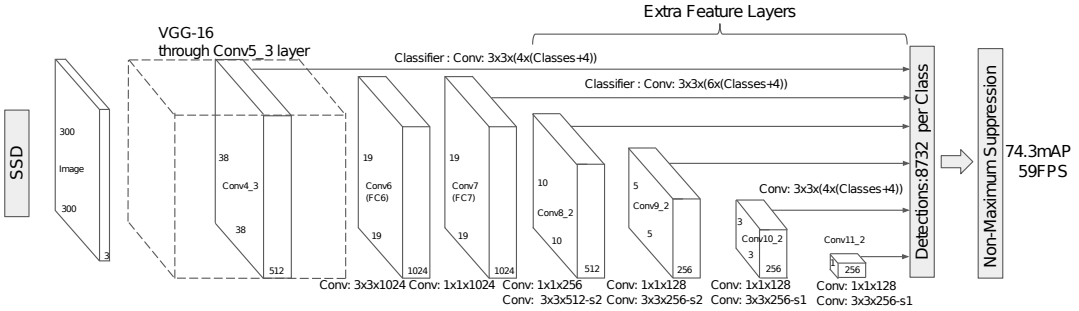


Figure 20: The architecture of Single Shot Multibox Detector consists of a base network that is responsible for extracting the features (in this case the VGGNet [80], but could also be the GoogleNet or similar). On this, multiple, progressively decreasing convolutions are applied. Each of these resulting feature maps is used as a grid to predict boxes on [56].

In conclusion, SSD is more accurate than YOLO, especially for objects with strongly varying shape sizes. However, SSD also contains more layers and, hence, is more complex. Multi-stage approaches like Faster R-CNN suffer under their slow evaluation time. This is why for real-time detection, mostly YOLO or SSD are used [68].

2.4.3. Datasets

State of the art results on semantic segmentation and bounding box detection are mostly achieved by supervised learning. Therefore, datasets with examples and corresponding label sets have to be generated. In the context of autonomous driving, images of typical street scenarios can be easily recorded, as it is sufficient to have a camera taking pictures during a drive. However, labels for semantic segmentation or object detection often need to be created by a human because they have to be accurate to achieve the best learning performance of neural networks. Hence, large datasets are expensive and can rarely be automated.

Nevertheless, deep neural networks require large-scale datasets to develop their full potential [12]. This is why there are well-known, publicly available datasets that provide many training examples and benchmarks to test the neural network's performance on. As this thesis also relies on such datasets, the next paragraphs introduce some of the most popular in the autonomous car context.

KITTI The KITTI (Karlsruhe Institute of Technology and Toyota Technological Institute) dataset [22] was released in 2012 and contains data for stereo, optical flow, visual odometry/SLAM and 3D object detection. The dataset was captured by driving around in rural areas and highways in Karlsruhe. Over six hours of traffic scenarios were recorded with several sensors like color and grayscale stereo cameras, a Velodyne 3D laser scanner and a high-precision GPS system. The 3D object detection challenge includes 7481 training and 7518 test images annotated by a special labeling tool. The task is separated into the detection of vehicles, pedestrians and cyclists as those are the most crucial objects for autonomous driving [39]. Figure 21 shows an example for the 3D object detection challenge.

However, KITTI is limited to a single street section where all sequences were recorded, hence lacking of diversity. Besides, there exist no official pixel-wise annotations for KITTI, so that it is less practical for the task of semantic segmentation.

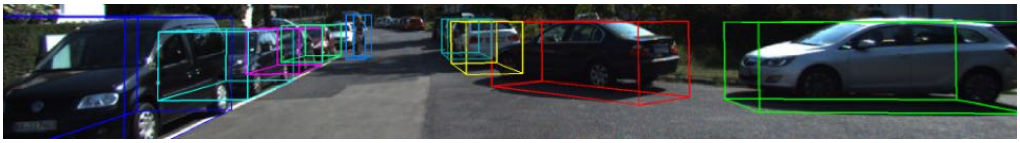
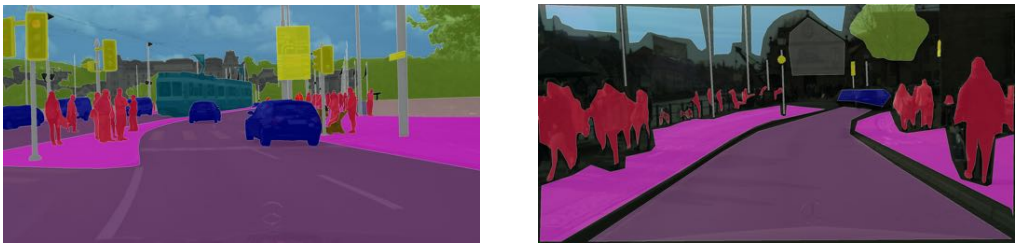


Figure 21: Visualization of 3d bounding boxes in KITTI [22]. Every object is approximated by a box with three shape dimensions and an additional orientation parameter.

Cityscapes The Cityscapes dataset [12] provides pixel-level and instance-level semantic labels of complex urban street scenes. The images were recorded in 50 different cities constituting a diverse set of examples with a resolution of 2048×1024 pixels. Overall, the dataset distinguishes between 30 different classes on 3475 fine annotated images for training and validation and 1525 test images. Also, 20,000 images have coarse annotations where only a part of the image is labeled. Especially the object boundaries are less accurate, but the annotation process is more than 10 times faster so that more labels can be created at the same time. In comparison to other datasets, Cityscapes has a high annotation quality and richness while covering more complex scenarios than, i.e. KITTI [12]. Examples of the dataset are shown in Figure 22.



(a) Fine annotation (Zurich) (b) Coarse annotation (Bamberg)

Figure 22: Example images with overlaying color-coded labels from Cityscapes. Cars are visualized in blue, pedestrians in red and pixels without a label are shown in black [12].

Mapillary Vistas An even larger dataset of street-level images constitutes Mapillary Vistas [62] which was published in 2017. 25,000 traffic scenes from all over the world are annotated into 66 object categories with instance-specific labels for 37 classes. The dataset includes a wide variety of weather conditions and daytimes as well as different imaging devices and perspectives. Therefore, the image resolution also differs with an average of about 9 megapixels. To create the Mapillary dataset, a similar quality assurance pipeline like Cityscapes was applied so that the accuracy of the images is similar high. Figure 23 summarizes some examples from this dataset.

Figure 23: Annotated images from Mapillary Vistas. The labels are more detailed containing lane markings and different kind of grounds like snow, and recorded from various perspectives compared to Cityscapes. The same color encondings like in Figure 22 are used to visualize the labels [62].

2.5. Related work

The following section briefly introduces related work which was taken into account for this project. First, the second version of YOLO, called YOLO9000, is discussed with a focus on the hierarchical classification. The second work proposes an architecture for training a CNN on multiple heterogeneous datasets for the task of semantic segmentation while hierarchically classifying certain objects.

2.5.1. YOLO9000

Common object detectors like YOLO [70] or SSD [56] are mostly limited to detect less than 50 different categories [18, 54]. However, [68] proposes to extend the YOLO architecture recognizing over 9000 different object categories. The key concept of their approach is a hierarchical classification with which YOLO can be trained on detection and classification datasets simultaneously.

First of all, when training a network with multiple, different datasets, the labels need to be matched. For the detection task, the well-known dataset MSCOCO (Microsoft Common Objects in Context) [54] is selected containing up to 91 different object types in 328k images. The ImageNet dataset with its 1,000 classes provides the classification images whereas the labels are based on the WordNet which is a language database structuring concepts and illustrating relations between them [75]. The problem of combining both datasets is that MSCOCO has, i.e. the class label “dog” while ImageNet distinguishes between 147 different dog species. To still train on both datasets, the labels can be structured as a tree or hierarchy based on WordNet with “physical object” as the root node. However, WordNet is not a tree but a directed graph. Therefore, one leaf might be reached by multiple paths from the root node. To convert WordNet to a tree, [68] proposes only to choose the shortest path and remove the others from the graph. Figure 24 visualizes the resulting label hierarchy.

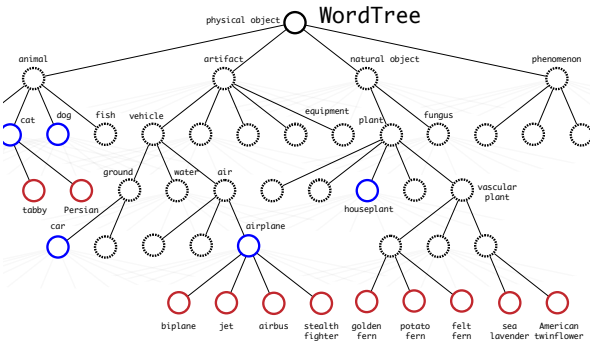
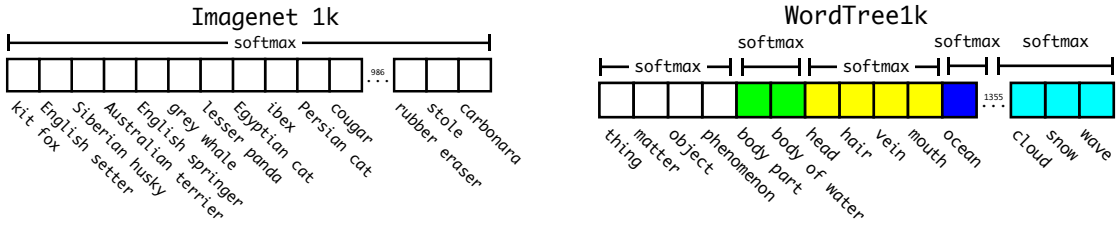


Figure 24: The combined label hierarchy of ImageNet and MSCOCO is based on WordNet. The Figure shows a simplified tree where blue nodes are classes from MSCOCO, red ones from ImageNet, and black nodes are additional classes within the hierarchy sticking the label sets together. Overall, the tree contains 1369 nodes [68].

After specifying the label set, the question remains how to train a network on such a hierarchy. Using a softmax for classification assumes that the classes are mutually exclusive [27, 68], but a hierarchy includes parent-child relations like car and ground vehicle. Therefore, a multi-label model is used where more than one class can be correct at the same time. Furthermore, as neighbor classes with the same ancestor node are yet mutually exclusive, the softmax function can be applied on class subsets shown in Figure 25. Hence, a label consists of multiple classes. For example, if an object is a car, the ground truth is 1 for the class car and all its ancestor, while all direct neighbor nodes have the label 0. Other classes like "dog" whose ancestor is not included in the path from the correct class to the root node, are ignored.



(a) Plain ImageNet class vector (b) Hierarchical WordNet class vector

Figure 25: (a) The 1000 classes from ImageNet are mutually exclusive, so that a softmax can be applied over all classes. (b) The hierarchical classification on WordNet has subclasses and relations between categories. Therefore, a multi-label structure is introduced with a softmax over those classes having the same ancestor [68].

Using the concept of partially applied softmax, only the conditional probabilities are learned by the network like $Pr\,(\text{car} \mid \text{ground vehicle})$. To determine the absolute proba-

bility of a certain class, all probabilities of nodes on the path from the root to the class are multiplied resulting in the following equation for *car*:

$$\begin{aligned} Pr\,(\text{car}) = & Pr\,(\text{car} \mid \text{ground vehicle}) \\ & \cdot Pr\,(\text{ground vehicle} \mid \text{vehicle}) \\ & \cdot Pr\,(\text{vehicle} \mid \text{artifact}) \\ & \cdot Pr\,(\text{artifact} \mid \text{physical object}) \end{aligned} \tag{2.9}$$

The ImageNet dataset only provides classification labels that are not sufficient for training a bounding box detector. However, as MSCOCO already contains object location labels for 91 classes, [68] proposes to solely train the classifier part of YOLO on ImageNet. Therefore, if an image with a single classification label is given during training, the box having the highest probability for the class label is selected and used to compute the classification loss. The regression loss for the box shape is ignored, as no information about the object location is provided. Hence, YOLO learns to locate objects by MSCOCO, but improves its classifier on ImageNet examples.

In experiments, the label hierarchy was even extended to over 9000 classes using the full ImageNet release with more than 14 million images. Evaluating YOLO9000 on the ImageNet detection task, the network performed well learning new species of animals but struggles with objects of categories which are not included in MSCOCO and, thus, lacking of location information. This is why items of clothing like "sunglasses" or "swimming trunks" are very poorly detected with an accuracy of 0.0%, whereas YOLO achieves over 61% on "armadillo" and "tiger" although it has not seen any bounding boxes for those classes.

2.5.2. Training of Convolutional Networks on Multiple Heterogeneous Datasets for Street Scene Semantic Segmentation

Recently, a new approach for hierarchical classification of semantic segmentation was published by [58]. The focus of this work is to train a CNN on multiple, heterogeneous datasets that have different label policies. One of the greatest challenge arising from the combination of, i.e. Cityscapes [12] and Mapillary Vistas [62] is that each dataset has different detailed semantic labels causing a conflict of supervision. For example, the road class in Cityscapes includes lane markings and bike lanes that are separated in extra classes in Mapillary Vistas. Also, the rider class is split up into bicyclist, motorcyclist and other riders. To still train on the detailed labels, a hierarchical approach is introduced.

Figure 26 shows the label hierarchy of [58]. Besides Cityscapes and Mapillary Vistas, the German Traffic Sign Detection Benchmark (GTSDB) [86] is used to get more detailed information of traffic signs. However, this dataset only contains bounding box

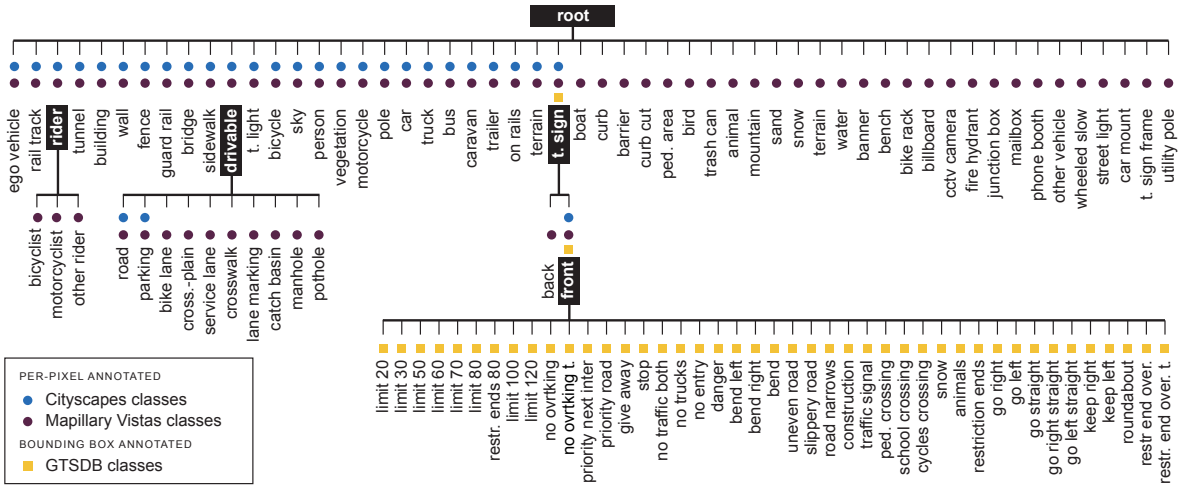


Figure 26: The label hierarchy of Cityscapes, Mapillary Vistas and GTSDB consists of 108 classes structured in 5 subsets with a maximum depth of 3. Classes that are marked black constitute an ancestor for all its subclasses [58].

annotations although the network is trained on pixel-level. To still use both label types, the bounding boxes are converted to semantic by annotating all pixels inside a box with the corresponding class and masking them out by the prediction of higher-level labels (in this case “traffic sign front”).

After specifying the label hierarchy as an approach for combining multiple datasets, the overall network architecture can be introduced. The input image is processed by a single fully convolutional feature extractor based on the ResNet [32] for computing a dense representation like most other networks for semantic segmentation. As proposed in YOLO9000, the hierarchy can be interpreted as a multi-label classification over all subsets. For this approach, not only the softmax is partially applied, but different deconvolution networks are used for each class subset adapted to the specific task. Thus, the classifier for the first stage has a much more complex architecture compared to the one of the traffic sign subclasses “front” and “back”. For the hierarchy of Figure 26, 5 subnetworks are applied resulting in the overall architecture visualized in Figure 27.

In experiments, all input images are downscaled to a resolution of 520×706 to increase the batch size to 4. As the amount of examples is strongly imbalanced across datasets, every batch contains one image from Cityscapes, one from GTSDB and two from Mapillary Vistas. The hierarchical approach is compared to a standard classifier that is trained and evaluated on all three datasets independently. Overall, the proposed architecture clearly outperforms the flat version proving the advantage of the hierarchical classification for combined heterogeneous datasets.

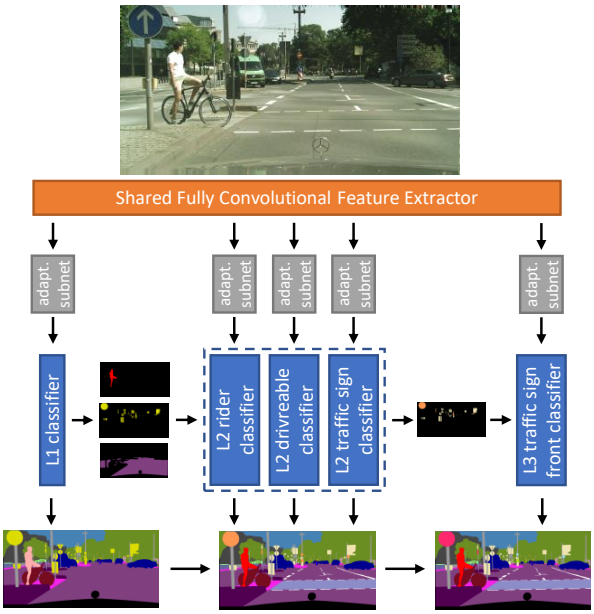


Figure 27: The network architecture consists of a shared feature extractor that transforms the input image to a compressed representation. Based on this, one deconvolution network is applied for each sub-classification for which the architecture is adapted. Finally, the outputs of all networks are combined to get the final pixel-wise segmentation [58].

3. Hierarchical Object Detection

This section discusses the methods and contributions of this thesis. First, the hierarchical classification is introduced and delimited from other literature. Furthermore, a concept for adding object attributes to the classifier is presented. Next, the techniques for improving the performance of rare classes is discussed focusing on variations of the loss function. The third part deals with the evaluation metric for which standard classifier metric are reviewed before addressing the adaptations for hierarchical classification. Moreover, training on such metrics is examined in the following sub-section concentrating on the Intersection over Union (IoU) for tree-based hierarchies. Finally, the calculation of the thresholds for uncertain predictions is discussed.

3.1. Hierarchical classification

Most research in the fields of machine learning and pattern recognition has focused on flat classification where no class taxonomy is considered [78]. However, many real-world classifications are naturally based on a hierarchical structure organizing the classes as a tree or a Directed Acyclic Graph (DAG). To introduce the combination of real-world class taxonomy and machine learning classifiers, the first part of this section clarifies the definition of a hierarchical label structure. Next, the task of hierarchical classification and the different variations that are used in common application domains are discussed. The third subsection addresses a novel approach for adding class attributes to the hierarchy. Finally, example label hierarchies for the context of autonomous driving are presented.

3.1.1. Class taxonomy

Today's state-of-the-art classifiers for semantic segmentation and bounding box detection are mostly based on a flat, multi-class classification, where one out of a set of classes is chosen for each data point [21, 39]. The underlying method often is a softmax applied on the final outputs of a neural network and selecting the class with the greatest score. However, the softmax requires that all classes are mutually exclusive as every class is compared to all others [27, 68]. In real-world problems like analyzing traffic scenarios, this might not always be the case. For example, consider the classes of Cityscapes [12] which include pedestrians, cars, buses and more. To obey all traffic rules, special vehicles like an American school bus must also be recognized. However, a school bus also belongs to the class *bus*, so that adding this class leads to a non-mutually exclusive class set. To handle this conflict, the classes can be organized in a hierarchy, as shown in Figure 28.

A hierarchy consists of a root node to which all classes belong. The relationship between the classes can be expressed by a *is-a* relationship formally represented by $\prec$. For example, the relation between *bus* and *schoolbus* can be expressed by $schoolbus \prec bus$. If a data point is classified as a certain subclass, it automatically belongs to all ancestors of the class as well. Therefore, a prediction consists of a path through the hierarchy

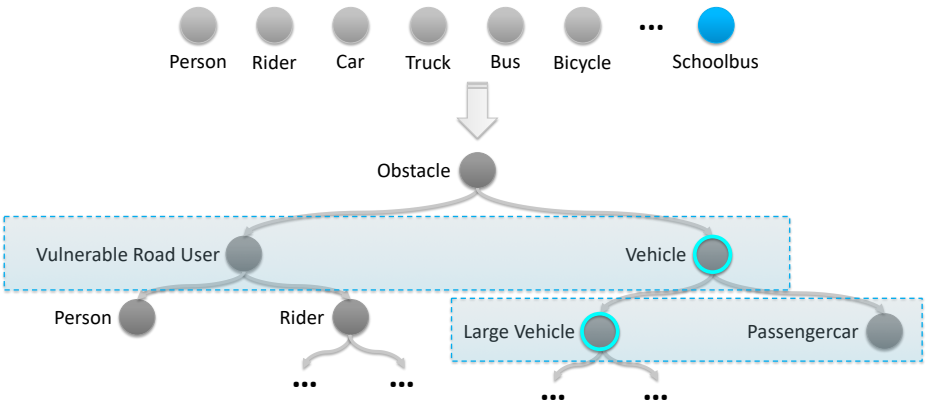


Figure 28: Most multi-class classifications are done in a One-Against-All scheme comparing each class against all others. If those classes are not mutually exclusive, like *bus* and *schoolbus*, a hierarchy can be used to organize the classes. On each level, one class is selected leading the path to the final prediction. To predict the class *schoolbus*, the classifier has to choose *vehicle*, *large vehicle* and so on until reaching the leaf *schoolbus*.

instead of a single class. The number of edges between the root node and a class is defined as the *depth* of the class. The hierarchy can be organized in *levels* that include all classes with the same depth [78]. In Figure 28, *large vehicle* and *passengercar* belong to the same level with a depth of 2, but also *person* and *rider* are assigned to this level.

Before discussing how a classifier operates on such a class structure, the hierarchy should be formally specified. Thus, a class taxonomy C must fulfill the following conditions to be called a hierarchy [78]:

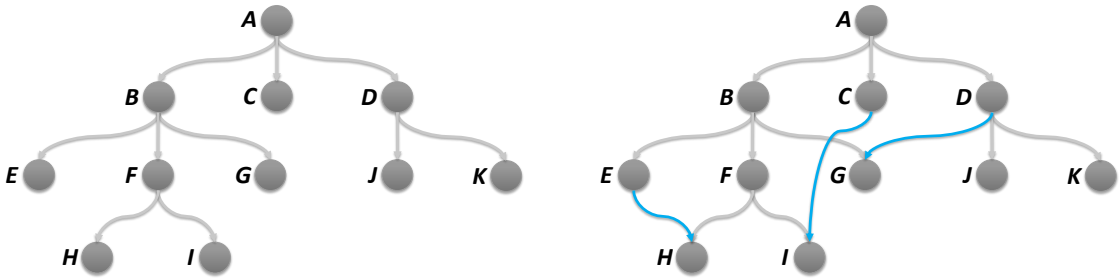
1. There is only one greatest element C_R, called the root
2. Asymmetric: $\forall c_i, c_j \in C$, if $c_i \prec c_j$ then $c_j \not\prec c_i$
3. Anti-reflexive: $\forall c_i \in C, c_i \not\prec c_i$
4. Transitive: $\forall c_i, c_j, c_k \in C, c_i \prec c_j$ and $c_j \prec c_k$ implies $c_i \prec c_k$

The asymmetric property ensures that two classes can not be each others ancestor leading to a circle in the hierarchy. Moreover, as a class should also not be the parent of itself, the taxonomy is anti-reflexive. Transitivity fulfills the definition by cascading the *is-a* relationship throughout levels. However, two different variations of hierarchies can be further distinguished: a tree and a Directed Acyclic Graph (DAG). In a tree, there exists exactly one, unique path from the root node to every class (see Figure 29a). The DAG, on the other hand, allows multiple paths to the same node, even with different lengths (see Figure 29b). Hence, a class can have more than one direct ancestor at the same time. For example, an amphibious vehicle can be assigned to the ancestor *boat* as well

as *ground vehicle*. To separate a tree from the more general DAG, the conditions above can be extended by:

5. $\forall c_i, c_j, c_k \in C$, if $c_i \prec c_j$ and $c_i \prec c_k$ then $c_j \prec c_k$ or $c_k \prec c_j$

A DAG with more than one direct ancestor for at least one node, as e.g. in Figure 29b, does not meet this requirement, because both $I \prec F$ and $I \prec C$ are true, but neither $F \prec C$ nor $C \prec F$ apply.



(a) Tree-based hierarchy (b) Directed Acyclic Graph

Figure 29: (a) Hierarchies are often organized as a tree where only one path from the root node to each class exist. The *is-a* relations between two subsequent classes are visualized by directed connections. (b) A DAG extends the definition of a tree by allowing multiple parents for a node. Hence, the class I belongs to F and B, but also to C. The additional connections compared to 29a are highlighted in blue.

Whether a tree or a DAG organizes the classes is determined by the complexity of the underlying problem. Most literature focuses on tree-based hierarchies [19, 58, 68] as these constitute an easier task than classifying in a DAG [78]. As this thesis examines novel methods for applying hierarchical classification on object detection for autonomous driving, the task is limited to solely considering tree taxonomies. Most of the techniques can also be used or adjusted for DAGs, but exceed the scope of the thesis. The hierarchies developed for the context of autonomous driving are presented in Section 3.1.4.

3.1.2. Multi-class classification on hierarchical structure

Besides the class structure, there are also different methods for performing classification in a hierarchy that need to be reviewed. Firstly, the classification can be implemented that either the selected category always is a leaf-node, or it can stop at any node of the hierarchy [78, 87]. Using the second approach has the benefit that the uncertainty of the network is considered for the class prediction so that for new, unknown objects the classifier can still select the correct class on shallower levels. For example, if the classifier is trained on recognizing cars, buses and trucks, an image of an excavator will

still lead to high confidence of the ancestor *vehicle*, but may not choose a leaf class as none of those fit.

