Master's Thesis

Department of Mathematics and Computer Science
Intelligent Systems and Robotics Lab

Construction site detection for autonomous vehicles using deep learning

submitted by
Felix Bröker
felix.broeker@posteo.de

Supervisor:
Prof. Dr. Daniel Göhring

Second examiner:
Prof. Dr. Raúl Rojas

February, 28 2017

Abstract

Currently big players in the automotive sector are racing each other for developing the first full autonomous vehicle. The most well-known among them are Tesla Inc. [1], VW AG [2] and even non automotive companies, such as Alphabet Inc. [3] and Apple Inc. [4] are taking part. However, there are still many pitfalls and major challenges they are faced with. Specifically unexpected situations in everyday road traffic have to be considered. Nissan [5] recently even plans to install a remote call center for providing human assitance for dealing with those situations and even has lost the hope of eventually developing a full autonomous vehicle one day. [6] One type of these critical situations is given by road construction sites. A road construction site is characterized by a broad range of unforeseeable temporal changes to the infrastructure that can invalidate any preexisting map information for navigation purposes. Furthermore, an undefined behavior or slow reactions on infrastructure changes can directly affect the road safety or obstruct the traffic flow. The present work specifically focuses on this problem domain, and examines how deep learning can be a part of a solution towards a fully autonomous handling of those situations. For this purpose, a dataset of hundreds of road construction site images has been created. Only images of construction sites that comply with german rules and regulations and images taken at daytime are considered. Further, a deep convolutional neural network (CNN) has been designed and trained to detect road construction sites as well as the corresponding changes to the actual driving lanes on a pixel level. The final outcome of this work basically demonstrates the powerfulness of deep learning with respect to the domain of construction sites and thus provides a proof of concept for future developments in autonomous driving.

[1] `www.tesla.com` (visited on 2/19/17)
[2] `http://volkswagenag.com/` (visited on 2/19/17)
[3] `https://abc.xyz/` (visited on 2/19/17)
[4] `www.apple.com` (visited on 2/19/17)
[5] `http://www.nissan-global.com/EN/COMPANY/PROFILE/` (visited on 2/19/17)
[6] `https://www.wired.com/2017/02/self-driving-cars-cant-even-construction-zones/` (visited on 2/19/17)

Zusammenfassung

Aktuell laufen die Big Player der Autobranche um die Wette, um als erster ein vollautonomes Fahrzeug zu entwickeln. Die bekanntesten unter ihnen sind Tesla Inc. [7], VW AG [8]und sogar branchenfremde Unternehmen wie Alphabet Inc.[9] und Apple Inc. [10] mischen nun mit. Allerdings gibt es noch viele Fallstricke und große Herausforderungen, mit denen sie konfrontiert sind. Besonders unerwartete Situationen im alltäglichen Straßenverkehr müssen berücksichtigt werden. Nissan[11] plant momentan sogar die Einrichtung eines Remote-Call-Centers, um die Autos in entsprechenden Situationen aus der Ferne mit menschlicher Hilfe unterstützen zu können. Der Automobilhersteller gibt damit bereits die Hoffnung auf, eines Tages ein vollautonomes Auto ohne menschliches Zutun zu entwickeln.[12] Ein Beispiel für besonders kritische Situationen im Straßenverkehr sind Straßenbaustellen. Eine Straßenbaustelle zeichnet sich durch eine breites Spektrum an unvorhersehbaren temporären Änderungen an der Straßeninfrastruktur aus, welche dazu führen können, dass sämtliche vorhandene Karteninformationen für die Navigation unbrauchbar werden. Darüber hinaus kann ein undefiniertes Verhalten oder zu langsame Reaktionen des selbstfahrenden Fahrzeugs auf diese Änderungen die Verkehrssicherheit direkt beeinträchtigen oder aber z.B. den Verkehrsfluss behindern. Die vorliegende Arbeit konzentriert sich daher speziell auf diese Problemdomäne und untersucht, inwiefern Deep Learning als Teil einer Lösung für ein vollständig autonomes Fahrzeug beitragen kann. Zu diesem Zweck wurde ein Datensatz von hunderten von Straßenbaustellenbildern erstellt. Dabei werden nur Bilder von Baustellen berücksichtigt, die der deutschen Straßenverkehrsordnung entsprechen und bei Tageslicht aufgenommen wurden. Weiterhin wurde ein "deep convolutional neural network", kurz "CNN" (lit. Bedeutung: tiefes faltendes neuronales Netzwerk) entworfen und trainiert, um Straßenbaustellen sowie Änderungen von Fahrspuren auf Pixelebene zu erfassen. Das finale Ergebnis dieser Arbeit zeigt grundsätzlich die Stärke von Deep Learning im Bereich von Straßenbaustellen und stellt somit einen Proof of Concept für zukünftige Entwicklungen im autonomen Fahren dar.