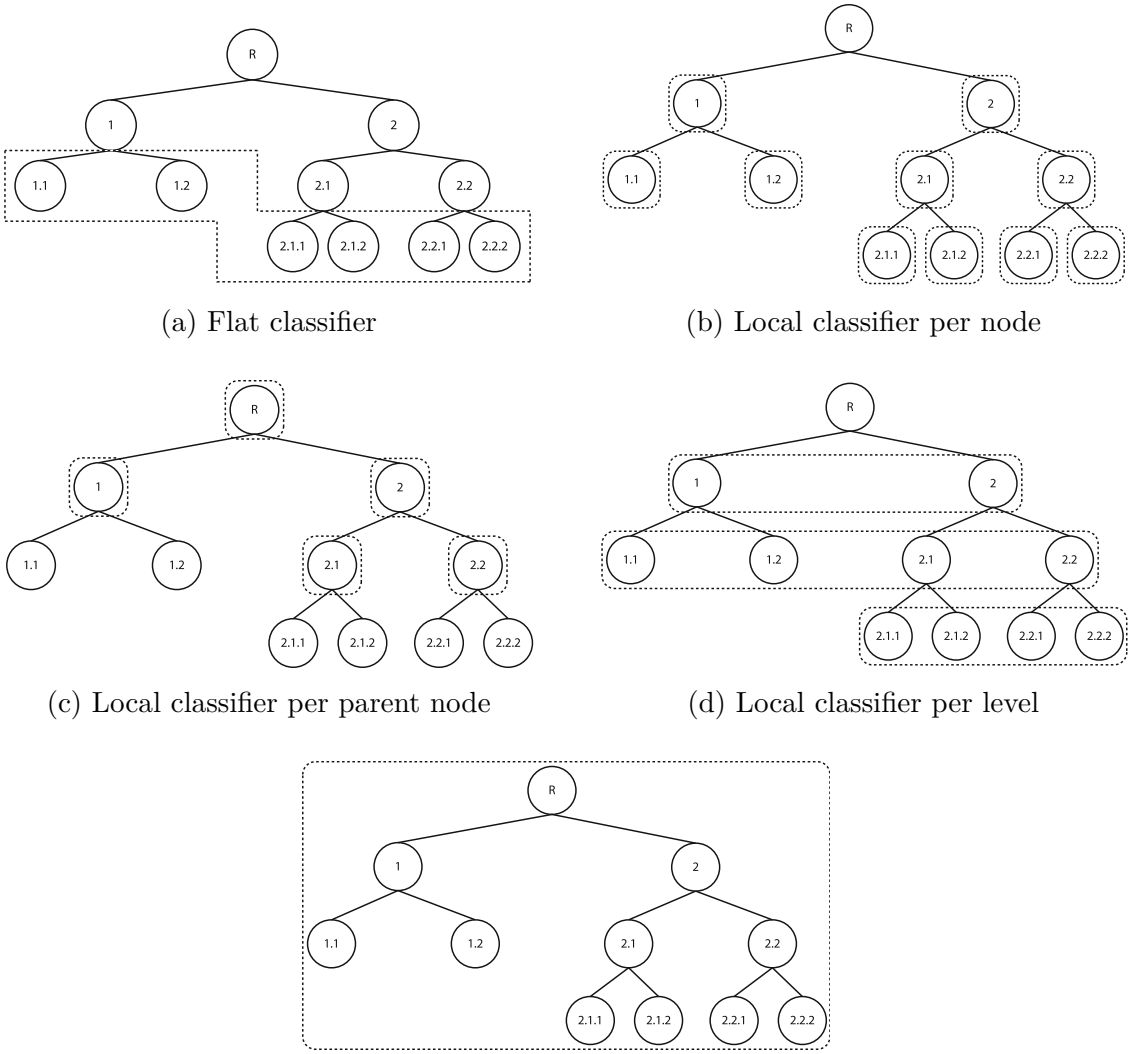
The second aspect for developing a classifier for a hierarchical task is how the structure is explored. Most literature [8, 44, 78, 87] distinguishes between the flat classifier ignoring the class relationships, the local (or top-down) classifier employing a set of classifiers, and the global (or big-bang) approach capturing the class structure in a single classifier. As selecting the best method is a fundamental part for the design of the classifier and significantly influences the performance, the three variations should be discussed in more detail visualized in Figure 30.

Flat classifier The flat classifier behaves like a traditional classification algorithm completely ignoring the hierarchical structure. Typically, the classifier only predicts leaf classes from which all its ancestors might be derived. The concept is visualized in Figure 30a. However, this simple approach has some drawbacks. A flat classifier does not explore any information about the class relationships. Furthermore, it is incapable of handling predictions for inner nodes of the hierarchy as only leaf nodes could be selected.

Local classifier Local classifiers are based on a divide-and-conquer approach. The system starts with predicting its highest level class. At the second level, another class has to be selected, but the choices are narrowed by the previously predicted class. The procedure continues recursively until either the prediction ends up in a leaf class or the classifier stops at an inner node due to, e.g. too low confidence. For each of these steps, a single classifier is required having a local information perspective. There are several ways how to apply a local classifier. The most popular according to Silla and Freitas [78] are:

Per node The local classifier *per node* applies a binary classifier for each node of the class hierarchy (see Figure 30b). There are several methods of selecting positive and negative examples for training as well as how to combine the predictions for a class prediction during inference. A widely used and intuitive approach is to train a classifier to be true if the label is the node's class itself or any of its ancestors. The prediction for unknown inputs is determined by following the path of the classifiers with the highest scores. Alternatively, the child classes can be ignored for training or even included in the negative examples. For testing, other methods have to be used. For example, positive predictions can be propagated upward so that a leaf class with a positive score overrules the negative predictions of all its ancestors. However, as all classes belong to an independent classifier, the class predictions might be inconsistent across different levels.

Per parent node Instead of using a binary classifier for each node, the local classifier *per parent node* subsumes siblings into a multi-class classifier so that only parent nodes are assigned to an own classifier (see Figure 30c). During



(a) Flat classifier

(b) Local classifier per node

(c) Local classifier per parent node

(d) Local classifier per level

(e) Global classifier

Figure 30: Comparison of the most popular variations of hierarchical classification. Circles represent classes whereas the context of a classifier is shown by dashed rectangle [78]. (a) The flat classifier is similar to a standard classifier and categories a data item into one of the leaf classes. (b) The local classifier per node approach consists of a binary classifier for each node of the hierarchy. (c) Using a multi-class classifier to distinguish between child nodes is called a local classifier per parent node. (d) The third variation of local classifiers considers nodes with the same depth, or level. The predictions are masked based on the upper-level selection. (e) A global classifier predicts scores for every class in a single model evaluation [78].

inference, the prediction is determined by selecting the child with the highest score directly beneath the root, and recursively continuing the process for each of the chosen nodes until ending up in a leaf node. Mostly, the same classification algorithm is used throughout the hierarchy. One example for this approach is the work of Meletis and Dubbelman [58] that is presented in Section 2.5.2. The labels are structured in a hierarchy with four nodes having one or more descendants besides the root node. For all those nodes, an independent decoder network is applied to distinguish between the children and predict a corresponding semantic segmentation. The encoder part of the network is shared among the different decoders so that the approach is a hybrid version of a local classifier per parent node and a global classifier.

Per level The third variation of local classifiers applies a multi-class classifier *per level* of the class hierarchy illustrated in Figure 30d. The training and inference can be done similarly to the local classifier per parent node whereas the decisions of the previous layers restrict the possible choices of the following classifier. Otherwise, class relationships are not considered leading to inconsistent predictions. For example, if the classifier of the first level selects the class 1 from the hierarchy of Figure 30d, but the second suggests 2.1, a homogeneous prediction cannot be given except by a post-processing correction method. Hence, the approach of a local classifier per level rarely occurs in literature [78].

Global classifier The global classifier approach, or big-bang approach, uses a single classifier for the prediction of all nodes of the hierarchy visualized in Figure 30e. A great benefit of this approach is that it is typically significantly smaller compared to local classifiers as only one classifier is needed. Furthermore, considering the entire class hierarchy results in a better understanding of class relationships and, thus, consistent predictions across levels. The drawbacks of a global classifier are the lack of modularity, as adding a new class requires retraining of the classifier and the complex architecture for the single model. One example for the approach of a global classifier constitutes YOLO9000 [68]. A single neural network evaluates an image and predicts bounding boxes including the class hierarchy. The method for obtaining the class prediction from the hierarchical scores is similar to a local classifier per parent node, but the network subsumes all classifiers in a multi-class manner.

After reviewing the existing approaches, a suitable method for the task of hierarchical object detection on images has to be selected. In the context of autonomous driving, the classifier needs to be executed in real-time analyzing more than 20 frames per second. Applying multiple, complex neural networks on high-resolution images, as it is required for local classifier approaches, is not practicable as it needs numerous high-end GPUs in the autonomous vehicle for solely this task, or a slow, sequential execution. Hence, this thesis relies on a global classifier approach for which the scores of all nodes are determined by a single network evaluation. Nevertheless, the local classifier per node

approach has the benefit of being modular and flexible as all classes are evaluated by an own binary classifier. Furthermore, the training examples for positive and negative examples can be picked regardless of other classes. This simplifies handling imbalanced class distributions as the training can be carefully adjusted to the desired performance.

To combine the global classifier with local classifiers per node, a neural network can be extended by using binary classifiers for each node of the hierarchy. Therefore, the scores are still predicted by a single classifier taking the entire class hierarchy into account, but the inference of the final class prediction is a post-processing step independently of the network. The disadvantage of the separation of classification and inference is that the class relationships are not encoded within the classifier. Thus, inconsistencies of the predictions might occur and must be handled. The simplest method is a top-down approach similar to the local classifier per parent node that determines the most likely descendant for the root node and all subsequently selected classes based on the predicted probabilities. As the network might be uncertain for objects that it has either never seen or are hard to recognize, a threshold can be specified which the child node's probability must exceed to continue the search through the hierarchy. If this is not the case for any of the descendants, the algorithm stops at the last node that has a threshold higher than the predefined value. However, handcrafting such thresholds is not intuitive and can reasonably influence the network's performance. Determining the thresholds can also be done automatically but requires a second, independent validation dataset to test the performance depending on the selected thresholds.

3.1.3. Object attributes

A hierarchical structure describes the relationships between classes. Still, objects can be further characterized by having attributes like whether the brake lights of a vehicle are activated or not. Representing attributes with an *is-a* relationship is not always practicable especially when multiple classes share the same attribute. For example, considering the brake lights, the attribute can be assigned to the class *vehicle* and all its descendants. Hence, introducing additional nodes for every subclass like *truck with brake lights on* significantly increases the size of the hierarchy. Moreover, as the classes are spread over different levels of the hierarchy, the attribute nodes are compared to the other descendants although a subclass like *truck* should be superior as it provides more specific information about the object, and the attribute can also be assigned at a deeper level. Very few literature deals with the task of integrating attributes in a hierarchical class structure. Therefore, a new, more modular approach needs to be developed here.

The neural network relies on a global classifier so that all scores are calculated in a single evaluation of a multi-class classifier, and the final class prediction is determined by a post-processing step. As the attributes can also be seen as an additional classification task, they are added to the class vector of the network like any other node of the hierarchy and also trained similarly. However, the post-processing part is where the

special properties of the attributes can be taken into account. When the inference stops at a leaf class or an inner node, the prediction can further be specified by evaluating the attributes that belong to the selected class and assign those to the prediction. The class-attribute relationship is represented by a *virtual edge* that is ignored during the class search. In Figure 31, an example of recognizing a school bus with this method is visualized. In the first part of the inference, the class *bus* is determined by following the path of the highest scores towards the leaf class. Next, the single attribute *duty* describing the function of a vehicle is evaluated assigning *school* to the prediction. If more attributes would belong to *bus* they would be considered as independent. Furthermore, the attributes can also be structured in a hierarchy and evaluated like the other classes.

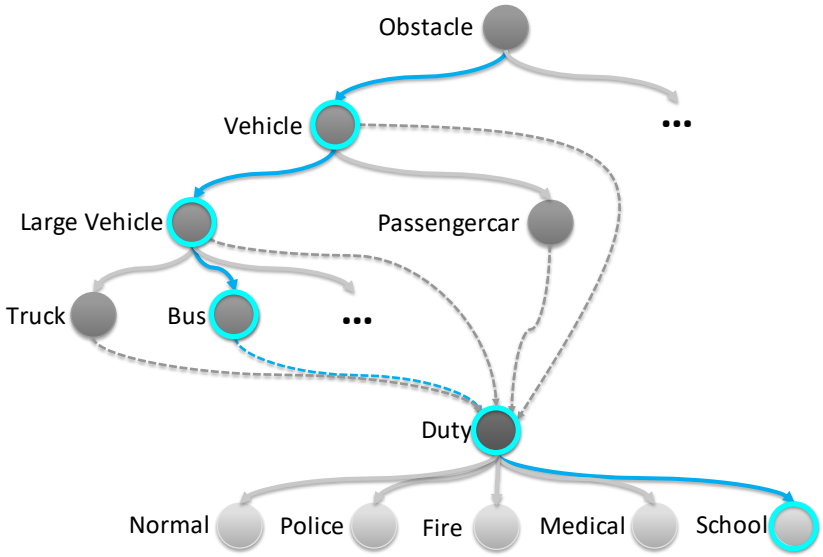


Figure 31: Object attributes in a hierarchical class structure are added as an additional classification task. Multiple classes can share the same attribute leading to a more efficient and modular prediction than using additional subclasses for each combination of node and attribute. Dashed lines represent virtual edges between classes and attributes, and the selected edges and classes for the prediction of a school bus are highlighted in blue.

The most significant benefit of this approach is its modularity. Attributes can be added or removed without alternating the class hierarchy. Moreover, multiple classes can share the same classifier for an attribute as the features are mostly very similar. Considering the example of *duty* in Figure 31, the visual properties of an ambulance or fire truck are nearly the same for similar cars. Especially for rare classes, modular attributes significantly increase the number of training examples and simplify the learning process for the network.

3.1.4. Label taxonomy for autonomous driving

This section gives an overview of the hierarchies that are used for experiments and developing methods for detecting rare classes in the context of this thesis. As the detection system has its application in the field of autonomous driving, the classes in which objects should be categorized are based on this domain. Multiple public state of the art datasets have already defined a class structure for urban street scenes, and thus the hierarchies in this thesis rely on the labels of Cityscapes [12] and Mapillary Vistas [62, 63]. Overall, three different hierarchies are specified varying in size and complexity. The class structures are based on object properties that are relevant for autonomous driving, and not solely on visual similarities. The illustrations of the hierarchies can be viewed in the Appendix B.

The smallest hierarchy is based on the 19 classes of the Cityscapes dataset on which the validation is performed. Nine additional inner nodes are added to form a hierarchical class structure. On the first level, the labels are distinguished between *ground*, *sky* and *obstacle*. The class *sky* has no descendants as for example clouds or airplanes are not relevant for a self-driving vehicle. However, detecting the type of surface like *sidewalk*, *terrain* and *road* is important to estimate the free space for driving. All kind of objects are subsumed in the class *obstacle* having the subclasses *vehicle*, *vulnerable road user*, *nature/vegetation* and *infrastructure*. As vehicles have different driving behaviours, they are further divided into typical *passenger cars*, *large vehicles* including buses and trucks, and *ridable vehicles* like bicycles and motorcycles. Riders of such vehicles are arranged as a child class of *vulnerable road user*, next to pedestrians. The class *infrastructure* contains subclasses like *building* and *pole*, but also *traffic signs* and *traffic lights*. The hierarchy is visualized in Figure 47.

More classes from Cityscapes that are not used for validation are added in a second hierarchy. The additional classes include *caravan* and *trailer* as children of *large vehicle*, and the infrastructures *bridge*, *tunnel* and *guardrail*. Furthermore, the hierarchy also contains some rare classes from Mapillary Vistas like *water* and *trash bin*. The class *rider* is split into *motorcyclist* and *bicyclist* as Mapillary Vistas contains specific labels for those. When training on an example of Cityscapes, the distinction of *motorcyclist* and *bicyclist* is ignored and only the ancestor *rider* is involved in the loss calculation. As the number of different infrastructures increased over 10, they are pre-categorized by additional inner nodes based on their general shape (vertical, elongated or sign). Besides, the attribute node *duty* is introduced that is assigned to most descendants of *vehicle* and *vulnerable road user*. Figure 48 shows the class relationships of all 48 nodes in detail. This hierarchy states the standard class structure for experiments in Section 4 and for further discussions in this section.

To test the scalability of the hierarchical classifier approach, the classes structure is extended by about 20 additional classes. For instance, the class *road* is split into *lane* representing the road class from Mapillary Vistas, *marking* further divided into general

lane markings and crosswalks, *hole* with the descendants *manhole* and *pothole*, and *bike lane* as last subclass. The class *curb* is added as a subclass of *sidewalk* to be consistent to the class definition of Cityscapes. Besides, a new type of infrastructures, called *box-like infrastructure*, is created subsuming classes with a rectangular shape like *junction box* and *bench*. Overall, the large-scale hierarchy provides a more detailed distinction of subclasses but introduces a significant imbalance for the subtree of *ground*. The class structure is visualized in Figure 49.

3.2. Optimization on rare classes

Datasets like Cityscapes [12] and Mapillary Vistas [62] are based on real-world road scenes containing various objects like pedestrians and cars. However, the number of examples for each class significantly differ as some object types are more likely to occur in a scene than others. In the case of semantic labeling, the imbalance is further enhanced by the number of pixels a class covers in average. Standard machine learning algorithms assume that the class distribution is roughly uniform [45]. If this is not the case, these algorithms might fail to represent the distributive characteristic of the dataset adequately and perform poorly on rare classes [30, 31]. As the hierarchies of Section 3.1.4 contain several classes that infrequently occur in datasets, this section reviews methods for dealing with such imbalances. First, the problems of rare classes are discussed more deeply. The second subsection presents techniques to balance the training examples for each class individually, whereas performance-based weights determined by moving average filters are proposed in the last paragraph. Another method for optimizing the predictions on rare classes is using a metric as a loss function. Due to its complexity, it is discussed later in Section 3.4.

3.2.1. Dataset distributions

Many practical applications of machine learning deal with imbalanced data like detecting software defects or cancer gene expressions [30]. Also, the task of environment perception for autonomous driving encounters the problem of recognizing various rare objects that infrequently occur at road scenes. For instance, the class distribution of Cityscapes is highly biased towards flat surfaces like *road* and obstacles as *buildings* and *vegetation*. To give a more detailed review of the class distribution, Appendix A summarizes each dataset by an overview of the average number of pixels per class based on the label hierarchies of Section 3.1.4. Furthermore, the ratio of positive and negative examples is recorded for each classifier showing the data imbalances within the hierarchy. As the optimization techniques in this thesis are developed regarding those datasets, a short discussion about their class distribution is stated here:

Cityscapes Over 65% of typical urban street scenes in Cityscapes are covered by parts of road, buildings and vegetation. The remaining pixels mainly divide into vehicles, sidewalks and other types of infrastructure. Although the class *sky* cannot

be counted as a rare class and occurs on most images, its positive and negative examples are imbalances as the node is arranged at the first level of the hierarchy. Regarding vehicles, passenger cars constitute the majority of the examples whereas both large and ridable vehicles have about eight times fewer examples in average. Especially classes like *caravan* and *trailer* that are ignored during evaluation are underrepresented in Cityscapes. With an average proportion of less than 0.01% of all labels has the class *guardrail* the fewest number of examples. Compared to the most common class road, it has about 3000 times fewer pixels. The class distribution is summarized in Figure 32.

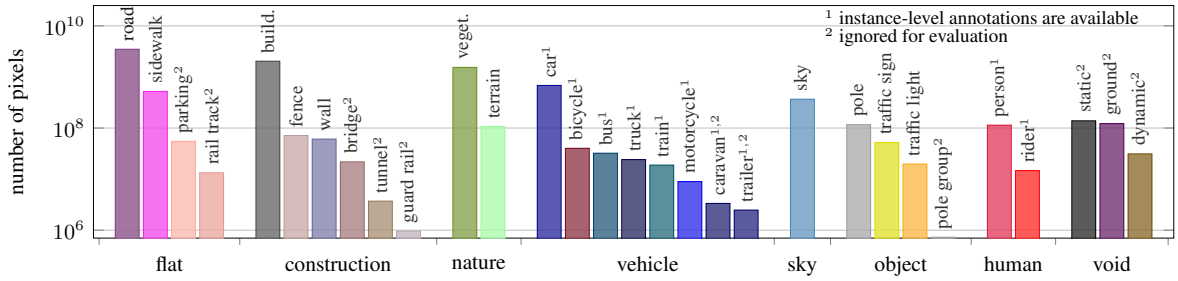


Figure 32: The bar chart shows the number of pixels for each class summed over the whole Cityscapes dataset. The classes are further ordered by their general category defined in [12]. Besides *pole group* which is subsumed in the node *pole* in all hierarchies, the number of pixels differs by a factor of up to 3000 [12].

Cityscapes Special The standard Cityscapes dataset does not provide any labels for the duty of a vehicle. As such labels are required to test out the approach for object attributes proposed in Section 3.1.3, the Cityscapes dataset is relabeled for special vehicles like police cars, fire trucks and ambulances. Those images that contain any vehicle with a specific duty are excluded from Cityscapes and collected in a new dataset, referred to as *Cityscapes Special*. Overall, the dataset contains 230 images including 117 with police vehicles, 90 with ambulances and 20 showing fire trucks. For all other labels that are assigned to the attribute *duty*, its ground truth is given by *normal*. The overall class distribution is similar to Cityscapes, but even though every image contains at least one object with a special duty, over 75% of all attribute labels are *normal*.

Mapillary Vistas Mapillary Vistas is based on a more detailed label policy compared to Cityscapes so that classes like *hydrant* and *trash bin* are included as well. However, those additional classes are more likely to have fewer examples as the labels of Cityscapes are based on objects that frequently occur in urban street scenes. In Mapillary Vistas, the images have a noticeably higher proportion of *sky* due to the different camera devices and perspectives. Also, guardrails appear more frequently because Mapillary includes images of highways from all over the world. *Animal* constitutes the rarest class with an average of 61 of the 2,000,000 pixels per image.

American school buses Next to police, medical and fire, an important type of vehicles with special duty constitute American school buses. As those do not occur in Europe, no examples of school buses can be found in Cityscapes. Still, to train a network on this duty, an initial internal dataset of 200 images of American school buses is used. The labels only include the school buses whereas all other pixels are ignored. By this labeling policy, the dataset could be created much faster but introduces a high bias towards the class *bus*.

Overall, the datasets are significantly imbalanced. Most machine learning algorithms, including neural networks, adapt to this bias as they are mostly optimized to achieve the best result on a randomly sampled data item. If the training dataset is biased, the optimal performance is also shifted so that recognizing common classes is more crucial. In the worst case, the algorithm can even learn to ignore a class if it occasionally occurs and does not significantly influence the mean result [31].

Techniques to encounter imbalanced data can be mostly classified into one of the following types [45]: Data-level methods adding or removing examples to balance the distribution, and algorithm-level methods modifying the existing learning algorithm to handle the bias. Creating new data focusing on the rare classes is not practicable in the context of high-resolution images for semantic segmentation as labeling a single image takes up to 1.5 hours [12, 62]. Thus, only data-level methods that reuse the existing examples like under- and oversampling can be used. Undersampling means that samples of majority classes are randomly discarded whereas oversampling repeats rare data items more frequently. However, if examples are shown too often, the network can overfit its parameters on specific details that are not decisive for the class. Thus, it might fail to recognize objects on new, unseen images.

For algorithm-level methods, the most common approach is adjusting the loss as it constitutes the performance measure during training [30]. Weighing the loss function so that all classes are equally represented is not practicable. For instance, the class *guardrail* would require a weight of more than 500 when solely considering the Cityscapes dataset. If the image contains labels for *guardrail*, the loss is enormously higher compared to other examples, even if the network performs well. The oscillating loss destabilizes the training, especially if several instances of rare classes occur in a row.

To spread rare examples equally over the dataset, it is split into multiple subsets of images containing objects of rare classes. However, no example should be significantly more often shown than others. So, the frequency with which an image is sampled from a subset depends on its size. For the algorithm part, more complex approaches are taken into account that are presented in the next subsections.

3.2.2. Independent classifiers

Every class has a different ratio of positive and negative examples. For a flat multi-class classifier of N classes, the average ratio of positive examples is $1 : N - 1$. Hence, the

flat classifier approach naturally introduces a strong imbalance for large class sets. In contrast, a hierarchical class structure already helps to reduce the number of negative examples as the predictions of a node are only compared to its siblings. Negative examples that are assigned to an ancestor are not passed to the subclasses while positives are propagated throughout the hierarchy. For instance, the descendant *natural ground* of the node *ground* should predict a probability close to 0 if the label is assigned to its sibling *street*, but is ignored if the label is a class of a different sub-tree like *vehicle* or *sky*. Thus, the above-mentioned imbalance can only occur between nodes with the same ancestor significantly reducing the number of classes that are taken into account. Especially for rare classes, the dataset is reasonably improved by the hierarchical class structure. Considering the distribution of Mapillary Vistas (see Table 7, Appendix A), the class *animal* would have a positive ratio of $61 : 2,097,152 = 1 : 34380$ for a flat classifier. Using the hierarchical classification, the node is only compared to *human* and has a ratio of $61 : 11,282 = 1 : 185$ reducing the imbalance by a factor of approximately 200.

Still, the labels are imbalanced and rarely uniformly distributed over all classes. A conventional method to balance positive and negative examples is weighing the loss for each of these classes differently so that the predictions of data items assigned to positive and negative labels equally contribute to the loss [30]. However, most multi-class classifiers rely on the softmax function where the resulting probabilities of each class depend on all others. Thus, optimizing a single score to 1 means to push all other probabilities to 0. If multiple rare classes are assigned to the same parent node, their weight for positive examples constitute a high factor for the loss of negatives labels for all sibling classes. A technique where the weighting can be easier and more accurately applied is using independent, binary classifiers for each node of the hierarchy as mentioned in Section 3.1.2. The loss is based on a binary cross entropy whereas the weight of each data item is determined by its label. The sum over all examples is the final loss that the network tries to optimize.

Balancing positive and negative examples for all classifiers to the same proportion also has some drawbacks. The mean loss over a set of examples can be interpreted as the average proportion of correct classifications. If the loss is desired to be similar for the negative and positive predictions, the classifier will also perform with a similar average quality on both classes. However, the absolute number of misclassifications significantly differs when the distribution is biased. For instance, if the classifier of *animal* recognizes 90% of the examples for both classes, it predicts up to 20 times more often an example of the class negative to be right than correctly classifying a positive sample. Thus, rare classes should be predicted less to get a better trade-off between correctly classified positives and negatives. The desired proportion of misclassifications is often defined by metrics which are further discussed in Section 3.3.

Instead of balancing the dataset, the loss can also be reshaped to focus on examples that are poorly classified. A similar approach is applied by Lin et al. [55] for object

detection called *focal loss*. The standard cross entropy is scaled by a factor that decreases in correlation to the confidence of the network's prediction. By the use of a *focusing* parameter γ, the decay rate of the loss can be adjusted. The weighting factor $\alpha \in [0, 1]$ further balances the losses for positive and negative labels. The focal loss is formally defined in Equation 3.1. To be consistent to the definition of the binary cross entropy in Equation 2.3, the prediction is given by $\hat{z}$ and the ground truth by z.

$$p_t = \begin{cases} \hat{z} & \text{if } z = 1 \\ 1 - \hat{z} & \text{otherwise} \end{cases}, \quad \alpha_t = \begin{cases} \alpha & \text{if } z = 1 \\ 1 - \alpha & \text{otherwise} \end{cases} \tag{3.1}$$
$$FL(p_t) = -\alpha_t \left(1 - p_t\right)^{\gamma} \log\left(p_t\right)$$

The default cross entropy loss is given by $\alpha = 0.5, \gamma = 0$. Increasing γ reduces the weight of the well-classified examples. For instance, if $\gamma = 1$, the loss for a prediction of $\hat{z} = 0.9, z = 1$ is 10 times smaller compared to the standard cross entropy. The focal loss is visualized for several values of γ in Figure 33.

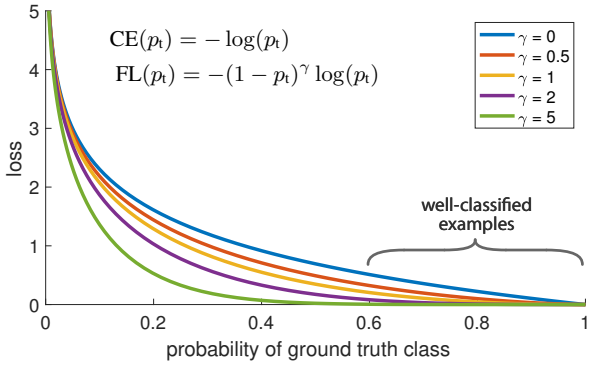


Figure 33: The focal loss scales the cross entropy by the factor $(1 - p_t)^{\gamma}$. The greater γ is set, the smaller is the loss for well-classified examples. The plot compares the loss function for different values of $\gamma \in [0, 5]$ [55].

Setting γ to a value higher than five is less practical because the loss for a random prediction of 0.5 almost is counted as a well-classified example of both positive and negative labels. In experiments by Lin et al. [55], a value of 2 for γ and 0.25 for α constitute the best setting for the imbalanced dataset of positive and negative examples for bounding box detection. As the classifiers within the hierarchical class structure encounter less imbalance, a lower value of γ might be used. However, the optimal set might correspond to the number of positive and negative examples. To test this hypothesis, more than 100 experiments with different values of γ and α for each classifier would be necessary. As this should not be the focus of the thesis, a default configuration of $\gamma = 1$ and $\alpha = 0.5$ is applied.

3.2.3. Moving average filters for class weights

The difficulty to learn a class does not always correspond with the number of examples. For instance, an American school bus is due to its distinctive yellow color easier to recognize for the network so that fewer examples are sufficient. Weighing the loss solely based on the dataset distribution can cause the network to focus on classes that are recognized well. Thus, a new approach that dynamically adjusts the weights to the current performance is developed.

Firstly, the performance has to be efficiently measured as the calculation is done during training and should be executed as often as possible. Determining the scores on the validation dataset requires larger step sizes as 500 images have to be processed per iteration. As an alternative, the mean distance of prediction and label during training can be used as a metric. The more confident the network is on a class, the more accurate are its predictions. The distance is determined merely by $dist_i = |z_i - \hat{z}_i|$ for all data items i (gathered by positive and negative examples) and can, therefore, be calculated by a negligible computational effort. As the prediction accuracy can strongly vary over images and the performance should ideally represent the whole dataset, a moving average filter (MAF) is applied that smooths the input by taking the average of the last M values. Thus, the metric is defined as the mean distance of the previous M batches.

Based on this metric, the weight factors for positive and negative examples can be determined. The best performance is stated by a distance of 0, while the worst is given by 1. The network has to focus on the class which is poorly recognized compared to the other. Thus, if the mean distance of the positive examples is close to zero, its weight factor should also be small whereas the loss of negatives is scaled up. To implement such a weighting method, the mean distances are divided by each other. For instance, if the mean distance of the positive examples is 0.17 and 0.1 of the negatives, the positive weight factor is calculated by $\frac{0.17}{0.1} = 1.7$. Similarly, the weight factor for negative examples is $\frac{0.1}{0.17} \approx 0.6$. The goal of the loss weighting is reduce the imbalance for each classifier so that the optimal setting is having both factors equals one. Figure 34 visualizes the weighting technique.

The mean distance is calculated for each class independently. However, as the class structure is based on a hierarchy, the relations of the classes need also to be taken into account for the weights. For example, most positive examples of the node *elongated infrastructure* belong to the descendant *building*. Calculating the average over all examples clearly prefers the data items where the labels are assigned to *building*. If the classifier performs well for *building* but not for other rare subclasses like *guardrail*, the loss is probably downscaled reducing the performance on rare classes. To introduce class relations into the moving average filters, the mean distance is determined regarding the labels. Thus, every class is assigned to a positive moving average filter for all its descendants and a filter for negative examples of all sibling nodes and their subclasses. The weight for a pixel is calculated by the value of the corresponding label-dependent filter divided

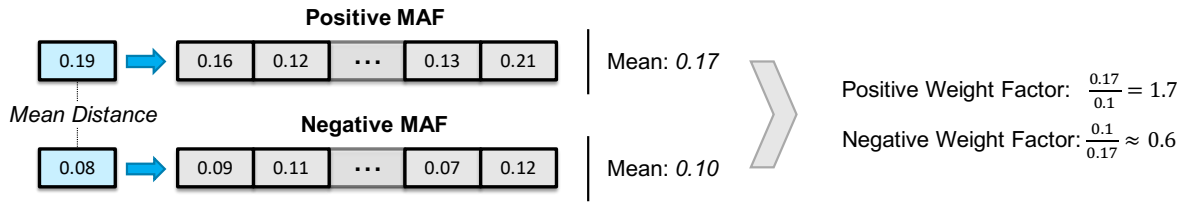


Figure 34: The performance of the network is measured by the mean distance of predictions and labels for both positive and negative examples. The value is smoothed by applying a moving average filter saving the distances of the last M training steps. The loss weight is the ratio of positive and negative mean distance.

by the average distance for the other binary class. Regarding the previous example, the positive loss of the class *elongated infrastructure* is now differently weighted whether the label is assigned to *building* or *guardrail*. However, both mean distances are compared to the average distance of negative examples which is determined by the moving average filter that is independent of the labels. The example is illustrated in Figure 35.

The method of label-dependent moving average filters helps to detect common confusions, even for frequently occurring classes. If the network often predicts *vegetation* instead of *nature*, the loss weight is increased to prevent such confusions. However, if the dataset is extremely imbalanced and a class rarely has positive examples, the weighting can lead to a fixed point where the average positive distance is much greater than the negative, but the calculated weight factor is not high enough to compensate the imbalance. In experiments, no such case could be observed for any of the classes in the hierarchies presented in Section 3.1.4. Another drawback of the proposed weighting technique is that the calculated factors are independent to the actual loss. If a class is harder to learn, the loss for positive and negative examples is greater than the average of other classes. An additional weight can make the network focusing on such a class and try to optimize the mean performance over all classes. For a hierarchical approach where multi-class classifier instead of binaries is used, this weighting method would be applied to the set of directly connected children to balance the dataset between them. Nevertheless, in Section 3.4, a loss function based on the Intersection over Union metric is proposed that considers each class independently to the number of examples. Thus, a weight factor for each class might be redundant to the higher loss a worse performance already causes.

3.3. Hierarchical metric

To compare different models, the performance of each need be determined. This is mostly done by metrics that compress the network predictions into a few scores representing the quality of the predictions. Although there exists multiple, common metrics for standard

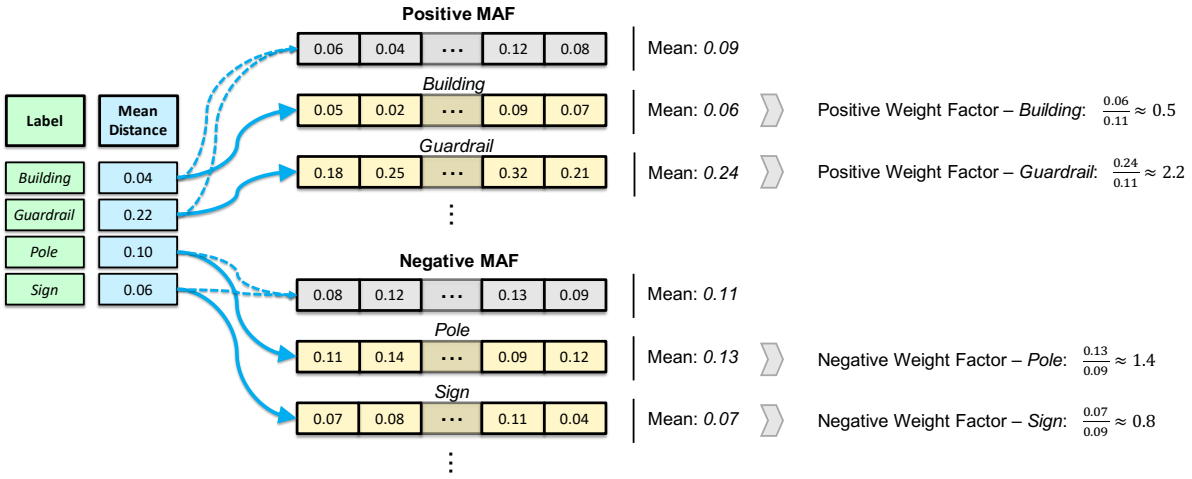


Figure 35: The figure shows a label-dependent weighting for the class *elongated infrastructure*. On the left, the mean distances between labels and predictions are stated for selected child- and sibling nodes. For each subclass (top row), a MAF for the positive distance measures is created. Besides, the average distance over all pixels independent of the labels constitutes the input to another filter. Similarly, sibling nodes and their children (bottom row) are assigned to a MAF for negative measures. The label-dependent weights are calculated by dividing the measure of the corresponding class MAF by the mean distance of the opposite binary class.

classification, semantic segmentation and object detection tasks, most of them do not consider relationships between classes. Confusing cars and trucks are less crucial than, e.g. poles and pedestrians so that the metric has to be adjusted to such relations. Therefore, this section firstly reviews popular metrics for flat classifiers, and secondly, discusses hierarchical approaches for measuring the performance of a network.

3.3.1. Review of classifier metrics

A common technique for evaluating the correctness of a classifier is to determine the numbers of correctly and incorrectly classified examples. Therefore, four parameters are defined for each class:

- True positives (TP) are examples that are correctly predicted as the class.
- True negatives (TN) are the cases where the prediction and the ground truth are both not the class.
- False positives (FP) are falsely assigned to the class.
- False negatives (FN) are examples that are incorrectly classified as not belonging to the class.

These parameters often are ordered in a matrix, called confusion matrix. For a binary classification task, where only the classes positive and negative are considered, a confusion matrix for the positive class is shown in Table 2. The columns represent the predictions of the classification algorithm, whereas the rows constitute the corresponding label. If both prediction and ground truth are assigned to the positive class, the example counts to the true positives. Similarly, if both are negative, it belongs to the true negatives. The cases where the prediction differs from the ground truth are divided into false positives and false negatives based on the predicted class.

		Prediction	
		Positive	Negative
Ground Truth	Positive	TP	FN
	Negative	FP	TN

Table 2: A confusion matrix for a binary classification task separates the evaluation results in the combinations of ground truth and prediction. The columns represent the selected class by the classifier and the rows the labeled class while the corresponding parameter is shown in the cell. The visualization is adapted from [14].

However, most classification problems have multiple classes. The confusion matrix can be extended to this task by listing the classes over rows and columns and recording the combination of prediction and ground truth label for every evaluation input. Table 3 represents an example of a confusion matrix for the classes *dog*, *car* and *person*. The parameters TP, FP, FN and TN can be similarly determined as for the binary classification. When looking at the class *car*, the number of true positives is $TP_{Car} = 100$ as there prediction and ground truth is assigned to the class. The true negatives are all examples where neither prediction nor ground truth belong to the class *car*, resulting in $TN_{Car} = 20 + 12 + 25 + 215 = 272$. The number of predictions for *car* where the ground truth is any of the other classes, are summed up to the false positives $FP_{Car} = 8 + 13 = 21$. The last parameter, the false negatives, is thus represented by the the cases where the ground truth is assigned to *car*, but the algorithm recognized it as *dog* or *person*: $FN_{Car} = 5 + 20 = 25$. Furthermore, the overall number of ground truth labels can be expressed by the parameters. The amount of positive labeled examples for the class *car* arise from the sum of its true positive and false negatives: $P_{Car} = TP_{Car} + FN_{Car} = 125$. Hence, the sum of true negatives and false positives constitute the amount of negative labels: $N_{Car} = FP_{Car} + TN_{Car} = 293$. The same values for the positive labels can be calculated by a sum over columns whereas the negatives are the sum of P from all other classes. Still, the first conversion is often used to simplify the computation of various metrics.

The confusion matrix visualizes the relation of the class predictions making it easier to spot common mistakes or confusions. In Table 3. For example, it gets obvious that the

		Prediction		
		Car	Dog	Person
Ground Truth	Car	100	5	20
	Dog	8	20	12
	Person	13	25	215

Table 3: The table summarizes an example of a confusion matrix for a multi-class classification of the classes *car*, *dog* and *person*. The numbers in the cells constitute the number of appearances for the specific combination of ground truth (row) and prediction (column).

algorithm is more likely to mix up the classes *dog* and *person* rather than predicting *car*. Nevertheless, when comparing two different classifiers, it is hard to determine the better performance of both as one might have a lower false positive rate for a certain class, but the other might recognize more examples resulting in a lower number of false negatives. To measure the performance of a classifier, multiple metrics exist. Therefore, the most common evaluations should be presented here and calculated for the example in Table 3.

Precision The precision measures how often a positive prediction for a certain class was correct. For example, the classifier of Table 3 predicted 121 times the class *car* from which 100 were correct. Thus, the precision is determined by $\text{Prec}_{Car} = \frac{100}{121} \approx 82.6\%$. The computation can also be represented by the parameters of the confusion matrix:

$$\text{Prec} = \frac{TP}{TP + FP} \tag{3.2}$$

Recall Similarly to the precision, the recall determines how many examples of a class were actually recognized by the classifier. As for the class *car*, 100 of 125 cars were correctly classified, the corresponding recall score is calculated by $\text{Rec}_{Car} = \frac{100}{125} = 80.0\%$. Again, the formula can also be expressed by the defined parameters:

$$\text{Rec} = \frac{TP}{TP + FN} = \frac{TP}{P} \tag{3.3}$$

F-score Both precision and recall can be combined into one single metric, called the F-score or F1-score. The measure is the harmonic mean of precision and recall which is the average if both values are equal but tends to the smaller number in the case of imbalance. The advantage of the harmonic mean in comparison to the arithmetic mean concerning classifier evaluation is that if an algorithm predicts every example as *car*, it would have a recall of 100% but a precision of 0%. Although the performance is bad, the arithmetic mean would result in a score 0f 50%. Despite, the harmonic mean is 0% in this context representing the issue

much better. Nevertheless, not always is recall and precision equally important. With an additional constant β, the trade-off between precision and recall can be adjusted to the specific task. Overall, the F-score is calculated by:

$$\text{Fscore} = \frac{(\beta^2 + 1) \cdot \text{Prec} \cdot \text{Rec}}{\beta^2 \cdot \text{Prec} + \text{Rec}} \tag{3.4}$$

Accuracy The accuracy rate evaluates a classifier by its percentage of correct predictions. Therefore, not only the true positives but also the true negatives are taken into account. For example, the classifier of Table 3 has 100 correctly recognized *car* objects and 272 examples where neither ground truth nor prediction was assigned to this class. Overall, the evaluation included 418 examples. Hence, the accuracy for the class *car* is determined by $\text{Acc}_{Car} = \frac{100+272}{418} \approx 89.0\%$. Concerning the parameters of the confusion matrix, the accuracy rate can be expressed by:

$$\text{Acc} = \frac{TN + TP}{TN + FN + TP + FP} = \frac{TN + TP}{N + P} \tag{3.5}$$

However, this measure should rather be used when the class distribution is nearly balanced. Otherwise, the accuracy rate can still be high for a class that is not recognized well. In the example above, the class *dog* has an accuracy of 88.0% although the precision and accuracy are both equals or less than 50%.

Intersection over Union The last performance measure that should be discussed here is the Intersection over Union (IoU), also called the Jaccard Index. This measure describes the similarity of predictions and labels over a set of examples by dividing the intersection by the size of the union. Regarding the confusion matrix, the intersection of predictions and labels are the true positive examples for a specific class, whereas the union is the number of examples for which either prediction, ground truth or both are assigned to the class. Therefore, the score can be calculated by:

$$IoU = \frac{TP}{FN + TP + FP} = \frac{TP}{P + FP} \tag{3.6}$$

The Intersection over Union is a common measure for semantic segmentation where every pixel constitutes an example, and the predicted and labeled region can actually be compared by visualization.

All presented metrics share the feature that a higher score means the better performance of a classifier. For multi-class classification, each measure is calculated for every class independently. However, to get a single final evaluation number, the scores are mostly averaged over classes. As an alternative, the parameters TP, TN, FP and FN can also be summed up over classes and used for calculating the scores. The disadvantage of this

method is that it ignores the class imbalance and is dominated by frequently occurring classes [82]. Furthermore, all measures that use the number of true negatives in a multi-class environment are mostly unstable regarding class imbalances as the performance of a single class is significantly influenced by the number of examples for other classes. When distinguishing between N classes, a random prediction would generate at least $N-2$ true negatives if only one class can be chosen at a time. Therefore, measures like Precision, Recall or IoU are preferred to multi-class classification in contrast to accuracy [82].

3.3.2. Depth-Dependent Distance Metric

All previous evaluation measures are inadequate for hierarchical classification models as they do not take the class topology into account. Thus, the fact is ignored that deeper classes might be harder to distinguish than shallow ones, but also that a false prediction of a sibling is less crucial than classes from a whole different part of the hierarchy. Various performance measure for the task of hierarchical classification exist that adjust better this domain and actively consider the relations between classes [14, 19, 78, 82]. Therefore, this section discusses the application of hierarchical metrics and their advantages and disadvantages in the context of this thesis.

Firstly, the precision and recall rate can be adapted to hierarchical classification. For this, [44] proposed to use the set of nodes on the path from the root to the prediction and label instead of single classes. The root itself is excluded from the sets since all examples belong to the root by default. The similarity is measured by the cardinality of the both paths' union divided by the cardinality of either the prediction (precision) or the ground truth (recall). Thus, the score of the hierarchical precision is determined by:

$$hPrec = \frac{|Ancestor\,(C_p) \cap Ancestor\,(C_t)\,|}{|Ancestor\,(C_p)\,|} \tag{3.7}$$

with C_p as the predicted class of the classifier algorithm, and C_t the ground truth. The recall replaces the denominator by the label set resulting in following equation:

$$hRec = \frac{|Ancestor\,(C_p) \cap Ancestor\,(C_t)\,|}{|Ancestor\,(C_t)\,|} \tag{3.8}$$

As an example, consider the hierarchy in Figure 36 where $C_p = G$ and $C_t = I$. The paths of both are determined by $Ancestor\,(C_p) = \{B, F, I\}$ and $Ancestor\,(C_t) = \{B, G\}$. The precision is hence $hPrec = \frac{|\{B,G\} \cap \{B,F,I\}|}{|\{B,F,I\}|} = \frac{|\{B\}|}{|\{B,F,I\}|} = \frac{1}{3}$ and the recall $hRec = \frac{|\{B\}|}{|\{B,G\}|} = \frac{1}{2}$. In contrast, using the precision and recall for flat classifiers, both would end up being 0.0% although the prediction was not so far away from the ground truth.

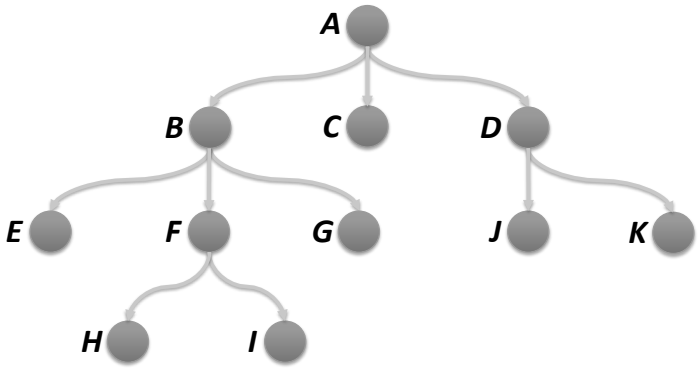


Figure 36: The nodes of the hierarchy are visualized with circle containing their class name. The class A constitutes the root node. The directed connections represent an *is-a* relationship between nodes. Therefore, the descendants of A are B, C and D.

However, as the evaluation contains more than only one data item, the different values need to be combined. Similar to the measure for multi-class classification, two common approaches exist. Firstly, the sum can be applied to all numerators and denominators and calculating the precision over all examples. But especially for the precision, one prediction in a false subtree can have a very high influence on the performance. For the hierarchy in Figure 36, a misclassification of an example from class J in C is weighted one third compared to a prediction of I although both miss the first hierarchy level. The second approach is calculating the measures on every single example and averaging the values afterwards. Nevertheless, the drawback is that it does not consider if the classifier missed the first level, but is also uncertain for deeper classes. For example, misclassifying C in B should be rated better than I.

Another, popular hierarchical metric focuses on the distance between classes [87]. The distance is determined by the number of links that are on the path between both classes. Furthermore, every false prediction is assigned to a contribution that defines the similarity of the prediction and the ground truth. For this contribution, an acceptable distance Dis_θ has to be specified by the user. If the distance between the prediction and label is smaller than the acceptable distance, the misclassification is considered to be close and, hence, contributes to the class. Otherwise, it leads to a negative contribution. To limit the maximum negative value for large hierarchies, the contribution is refined to be in the range of $[-1, 1]$. Formally, the contribution of a prediction C_p for the label C_t and the input $\mathbf{x}$ can be calculated by:

$$\begin{aligned} Con\left(\mathbf{x}, C_p\right) &= 1 - \frac{Dis\left(C_p, C_t\right)}{Dis_\theta} \\ RCon\left(\mathbf{x}, C_p\right) &= \min\left(1, \max\left(-1, Con\left(\mathbf{x}, C_p\right)\right)\right) \end{aligned} \tag{3.9}$$

For example, if the accepted distance is defined as $Dis_\theta = 3$, the prediction $C_p = H$ for an input assigned to the label $C_t = I$ results in a contribution of $Con(\mathbf{x}, H) = 1 - \frac{Dis(H,I)}{3} = 1 - \frac{2}{3} = \frac{1}{3}$. In contrast, if the prediction would have been $C_p = K$, the contribution is determined by $Con(\mathbf{x}, K) = 1 - \frac{Dis(K,I)}{3} = 1 - \frac{5}{3} = -\frac{2}{3}$. To get a final metric, the contributions are recorded and summed for all false positive and false negative predictions. With those parameters, the standard metrics for flat classification can be adjusted for the hierarchical class structure:

$$\begin{aligned} \text{Prec} &= \frac{\max\left(0, TP + FPCon + FNCon\right)}{TP + FP + FNCon} \\ \text{Rec} &= \frac{\max\left(0, TP + FPCon + FNCon\right)}{TP + FP + FPCon} \end{aligned} \tag{3.10}$$

However, the distance metric does not consider the depth of the single classes. Thus, the contribution of C and D is the same as H and I, although the first pair is much more crucial as it is on the first hierarchy level. To overcome this drawback, [8] proposes a depth-dependent distance metric that weights the edges by exponentially decreasing the value with the depth. The distance between two classes is specified by the summation of the edges' weights instead of counting. So, the contribution of deeper, nearby classes is greater compared to high-level misclassifications. Figure 37 visualizes an example of the metric.

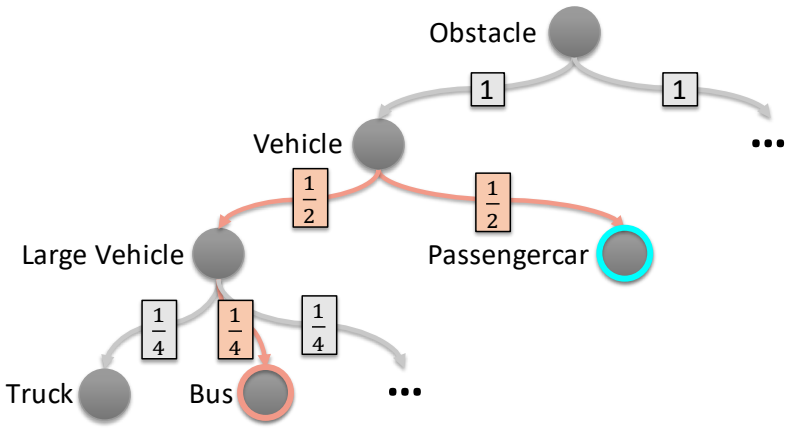


Figure 37: For a depth-dependent distance metric, the edges are weighted by an exponentially decreasing factor. In this example, a basis weight of 0.5 is chosen, and the class *passenger car* in blue is the ground truth label whereas the red shaped node *bus* was predicted. The distance between both classes is calculated by the sum of the edges' weights that form the path from *bus* to *passenger car* (highlighted in light red). Thus, the distance is determined by 1.25.

This new distance measure can be implemented in the contribution concept of equation 3.9. However, introducing weights depending on the depth of a node arises new

problems, as discussed in [14]. If the hierarchy is imbalanced, the distances might significantly vary as deeper classes naturally come along with greater distances. Thus, a misclassification in a subtree can have a greater distance than two classes on the highest level when using a factor greater than 0.5. Furthermore, if the hierarchy is not a tree, but a directed acyclic graph (DAG), then it is more difficult to define the *depth* of a node if it has multiple paths to the root.

Another problem is that the depth of a node does not necessarily correspond to the richness of information it holds. Considering the hierarchy of Figure 48, the class *nature* in the sub-tree *obstacle* is as specific as *terrain* for *ground*, but the depth of both is different. Hence, the weights could also be specified by the class similarity, known as semantics-based measure [87]. This similarity can be calculated in various ways, like describing each class with a feature vector and determine the distance between both. However, the selection of the similarity is mostly subjective, and the hierarchy itself should already map the class structure.