[7] `www.tesla.com` (visited on 2/19/17)
[8] `http://volkswagenag.com/` (visited on 2/19/17)
[9] `https://abc.xyz/` (visited on 2/19/17)
[10] `www.apple.com` (visited on 2/19/17)
[11] `http://www.nissan-global.com/EN/COMPANY/PROFILE/` (visited on 2/19/17)
[12] `https://www.wired.com/2017/02/self-driving-cars-cant-even-construction-zones/` (visited on 2/19/17)

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben. Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Datum	Unterschrift

Declaration of authorship

I hereby certify that this work has been written by none other than my person. All tools, such as reports, books, internet pages or similar, are listed in the bibliography. Quotes from other works are as such marked. The work has so far been presented in the same or similar form to no other examination committee and has not been published.

Date	Signature

Contents

1 Introduction **1**

1.1 Methods . . . 3
1.2 Limitations of this work . . . 5
1.3 Description of contents . . . 5

2 Fundamentals **7**

2.1 Artificial neural networks (ANNs) . . . 7
2.1.1 Training . . . 9
2.2 Convolutional neural networks (CNNs) . . . 14
2.3 Fully convolutional neural networks (FCNs) . . . 18

3 Tool setup and methods **20**

3.1 TensorFlow . . . 20
3.1.1 Programming Model . . . 21
3.1.2 Partial Execution . . . 22
3.1.3 Auto gradient computation . . . 23
3.2 Python and TensorFlow . . . 24
3.3 Infrastructure as a Service (IaaS) . . . 24
3.4 Development strategy . . . 25

4 Construction site classification **27**

4.1 Bottom-up approach . . . 27
4.2 Classification task . . . 28
4.2.1 Data acquisition . . . 28
4.3 Implementation . . . 32
4.3.1 Scope of implementation . . . 33
4.3.2 Final architecture . . . 38
4.4 Results . . . 40

5 Construction site detection **48**

5.1 Bottom-up approach . . . 48

5.2 Detection Task 48
5.2.1 Interpretation as semantic segmentation task 51
5.2.2 Data annotation 52
5.3 Implementation 55
5.3.1 From convolutional to fully convolutional (3-4) 55
5.3.2 Loss (5) 57
5.3.3 Final detection architecture 59
5.4 Results 60
5.4.1 Accuracy metrics 60
5.4.2 Additional mean IU derived loss function 61
5.4.3 Accuracy results 61
5.4.4 Visual results 63

6 Conclusion **70**

Bibliography **71**

List of Tables **75**

List of Figures **76**

Appendices **80**
Implementation 81
Classification architecture 81
Detection architecture 86
Attached media 91