To select a suitable evaluation metric for the task of hierarchical classification for autonomous driving, the class structure needs to be reviewed (see Appendix B). First of all, all proposed hierarchies are trees so that there exists exactly one path from the root to every other node so that an explicit depth-dependent edge weighting is possible. However, the structures are significantly imbalanced as on the first hierarchy level, *sky* has no children at all, *ground* has between 6 and 13 with a depth of 2 and 3, and *object* has more than 25 classes in both standard and large-scale hierarchy. The maximal depth is limited to 6 with one additional attribute. Overall, the hierarchies include up to 65 different classes.

The metric should be suitable for object detection, and semantic segmentation as both tasks are evaluated in this theses. Therefore, the precision, recall and IoU measure need to be adjusted to the hierarchical classification. Due to the considerable imbalance, especially with a single class on the first level, an exponentially decreasing edge weight might be able to represent the semantic structure the best. Furthermore, as the hierarchical adjustments are performed on the confusion matrix's parameters, all flat metrics stay unchanged, and the performance of the network can be easier compared to a flat classifier. For the hierarchies in appendix B, a factor of 0.5 would lead to a minimal edge weight of 0.0625 that is assigned to, e.g. *rider* $\rightarrow$ *bicyclist*, whereas the distance between *passenger car* and *bus* is ten times higher (0.625). Hence, finding a suitable, acceptable distance with this factor is not easy as there is still a difference between *rider* and *bicyclist*, but misclassifications of large vehicles as regular cars should also contribute. To overcome this problem, a higher factor of 0.8 is chosen for all hierarchies resulting in a minimal weight of approximately 0.4096. An acceptable distance of 1.5 allows small correlations between nearby classes whereas a mistake on the first hierarchy level is not tolerated.

Still, a negative contribution would distort the measures and making it difficult to

compare to flat metrics as these do not include additional penalties for *very* bad misclassifications. Thus, the contribution is refined to the range of $[0, 1]$ resulting in the following adaptation of equation 3.9:

$$RCon\left(\mathbf{x}, C_p\right) = \min\left(1, \max\left(0, Con\left(\mathbf{x}, C_p\right)\right)\right) \tag{3.11}$$

Furthermore, instead of solely using the contribution for precision and recall as proposed by [87], all parameters of the confusion matrix are adapted to close predictions. The idea is that if the classifier predicts *rider* instead of *bicyclist* resulting in a contribution of $Con(\mathbf{x}, rider) = 1 - \frac{0.4096}{1.5} \approx 0.727$, the example is treated as 72.7% true positive and 27.3% false negative for the class *bicyclist*. Similar, the false positive value of *rider* is increased by 27.3% and the true negatives by 72.7%. Thus, the hierarchical confusion matrix's parameters can be formally defined as:

$$\begin{aligned} \overline{TP} &= TP + FNCon \\ \overline{TN} &= TN + FPCon \\ \overline{FP} &= FP - FPCon \\ \overline{FN} &= FN - FNCon \end{aligned} \tag{3.12}$$

Moreover, the hierarchical precision, recall and Intersection over Union measure are defined by:

$$\begin{aligned} \overline{Prec} &= \frac{\overline{TP}}{\overline{TP} + \overline{FP}} = \frac{TP + FNCon}{TP + FNCon + FP - FPCon} \\ \overline{Rec} &= \frac{\overline{TP}}{\overline{TP} + \overline{FN}} = \frac{TP + FNCon}{TP + FN} \\ \overline{IoU} &= \frac{\overline{TP}}{\overline{TP} + \overline{FP} + \overline{FN}} = \frac{TP + FNCon}{TP + FN + FP - FPCon} \end{aligned} \tag{3.13}$$

However, the metric still has some drawbacks. The depth-dependent distance does not always fit the semantic distance between classes. One example for this is *terrain* and *nature*. Although both are sometimes hard to distinguish as patches of grass belong to *terrain* and plants to *nature*, but small plants can be annotated as each of these classes, the metric distance is 4.24 and, thus, much greater than the accepted distance of 1.5. Nevertheless, this problem can also be solved by an improved class structure that considers all relations between classes, but this hierarchy would be a more complex DAG and goes beyond the scope of this thesis.

Besides the depth-dependent distance metric, the average distance for false positives and false negatives of each class are recorded to get an inside view of how far away the classifier actually was. Furthermore, the measures are also evaluated without the contribution to have an exact comparison between flat and hierarchical classification.

As the evaluation labels from datasets like Cityscapes [12] and Mapillary Vistas [62] are based on a flat, single-class classification, the metric needs to be adjusted to a multi-classification task. Therefore, if, e.g. the class label is H based on the hierarchy of Figure 36, the prediction is also evaluated on the ancestors F and B by using the same false positive/negative contribution as H. Still, the subclasses are subsumed from the perspective of an ancestor node, so that misclassification of I instead of H is a true positive for the parent F.

3.4. Metric Loss Function

The Intersection over Union metric handles imbalanced data by normalizing the score of a single class by the number of examples and taking the mean score of all classes as the final performance. Since the loss function has to handle imbalances as well, it can be considered to train on the metric itself. Very few literature [7, 67] exist dealing with the IoU metric as a loss function, especially when considering a hierarchical class taxonomy. Therefore, a novel approach to train a neural network on the Intersection over Union for hierarchical classification will be deepened in this section. The first part deals with the mathematical derivation of the IoU score as loss function, whereas the second presents methods to consider the hierarchical class structure for the IoU. Various weighting techniques for rare classes are discussed in the last subsection. All examples within this section are based on the mid-size hierarchy of Figure 48 in Appendix B.

3.4.1. Using IoU score as loss function

For semantic segmentation, the Intersection over Union (IoU) score represents a standard performance measure. As introduced in Section 3.3.1, the IoU is determined by the similarity between the predicted region and the ground truth over a set of images. Based on the confusion matrix, it is calculated by $IoU = \frac{TP}{P+FP}$.

One of the advantages of the IoU score is that it is regardless of the number of examples per class. The only part which is influenced by the dataset distribution is the proportion of false positives as it gets more likely to falsely predict a class with an increasing number of negative examples while the positive remain unchanged. Still, for rare classes, the number of negatives is significantly higher compared to standard categories, but it is also intended that the rate of false positives is lower for such classes.

However, the IoU score is a discrete, count-based measure whereas the outputs of a CNN are probabilities. The loss function needs to be globally differentiable as the network's parameters are optimized regarding the gradients of the single loss terms. So, the measure cannot be accurately determined but approximated with the output probabilities. A similar approach is proposed by [67] where the network is trained on object category segmentation labeling each pixel as being part of a given object or not. In this task, the distribution of positive and negative labels also are significantly imbalanced. Nevertheless, as in this thesis the classifier is based on a hierarchical structure, the loss

function has to be adapted to this concept. This is why the following paragraphs discuss the mathematical derivation of using the Intersection over Union score as loss function for hierarchical classification.

First of all, to be consistent with [67], the same mathematical notation is used for the derivation. Let V be the set of all pixels of all the images in the training set (in this case a batch) and $X \in [0,1]^n$ be the output of the network representing pixel probabilities over the set V for all n classes. $Y \in \{-1,0,1\}^n$ gives the ground-truth assignment for the set V whereas 1 represents labels where the class is correct, 0 where the class is false, and -1 where the class is ignored. C is a set constituting the particular class label for every pixel in V. Based on this notation, the metric parameters are approximated by:

$$\begin{aligned} P &= \sum_{v \in V} (Y_v == 1) \\ N &= \sum_{v \in V} (Y_v == 0) \\ TP &= \sum_{v \in V} X_v \cdot (Y_v == 1) \\ FP &= \sum_{v \in V} X_v \cdot (Y_v == 0) \end{aligned} \tag{3.14}$$

Note that FN and TN can be similarly calculated but neglected as these parameters are subsumed by P and N. The IoU score is approximated by applying TP, FP and P in equation 3.6:

$$IoU = \frac{TP}{P + FP} = \frac{\sum_{v \in V} X_v \cdot (Y_v == 1)}{\sum_{v \in V} (Y_v == 1) + \sum_{v \in V} X_v \cdot (Y_v == 0)} \tag{3.15}$$

This function is continuous and globally differentiable regarding X and, hence, to the outputs. Furthermore, the loss can be specified by applying a negative logarithm on the IoU: loss $= -\log(IoU)$. The logarithm has the benefit of an increasing loss for minimal values and corresponds to the log-likelihood. To optimize the internal parameters of the network, the gradients of the loss function concerning the outputs X need to be

determined. For a single prediction X_v, the gradients are calculated by:

$$\begin{aligned} -\frac{\partial \log(IoU)}{\partial X_v} &= -\frac{1}{IoU} \cdot \frac{\partial IoU}{\partial X_v} \\ &= -\frac{P+FP}{TP} \cdot \left((Y_v == 1) \cdot \frac{1}{P+FP} - (Y_v == 0) \cdot \frac{TP}{(P+FP)^2} \right) \\ \Rightarrow -\frac{\partial \log(IoU)}{\partial X_v} &= \begin{cases} -\frac{1}{TP} & \text{if } Y_v = 1 \\ \frac{1}{P+FP} & \text{if } Y_v = 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.16}$$

The gradients for predictions with a positive label are anti-proportional to the approximated true positives. This means that if the network performs well on recognizing the class, the gradients are smaller and, thus, the parameters are less adjusted. In contrast, if the desired output is 0, the gradients are anti-proportional to the number of positives plus the false positive predictions. Bad performance of the network leads to a high amount of false positives so that the gradients are small. Although one would expect that the gradients increase for worse predictions, the IoU actually exhibits greater changes regarding FP if only a few are misclassified. For example, the IoU for a prediction of $FP = 10$ and $TP = P = 5$ is $IoU = \frac{5}{5+10} \approx 33.3\%$. Reducing the FP score by 1 increases the IoU by about 2.3%. In comparison, if the score is $FP = 2$, a single less false prediction results in a rise of the IoU score from 71.4% to 83.3%. Thus, it is even more important that the network reduces its false positive predictions if it already performs quite well. Although the gradients appear counter-intuitive, experiments have demonstrated that applying a logarithm on the IoU loss function achieves significantly better results than without.

3.4.2. Adaption of IoU for hierarchical classification

The loss of a multi-class classifier is mostly an average over the losses of all classes. However, when using a hierarchical classification approach, the relationships between classes are ignored by a simple mean. If the network performs badly on the nodes of the first level, all descendants suffer under the false predictions as well. Thus, the IoU score has to be adjusted to the class structure.

As mentioned earlier, one of the benefits of the IoU is its invariance to class imbalances. However, when optimizing the IoU of the class *obstacle*, the imbalance of its children is not handled. Considering the distribution of only 2.56% positive labels for *vulnerable road user* compared to over 50% for the class *infrastructure*, it is much more likely that

the network achieves a high score on *obstacle* for pixels that are labeled as *infrastructure* than as *vulnerable road user*. To preserve the invariance to imbalances for descendants, the IoU for a class c needs to be calculated in two different manners:

$$IoU_c = \begin{cases} \dfrac{TP}{P + FP} & \text{if class } c \text{ has no subclasses} \\ \dfrac{1}{M} \cdot \displaystyle\sum_{c_s \prec c} \left(\dfrac{TP_{c_s}}{P_{c_s} + FP \cdot \left(\frac{P_{c_s}}{P}\right)} \right) & \text{otherwise} \end{cases} \tag{3.17}$$

M constitutes the number of descendants c_s of the class c, P_{c_s} the amount of positive labels for the subclass c_s, and TP_{c_s} the prediction of the class c for the labels of c_s. The above equation expresses that the IoU is a mean over conditional IoUs based on the descendants of a class. Note that only subclasses are considered for which at least one positive label exist: $P_{c_s} > 0$. The false positives are weighted proportionally to the number of positive labels for the subclass assuming a similar distribution of false predictions as true labels. Although the exact number of false positives for each class could be determined, it is not considered here as it requires a huge computational effort when operating on high-resolution images with more than 50 classes. In Section 3.4.3, a more extensive discussion of the equation shows that the false positives FP are independent to the subclasses and the conditional part is only necessary for true positives.

An alternative is applying the logarithm before averaging the conditional IoUs. This variation focuses more on the classes with low accuracy as the logarithm tends to infinity for zero. For a prediction with an IoU of 0.1 and 0.9 of two subclasses, the loss is calculated by $\frac{1}{2} \cdot (-\log 0.1 - \log 0.9) \approx \frac{1}{2} \cdot (2.3 + 0.1) = 1.2$. Thus, the loss for 0.1 is 23 times greater than for 0.9 leading to corresponding high gradients. However, such great weightings can cause instability within the network. If a class is represented by a single, small object in a batch which is not detected, the loss could explode although the classifier performs well on other descendants. This assumption is established by experiments in which the loss function with early applied logarithm resulted in a significantly worse performance compared to the reversed order of operations. Besides, the goal of the metric loss function is to optimize the *mean* Intersection over Union. Every class is considered equally regardless of the actual performance. Thus, the weighting in the loss function should be applied similarly.

Still, with the current approach, the loss function is disregarding the position of a class in the hierarchy. Especially base classes like *obstacle* are treated equally to leafs like *caravan* although the performance of the first class influences all its descendants. A simple idea of weighting classes corresponding to its subclasses is having losses along the whole path for each single class. So, a class's loss is defined by the sum of its losses

at each level:

$$\text{loss}_c = -\sum_{c \prec c_p} \log\left(IoU_{(c_p|c)}\right) \tag{3.18}$$

$IoU_{(c_p|c)}$ represents the IoU of ancestor c_p under the condition the prediction or label is assigned to c. However, this approach has two major drawbacks. Firstly, its loss shows the same instability as the early applied logarithm. Rare classes can cause great losses at levels on which the detection for other classes is quite well. Secondly, leafs at a shallow depth level, like the class *sky*, are weighted up to 40 times less than its neighbors *obstacle* and *ground* . So, the network might badly perform on *sky* generating many false positive predictions and tearing down the other classes. To prevent this, another weighting technique has to be developed. Therefore, the mean IoU calculation defined in Equation 3.17 can be extended by weighting the class loss by the number of *active* descendants. A leaf is called *active* if the batch contains at least one example of the class in its labels C. To reduce the imbalance at shallow levels, the weight is calculated by the square root of the number of active descendants:

$$\text{loss}_c = -\sqrt{\sum_{c_s \in C} \mathbb{1}\left((c_s \prec c) \wedge (P_{c_s} > 0)\right)} \cdot \log\left(IoU_c\right) \tag{3.19}$$

The square root is necessary to prevent too large weights. Considering the class *obstacle* for the label taxonomy in Figure 48 (Appendix B), the weight can become greater than 30 ignoring leaf classes like *pedestrian* with a weight of one. Nevertheless, the relationships between classes are not fully implemented yet, especially for the false positive. In a hierarchical structure, the goal is that misclassifications are detected by stopping the prediction at an earlier level. Currently, the network only gets punished for classes at levels where it has predicted incorrectly. Deeper descendants that are selected in succession are ignored although the network should classify all of them as false to stop the prediction. In addition, rare classes can suffer under too deep predictions as such false positives decrease the IoU score. The problem can be handled either by a top-down approach considering the predictions of all descendants for the approximation of the false positives' depth, or by a bottom-up approach propagating misclassifications of shallower levels to each descendant. The first method might not take the class imbalance into account as falsely predicting *truck* is less crucial for the IoU score than *caravan* but both are on the same level. In contrast, the bottom-up approach naturally deals with the imbalance by adding the extra false positives to the IoU score of the predicted class. Formally, the propagation for a class c is expressed by:

$$\widetilde{FP}_c = FP_c + \sum_{v \in V} \sum_{c \prec c_p} \left(\left(Y_{v,c_p} == 0\right) \cdot \gamma^{(|P(c_p,c)|-1)} \cdot \prod_{c_k \in P(c_p,c)} X_{v,c_k} \right) \tag{3.20}$$

$P\left(c_1, c_2\right)$ constitutes the set of classes on the path from c_1 to c_2, and γ a weight decay to

regularize the influence of previous levels on the class's false positives. For each ancestor c_p of c, the false positives are propagated by multiplying the probabilities of all classifiers between both nodes where the label of c_p is false (probability of 0). Note that no pixel is considered more than once as the masks of $\left(Y_{v,c_p} == 0\right)$ are exclusive. For instance, the pixels for which $(Y_{v,street} == 0)$ is true do not overlap with those from $(Y_{v,ground} == 0)$ as the first set is labeled as ignore (-1) if the ground truth does not contain the class *ground*. γ should be equals or less than one as the major misclassifications are made by a different classifier. To clarify the application of the approach, the propagated false positives $\widetilde{FP}$ of the class *road* are calculated by:

$$\begin{aligned}\widetilde{FP}_{road} = \sum_{v \in V} &(Y_{v,road} == 0) \cdot X_{v,road} \\ &+ (Y_{v,street} == 0) \cdot \gamma \cdot X_{v,street} \cdot X_{v,road} \\ &+ (Y_{v,ground} == 0) \cdot \gamma^2 \cdot X_{v,ground} \cdot X_{v,street} \cdot X_{v,road}\end{aligned} \tag{3.21}$$

The last two multiplication terms are the propagated false positives by *street* and *ground*. If the network predicts *road* for a pixel labeled with *sky* or similar, the last term of $(Y_{v,ground} == 0)$ is true and adds the misclassification to the false positive count of *road*.

Implementing a similar approach for true positives is not as easy because if any classifier on the path from the root node to the class does not predict to be true, the classification is not correct. The method for combining the predictions determines how the network will try to optimize its predictions. A multiplication of all probabilities, as it is applied for YOLO9000 [68], does not provide sufficient gradients and rewards. For example, if the network predicts 4 out of 5 classifiers correctly, but the last close to 0, the complete classification ends up in a probability of 0. Furthermore, a concatenation of 5 classifiers each with a prediction of 0.7 leads to a combined probability of only $0.7^5 = 0.168$ although the prediction probably is a true positive during the inference. Nevertheless, the class relationships are already taken into account by the mean IoU over all descendants as described in Equation 3.17. In the worst case, the true positives of the ancestor and the descendant share no pixels at all although both have an IoU score of 0.5. As the network constitutes a global classifier and all predictions influence each other, it is improbable that such a worst case scenario happens.

In conclusion, the hierarchical label taxonomy can be integrated into the loss function by using the mean Intersection over Union of all descendants for classes with children. As it is fundamental that the network performs on classes with multiple descendants the best, their loss is weighted by the number of active subclasses. The class relationships can be implemented by propagating false positives from each ancestor to all its children. To find the most suitable value for the decay parameter γ, Section 4.2.2 reviews experiments performed with different settings of this approach.

3.4.3. Weighting IoU for rare classes

Although the IoU is invariant to class imbalances, it can only be determined for a single batch and might miss out some rare classes. Furthermore, certain classes might be easier to learn than others, and thus, the network should concentrate on the hard classes as those provide the greatest chance for improvement. As presented in Section 3.2.3, a moving average filter over the losses or probabilities can help to find the optimal weights for a class set. However, the positive and negative predictions are weighted differently to balance the examples for a single class. Within this section, the weight factor for positive predictions is referred as α and the negative respectively as β, defined by:

$$\alpha = \frac{MAF_1}{MAF_0}, \quad \beta = \frac{MAF_0}{MAF_1} \tag{3.22}$$

The IoU loss subsumes both label types into a single score. To weight the loss of positive and negative predictions differently, the function has to be split into two parts. By applying the logarithm laws, the loss of the IoU for a leaf class can be simplified by:

$$\begin{aligned} \log(IoU) &= \log\left(\frac{TP}{P+FP}\right) \\ &= \log(TP) - \log\left(P+FP\right) \\ &\Rightarrow \alpha \cdot \log\left(TP\right) - \beta \cdot \log\left(P+FP\right) \end{aligned} \tag{3.23}$$

Thus, the losses of positive and negative predictions can be calculated and weighted independently. For classes with one or more descendant, the loss function is changed to:

$$\begin{aligned} \log(IoU) &= \log\left(\frac{1}{M} \cdot \sum_{c_s} \frac{TP_{c_s}}{P_{c_s} + FP \cdot \left(\frac{P_{c_s}}{P}\right)}\right) \\ &= \log\left(\frac{1}{M} \cdot \sum_{c_s} \frac{TP_{c_s}}{P_{c_s} \cdot \left(1 + \frac{FP}{P}\right)}\right) \\ &= \log\left(\frac{1}{1+\frac{FP}{P}} \cdot \frac{1}{M} \cdot \sum_{c_s} \frac{TP_{c_s}}{P_{c_s}}\right) \\ &= \log\left(\frac{1}{M} \cdot \sum_{c_s} \frac{TP_{c_s}}{P_{c_s}}\right) - \log\left(1 + \frac{FP}{P}\right) \\ &\Rightarrow \alpha \cdot \log\left(\frac{\sum_{c_s} \alpha_{c_s} \cdot \left(\frac{TP_{c_s}}{P_{c_s}}\right)}{\sum_{c_s} \alpha_{c_s}}\right) - \beta \cdot \log\left(1 + \frac{FP}{P}\right) \end{aligned} \tag{3.24}$$

α_c constitutes a label-dependent weight factor for the descendant c. As mentioned earlier in Section 3.4.2, in this form it gets obvious that only the loss function for true positives needs to handle the conditional IoUs, and the false positives are similarly calculated as for a class without descendants. To be formally consistent with the definition of the positive and negative loss function, the IoU score without subclasses is adapted to the same function as a class with a single descendant:

$$\begin{aligned} \log(IoU) &= \log(TP) - \log(P + FP) \\ &= \log(TP) - \log\left(1 + \frac{FP}{P}\right) - \log(P) \\ &= \log\left(\frac{TP}{P}\right) - \log\left(1 + \frac{FP}{P}\right) \end{aligned} \tag{3.25}$$

Label-dependent weight factors can also be introduced for false positives. The goal is to detect common confusions over several layers, like *sky* to *water*, and weight the class loss regarding to the mean performance on pixels assigned to certain labels. Therefore, the formula for the false positives in Equation 3.14 is changed to:

$$FP = \sum_{v \in V} \beta_{C_v} \cdot X_v \cdot (Y_v == 0) \tag{3.26}$$

The moving average filters measure the performance of each class. Still, the distribution of a batch can significantly differ from the whole dataset regarding which the prediction should be optimized. Especially, if a batch contains a minimal number of labels for a class, the true positive loss can easily blow up as the network might miss out the few pixels. To prevent such great losses and stabilize the training, a minimal number of labels is determined by the average amount in the training dataset. If a batch contains less than half of the average number of labels, the true positives are filled up with default predictions of 1 until the minimum label number is reached. Other default values might also be possible, but as greater objects are more important for the IoU score than small examples, a perfect prediction is implied ensuring stable training. Formally, the label adjustment can be defined by:

$$\begin{aligned} P_{min} &= \frac{1}{2} \cdot P_{avg} \\ P &= \begin{cases} P & P \geq P_{min} \\ P_{min} & \text{otherwise} \end{cases} \\ TP &= \begin{cases} TP & P \geq P_{min} \\ TP + (P_{min} - P) & \text{otherwise} \end{cases} \end{aligned} \tag{3.27}$$

The number of labels can also be adjusted for the false positives. As the desired performance for the false positives of a class depends on the ratio of positive and negative

labels, the weight can be fitted to the batch distribution. Thus, the number of labels for the false positive is regulated by the number of negative examples in a batch. Furthermore, even if there are very few negative examples, the loss cannot blow up because the gradients even decrease for a great number of false positives. The loss adaptation is expressed by:

$$\begin{aligned} P_{FP} &= P_{avg} \cdot \frac{N}{N_{avg}} \\ \Rightarrow \log\left(IoU\right) &= \alpha \cdot \log\left(\frac{TP}{P}\right) - \beta \cdot \log\left(1 + \frac{FP}{P_{FP}}\right) \end{aligned} \tag{3.28}$$

However, there is still a drawback for rare classes when calculating the false positives by adding up the probabilities for all labels. When considering the class *animal*, it has only 61 positive examples per image in average compared to over 11,000 negatives (see Table 7). Even a prediction of 0.01 for all negative examples results in an approximated false positive score of $0.01 \cdot 11,282 = 112.82$. This is nearly twice as high as the number of positive labels and suggests an IoU score of lower than 0.4 although none of the prediction might be a false positive. A reasonably lower prediction is due to the sigmoid function hardly possible as the gradients decrease to 0 for such changes. To overcome this problem, the false positives can be approximated by the distance between the prediction for the labeled class and its siblings that should have a lower probability. If the distance is lower than a certain threshold, the false positive is set to 0. Otherwise, the prediction is counted as a false positive weighted by the size of the distance. For a threshold δ, the new false positive approximation can be defined by:

$$FP = \sum_{v \in V} \beta_{C_v} \cdot (Y_v == 0) \cdot (\max\left(X_v - X_{v,Cv}, -\delta\right) + \delta) \tag{3.29}$$

Furthermore, the propagated false positives introduced in Equation 3.20 can also be adjusted to the distance measure. The probability threshold that is used for determining whether the prediction should go a level deeper or not is applied as a distance threshold δ for those classes that do not have a label. For the example of falsely predicting *road* instead of *sky*, this means that the false positive for *street* and *road* is measured by distance to the probability threshold, while *road* uses the prediction of *sky*.

A similar approach could be applied for true positives, but with a different motivation. From the perspective of the approximated IoU score, the gradients of a prediction of 0.1 are the same compared to those of 0.9 although the second is probably already a true positive during inference. To focus on scores that need to be improved, the true positives are approximated by the distance to the greatest probability of the class's sibling. Similar to the false positives, a threshold is defined that specifies the minimum distance over which a prediction is considered as a full true positive and the loss is set to zero. Similar

to Equation 3.29, the distance-based true positives are defined by:

$$TP = \sum_{v \in V} (Y_v == 1) \cdot \left(1 - \frac{1}{1+\delta} \cdot \left(\max\left(\max_{c_s \neq c} (X_{v,c_s}) - X_v, -\delta\right) + \delta\right)\right) \tag{3.30}$$

Besides, when training on the IoU score, it is essential that the batch represents the class distribution as accurately as possible. Furthermore, the loss is invariant to the number of pixels/examples, as only the proportion of true positives to positive labels is taken into account (similar to false positives). However, some classes like *caravan* or *trailer* have instances on only a few images so that some batches exclude rare classes when selecting the images randomly. Thus, dividing the dataset into multiple subsets as described in Section 3.2.1 is crucial for this task.

Overall, the Intersection over Union can be weighted in many different ways to optimize rare classes. To evaluate the effects of the proposed methods, Section 4.2.2 presents experiments on different parameter settings for the distance-based measures of Equation 3.29 and 3.30. The weighting of hard classes by moving average filters are used in every experiment as well as the limitation of minimum labels. Both have shown to stabilize the training and are essential for reasonable performance on an imbalanced dataset.