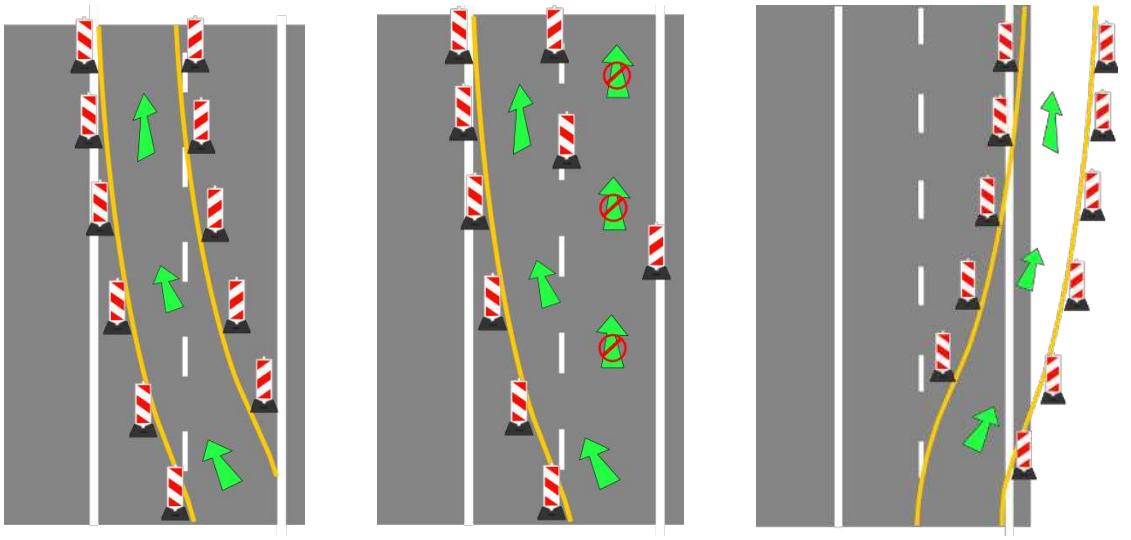
1 Introduction

> "This gets to the central challenge of autonomous driving: : How do you teach machines to deal with the chaotic, grubby humanity of our roads, where the rules bend so easily?"[1]

This quotation in essence outlines the specific characteristics of the challenges that still have to be overcome to eventually enable full autonomous vehicles to drive without any human interaction. Currently almost any larger automotive group extensively does research in the field of autonmous driving, such as Tesla Inc. [2], VW AG [3] and Nissan [4]. Even non-automotive companies, such as Apple Inc. [5] or Alphabet Inc. [6] are researching and developing autonomous vehicles. [7]

However, there are still lots of big challenges to deal with. As the introductory quotation indicates, these challenges are mainly characterized by the human factor and in general by the chaotic reality itself. Although a broad range of laws and regulations exist that should regulate the road traffic, various unexpected situations occur that do not fit those constraints. Examples for these situations are road construction sites. For example, a self-driving vehicle equipped with detailed map information and a GPS sensor could be considered to be able to autonomously drive the road. However, as soon as a road construction site occurs, which blocks a lane or places new lane markings, or the traffic is diverted, it has no idea how to react to the changed circumstances. Furthermore, in terms of safety and congestion avoidance, this behaviour has significant drawbacks. Specifically for dealing with these types of situations, the automotive group Nissan currently even plans to establish a remote call center to guide autonomous

[1] https://www.wired.com/2017/02/self-driving-cars-cant-even-construction-zones/ (visited on 2/19/17)
[2] www.tesla.com (visited on 2/19/17)
[3] http://volkswagenag.com/ (visited on 2/19/17)
[4] http://www.nissan-global.com/EN/COMPANY/PROFILE/ (visited on 2/19/17)
[5] www.apple.com (visited on 2/19/17)
[6] https://abc.xyz/ (visited on 2/19/17)
[7] https://www.cbinsights.com/blog/autonomous-driverless-vehicles-corporations-list/ (visited on 2/19/17)



(a) Lane changes to opposite lane

(b) Vertical panels not properly set up

(c) Road continues next to the actual road

Figure 1.1: Example scenarios

vehicles through those confusing situations. [8]

The present work focuses right here and tries to facilitate the automation of these situations to some extent, by visually detecting road construction sites. The goal is not only to detect corresponding construction site elements like vertical panels, construction vehicles and many more, but in conjunction to provide also important information on changes to the drivable road and changes to the lanes as stated above, to optimally supporting an autonomous vehicle with relevant information. Thus the objective of this work can also be considered as a visual construction site detection combined with lane detection support specifically aiming at road construction sites.

Figure 1.1 illustrates some example scenarios that have been taken into account. 1.1 (a) shows an example scenario, where the driving lane has been laid to the left, which has actually been the current driving lane. A self-driving vehicle, as characterized above, would be overstrained by this unexpected situation and probably would stop, as there are several obstacles (the vertical panels of the construction site) on the road. Besides the need of human intervention, also the road safety could be affected with respect to other road users.