4. Experiments

To evaluate the proposed methods of Section 3, several experiments are implemented in the context of this thesis. Therefore, the first subsection reviews the particular experimental setup including the network architecture and implementation details of the approaches. Next, the results of the different training techniques are presented and compared to a flat classifier. The third subsection discusses the scalability of the hierarchical class structure by comparing the results of three different hierarchies. In the last part of this section, the relevance of thresholds and uncertainty detection is explained.

4.1. Experimental setup

This section describes the settings that are used across all experiments. Therefore, the first subsection reviews the network architecture, whereas the second part concentrates on implementation details regarding the training.

4.1.1. Network architecture

The hierarchical classification is tested on the task of semantic segmentation. Therefore, a fully convolutional network [57] is applied where the convolutional network consists of a GoogLeNet-v1 Inception architecture [89]. As the resolution of images from Cityscapes [12] and Mapillary Vistas [62] is much higher than from ImageNet [75] for which the network is originally designed, a max pooling operation with two subsequent inception modules is added to the end of the network, similarly to the approach of Uhrig *et al* [92]. Upscaling the features to a pixel-wise prediction is implemented by skip connections and deconvolution operations, as proposed in [57].

For training, the network's parameters are initialized by a pre-trained version on ImageNet and afterwards fine-tuned on the task of semantic segmentation using the described datasets of Section 3.2.1. To minimize the loss function, the Adam optimizer [43] is applied to improve the network's parameters after each iteration. The learning rate is set to $4 \cdot 10^{-5}$ for all experiments, and a weight decay of $2 \cdot 10^{-6}$ is introduced to regularize the range of the weights [27, 47]. Several augmentation methods are applied to the input image that alternate the RGB appearance without significantly influencing the labels, like, i.e. adding noise or flipping the image (labels are flipped accordingly). Augmentation helps the network to generalize as it is trained to be invariant to small changes [27]. However, as the color is a significant feature for special vehicles like police cars, no augmentation method is applied that fundamentally changes the colors, like, i.e. swapping the RGB channels.

4.1.2. Implementation details

The experiments are implemented using the TensorFlow framework [1] and executed on NVIDIA V100 GPUs. When applying the Intersection over Union as loss function, it

is important to train on a large number of images in parallel to accurately represent the dataset distribution. Hence, a multi-GPU training is developed where images are processed on different GPUs but contributing to the same loss function. Each GPU determines the approximated true and false positives for its images, and forward that information to the CPU that combines the parameters of all GPUs. After the loss is calculated, the gradients are estimated on each GPU separately but combined to a final update of the network's weights. For all experiments, a default configuration of 3 GPUs is applied, each processing two images. Experiments with a single GPU achieve significantly worse results (about 10% on mIoU), but also noticeably greater sizes (i.e. using 8 GPUs) have shown to improve less while increasing the time per iteration.

Furthermore, the resolution of the input is reduced to 1536×768 pixels during training. This enables training of 2 images on GPUs that only provide 12GB memory. The reduction is performed by a random down-scaling and cropping to make use of the whole image. Still, the network is evaluated on the original images of 2048×1024 pixels to compare the result to other approaches.

4.2. Hierarchical classifier for Semantic Segmentation

The hierarchical classification is evaluated on the task of semantic segmentation. The performance is measured by the mean Intersection over Union and the depth-dependent distance metric (see Section 3.3.2) of the 19 classes of the Cityscapes validation dataset. As a baseline, a flat classifier is trained in a similar experimental setting presented in the first subsection. Next, the various methods based on the metric loss function of Section 3.4 are discussed. The final subsection presents the results on object attributes.

4.2.1. Baseline

To compare a hierarchical classifier with an ordinary flat version, both have to be trained in a similar setting (network architecture, datasets, ...). Thus, a flat classifier is applied to the leaf classes of the standard hierarchy of Figure 48. However, the attributes cannot be converted so quickly as they are based on a multi-class approach and the flat classifier predicts a single class per pixel. To train the network on the special classes, new leafs are introduced that represent all possible combinations of the base class and the attribute *duty* occurring in the dataset. Overall, the classifier is trained on 32 classes including the 19 classes of the Cityscapes validation.

As a result, the network achieves a mean IoU score of 72.43%. The best recognized class is *road* with a score of 97.76%, whereas the worst classes, *rider* and *motorbike*, are detected with a IoU of approximately 54%. Also, the network performs poorly on rare classes that are not considered for the Cityscapes evaluation. For instance, the classes *trailer* and *caravan* achieve a score of only 5.38% and 11.06% respectively.

4.2.2. Evaluation of metric loss function

Various methods for training a neural network on hierarchical classification for semantic segmentation are proposed in Section 3. To test their capability, several experiments of different settings are implemented and performed.

The network is based on a hybrid approach of a global classifier and local classifiers per node (see Section 3.1.2). As label taxonomy, the mid-size hierarchy with 48 classes is selected providing examples of rare classes and the object attribute *duty* (see Appendix B, Figure 48). Also, label-dependent moving average filters as described in Section 3.2.3 are applied to all experiments.

In a first experiment, the hierarchical classifier is trained on the binary cross entropy. As described in Section 3.2.2, the cross-entropy is extended by a focal loss with the parameter setting of $\gamma = 1$ and $\alpha = 0.5$. During inference, a class for all sets of descendants need to be determined. To consider every single classification as equally important, the loss is averaged over these sets independent to the number of labels as leaf classes are less likely to occur frequently. For instance, the loss for the children of the class *overpass* is equally scaled to the subclasses of the node *ground*. By applying this loss function, the network achieves a mean IoU score of only 66.89%. The classifier performs poorly on classes where the number of positive and negative examples is significantly imbalanced. The class *trailer*, for example, has a IoU score of 0.0%.

In contrast, the metric loss function is developed to overcome such imbalances. However, applying this loss without any improvements proposed in Section 3.4.2 and 3.4.3, the classifier performs worse on the first level of the hierarchy. Different works have shown that combining multiple loss functions can be beneficial when training a deep neural network on various tasks [42, 6]. Thus, in further experiments, the metric loss function is simultaneously applied with a binary cross entropy. To equally consider the losses independently to their scale, the method of uncertainty weighting [42] is used that determines the weights of each loss based on its scale and noise magnitude. Besides, the MAF filters are shared among all loss functions.

In further experiments, various parameter settings, mostly for the metric loss function are tested for the combination of both losses. To structure and name the experiments, the following abbreviations are used:

H Identifying a hierarchical class structure.

IoU The default setting with the basic metric loss function and binary cross entropy as described before.

NWB - Non-weighted binary cross entropy. The loss is not averaged over the different sets of descendants, but is the mean over all pixels. Thus, the loss of a class depends on its number of labels. MAF weights are still applied.

WBL For the metric loss function, the classes are weighted by number of active leafs (see Section 3.4.2, Equation 3.19).

NL Combining both *NWB* and *WBL*

FD The metric loss function is extended by approximating the false positives by the distance to the prediction of correct class. The subsequent number constitutes the value of δ_{FP} (see Section 3.4.3, Equation 3.29).

C False positives are propagated to their descendants (C for contribution). The subsequent number constitutes the decay factor γ (see Section 3.4.2, Equation 3.20).

TD Similarly to *FD*, the true positives are approximated by the distance to the greatest false prediction. The subsequent number constitutes the value of δ_{TP} (see Section 3.4.3, Equation 3.30).

The experiments are evaluated on the IoU score ($\text{IoU}_e xact$) and the depth-dependent distance metric where the contributions of false positives and false negatives are considered for the IoU (IoU_{cont}). The performances of different experiments are summarized in the following table:

Abbreviation	NWB	WBL	δ_{FP}	γ	δ_{TP}	IoU_{exact}	IoU_{cont}
Baseline						72.43%	72.88%
H_IoU			-	-	-	64.81%	66.89%
H_NWB	✓		-	-	-	68.27%	70.06%
H_WBL		✓	-	-	-	68.14%	70.09%
H_NL	✓	✓	-	-	-	72.01%	73.70%
H_NL_FD075	✓	✓	0.75	-	-	72.37%	74.20%
H_NL_FD05	✓	✓	0.5	-	-	72.95%	74.75%
H_NL_FD05_C1	✓	✓	0.5	1.0	-	71.63%	74.59%
H_NL_FD05_C05	✓	✓	0.5	0.5	-	73.98%	76.25%
H_NL_FD075_C025	✓	✓	0.75	0.25	-	**74.24%**	**76.48%**
H_NL_FD075_C025_TD075	✓	✓	0.75	0.25	0.75	73.95%	76.26%
H_NL_FD05_C05_TD075	✓	✓	0.5	0.5	0.75	74.05%	76.34%

Table 4: The table shows the achieved results for different parameter settings. The baseline of the flat classifier is listed at the top. For the hierarchical classifiers, the additional rows represent the applied parameters. The highest scores are highlighted in bold font.

Combing both metric loss function and binary cross entropy seem to have an adverse effect first. The score is worse compared to the training of purely the cross entropy. However, when adding the *weighting by active leafs* (H_WBL) or changing the binary cross entropy to a non-weighted version (H_NWB), the exact and depth-dependent IoU score significantly increases by more than 3%. Both methods weight the losses for classes on a shallower hierarchy level higher, as those frequently occur in images and the detection of all descendants dependent on the ancestor. Moreover, when WBL and NWB are simultaneously used (H_NL), the network again experiences a substantial improvement of 3.7% for both scores almost reaching the score of the baseline.

Next, the variations of the metric loss function proposed in Section 3.4 are applied. Approximating the false positives by the distance to correct labeled predictions should initially help to reduce the false positives for classes with a large label imbalance. The parameter δ_{FP} constitutes the distance over which a prediction is ignored. For a value of $\delta_{FP} = 0.75$ (H_NL_FD075), the network slightly performs better. However, when using a lower value of $\delta_{FP} = 0.5$ (H_NL_FD05), the performance considerably increases to an IoU score of 72.95%. Even smaller values are not considered, as experiments have shown that the loss of false positives would be too small compared to the true positives to get a stable prediction.

Another proposed change of the metric loss function is propagating false positives from a node to all its descendants. When using a decay factor of $\gamma = 1$ (H_NL_FD05_C1) so that the losses for misclassifications at a higher level are not reduced for child nodes, the exact IoU score drops by 1.3%. However, when considering the depth-dependent distance metric, the score almost stays the same compared to the experiment proving that the predictions of the network are close to the actual label. A more in-depth analysis shows that a network trained without propagating the false positives to descendants is less likely to stop at an inner node. When weighing the false positives too high, the network only predicts a leaf class if it is very confident. Reducing the decay factor to $\gamma = 0.5$ (H_NL_FD05_C05) significantly outperforms the previous experiment H_NL_FD05 in both exact IoU score and depth-dependent contributions. Still, the number of false positives is increased by applying the propagation. Using a smaller decay factor of $\gamma = 0.25$ but a greater distance of $\delta_{FP} = 0.75$ reduces the false positives for rare classes where the ancestor frequently occurs. The corresponding experiment H_NL_FD075_C025 has a slightly better score of 74.24% for the exact IoU and 76.48% for the depth-dependent distance metric. The results of the different decay factors can also be visually compared, as illustrated in Figure 38.

Applying the distance based loss for true positives shows no considerable improvements and has slightly worse scores than H_NL_FD075_C025. In the experiment H_NL_FD05_C05_TD075, setting both γ and δ_{FP} to 0.5 indicates a 0.1% higher score for both metrics than the similar approach with the values 0.25 and 0.75. However, the scores for every experiment are based on a single training, so that several random choices like the order of the training images can also cause small noise in the results.

(a) $\gamma = 0$ (b) $\gamma = 1$ (c) $\gamma = 0.25$

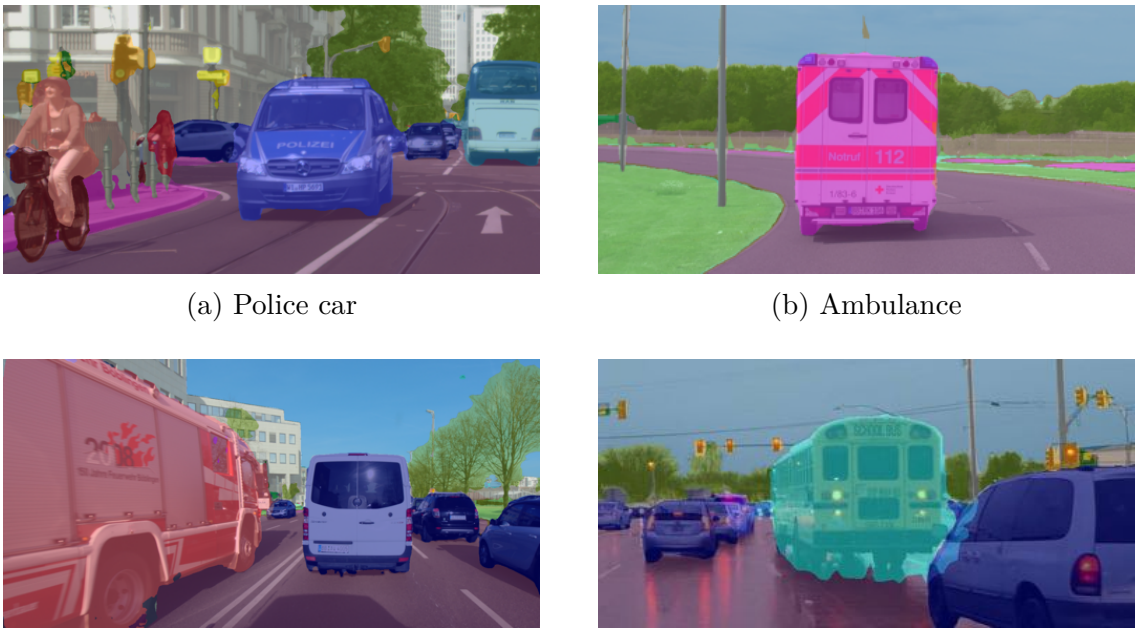
Figure 38: (a) Networks trained without propagating the false positives throughout the hierarchy rarely predict inner nodes. Thus, the boundaries between the bicyclist and the bicycle are inaccurate causing misclassifications of both classes. (b) High values of γ cause the network to stay at shallow levels of the hierarchy. The bicyclist is classified as *rider*, whereas the space in between the person and the bicycle is assigned to *vulnerable road user*. (c) If the decay factor is decreased, predictions more often end up in lead nodes, as the rider is recognized as *bicyclist*. For the pixels near the boundary of both classes, the network predicts their first common ancestor *obstacle* being even more accurate than the network trained with a high value of γ.

Overall, the hierarchical classifier shows a significant improvement compared to the flat baseline. The class for which the score increased the most is *train*. The performance of the baseline is about 56.82% whereas the best hierarchical classifier of experiment H_NL_FD075_C025 achieves a score of 73.69%. The reason for that might be the hierarchical class structure as *train* is a subclass of *large vehicle* and stand out from other child nodes by having significant characteristics like driving on rails. Besides, the score of the class *rider* is also noticeably improved by 7%. Most common classes like *passenger-car* show a similar score to the baseline as they are already recognized well.

4.2.3. Evaluation of object attributes

Attributes of objects are introduced in the hierarchy by virtual edges that can be connected to multiple nodes. To evaluate the performance of this approach, the attribute *duty* is included in the standard class structure of Figure 48 for several vehicles and humans. The training is based on a small, internal dataset of school buses and a relabeled subset of Cityscapes as described in Section 3.2.1. Due to the lack of labeled images, the evaluation is performed on a small set of coarse-labeled examples so that the results do not accurately reflect the detection quality. The Cityscapes validation set of 500 images constitute the instances of objects that do not have any particular duty. Furthermore, the visualization of the predictions can give a first glance at how good the network has learned to recognize the special vehicles.

Overall, the results prove that the most natural duty to detect is *school*. The network achieves an IoU score of about 98% on the validation dataset that is similar to the training examples. However, as the images of school buses are recorded with a different camera and location than all other images in the dataset, one might expect that the classifier has overfitted on this specific setting. Testing the detections on other recordings like Figure 39d, show that the network can recognize school buses in general.



(a) Police car

(b) Ambulance

(c) Fire truck

(d) School bus

Figure 39: (a) - (d) Examples of recognized vehicles with special duty. The predictions are visualized by combining the base class with the color of the attribute. Thus, a police car is shown in a light blue and school buses in turquoise. The first image is from the Cityscapes validation dataset, the other three from internal recordings.

Objects with other duties are detected slightly worse. Police cars constitute the groups on which the network performs second best with a score of approximately 80%. Examples like in Figure 39a are detected accurately independent to the specific class of the vehicle. With 117 images, the class *police* also has the most training examples compared to the other duties in the Cityscapes Special dataset (see Section 3.2.1). Especially the class *fire* suffers under the lack of variety in its 20 images. Whereas the fire truck in Figure 39c is detected quite well, vans or cars with the same duty are mostly misclassified as *medical* because the network has never seen such examples. Thus, the IoU score is only between 40% and 50%. Ambulances as in Figure 39b have a much better recall, but the false positives on *fire* significantly influence their performance on the IoU.

In conclusion, the approach of introducing object attributes within the class hierarchy shows great potential. When comparing the results to the flat classifier of Section 4.2.1, the hierarchical approach noticeably detects more special objects with the correct underlying base class like *medical car* versus *medical truck*. The greatest, remaining challenges derive from the dataset. The lack of examples can also be noticed for the classes *police* and *medical*, because most vehicles are recorded from a greater distance. Also, unseen vehicles that have a different duty like tow trucks might be misclassified. The problems are summarized in Figure 40.

(a) Large vehicles (b) Unseen duty

Figure 40: (a) The police car on the left is recognized as *passenger car* without duty. Large objects assigned to *police* and *medical* are less likely to be detected as the Cityscapes Special dataset mostly contains examples that are far away from the ego vehicle. (b) A siren is typical for many vehicles with a special duty. On the image from the Cityscapes validation dataset, the car of a tow service is mostly classified as *normal*, but near to the siren the prediction switches to *medical*.

Nevertheless, the dataset of school buses has proved that it is also possible to train on images where solely the desired object is labeled significantly simplifying the labeling task. Thus, extending the dataset by such examples will probably improve the performance to sufficient quality.

4.3. Testing scalability of hierarchical classification

A hierarchical class structure has the benefit that it can easily be expanded. However, most classifiers perform worse if several classes are added to the task. To test whether the performance also significantly decreases if a hierarchical classification is used, this section reviews experiments on a smaller and larger hierarchy which are described in Section 3.1.4 and visualized in Appendix B. The different level of detail of the three hierarchies is illustrated in Figure 41.

(a) Small-scale hierarchy

(b) Standard hierarchy

(c) Large-scale hierarchy

Figure 41: (a) The small-scale hierarchy solely contains the 19 Cityscapes classes with nine inner nodes. Thus, bicyclist and motorcyclist are both recognized as *rider*, and no object attributes are detected. Nevertheless, with a mean IoU score of 75.99%, it constitutes the best performing configuration on Cityscapes. (c) The standard hierarchy extends the small version by the object attribute *duty* and six additional classes from Mapillary Vistas like distinguishing between bicyclist and motorcyclist. The police car on the left image is therefore recognized with its corresponding duty. (c) The deepest level of detail provides the large-scale hierarchy. Especially the class *road* is extended by adding six new leaf classes and two inner nodes as descendants. On the example images, the subclasses *lane markings general* (white) and *manholes* (dark blue) can be seen. Also, curbs are detected as a subclass of *sidewalk* (visualized in grey).

4.3.1. Small Cityscapes hierarchy

The small-scaled hierarchy is based on the 19 classes on which Cityscapes is evaluated, and nine inner nodes to form the hierarchical class structure. With 28 classes in summary, it has approximately half as many nodes as the hierarchy used for experiments in Section 4.2.2. The settings regarding network architecture, datasets and training parameters, stay the same for experiments with the smaller hierarchy. The metric loss function is designed as constituted in best performing experiment H_NL_FD075_C025. Figure 41a shows two example predictions of this experiment.

The small hierarchy outperforms all previous experiments and achieves a mean IoU score of 75.99%. The most significant improvements appear in subclasses of *large vehicle*. The score of *train* rises by 2%, *truck* by 3% and *bus* by 4.5%. However, when comparing the IoU value of the ancestor, the score only slightly improves by 0.8%. Hence, the rare classes *caravan* and *trailer* which are not included in the small hierarchy, seem to have a significant effect on the performance of its sibling nodes. This is supported by the fact that the contribution of false positives and false negatives is noticeably higher for descendants in the standard hierarchy. A higher contribution indicates that the misclassification occurs on a deeper level, or the prediction stops at an ancestor. Thus, when adding rare classes, the network is more likely to create uncertain predictions, also for common classes.

Another significant improvement can be experienced for the class *wall* for which the exact IoU score is increased from 52.14% to 57.98%. This might derive from the additional inner node *barrier* that summarizes *wall* and *fence*. In the standard hierarchy, both classes are siblings of the frequently appearing class *building* introducing a strong imbalance of positive and negative examples. The additional inner node stabilizes the ratio as it subsumes both labels of *fence* and *wall* to compare to *building*.

However, there are also some classes that are less affected by the alternated class structure. All nodes on the first level have an almost unchanged score with a maximum difference of 0.1%. Moreover, classes like *passenger car* and *building* on which the standard hierarchy already achieved sufficient results of more than 90% are only slightly improved. One reason for this is that it is even harder to increase the absolute score for such classes, as its false positives are affected by the predictions of other classes, and the ground truth also contains some mistakes [12]. However, another possible factor is that the method *weighting by active leafs* of the IoU score is significantly influenced by the hierarchical structure. Thus, the same classifiers might be weighted differently in the small and standard hierarchy.

4.3.2. Large-scale Mapillary hierarchy

Mapillary Vistas is based on a label hierarchy with more than 60 classes. To use this detailed information, the standard hierarchy is extended by several subclasses. For in-

stance, the class *road* is divided into *marking*, *hole*, *bike lane* and *car lane*. Furthermore, markings and holes have each two additional descendants. To test whether the network can handle several rare classes at the same time, classes of small objects with a rectangular shape are subsumed in a new infrastructure type, called *box infrastructure*. Overall, the large-scale hierarchy consists of 69 classes. Note that a hierarchical prediction of over 9000 classes like in [68] is not practicable for semantic segmentation on full-resolution images of Cityscapes, as this results in an output of several Gigabytes size. For experiments, the same settings as for the small and standard hierarchy are applied, but the number of parallel GPUs is increased to 8. The time needed for a full training considerably rises as more classes need to be evaluated while using multiple GPUs synchronously additionally slows down. Thus, the training was stopped after 500,000 iterations although the performance might still slightly improve afterwards.

Using the large-scale hierarchy, the network achieves a mean IoU score of 72.40% on Cityscapes. Example predictions are illustrated in Figure 41c. The network performs similarly to the baseline of the flat classifier but recognizes about 30 more classes. However, compared to the standard hierarchy, the score drops by 1.76%. Some significant drawbacks include a 3.4% smaller value for the class *traffic light*. This can be traced back to the shifted weights due to the additional class *ad sign* as well as the two new descendants of *traffic sign*. Also, the IoU score of the class *rider* decreases by 3% which might also be related to the enlarged tree imbalance. Compared to the standard hierarchy, the large-scale version mainly includes more classes for *infrastructure* and *street*. Other classes might be discriminated due to the method of *weighting by active leafs*.

Besides, the new classes added to the hierarchy are detected quite reasonably. For instance, the network achieves an IoU score of over 90% on the class *marking*. Furthermore, rare classes like *animal* and *buggy* have a much lower score of 15% to 20%. Regarding the depth-dependent distance metric, contributions for close misclassifiations show significant improvements for the scores of uncommon classes. *Trailer* has an exact IoU score of only 12.72%, but is nearly doubled when taking the contributions into account although a prediction of *passenger car* is not considered to be close.

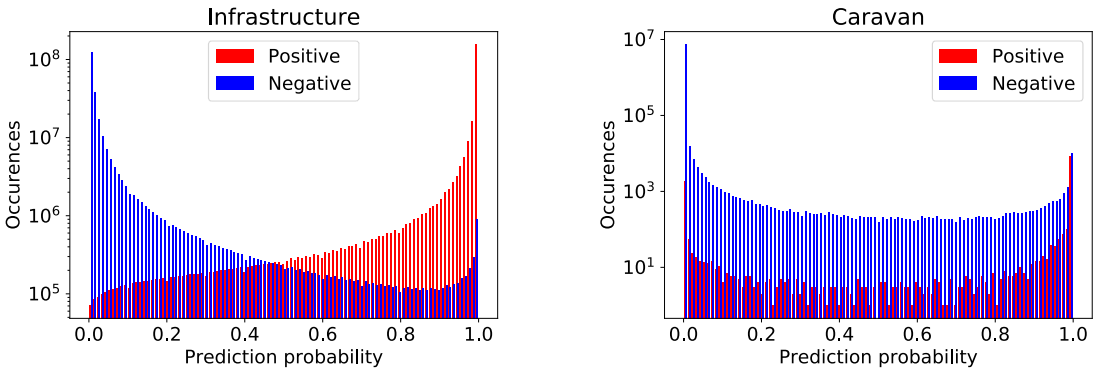
In conclusion, the larger the hierarchy is, the more uncertain predictions the network might generate, especially if several rare classes are included. The experiment proves that it is possible to train a classifier on a large number of categories simultaneously when using multiple datasets with different label sets. A drop of 1% to 2% regarding the IoU score can almost be neglected when considering that 30 new object classes are added. This method can be helpful to build up a modular dataset where new classes can easily be included by providing sufficient examples with the required level of detail. Other, already labeled data does not need to be reviewed if the new class can be expressed by a *is-a* relationship or is ignored in the previous datasets.

4.4. Threshold adaptation

A critical aspect of hierarchical classification is that uncertain predictions can end at any inner node of the class structure. The most common approach to detect uncertainty is by specifying a threshold over which the confidence of a prediction needs to be [78]. Finding the optimal thresholds for each classifier by hand is hardly possible as it depends on how the network has optimized its internal parameters. To automate the process, a test dataset is used which is processed by the network. Based on these predictions, the thresholds are optimized to gain the best performance for each classifier. This technique referred to as *threshold adaptation* in the context of the thesis, is further presented and applied in the first subsection. In the second part, more complex approaches are discussed that take the class relationships into account.

4.4.1. Evaluating prediction distribution

Automatically determining the optimal set of thresholds to detect uncertain predictions is the aim of the threshold adaptation. Firstly, the network processes a set of images for which the ground truth is known. The validation datasets of Cityscapes and Mapillary Vistas are used as those constitute new, unseen images. For every classifier, a distribution of the predicted confidences on the dataset is determined whereas it is distinguished between examples for which the label is positive or negative. Visualizations of such distributions are shown in Figure 42.



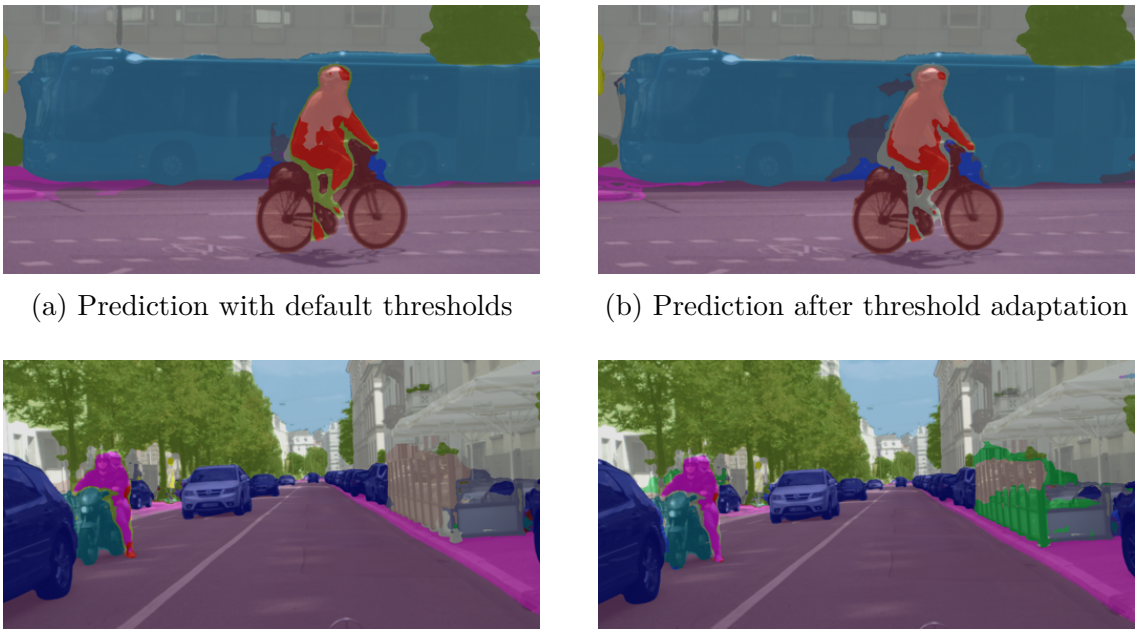
(a) Class *infrastructure* (b) Class *caravan*

Figure 42: (a) - (b) The bar charts summarize the distribution of the network's predictions for the classes *infrastructure* and *caravan*. The examples that are labeled positive are visualized in red and negatives in blue. The x-axis shows the probability that is predicted by the network in the range of 0 to 1 with a step size of 0.01. The height of a bar constitutes the number of examples that are assigned to a certain probability in a logarithm scale.

The optimal distribution would be where the predictions for all positive examples are 1.0 while all negatives are assigned to 0.0. As the network occasionally misclassifies an example, the predictions of both subsets overlap and are spread over the probabilities. When applying a threshold to this distribution, the confusion matrix parameters for the classifier performance can be visually determined. All values that are assigned to a negative label (blue) and are smaller than the thresholds (being left to the value) constitute the true negatives. The false negatives are therefore the values with positive labels on the left. The values greater than the threshold (being right to the value) split into true positive if the assigned label is positive, and false positives for those with negative labels. As the metric for semantic segmentation relies on the Intersection over Union, the measure is also applied here for various thresholds. The threshold which optimizes the IoU score is chosen as the final threshold of the classifier. For instance, the optimal value for the distribution of the classifier *infrastructure* in Figure 42a is about 0.45.

However, the number of examples that a classifier needs to categorize depends on its ancestor. If the parent node assigns a lower probability than the threshold to a pixel, then the prediction of the descendant can be ignored. Thus, the thresholds are determined in a top-down order of the hierarchy starting with the nodes on the first level. After i.e. the threshold for *obstacle* is determined, only the predictions where the probability of *obstacle* is actually greater than the specified threshold are used to set up the distribution of its child nodes *nature* and *vehicle*. So, all true and false negatives of a node are ignored for the threshold adaptation of its descendants as the predictions of those are not considered during inference if the ancestor predicts false.

For the experiments in Section 4.2.2, a default threshold of 0.5 for all classifiers is applied. When applying the threshold adaptation on the network of the experiment (0027), the IoU score is improved from 74.24% to 74.73% and the depth-dependent metric from 76.48% to 77.22% on Cityscapes validation. Especially the score of rare classes like *caravan* and *fire* climbs by 20% to 30%. The increase can be mainly traced back to the reduction of false positives that are less likely to occur in leaf classes. A comparison of predictions before and after applying the threshold adaptation is shown in Figure 43. However, the validation and the threshold adaptation are both performed on the Cityscapes validation dataset so that the thresholds are specifically optimized for this dataset. Experiments using, i.e. Mapillary Vistas for the threshold adaptation have shown moderate success as the images significantly differ from Cityscapes. Thus, the optimal solution would be to have another subset of Cityscapes that is exclusive to the validation set to perform the threshold adaptation on.



(a) Prediction with default thresholds

(b) Prediction after threshold adaptation

(c) Prediction with default thresholds

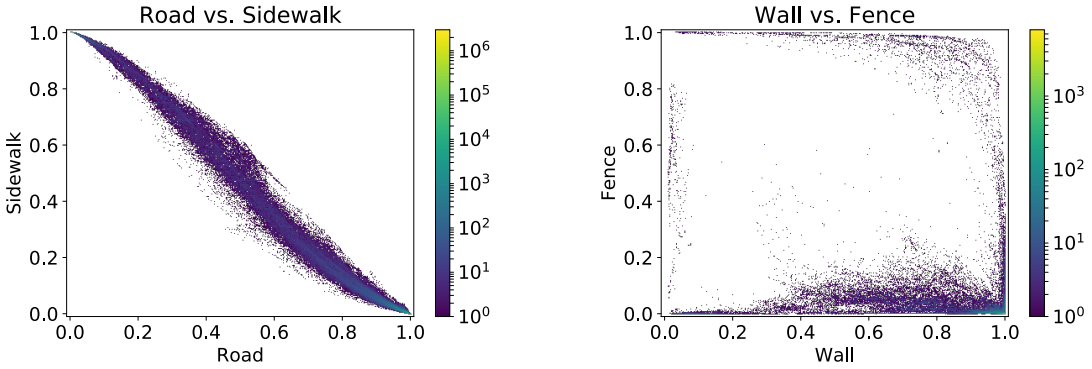
(d) Prediction after threshold adaptation

Figure 43: (a) - (b) The threshold adaptation helps to distinguish between uncertain and confident predictions. The space between the rider and the bicycle is previously classified as *vulnerable road user*, but changed to *obstacle* by applying the adjusted thresholds. However, the part of the bus near to the rider is assigned to *large vehicle* instead of staying at *bus*. (c) - (d) For the glass facade on the right, no leaf class is clearly specified but it is a infrastructure with horizontal alignment. By adapting the thresholds, the predictions of *wall* and *fence* are partially moved to its ancestor *elongated infrastructure* that might fit the best.

4.4.2. Class correlations

When applying local classifiers for each node, the predicted probabilities of sibling classes are independent. Thus, multiple nodes can simultaneously have a score which is greater than the specified threshold. A common method to select a class in such a situation is taking the node with the highest probability [17]. But especially rare classes have very low thresholds of i.e. 0.15 for *fire* whereas for its sibling node *normal* a threshold of 0.9 is specified. Therefore, a prediction of 0.9 from the classifier *fire* might be more informative than the same from *normal*. To check this hypothesis, the correlation of sibling classes based on their predictions is examined here.

Similar to the threshold adaptation, the network is evaluated on a validation dataset, and its predictions are plotted in a bar chart. However, the probabilities are shown regarding the predictions of a sibling node. The corresponding two-dimensional diagrams illustrate the correlation between the predictions of both classifiers (see Figure 44).



(a) *Road* vs. *sidewalk*

(b) *Wall* vs. *fence*

Figure 44: The two-dimensional bar charts visualize the distribution of predictions for two sibling nodes. The classifiers are evaluated on examples where the first mentioned class is assigned to a positive label. The x- and y-axis illustrate the predicted probability of both classes, so that a prediction of 1.0 for the correct class and 0.0 for the sibling is ordered in the bottom right corner. The color of the data points determines how often the combination of the class probabilities occurs (legend on the right). White spaces constitute no occurrence at all. (a) The diagrams shows the predicted probabilities of the classes *road* and *sidewalk* for positive examples of *road*. Comparing the predictions of both sibling nodes manifests that although the classifiers are trained independently, the correlation within the network typically leads to a prediction where the sum of the probabilities of both classes is close to 1. Most pixels are correctly classified shown in the bottom right corner. (b) The predictions of the classes *wall* and *fence* for positive examples of *wall* are analyzed in the diagram on the right. Both classifiers seem to be independent as sometimes *wall* and *fence* predict a high score close to 1 simultaneously (upper right corner). Still, the main proportion of the examples is correctly classified as *wall* while the classifier *fence* predicts a low score.

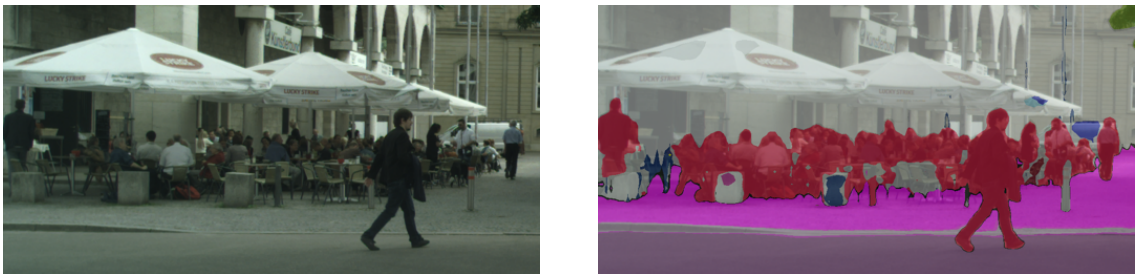
When considering a class that has a single sibling, it gets obvious that one classifier mainly learns to invert the other. Figure 44a illustrates the chart for the classes *road* and *sidewalk* for which different optimal thresholds are determined (0.6 and 0.3 respectively). Most data points are close to the diagonal between the upper left and bottom right corner where the sum of both predictions is 1. Although both classifiers are independently trained, they behave like a softmax function is applied to the network's output. The chart for the examples with positive labels of the class *sidewalk* is very similar to the chart for *road*.

Other classes that have more than one sibling show different behavior. In Figure 44b, the class *wall* and *fence* are compared. The chart proves that for some data points, both

classifiers predict a high score close to 1. However, the chart for the positive examples of *fence* shows similar data points so that no clear decision can be made to which class the predictions belong. Furthermore, experiments with different treatments of such cases have proved that in the current setting, selecting the classifier with the highest probability achieves the best results.

Nevertheless, next to applying thresholds on the predictions, there are also more complex approaches to find the path through the class hierarchy [5, 17, 94]. Mostly, they are based on two ideas: positive predictions of a node influences all its ancestors, whereas descendants consider negative predictions of their parent nodes. To see how the class selections might change by using such approaches, Figure 45 visualizes an example where the prediction stopped at the inner node *obstacle* or *infrastructure* for chairs that belong to the class *bench*. The parent class *box infrastructure* predicts a probability that is below the specified threshold although the classifier *bench* has high confidence on that pixels. Still, other leaf classes like *building* and *fence* have similar predictions. Hence, by considering the probabilities of the descendants for each node, both *box infrastructure* and *elongated infrastructure* would increase their scores whereas the second has a higher probability for multiple child nodes. Probably, either *building* or *fence* would be selected as the improved prediction still missing the correct class *bench*. Also for other examples, the classifiers seem to predict random scores if the ancestor has a very low probability.

In conclusion, selecting the correct class within the hierarchy remains a challenging task. Changes at this part of the inference have considerable influence on the network's performance and are therefore very important. However, the training might have to be adjusted so that classifiers of nodes in a different subtree of the label are not ignored, but pushed to be 0. Also, dropping out classes during training to introduce labels of inner nodes where all descendants are 0 and not ignored, can help the network to learn uncertainty.



(a) Original test image

(b) Classifier *box infrastructure*

(c) Classifier *bench*

(d) Classifier *box infrastructure*

(e) Classifier *building*

(f) Classifier *fence*

Figure 45: (a) The test image of Cityscapes [12] contains people sitting on chairs. In the label policy of Mapillary Vistas, these objects belong to the class *bench*. (b) The network is uncertain what class the chairs belong to and predicts *obstacle* or *infrastructure*. Color enconding according to Figure 49. (c) - (f) The predictions of the classifiers are overlayed on the test image. The color intensity specifies the confidence of the classifier. The chairs are recognized on the leaf class *bench*, but its ancestor *box infrastructure* has a low score over the whole image patch. However, also other leaf classes show high confidence like *building* and *fence*.

5. Outlook

Hierarchical classification has shown great success for the task of semantic segmentation. Compared to flat classifiers, networks trained on hierarchical class structures enable uncertain predictions while being as specific as possible. Also, they outperform flat versions on standard metrics like the Intersection over Union. This performance is facilitated by the extensive research on the training methods proposed in Section 3 that consider class relationships and dataset imbalances for the loss function. Nevertheless, this thesis only constitutes a first step in the vast, promising field of hierarchical classification for computer vision in the context of autonomous driving. There are many open challenges, but also new possibilities that are enabled by the concept of a hierarchy. In this section, five of these topics should be discussed to show possible future work.

Dataset modularity One of the significant benefits of a hierarchical class structure is its modularity. New classes can be added as subclasses or attributes without requiring that existing datasets are relabeled. This enables to build a modular dataset that consists of multiple subsets each containing different labels. For example, to detect whether a car has the turn signal activated or not, it is sufficient for the hierarchy to provide a small dataset of objects with different states of the turn signal. Previous datasets do not need to be changed although they also include objects that possess this attribute. Moreover, the additional classes might either be learned by a dataset similar to the school bus where only the regarded object is labeled, or an existing dataset is expanded by this information similarly to Cityscapes Special. This reduced the label time enormously without showing any apparent drawbacks.

Once the network is trained, it can detect all classes independent of which dataset the image comes from. If the network's performance is good enough, it can be used to relabel existing data with subclasses, like a network trained on the large-scale hierarchy can recognize lane markings and curbs in Cityscapes. For this task, a more complex network architecture than the GoogLeNet is necessary as the accuracy is much more critical than the runtime. One of the leading models on the Cityscapes Benchmark [13] is DeepLabv3+ [9] which applies a deeper, modified version of the Xception network [11]. In first experiments, the DeepLab model achieved better results on the large-scale hierarchy than the GoogLeNet based on the small-scale class structure. Thus, implementing such a model can help to generate more detailed labels on high-level datasets and use these to train shallower networks like the GoogLeNet with more examples.

However, there are also some limitations. When combining multiple, heterogeneous datasets with different annotation types, it is essential that the label definitions fit for each class. This is not always the case for the datasets Cityscapes [12] and Mapillary Vistas [62]. For instance, the class *rail track* includes in Mapillary Vistas the tracks which are embedded in a conventional roadblock [63]. In contrast, Cityscapes only considers rail tracks that are not drivable by cars. Otherwise, the part of the ground is

labeled as *road*. Training on both datasets in parallel probably causes a high loss as the same image would be labeled differently in Cityscapes and Mapillary Vistas. This is why the class *rail track* is excluded from the deployed hierarchies in Section 3.1.4. Alternatively, the class can also be ignored in one of the datasets having the drawback of losing many examples for the class. Thus, if such a class is relevant for autonomous driving and a single dataset provides few examples, there must be developed a technique for handling these inconsistencies to use all labels.

Hierarchical metric Reviewing the metrics for hierarchical classification, the depth-dependent distance metric might not be the best choice. The accepted distance, as well as the edges' weights, are handcrafted parameters. Selecting different values significantly influences the scores, so that only networks evaluated on the same parameter setting can be fairly compared. Furthermore, the decision of which distance between prediction and ground truth is acceptable can rarely be answered objectively. For the task of autonomous driving, this parameter might be determined by an expert who defines a value based on extensive research. Another challenge is that an edge's weight cannot be solely determined by its depth. For example, in case of the large-scale hierarchy, a confusion of *trash bin* and *mailbox* is assigned to the same distance as *bus* and *truck* although the semantic similarity might differ. In contrast, a metric purely resting on semantics is also not practicable as deeper classes require more correct classifications and are therefore more difficult to predict than shallower nodes.

Besides, the metric should be suitable to compare hierarchies for the same leaf nodes but different inner structures. The benefit of a fine-grained hierarchy is that the network can localize an uncertain prediction as specific as possible. Still, more internal nodes require more classifiers that need to be trained, increasing the complexity of the network. Overall, many challenges are remaining for the metric. When evaluating a hierarchical classifier, it has to be considered how valuable a prediction of an ancestor or a sibling is. This might be specific to the application but needs to be scalable as well. The importance of a metric should not be neglected as, without an evaluation method, it can not be determined how a classifier performs and whether a new approach improves the predictions or not.

Optimization techniques for rare classes One of the critical aspects of this thesis is to focus on classes that do not occur frequently. Therefore, various methods are developed and applied including a new weighting technique based on moving average filters (see Section 3.2.3) and a metric loss function of the Intersection over Union (see Section 3.4). Even if this improved the training, the performance on very rare classes often is much worse than on frequent categories. Currently, the moving average filters are used to compare the scores of positive and negative examples for each class independently. However, to focus on rare classes, the weight factors can be determined by considering

the mean predictions of other classes. Two problems arise from this idea that need to be handled. First of all, the performance of shallower nodes with several subnodes is more crucial than for a leaf node. Although applying the method of *weighting by active leafs* ensures a higher default weight for parent classes, the values provided by the moving average filters can be much greater exceeding the previous weights. Furthermore, the mean prediction does not always give a reasonable estimation of the classifier performance. During validation, the measure is the Intersection over Union. Thus, the discrete number of false and true positives can be counted on each training batch and used as input for the moving average filters. First experiments have shown that such weights can almost explode for rare classes. For instance, the class *animal* is assigned to a negative weight of over 200 in the first thousand iterations and slowly decreases. Even reducing high weights by applying the square root resulted in significantly worse performances on both the 19 Cityscapes and other rare classes. Thus, future work has to deal with the development of an improved weighting method that approximates the evaluation performance better than a simple mean prediction.

Directed Acyclic Graphs The proposed methods for hierarchical classification in Section 3 focus on the application on tree class structures. However, by allowing multiple parents for a node, more complex relationships can be implemented. The classes *nature* and *terrain* share significant features but differ in the first level of the hierarchy so that both are located in completely different subtrees of the class structure. Also, when looking at the predictions of the classifiers, both have a high score on the same pixels. Thus, combining the classes into a single node *natural* having the ancestors *ground* and *obstacle* reduces the size of the hierarchy and simplifies the learning process. In this case, only one of the parent nodes can be correct. In contrast, directed acyclic graphs can also be created where the right ancestor is not specified. An amphibian vehicle simultaneously is a boat and car, but the network might decide which ancestor it selects.

Especially for those type of DAGs, evaluating the classifier is more difficult. Multiple paths of different length might lead to the correct class. Hence, the distance between prediction and ground truth is not explicitly specified and represents another challenge for the hierarchical metric. Furthermore, the loss function needs to be reviewed when the correct ancestor is not stated. When an amphibian vehicle is driving on the road, the network probably classifies it as a ground vehicle and neglects the second ancestor boat. The prediction would be reversed if the amphibian occurs swimming. Thus, there might be a better training method than optimizing the network to predict a high score for all ancestors of a node. Developing such a technique should be addressed by future work if more complex class structures than a tree are applied.

Open world problem One of the hardest, remaining challenge is how to learn to be uncertain if an object is detected that does not fit to any known leaf class. Unknown objects are not covered by the training dataset and are therefore out of the represented

distribution. In the context of autonomous driving, such situations can frequently occur. For instance, existing datasets like Cityscapes [12] and Mapillary Vistas [62] already include some objects that could not be unambiguously assigned to a predefined class (see Figure 46). Still, the detection system needs to handle unknown objects by providing the most specific information on which it is certain.

(a) Train on street (b) Electric wheelchair

Figure 46: (a) In Cityscapes, a train on tires is labeled as a bus. Still, the bicycle taxi in the background constitutes a ridable vehicle, but significantly differs from other bikes. (b) An electric wheelchair for elder persons is recorded in Mapillary Vistas. However, the vehicle is neither a motorcycle nor a bicycle as it probably drives much slower and behaves differently.

Typically, deep neural networks are highly confident in their predictions regardless of the example being in- or out-of-distribution [33, 51]. However, hierarchical classifiers have the benefit that they can predict the closest parent class if the classifier is uncertain about the object category. Experiments of Section 4 have shown that adapted thresholds and hierarchical loss functions can push the network to make use of uncertainty.

Nevertheless, there also are more advanced methods like the *Dual Accuracy Trade-off Search* (DARTS) [16] or *leaving-one-out* [52] to explicitly train uncertain predictions and/or to detect novel objects. A crucial aspect of training this behavior is finding the best trade-off between accuracy and specificity of the predictions. If the network only selects classes with very high confidence, the inference path stops at a shallow level so that the number of false positives of deeper classes is reduced, but the predictions are very unspecific. The best trade-off might be defined by the user, or can also be learned during training.

Due to the time limitation, approaches for dealing with novel objects have not been tested for the application of understanding urban street scenes yet. As Mapillary Vistas already supplies some labels for out-of-distribution examples, first experiments might be implemented on the classes *other vehicle* and *other rider*. Novelty detection extends hierarchical classifiers by the possibility to solve the open world problem dealing with all possible objects that can occur in front of an autonomous vehicle.

6. Conclusion

Understanding complex urban street scenes is a challenging task for autonomous driving. Mostly, convolutional neural networks are applied to process camera images and localize various objects including pedestrians and other vehicles. One conventional method for detecting objects in images is semantic segmentation where every pixel is assigned to a class out of a predefined set. To select the correct category, the highest prediction among the class set is determined by compared all classes against each other. When adding a new class that is a subset of an already existing category like it is the case for *truck* and *fire truck*, the one-against-all scheme is inefficient and unstable. Furthermore, in previously labeled datasets, the subclass might be subsumed by the ancestor so that training on all datasets without expensive relabeling is not possible. To address this problem, the class structure can be ordered in a hierarchy implementing relationships and similarities. Hence, this thesis presents an approach to apply hierarchical classification for the task of semantic segmentation in the context of autonomous driving.

To simplify the architecture, the network simultaneously predicts a score for every class similarly to a flat classifier. Instead of applying a softmax, independent classifiers are used to obtain the classes' probabilities. The final prediction is determined by selecting the node with the highest score on the first level of the hierarchy. Afterwards, the search is recursively continued with the descendants of the selected class until either a leaf is reached or the scores of all classes are lower than a predefined threshold. A great challenge for which a hierarchical class structure might constitute an improvement, is learning rare classes. As a node is only compared to its siblings, the ratio of positive and negative examples for a class is significantly more balanced than for a flat classifier. Also, moving average filters are applied measuring the current performance to weight the loss for each class. A novel metric loss function is developed based on a hierarchical Intersection over Union (IoU). The IoU normalizes the score by the number of examples so that the value is independent to the class imbalance. To evaluate a hierarchical classifier, standard metrics are not sufficient as they do not consider the prediction of inner nodes. Thus, a depth-dependent distance metric is presented taking the depth-decaying edge weights between prediction and ground truth into account.

In experiments, three different hierarchies are tested differing in their size and complexity. All class structures are based on a tree to simplify the task. Compared to a flat classifier, the hierarchical networks achieve a considerably higher score when evaluated on the standard IoU metric for the Cityscapes validation dataset. The depth-dependent distance metric establishes that the hierarchical classifier predicts adjacent inner nodes when it is uncertain whereas the flat classifier more frequently selects a far-away leaf. Also, by applying a hierarchy with 69 classes, a higher level of detail can be reached for predictions while keeping the accuracy for other classes at a reasonable level. However, several challenges are remaining including detecting novel objects and finding a more suitable metric to compare the prediction across different class structures.

References

[1] Abadi, M. *et al.*: Tensorflow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283, 2016.

[2] Abbeel, P. and Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the twenty-first international conference on Machine learning, p. 1. ACM, 2004.

[3] Aghdam, H.H. and Heravi, E.J.: Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification. Springer International Publishing, Cham (Switzerland), first edition, 2017.

[4] Ba, J.L./ Kiros, J.R. and Hinton, G.E.: Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] Barutcuoglu, Z. and DeCoro, C.: Hierarchical Shape Classification Using Bayesian Aggregation. In: IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06), pp. 44–48. IEEE, Matsushima, Japan, 2006.

[6] BenTaieb, A. and Hamarneh, G.: Uncertainty driven multi-loss fully convolutional networks for histopathology. In: M.J. Cardoso/ T. Arbel/ S.L. Lee/ V. Cheplygina/ S. Balocco/ D. Mateus/ G. Zahnd/ L. Maier-Hein/ S. Demirci/ E. Granger/ L. Duong/ M.A. Carbonneau/ S. Albarqouni and G. Carneiro, editors, Intravascular Imaging and Computer Assisted Stenting, and Large-Scale Annotation of Biomedical Data and Expert Label Synthesis, pp. 155–163. Springer International Publishing, Cham, 2017.

[7] Berman, M. and Blaschko, M.B.: Optimization of the Jaccard index for image segmentation with the Lovász hinge. *Arxiv e-prints*, 2017. 1705.08790.

[8] Blockeel, H. *et al.*: Hierarchical multi-classification. In: Proceedings of the ACM SIGKDD Workshop on Multi-Relational Data Mining, pp. 21–35, 2002.

[9] Chen, L.C. *et al.*: Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. *arXiv preprint arXiv:1802.02611*, 2018. 1802.02611.

[10] Cho, K. *et al.*: On the properties of neural machine translation: Encoder-decoder approaches. *Computing Research Repository (CoRR)*, volume abs/1409.1259, 2014.

[11] Chollet, F.: Xception: Deep Learning with Separable Convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1800–1807, 2017. 1610.02357.

[12] Cordts, M. *et al.*: The Cityscapes Dataset for Semantic Urban Scene Understanding. In: Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[13] Cordts, M. *et al.*: The Cityscapes Dataset: Benchmark Suite, 2018. Retrieved from https://www.cityscapes-dataset.com/benchmarks/#pixel-level-results, last viewed on 17 August 2018.