1.1 (b) outlines a further situation, which can be even more dangerous for a self-driving car without specific knowledge on real-world road construction sites. It represents a

[8] `https://www.wired.com/2017/02/self-driving-cars-cant-even-construction-zones/` (visited on 2/19/17)

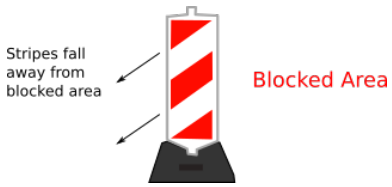


Figure 1.2: Example of construction site specific rules and regulations

slightly modificated situation with respect to 1.1 (a). The current driving lane is not fully blocked by vertical panels. A vehicle without the ability to recognize the situation precisely, could classify both panels placed at the roadside simply as smaller obstacles and nonetheless drive towards the construction site. Construction site detection systems that focus only on single elements like vertical panels and their placement and orientation with respect to the laws and regulations have their drawbacks. Figure 1.2 for instance, illustrates the rule of placing a red-white striped vertical panel correctly. The stripes have to fall away from the blocked area (the construction site). Transferred to the example situation of 1.1 (b), the panel on the right would indicate that a construction site is located only at the right roadside. Then the placement of the following panel on the left could indicate that there is a further construction site on the opposite lane. Eventually it would take the same dangerous path as the vehicle without any construction site detection would. Thus, this example scenario only outlines possible imperfections that have to be dealed with in reality and which are specifically addressed by the present work. One last slightly adjusted scenario is given by 1.1 (c). Here the current driving lane has even been laid to the right of the edge of the road. This change to the road actually requires completely new detailed map information to successfully navigate along the construction site. The detection approach provided by the present work is intended to also support these situations with relevant information.

1.1 Methods

For achieving the described objectives of this work, a technique being referred to as *deep learning* is used. This technique is characterized by using deep convolutional neural networks , which represent a state-of-the-art approach for visual recognition and detection [aLoBeHnt]. The key property of this approach is the ability of deep learning, to learn important visual features and generalize relevant classification information from large datasets without much engineering by hand [aLoBeHnt].

As figure 1.3 illustrates, the motivation behind applying deep learning is also the fact that construction sites have, compared to everyday road scenes, specific inherent striking

Figure 1.3: Construction sites are characterized by specific features

features and visual properties. This is not only due to the laws and regulations, which stipulate that specific elements have to be present, like signs with loud colors and specific patterns. Also the fact that construction work affects the usual road environment, for example by excavating the road, this kind of road scene is highly specific to road construction sites.

As figure 1.3 indicates, the overall approach of the present work is completely based on visual information. This is explained by the mentioned visual characteristics of road construction sites, but also is driven by the fact that any self-driving vehicle usually at least has access to high resolution camera images, as they are cheap and available in a high temporal resolution.

For the purposes of this work, a training dataset of construction site images has been collected and annotated. The open source deep learning framework TensorFlow[AAB+16] has been used to implement specifically adapted neural network models to solve the targeted detection task. The focus of the implementation and design has specifically been on a cross-platform capability as well as a fast execution time in order of being representative and providing a suitable proof of concept.

1.2 Limitations of this work

As already stated in the introductary section, this work does not claim to provide an overall solution to the broad spectrum of dealing with real-world road construction site environments. It specifically focuses on the static properties of a construction site and tries to outline a possible approach using deep learning, to deal with the two most important static types of information related to a construction site: The place and spatial extent of the construction site as well as the place and spatial extent of the drivable road with respect to the construction site. The particular dynamics, such as moving construction site vehicles and construction workers, or very detailed information on additional important elements, such as particular installations of traffic lights, are not taken into account.

In essence, the present work is considered to provide a proof of concept of applying deep learning techniques to the highly topical problem of road construction sites with respect to the development of autonomous driving vehicles.

Finally, the present work only deals with construction sites that correspond to the german road traffic regulations and only covers road construction sites by day and not by night.

1.3 Description of contents

The present work is roughly divided up into three parts.

The first two chapters 2 and 3 deal with the specific fundamental concepts, tools and methods that are used throughout the present work.

The following chapters 4 and 5 then describe how the targeted proof of concept is realized in detail. This includes the collection and preprocessing of the required training datasets as well as the concrete design and implementation of a suitable deep learning architecture. Both chapters follow a bottom-up approach and build upon each other. In 4, first a deep learning architecture for classifying road construction site images is designed and implemented. In 5, this solution is then extended to the targeted detection task architecture that finally represents the proof of concept.

Finally, the conclusion chapter (6) sums up the most important results and insights gained by this work. Furthermore, it briefly outlines potential future work and discusses how the provided proof of concept can be extended to act as a more practical and applicable solution.

The appendix (6) contains additional material with respect to the results of this work.

It gives an overview of the content of the attached medium, which contains all source code files, datasets, training outputs as well as additional documenting material, such as a detailed HTML documentation with respect to the training results and other.

2 Fundamentals

2.1 Artificial neural networks (ANNs)

Artifical neural networks form the absolute basis of every *deep learning* approach. In the field of pattern recognition they provide a powerful concept for classifying patterns and resolving complex problems [SK13]. Their key strength derives from the general ability to approximate any function [KKvdSS93] by a self-adaptive trainable process, and therefore also allows to learn complex input-output relationships [SK13] (e.g. decision boundaries for classification).

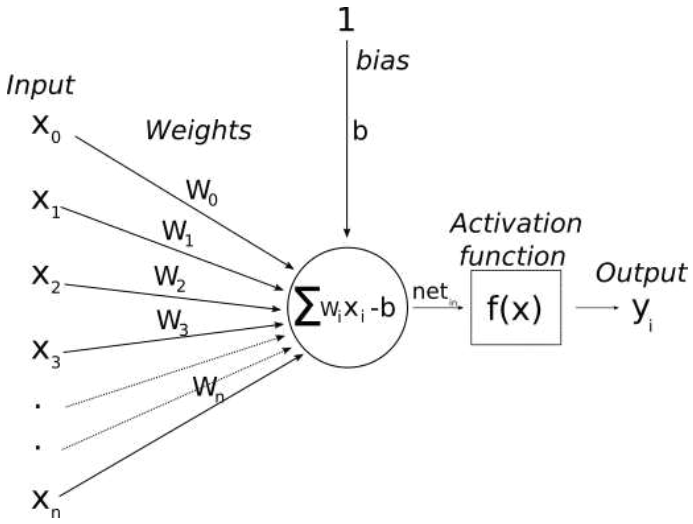
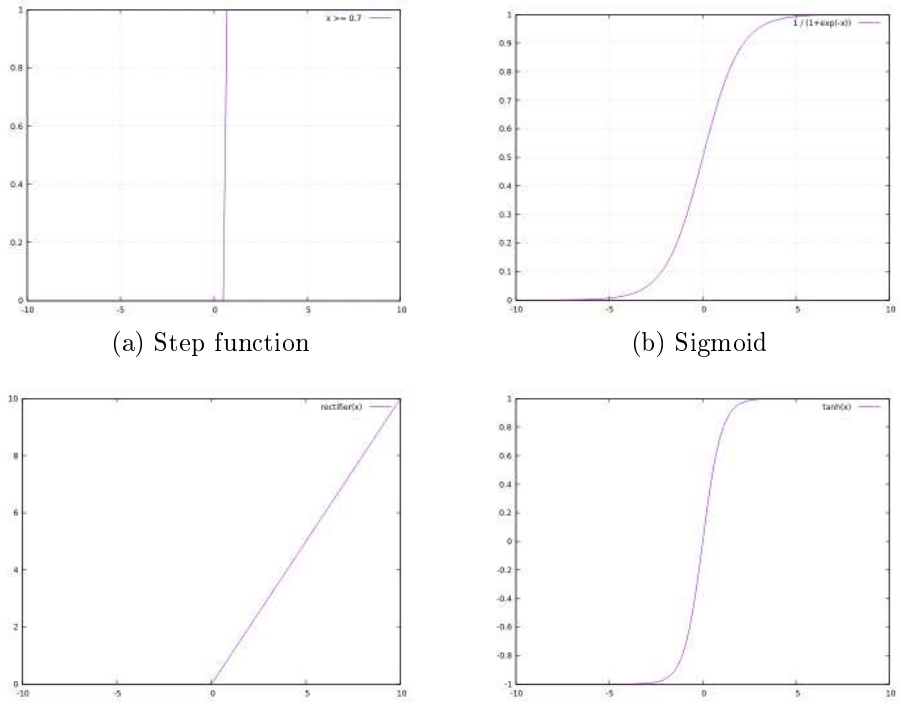