[14] Costa, E.P. *et al.*: A Review of Performance Evaluation Measures for Hierarchical Classifiers. In: Evaluation Methods for Machine Learning II: papers from the AAAI-2007 Workshop, pp. 1—-6, 2007. 10.1.1.183.1219.

[15] da Silva, I.N. *et al.*: Artificial Neural Networks: A Practical Course. Springer International Publishing, Switzerland, first edition, 2017.

[16] Deng, J. *et al.*: Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3450–3457, 2012.

[17] Dumais, S. and Chen, H.: Hierarchical classification of Web content. In: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval, pp. 256–263. ACM (New York, NY, USA), Athens, Greece, 2000.

[18] Everingham, M. *et al.*: The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, volume 88, no. 2, pp. 303–338, June 2010.

[19] Feng, S./ Fu, P. and Zheng, W.: A Hierarchical multi-label classification algorithm for gene function prediction. *Algorithms*, volume 10, no. 4, pp. 1–14, 2017.

[20] Finn, C./ Goodfellow, I.J. and Levine, S.: Unsupervised learning for physical interaction through video prediction. *Computing Research Repository (CoRR)*, volume abs/1605.07157, 2016.

[21] Garcia-Garcia, A. *et al.*: A Review on Deep Learning Techniques Applied to Semantic Segmentation. *arXiv preprint arXiv:1704.06857*, 2017. 1704.06857.

[22] Geiger, A./ Lenz, P. and Urtasun, R.: Are we ready for autonomous driving? the KITTI vision benchmark suite. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 3354–3361, 2012. 1612.07695.

[23] Girshick, R.: Fast R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 1440–1448. IEEE Computer Society, 2015. 1504.08083.

[24] Girshick, R. *et al.*: Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 580–587, 2014. 1311.2524.

[25] Glorot, X./ Bordes, A. and Bengio, Y.: Deep sparse rectifier neural networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, volume 15, pp. 315–323, 2011. 1502.03167.

[26] Goldberg, Y.: A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, volume 57, pp. 345–420, 2016.

[27] Goodfellow, I./ Bengio, Y. and Courville, A.: Deep Learning. MIT Press, Cambridge, Massachusetts, first edition, 2016.

[28] Goodfellow, I. *et al.*: Generative Adversarial Nets. In: Z. Ghahramani/ M. Welling/ C. Cortes/ N.D. Lawrence and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems 27, pp. 2672–2680. Curran Associates, Inc., 2014.

[29] Goodfellow, I.J.: NIPS 2016 tutorial: Generative adversarial networks. *Computing Research Repository (CoRR)*, volume abs/1701.00160, 2017.

[30] Haixiang, G. *et al.*: Learning from class-imbalanced data: Review of methods and applications. In: Expert Systems with Applications, volume 73, pp. 220–239. Elsevier Ltd, 2017.

[31] He, H. and Ma, Y.: Imbalanced Learning: foundations, algorithms, and applications. John Wiley & Sons, 2013.

[32] He, K. *et al.*: Deep Residual Learning for Image Recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, 2016. 1703.10722.

[33] Hendrycks, D. and Gimpel, K.: A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In: Proceedings of International Conference on Learning Representations, pp. 1–13, 2017.

[34] Hochreiter, S.: Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.

[35] Hochreiter, S. and Schmidhuber, J.: Long Short-Term Memory. *Neural Computation*, volume 9, no. 8, pp. 1735–1780, 1997.

[36] Holroyd, C.B. and Coles, M.G.: The neural basis of human error processing: reinforcement learning, dopamine, and the error-related negativity. *Psychological review*, volume 109, no. 4, p. 679, 2002.

[37] Huang, G. *et al.*: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2261–2269, 2017.

[38] Ioffe, S. and Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: F. Bach and D. Blei, editors, Proceedings of the 32nd International Conference on Machine Learning, *Proceedings of Machine Learning Research*, volume 37, pp. 448–456. PMLR, Lille, France, 07–09 Jul 2015.

[39] Janai, J. *et al.*: Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art. *arXiv preprint arXiv:1704.05519*, 2017. 1704.05519.

[40] Kaelbling, L.P./ Littman, M.L. and Moore, A.W.: Reinforcement learning: A survey. *Journal of artificial intelligence research*, volume 4, pp. 237–285, 1996.

[41] Karpathy, A.: CS231n Convolutional Neural Networks for Visual Recognition, 2017. Retrieved from http://cs231n.github.io/convolutional-networks/, last viewed on 14 May 2018.

[42] Kendall, A./ Gal, Y. and Cipolla, R.: Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition ({CVPR}), 2018. 1705.07115.

[43] Kingma, D.P. and Ba, J.: Adam: A Method for Stochastic Optimization. In: International Conference on Learning Representations (ICLR) 2015, 2015. 1412.6980.

[44] Kiritchenko, S. *et al.*: Learning and Evaluation in the Presence of Class Hierarchies: Application to Text Categorization. In: Conference of the Canadian Society for Computational Studies of Intelligence, pp. 395—-406. Springer, Berlin, Heidelberg, 2006.

[45] Krawczyk, B.: Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, volume 5, no. 4, pp. 221–232, 2016.

[46] Krizhevsky, A./ Sutskever, I. and Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105, 2012.

[47] Krogh, A. and Hertz, J.A.: A simple weight decay can improve generalization. In: J.E. Moody/ S.J. Hanson and R.P. Lippmann, editors, Advances in Neural Information Processing Systems 4, pp. 950–957. Morgan-Kaufmann, 1992.

[48] LeCun, Y./ Bengio, Y. and Hinton, G.: Deep learning. *Nature*, volume 521, no. 7553, pp. 436–444, 2015.

[49] LeCun, Y. *et al.*: Backpropagation applied to handwritten zip code recognition. *Neural computation*, volume 1, no. 4, pp. 541–551, 1989.

[50] LeCun, Y. *et al.*: Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, volume 86, no. 11, pp. 2278–2324, 1998.

[51] Lee, K. *et al.*: Training Confidence-calibrated Classifiers for Detecting Out-of-Distribution Samples. *CoRR*, volume abs/1711.0, pp. 1–16, 2017. 1711.09325.

[52] Lee, K. *et al.*: Hierarchical Novelty Detection for Visual Object Recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1034–1042, 2018. 1804.00722.

[53] Lin, G. *et al.*: RefineNet: Multi-path refinement networks for high-resolution semantic segmentation. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-January, pp. 5168–5177, 2017. 1611.06612.

[54] Lin, T. *et al.*: Microsoft COCO: common objects in context. *Computing Research Repository (CoRR)*, volume abs/1405.0312, 2014.

[55] Lin, T. *et al.*: Focal Loss for Dense Object Detection. In: IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017, pp. 2999–3007, 2017.

[56] Liu, W. *et al.*: SSD: Single Shot Multibox Detector. In: European conference on computer vision, pp. 21–37. Springer, 2016.

[57] Long, J./ Shelhamer, E. and Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 3431–3440, 2015. 1411.4038.

[58] Meletis, P. and Dubbelman, G.: Training of Convolutional Networks on Multiple Heterogeneous Datasets for Street Scene Semantic Segmentation. In: IEEE Intelligent Vehicles Symposium (IV). IEEE Computer Society, 2018. arXiv:1803.05675v1.

[59] Mitchell, T.M.: Machine Learning. McGraw-Hill, Inc., New York, NY, USA, first edition, 1997.

[60] Mnih, V. *et al.*: Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*, 2013.

[61] Nair, V. and Hinton, G.E.: Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, , no. 3, pp. 807–814, 2010. 1111.6189v1.

[62] Neuhold, G. *et al.*: The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes. In: Proceedings of the International Conference on Computer Vision (ICCV), Venice, Italy, pp. 22–29, 2017.

[63] Neuhold, G. *et al.*: The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes Supplementary Materia, 2017. Retrieved from http://research.mapillary.com/img/publications/ICCV17a_supp.pdf, last viewed on 13 July 2018.

[64] Nguyen-Tuong, D. and Peters, J.: Model learning for robot control: a survey. *Cognitive processing*, volume 12, no. 4, pp. 319–340, 2011.

[65] Noh, H./ Hong, S. and Han, B.: Learning deconvolution network for semantic segmentation. *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1520–1528, 2015. 1505.04366.

[66] Olah, C.: Understanding LSTM Networks, August 2015. Retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/, last viewed on 16 May 2018.

[67] Rahman, A. and Wang, Y.: Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation. In: International Symposium on Visual Computing, pp. 234–244. Springer, 2016.

[68] Redmon, J. and Farhadi, A.: YOLO9000: Better, Faster, Stronger. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6517–6525, 2017. 1612.08242.

[69] Redmon, J. and Farhadi, A.: YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767*, apr 2018. 1804.02767.

[70] Redmon, J. *et al.*: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788, 2016.

[71] Ren, S. *et al.*: Faster R-CNN: Towards Real-Time Object Detection with. In: C. Cortes/ N.D. Lawrence/ D.D. Lee/ M. Sugiyama and R. Garnett, editors, Advances in Neural Information Processing Systems 28, pp. 91–99. Curran Associates, Inc., 2015. 1506.01497.

[72] Ronneberger, O./ Fischer, P. and Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention, pp. 234–241. Springer, 2015.

[73] Rowley, H.A./ Baluja, S. and Kanade, T.: Neural Network-Based Face Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, volume 20, no. 1, pp. 23–38, January 1998.

[74] Rumelhart, D.E. *et al.*: Learning representations by back-propagating errors. *Cognitive modeling*, volume 5, no. 3, p. 1, 1988.

[75] Russakovsky, O. *et al.*: ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, volume 115, no. 3, pp. 211–252, 2015.

[76] Sak, H./ Senior, A. and Beaufays, F.: Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[77] Sermanet, P. *et al.*: Pedestrian Detection with Unsupervised Multi-stage Feature Learning. In: Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '13, pp. 3626–3633. IEEE Computer Society, Washington, DC, USA, 2013.

[78] Silla, C.N. and Freitas, A.A.: A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, volume 22, no. 1-2, pp. 31–72, 2011. arXiv:1507.02293v1.

[79] Silver, D. *et al.*: Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, volume 529, no. 7587, pp. 484–489, January 2016.

[80] Simonyan, K. and Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: International Conference on Learning Representations, pp. 1–14, 2014. 1409.1556.

[81] Singh, S.P. *et al.*: Robust reinforcement learning in motion planning. In: Advances in neural information processing systems, pp. 655–662, 1994.

[82] Sokolova, M. and Lapalme, G.: A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, volume 45, no. 4, pp. 427–437, 2009.

[83] Soutner, D. and Müller, L.: Application of LSTM Neural Networks in Language Modelling, pp. 105–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[84] Srivastava, N. *et al.*: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, volume 15, pp. 1929–1958, 2014. 1102.4807.

[85] Srivastava, R.K./ Greff, K. and Schmidhuber, J.: Highway Networks. *arXiv preprint arXiv:1505.00387*, 2015. 1505.00387.

[86] Stallkamp, J. *et al.*: The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In: Proceedings of the International Joint Conference on Neural Networks, pp. 1453–1460, 2011. 1302.1700.

[87] Sun, A.S.A. and Lim, E.P.L.E.P.: Hierarchical text classification and evaluation. In: Proceedings 2001 IEEE International Conference on Data Mining, pp. 521–528, 2001.

[88] Sutton, R.S. and Barto, A.G.: Introduction to Reinforcement Learning. MIT Press, Cambridge, MA, USA, first edition, 1998.

[89] Szegedy, C. *et al.*: Going Deeper with Convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9, 2014. 1409.4842.

[90] Szegedy, C. *et al.*: Rethinking the Inception Architecture for Computer Vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818—-2826, 2016. 1512.00567.

[91] Szegedy, C. *et al.*: Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In: Proceedings of the Thirty-First {AAAI} Conference on Artificial Intelligence, pp. 4278—-4284, 2017. 1602.07261.

[92] Uhrig, J. *et al.*: Box2pix: Single-shot instance segmentation by assigning pixels to object boxes. In: IEEE Intelligent Vehicles Symposium (IV), 2018.

[93] Uijlings, J. *et al.*: Selective Search for Object Recognition. *International Journal of Computer Vision*, 2013.

[94] Valentini, G.: True Path Rule Hierarchical Ensembles. In: J.A. Benediktsson/ / J. Kittler/ and F. Roli, editors, Multiple Classifier Systems, pp. 232–241. Springer Berlin Heidelberg, 2009. arXiv:1011.1669v3.

[95] Veit, A./ Wilber, M. and Belongie, S.: Residual Networks Behave Like Ensembles of Relatively Shallow Networks. In: D.D. Lee/ M. Sugiyama/ U.V. Luxburg/ I. Guyon and R. Garnett, editors, Advances in Neural Information Processing Systems, pp. 550—-558. Curran Associates, Inc., 2016. 1605.06431.

[96] Versteegh, M. *et al.*: The zero resource speech challenge 2015. In: Interspeech, pp. 3169–3173, 2015.

[97] Villegas, R. *et al.*: Decomposing motion and content for natural video sequence prediction. In: Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[98] Xie, M. *et al.*: Active and intelligent sensing of road obstacles: Application to the European Eureka-PROMETHEUS project. In: Computer Vision, 1993. Proceedings., Fourth International Conference on, pp. 616–623. IEEE, 1993.

[99] Zeiler, M.D. and Fergus, R.: Visualizing and understanding convolutional networks. In: European conference on computer vision, volume 8689 LNCS, pp. 818–833. Springer, 2014. 1311.2901.

[100] Zhao, H. *et al.*: Pyramid scene parsing network. In: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, pp. 6230–6239, 2017. 1612.01105.

[101] Ziegler, J. *et al.*: Making Bertha drive—An autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine*, volume 6, no. 2, pp. 8–20, 2014.

A. Dataset distribution

This section reviews the exact class distributions of the datasets Cityscapes [12], Cityscapes Special and Mapillary Vistas [62]. Therefore, tables list the number of positive and negative examples for each class of the hierarchy. The values are normalized by the number of pixels per image to get the expected distribution for a random image. Besides, the ratio of positives and the overall proportion of the class labels in the whole dataset are shown. The number of images the class occurs in is stated on the right side of the tables. The tables for Cityscapes and Cityscapes Special are based on the standard hierarchy visualized in Figure 48 (Appendix B). The dashed lines group sibling nodes together that are mutually exclusive. As the large-scale hierarchy mainly differs from the standard structure by having additional classes from Mapillary Vistas, the table of this dataset is based on the greater hierarchy. The values can be easily adopted for the standard or small hierarchy. To give a better overview of the datasets, the first subsection subsumes all data that is used for training whereas the second shows the validation datasets.

A.1. Training Datasets

Cityscapes Training

Class	Positive	Negative	Ratio	Proportion	Images
Ground	861,981	1,047,096	45.15%	41.10%	2,903
Sky	74,505	1,834,572	3.90%	3.55%	2,626
Obstacle	972,591	936,486	50.95%	46.38%	2,903
Street	815,721	21,848	97.39%	38.90%	2,880
Natural ground	21,848	815,721	2.61%	1.04%	1,635
Road	685,119	113,978	85.74%	32.67%	2,866
Sidewalk	113,978	685,119	14.26%	5.43%	2,749
Terrain	21,848	0	100.00%	1.04%	1,635
Water	0	21,848	0.00%	0.00%	0
Vehicle	154,075	818,514	15.84%	7.35%	2,865
Vulnerable road user	24,939	947,650	2.56%	1.19%	2,443
Nature	295,798	676,791	30.41%	14.10%	2,821
Infrastructure	497,777	474,812	51.18%	23.74%	2,902
Passenger car	129,556	24,518	84.09%	6.18%	2,765
Large vehicle	15,073	139,001	9.78%	0.72%	803
Ridable vehicle	9,445	144,629	6.13%	0.45%	1,789
Truck	4,938	10,134	32.76%	0.24%	347

Cityscapes Training					
Class	Positive	Negative	Ratio	Proportion	Images
Caravan	847	14,225	5.62%	0.04%	56
Bus	4,344	10,728	28.82%	0.21%	269
Train	4,431	10,641	29.40%	0.21%	142
Trailer	512	14,560	3.40%	0.02%	82
Bicycle	7,688	1,757	81.40%	0.37%	1,630
Motorbike	1,757	7,688	18.60%	0.08%	510
Animal	0	24,939	0.00%	0.00%	0
Human	24,939	0	100.00%	1.19%	2,443
Pedestrian	22,443	2,495	90.00%	1.07%	2,290
Rider	2,495	22,443	10.00%	0.12%	1,012
Bicyclist	0	0	0.00%	0.00%	0
Motorcyclist	0	0	0.00%	0.00%	0
Elongated infrastructure	460,449	37,327	92.50%	21.96%	2,895
Vertical infrastructure	22,988	474,788	4.62%	1.10%	2,878
Sign	14,339	483,437	2.88%	0.68%	2,789
Overpass	6,889	453,558	1.50%	0.33%	245
Building	424,513	35,934	92.20%	20.24%	2,865
Wall	12,495	447,952	2.71%	0.60%	978
Guardrail	195	460,252	0.04%	0.01%	20
Fence	16,355	444,092	3.55%	0.78%	1,293
Bridge	5,732	1,156	83.22%	0.27%	224
Tunnel	1,156	5,732	16.78%	0.06%	23
Hydrant	0	22,988	0.00%	0.00%	0
Pole	22,988	0	100.00%	1.10%	2,878
Trash bin	0	22,988	0.00%	0.00%	0
Traffic sign	10,435	3,903	72.78%	0.50%	2,760
Traffic light	3,903	10,435	27.22%	0.19%	1,624
Duty - Normal	161,283	0	100.00%	7.69%	2,887
Duty - Police	0	161,283	0.00%	0.00%	0
Duty - Fire	0	161,283	0.00%	0.00%	0
Duty - Medical	0	161,283	0.00%	0.00%	0

Cityscapes Training					
Class	Positive	Negative	Ratio	Proportion	Images
Duty - School	0	161,283	0.00%	0.00%	0
Total	2,097,152	-	100%	100%	2,903

Table 5: The Cityscapes training dataset contains 2975 images. As 72 of those are subsumed in the Cityscapes Special dataset, the table shows the distribution of the 2903 remaining images. Overall, the most common leaf classes include *road* and *building*, whereas *guardrail* and *trailer* are the rarest classes.

Cityscapes Special Training					
Class	Positive	Negative	Ratio	Proportion	Images
Ground	723,935	790,070	47.82%	34.52%	230
Sky	74,655	1,439,350	4.93%	3.56%	216
Obstacle	715,415	798,590	47.25%	34.11%	230
Street	706,016	11,579	98.39%	33.67%	230
Naturalground	11,579	706,016	1.61%	0.55%	48
Road	633,000	65,001	90.69%	30.18%	230
Sidewalk	65,001	633,000	9.31%	3.10%	194
Terrain	11,579	0	100.00%	0.55%	48
Water	0	11,579	0.00%	0.00%	0
Vehicle	153,061	562,353	21.39%	7.30%	230
Vulnerableroaduser	11,470	703,944	1.60%	0.55%	145
Nature	206,044	509,370	28.80%	9.82%	221
Infrastructure	344,839	370,575	48.20%	16.44%	230
Passengercar	124,296	28,764	81.21%	5.93%	223
Largevehicle	25,380	127,680	16.58%	1.21%	108
Ridablevehicle	3,384	149,676	2.21%	0.16%	84
Truck	15,147	10,231	59.69%	0.72%	74
Caravan	303	25,075	1.19%	0.01%	5
Bus	7,386	17,992	29.10%	0.35%	33
Train	2,414	22,964	9.51%	0.12%	5
Trailer	128	25,250	0.50%	0.01%	4
Bicycle	2,819	565	83.30%	0.13%	75
Motorbike	565	2,819	16.70%	0.03%	19
Animal	0	11,470	0.00%	0.00%	0
Human	11,470	0	100.00%	0.55%	145
Pedestrian	10,372	1,098	90.43%	0.49%	130
Rider	1,098	10,372	9.57%	0.05%	54
Bicyclist	0	0	0.00%	0.00%	0
Motorcyclist	0	0	0.00%	0.00%	0
Elongatedinfrastructure	320,822	24,015	93.04%	15.30%	228
Verticalinfrastructure	14,174	330,663	4.11%	0.68%	222

Cityscapes Special Training					
Class	Positive	Negative	Ratio	Proportion	Images
Sign	9,841	334,996	2.85%	0.47%	196
Overpass	2,066	318,755	0.64%	0.10%	5
Building	289,587	31,234	90.26%	13.81%	221
Wall	14,223	306,598	4.43%	0.68%	59
Guardrail	575	320,246	0.18%	0.03%	2
Fence	14,370	306,451	4.48%	0.69%	88
Bridge	903	1,162	43.73%	0.04%	4
Tunnel	1,162	903	56.27%	0.06%	1
Hydrant	0	14,174	0.00%	0.00%	0
Pole	14,174	0	100.00%	0.68%	222
Trashbin	0	14,174	0.00%	0.00%	0
Trafficsign	7,155	2,685	72.71%	0.34%	190
Trafficlight	2,685	7,155	27.29%	0.13%	125
Duty - Normal	118,245	38,956	75.22%	5.64%	221
Duty - Police	19,463	137,738	12.38%	0.93%	117
Duty - Fire	5,787	151,414	3.68%	0.28%	20
Duty - Medical	13,706	143,495	8.72%	0.65%	90
Duty - School	0	157,201	0.00%	0.00%	0
Total	2,097,152	-	100%	100%	230

Table 6: The Cityscapes Special training dataset is a subset of the Cityscapes training and testing. It constitutes an initial dataset to test the performance on learning object attributes. Overall, 230 images were collected and relabeled with attribute labels for *duty*. The most rarest class constitutes *fire* that only occurs on 20 images. Thus, the variety of the examples are limited and the network can less generalize. The other attribute nodes *police* and *medical* have about 100 different images where objects are assigned to those labels. The distribution of the other classes is similar to Cityscapes training.

Mapillary Large-Scale Training					
Class	Positive	Negative	Ratio	Proportion	Images
Ground	677,129	1,327,512	33.78%	32.29%	17,049
Sky	379,663	1,624,978	18.94%	18.10%	16,391
Obstacle	947,849	1,056,792	47.28%	45.20%	17,103
Street	642,839	34,289	94.94%	30.65%	17,028
Natural ground	34,289	642,839	5.06%	1.64%	8,363
Road	528,214	94,412	84.84%	25.19%	16,945
Sidewalk	94,412	528,214	15.16%	4.50%	14,642
Lane	469,149	59,063	88.82%	22.37%	16,872
Marking	50,318	477,894	9.53%	2.40%	15,471
Hole	1,348	526,864	0.26%	0.06%	4,630
Bike lane	7,397	520,815	1.40%	0.35%	1,253
Lane marking (general)	34,540	15,777	68.64%	1.65%	15,149
Lane marking (crosswalk)	15,777	34,540	31.36%	0.75%	5,528
Manhole	1,254	94	93.03%	0.06%	4,391
Pothole	94	1,254	6.97%	0.00%	405
Pedestrian sidewalk	70,698	23,714	74.88%	3.37%	12,420
Curb	23,714	70,698	25.12%	1.13%	14,363
Terrain	32,727	1,562	95.44%	1.56%	8,179
Water	1,562	32,727	4.56%	0.07%	426
Vehicle	130,371	817,476	13.75%	6.22%	16,380
Vulnerable road user	11,344	936,503	1.20%	0.54%	9,697
Nature	344,469	603,378	36.34%	16.43%	16,576
Infrastructure	461,663	486,184	48.71%	22.01%	17,080
Passenger car	104,222	24,931	80.70%	4.97%	15,988
Large vehicle	20,926	108,227	16.20%	1.00%	6,458
Ridable vehicle	4,005	125,148	3.10%	0.19%	5,071
Truck	11,281	9,643	53.91%	0.54%	4,158
Caravan	99	20,825	0.47%	0.00%	114
Bus	8,243	12,681	39.39%	0.39%	2,790
Train	728	20,196	3.48%	0.03%	223
Trailer	302	20,622	1.44%	0.01%	188

Mapillary Large-Scale Training					
Class	Positive	Negative	Ratio	Proportion	Images
Boat	271	20,653	1.30%	0.01%	122
Bicycle	1,917	2,087	47.88%	0.09%	2,828
Motorbike	1,892	2,112	47.25%	0.09%	2,644
Buggy	195	3,809	4.87%	0.01%	678
Animal	61	11,282	0.54%	0.00%	791
Human	11,282	61	99.46%	0.54%	9,531
Pedestrian	9,468	1,814	83.92%	0.45%	8,679
Rider	1,814	9,468	16.08%	0.09%	3,398
Bicyclist	1,082	681	61.37%	0.05%	1,771
Motorcyclist	681	1,082	38.63%	0.03%	1,824
Elongated infrastructure	405,091	56,571	87.75%	19.32%	16,976
Vertical infrastructure	32,398	429,264	7.02%	1.54%	16,974
Box-like infrastructure	3,436	458,226	0.74%	0.16%	6,911
Sign	20,737	440,925	4.49%	0.99%	16,205
Overpass	27,513	377,576	6.79%	1.31%	3,759
Building	293,589	111,500	72.48%	14.00%	16,057
Barrier	83,987	321,102	20.73%	4.00%	14,170
Bridge	22,638	4,874	82.28%	1.08%	2,394
Tunnel	2,525	24,987	9.18%	0.12%	202
Traffic sign frame	2,349	25,163	8.54%	0.11%	2,006
Wall	23,651	46,784	33.58%	1.13%	7,953
Guardrail	8,164	62,271	11.59%	0.39%	2,877
Fence	38,620	31,815	54.83%	1.84%	10,724
Hydrant	171	32,226	0.53%	0.01%	1,663
Pole	32,226	171	99.47%	1.54%	16,974
Bench	390	3,045	11.35%	0.02%	964
Trash bin	1,788	1,647	52.05%	0.09%	3,678
Junction box	1,131	2,304	32.93%	0.05%	3,541
Mailbox	126	3,309	3.67%	0.01%	541
Traffic sign	13,662	7,074	65.89%	0.65%	15,974
Traffic light	5,430	15,306	26.19%	0.26%	8,763

Mapillary Large-Scale Training					
Class	Positive	Negative	Ratio	Proportion	Images
Ad sign	1,644	19,092	7.93%	0.08%	3,172
Traffic sign (front)	11,566	2,095	84.66%	0.55%	15,534
Traffic sign (back)	2,095	11,566	15.34%	0.10%	11,311
Duty - Normal	0	0	0.00%	0.00%	0
Duty - Police	0	0	0.00%	0.00%	0
Duty - Fire	0	0	0.00%	0.00%	0
Duty - Medical	0	0	0.00%	0.00%	0
Duty - School	0	0	0.00%	0.00%	0
Total	2,097,152	-	100%	100%	17,106

Table 7: The Mapillary Vistas training dataset contains labels for much more classes than Cityscapes. Therefore, the table shows the class distribution regarding to the large-scale hierarchy. Within the 17,106 images, the class *road* appears most often. In contrast to Cityscapes, the class *caravan* has about 10 times less pixels in average. Thus, it is one of the rarest classes in Mapillary Vistas next to *pothole* and *animal*.

A.2. Validation Datasets

Cityscapes Validation					
Class	Positive	Negative	Ratio	Proportion	Images
Ground	851,283	1,032,050	45.20%	40.59%	495
Sky	61,804	1,821,529	3.28%	2.95%	438
Obstacle	970,246	913,087	51.52%	46.26%	495
Street	797,978	15,463	98.10%	38.05%	482
Natural ground	15,463	797,978	1.90%	0.74%	240
Road	689,487	99,253	87.42%	32.88%	479
Sidewalk	99,253	689,487	12.58%	4.73%	461
Terrain	15,463	0	100.00%	0.74%	240
Water	0	15,463	0.00%	0.00%	0
Vehicle	149,444	820,801	15.40%	7.13%	485
Vulnerable road user	27,906	942,339	2.88%	1.33%	439
Nature	318,656	651,589	32.84%	15.19%	482
Infrastructure	474,239	496,006	48.88%	22.61%	495
Passenger car	119,443	29,999	79.93%	5.70%	474
Large vehicle	15,349	134,093	10.27%	0.73%	173
Ridable vehicle	14,650	134,792	9.80%	0.70%	363
Truck	5,609	9,739	36.55%	0.27%	83
Caravan	26	15,322	0.17%	0.00%	5
Bus	7,156	8,192	46.62%	0.34%	75
Train	2,067	13,281	13.47%	0.10%	22
Trailer	490	14,858	3.19%	0.02%	16
Bicycle	13,177	1,473	89.95%	0.63%	347
Motorbike	1,473	13,177	10.05%	0.07%	90
Animal	0	27,906	0.00%	0.00%	0
Human	27,906	0	100.00%	1.33%	439
Pedestrian	23,899	4,007	85.64%	1.14%	402
Rider	4,007	23,899	14.36%	0.19%	253
Bicyclist	0	0	0.00%	0.00%	0
Motorcyclist	0	0	0.00%	0.00%	0

Cityscapes Validation					
Class	Positive	Negative	Ratio	Proportion	Images
Elongated infrastructure	430,872	43,366	90.86%	20.55%	492
Vertical infrastructure	27,274	446,964	5.75%	1.30%	486
Sign	16,092	458,146	3.39%	0.77%	473
Overpass	629	430,240	0.15%	0.03%	14
Building	400,993	29,876	93.07%	19.12%	486
Wall	13,763	417,106	3.19%	0.66%	203
Guardrail	78	430,791	0.02%	0.00%	1
Fence	15,406	415,463	3.58%	0.73%	196
Bridge	629	0	100.00%	0.03%	14
Tunnel	0	629	0.00%	0.00%	0
Hydrant	0	27,274	0.00%	0.00%	0
Pole	27,274	0	100.00%	1.30%	486
Trash bin	0	27,274	0.00%	0.00%	0
Traffic sign	12,364	3,728	76.83%	0.59%	469
Traffic light	3,728	12,364	23.17%	0.18%	289
Duty - Normal	156,108	0	100.00%	7.44%	488
Duty - Police	0	156,108	0.00%	0.00%	0
Duty - Fire	0	156,108	0.00%	0.00%	0
Duty - Medical	0	156,108	0.00%	0.00%	0
Duty - School	0	156,108	0.00%	0.00%	0
Total	2,097,152	-	100%	100%	500

Table 8: The Cityscapes validation dataset contains 500 images from three cities that are not included in the training. The distribution of both sets is similar but differs especially for rare classes. For instance, the class *caravan* is about 40 times less likely to be assigned to label in the validation dataset. With only 26 pixels per image in average, it is the rarest class in any of the datasets. The distribution for common classes on which the Cityscapes validation is performed only slightly deviates from the training set.

Mapillary Large-Scale Validation					
Class	Positive	Negative	Ratio	Proportion	Images
Ground	693,002	1,307,489	34.64%	33.04%	1,880
Sky	383,099	1,617,392	19.15%	18.27%	1,821
Obstacle	924,390	1,076,101	46.21%	44.08%	1,886
Street	654,755	38,246	94.48%	31.22%	1,877
Natural ground	38,246	654,755	5.52%	1.82%	965
Road	531,977	99,271	84.27%	25.37%	1,866
Sidewalk	99,271	531,977	15.73%	4.73%	1,600
Lane	473,171	58,804	88.95%	22.56%	1,854
Marking	49,969	482,006	9.39%	2.38%	1,697
Hole	1,419	530,556	0.27%	0.07%	511
Bike lane	7,416	524,559	1.39%	0.35%	137
Lane marking (general)	33,386	16,582	66.81%	1.59%	1,661
Lane marking (crosswalk)	16,582	33,386	33.19%	0.79%	593
Manhole	1,196	223	84.28%	0.06%	488
Pothole	223	1,196	15.72%	0.01%	34
Pedestrian sidewalk	74,895	24,376	75.44%	3.57%	1,381
Curb	24,376	74,895	24.56%	1.16%	1,579
Terrain	35,734	2,512	93.43%	1.70%	941
Water	2,512	35,734	6.57%	0.12%	52
Vehicle	125,327	799,062	13.56%	5.98%	1,801
Vulnerable road user	11,263	913,126	1.22%	0.54%	1,133
Nature	337,202	587,187	36.48%	16.08%	1,842
Infrastructure	450,597	473,792	48.75%	21.49%	1,883
Passenger car	100,734	23,288	81.22%	4.80%	1,758
Large vehicle	19,685	104,337	15.87%	0.94%	716
Ridable vehicle	3,603	120,419	2.91%	0.17%	576
Truck	11,025	8,659	56.01%	0.53%	453
Caravan	193	19,491	0.98%	0.01%	9
Bus	7,064	12,620	35.89%	0.34%	298
Train	944	18,740	4.80%	0.05%	30
Trailer	296	19,388	1.50%	0.01%	25

Mapillary Large-Scale Validation					
Class	Positive	Negative	Ratio	Proportion	Images
Boat	162	19,522	0.82%	0.01%	20
Bicycle	1,666	1,936	46.25%	0.08%	317
Motorbike	1,744	1,858	48.42%	0.08%	310
Buggy	192	3,410	5.33%	0.01%	83
Animal	91	11,171	0.81%	0.00%	99
Human	11,171	91	99.19%	0.53%	1,113
Pedestrian	9,365	1,806	83.83%	0.45%	1,021
Rider	1,806	9,365	16.17%	0.09%	381
Bicyclist	1,072	714	60.02%	0.05%	194
Motorcyclist	714	1,072	39.98%	0.03%	209
Elongated infrastructure	390,771	59,824	86.72%	18.63%	1,869
Vertical infrastructure	33,683	416,912	7.48%	1.61%	1,873
Box-like infrastructure	3,864	446,731	0.86%	0.18%	783
Sign	22,277	428,318	4.94%	1.06%	1,782
Overpass	23,722	367,048	6.07%	1.13%	395
Building	285,217	105,553	72.99%	13.60%	1,756
Barrier	81,831	308,939	20.94%	3.90%	1,537
Bridge	19,357	4,364	81.60%	0.92%	273
Tunnel	1,952	21,769	8.23%	0.09%	17
Traffic sign frame	2,412	21,309	10.17%	0.12%	197
Wall	25,358	45,037	36.02%	1.21%	883
Guardrail	6,959	63,436	9.89%	0.33%	312
Fence	38,078	32,317	54.09%	1.82%	1,167
Hydrant	134	33,549	0.40%	0.01%	183
Pole	33,549	134	99.60%	1.60%	1,873
Bench	405	3,457	10.49%	0.02%	108
Trash bin	1,644	2,218	42.57%	0.08%	413
Junction box	1,724	2,138	44.64%	0.08%	431
Mailbox	89	3,773	2.30%	0.00%	61
Traffic sign	15,381	6,895	69.05%	0.73%	1,758
Traffic light	5,182	17,094	23.26%	0.25%	943

Mapillary Large-Scale Validation					
Class	Positive	Negative	Ratio	Proportion	Images
Ad sign	1,713	20,563	7.69%	0.08%	350
Traffic sign (front)	13,234	2,147	86.04%	0.63%	1,718
Traffic sign (back)	2,147	13,234	13.96%	0.10%	1,260
Duty - Normal	0	0	0.00%	0.00%	0
Duty - Police	0	0	0.00%	0.00%	0
Duty - Fire	0	0	0.00%	0.00%	0
Duty - Medical	0	0	0.00%	0.00%	0
Duty - School	0	0	0.00%	0.00%	0
Total	2,097,152	-	100%	100%	1,886

Table 9: A set of 1886 images constitute the Mapillary Vistas validation dataset. It is very similar to the training dataset, but contains some more pixels for rare classes like *animal* and *pothole*.

B. Label Hierarchies

This section summarizes the visualizations of the three class hierarchies used for experiments in Section 4. Each node is illustrated by a filled circle. The label id and the name of the class is written inside the node. The color of a node shows the color-mapping for example images in Section 4. The color for object attributes are a combination of the base class and the attribute node color.

The directed connections between nodes constitute an *is-a* relationship. The number written next to the arrow is the edges weight for the depth-dependent distance metric (see Section 3.3.2). Virtual edges to object attributes are visualized by a dashed line. The root node is given by *label* with the id -1. For a detailed discussion of the class structures, see Section 3.1.4.

B.1. Small-scale hierarchy

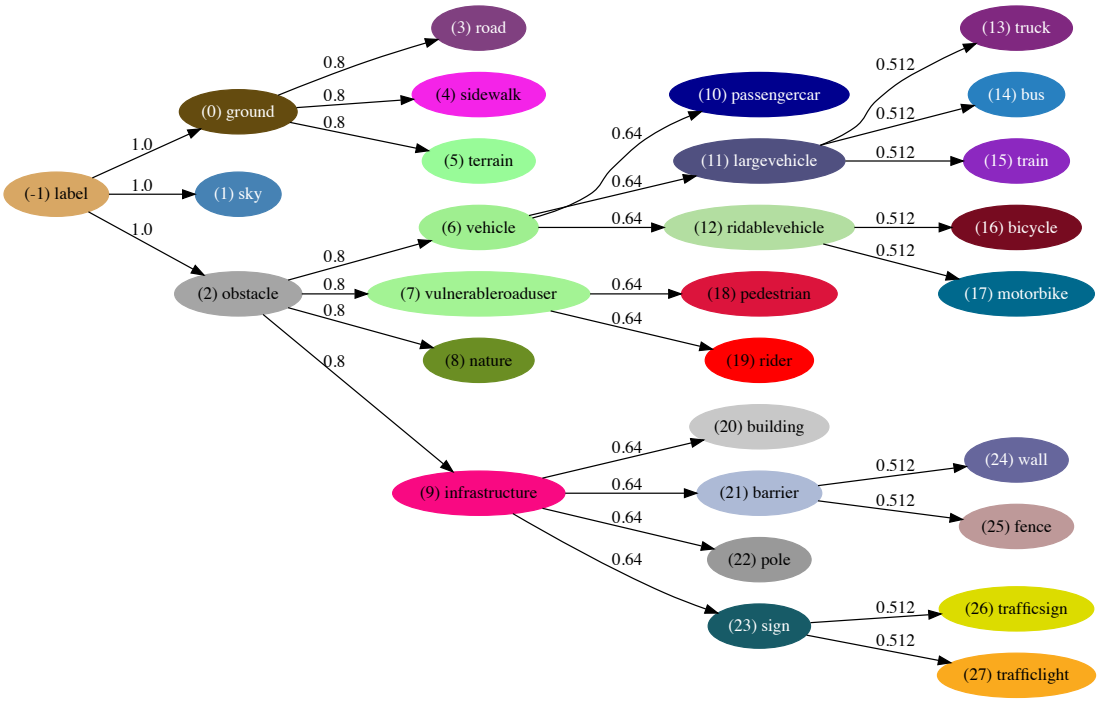


Figure 47: The small-scale hierarchy consists of the 19 Cityscapes validation classes and nine inner nodes.

B.2. Standard hierarchy

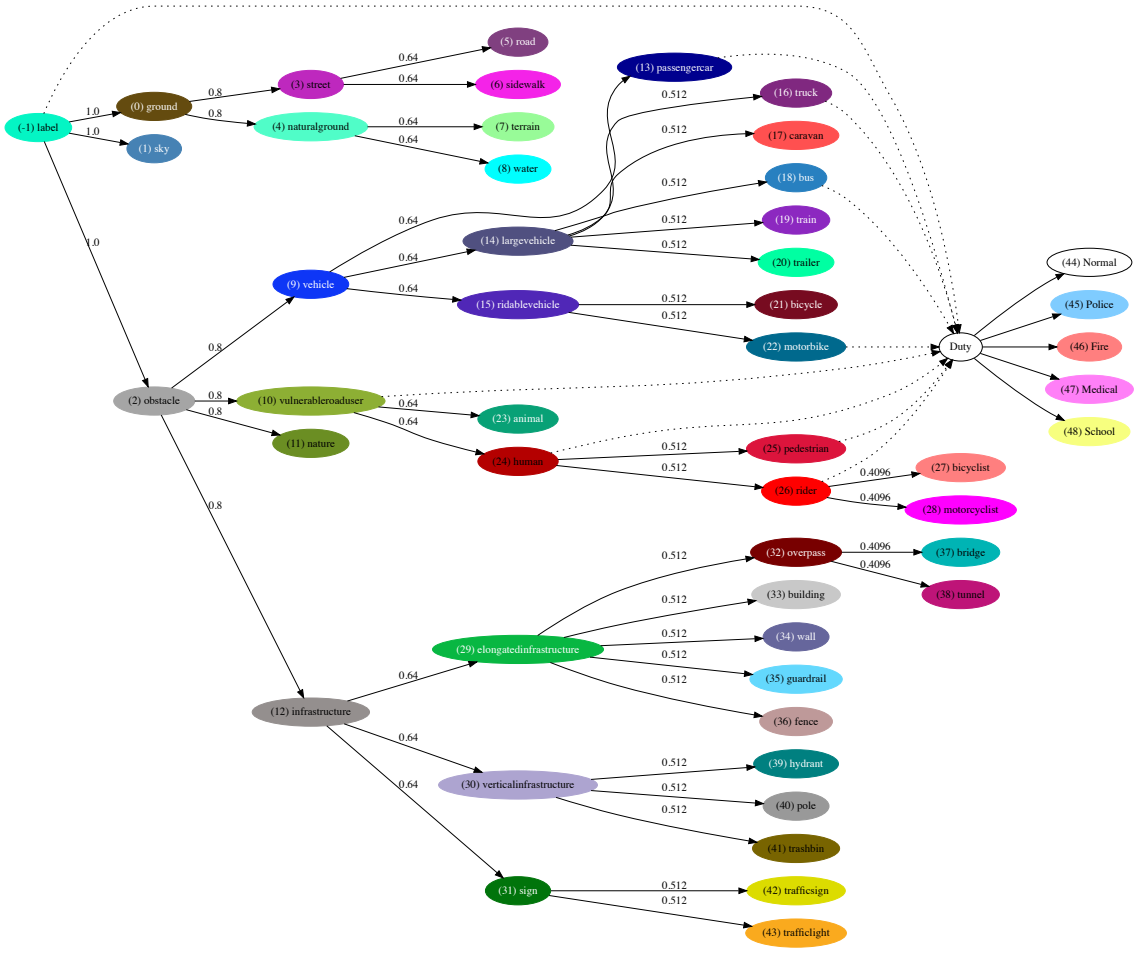


Figure 48: The standard hierarchy consists of 48 nodes extending the small-scale hierarchy by rare classes like *caravan* and *trailer*. Also, the attribute *duty* is added.

B.3. Large-scale hierarchy

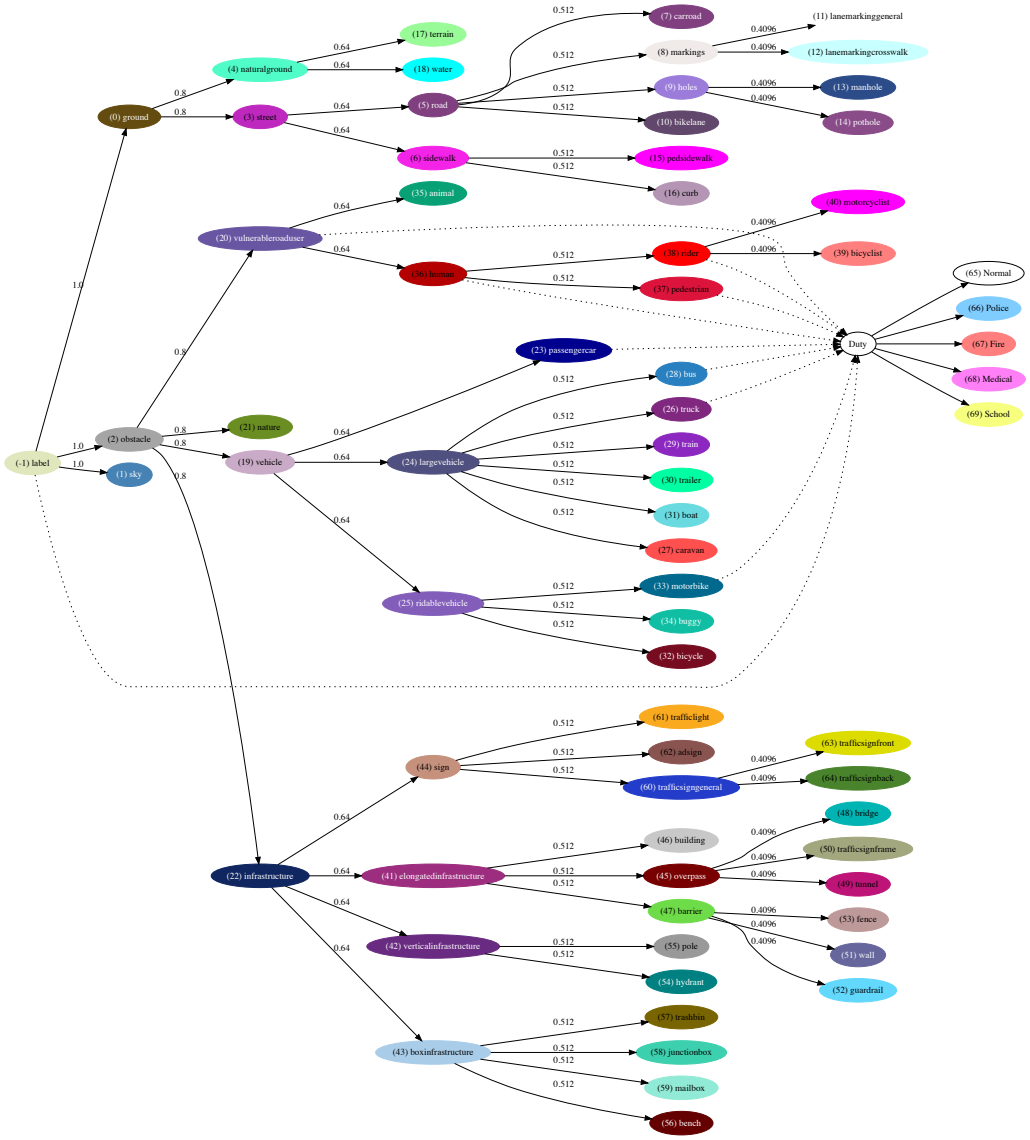


Figure 49: The large-scale hierarchy consists of 69 nodes. Additional classes compared to the standard hierarchy include *lane markings* and *buggy*.

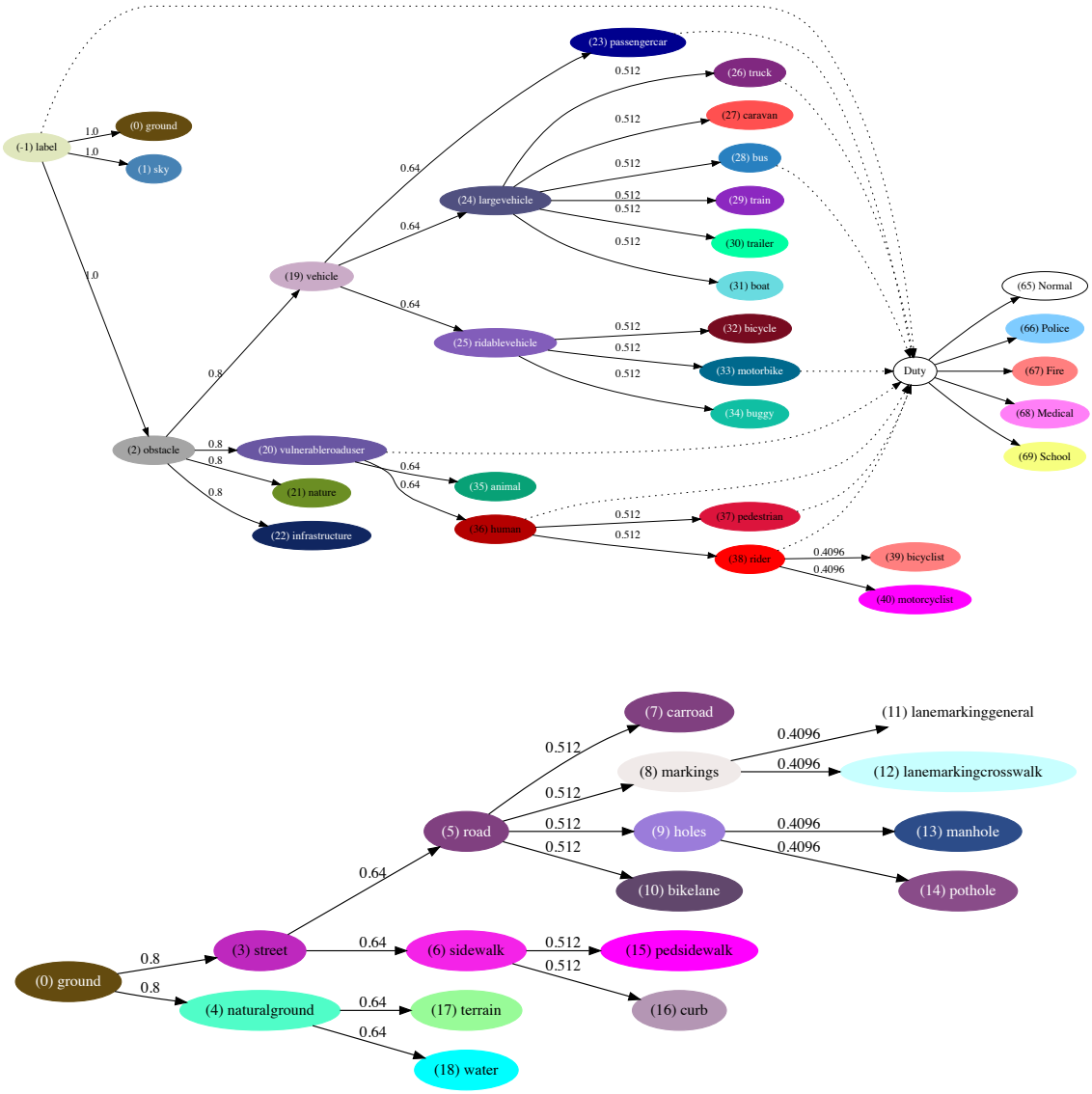


Figure 50: Larger figures of large-scale hierarchy for printed version.

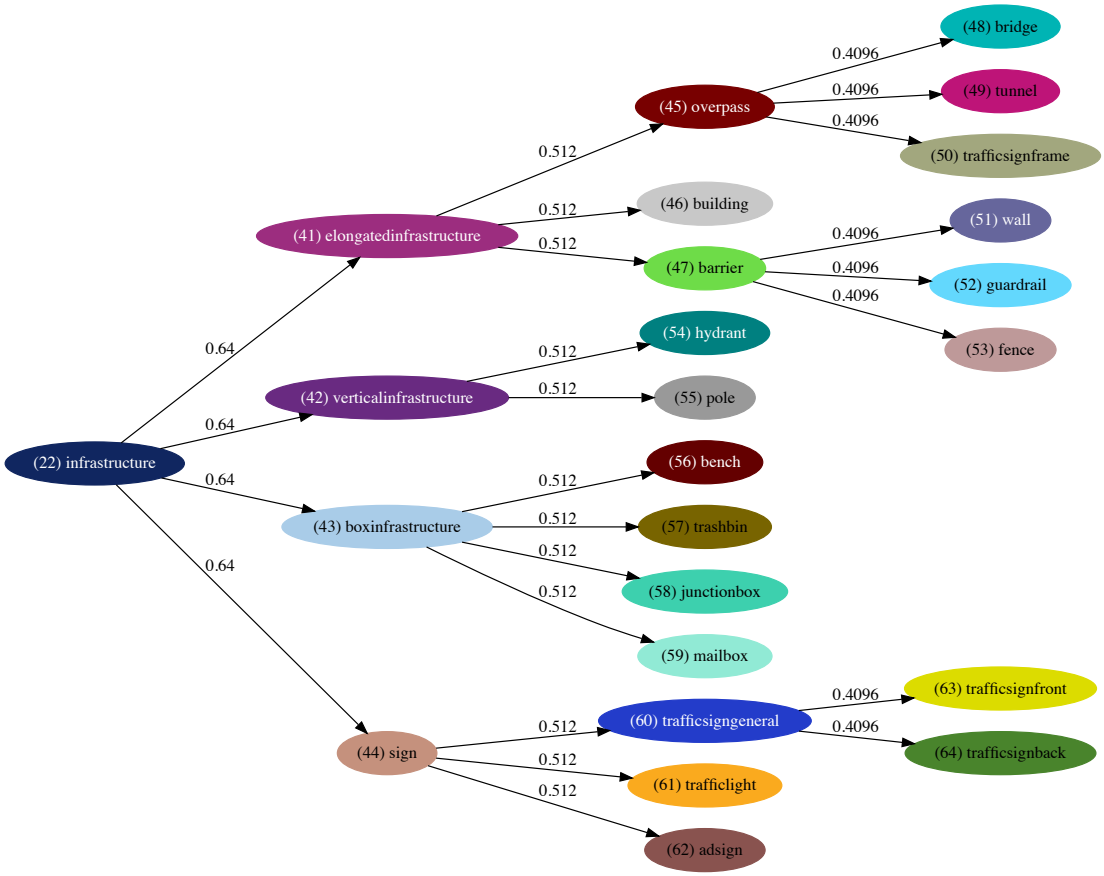


Figure 51: Larger figures of large-scale hierarchy for printed version (subtree infrastructure).