Figure 2.1: Artificial neuron (Perceptron)

In general, an ANN can be considered as a computational model that originally has been inspired by the human brain. The human brain consists of billions of neurons. They are interconnected by synapses and dendrites, and communicate using electrical impulses. An ANN is analogously built up of abstract representations of biological neurons, the artificial neurons (also called perceptrons). The artificial neuron acts as the most basic computing unit and building block of an ANN. Figure 2.1 illustrates the general structure. The neuron is connected with n input elements x_i by incoming edges



(a) Step function (b) Sigmoid

(c) Rectifier (d) tanh(x) - Hyperbolic tangent

Figure 2.2: Activation functions - examples

that are weighted with a factor w_i. The following component (comparable to the soma of the biological counterpart) simply computes the linear combination of those weights and inputs. An additional value, the *bias*, is then subtracted. It can be considered as the equivalent to the biological threshold potential. The computed value so far (net_{in}) is then further transmitted to the *activation function* $f(x)$. The overall output of the artificial neuron is then given by the output of $f(x)$.

Figure 2.2 shows some sample nonlinear activation functions. They all have in common that they reflect the behaviour of an activation function in terms of the activation potential of a biological neuron. Essentially they only have two output values corresponding to a logical "0" and "1" (the neuron is firing). The most simple is the step function $f(x) = \{1 \quad \text{if}\, x \geq 0, else\, 0$. Other ones are the rectifier $f(x) = max(0, x)$, the sigmoid $f(x) = \frac{1}{1+e^{-x}}$ and the hyperbolic tangent $f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

After defining the basic properties of a single artificial neuron, figure 2.3 illustrates a network of several single artificial neurons known as artificial neural network (ANN). Though the neuron of figure 2.1 also can be interpreted as the most basic ANN, only consisting of a single neuron.

There are basically three layer types. The first is the input layer, which receives the

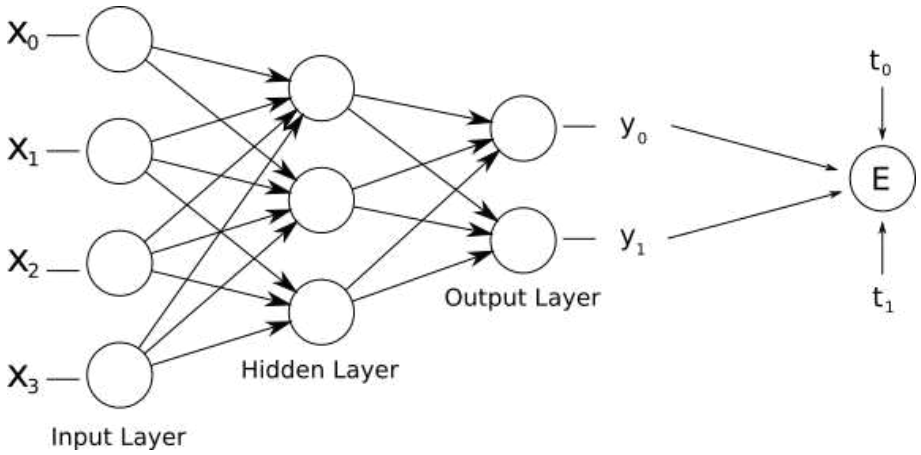


Figure 2.3: Artificial neural network (Multi-layer perceptron (MLP))

input values x_i. Then one or more hidden layers follow consisting of artificial neurons as described above, whereas any neuron is fully connected to any output of the preceding layer. Finally, one output layer delivers the overall network output, in this case the two outputs y_0 and y_1. Thus the network essentially computes a network function that maps the input vector x to an output vector y.

2.1.1 Training

Backpropagation

The main purpose of neural networks in this work is the application for classifying input patterns. Given an input pattern $x \in \mathcal{R}^n$, the corresponding output pattern $y \in \{0,1\}^m$ representing the required class information is requested.

Before applying further algorithmic steps, the network in 2.3 has first to be extended by an adequate loss function E (already visually added to the network in 2.3), which basically provides a measure of difference between the network output y and the actual desired output t. One of the most frequently used loss functions in the ANN literature is the squared-error cost function [JMM96]. With respect to the network in 2.3, the loss function is defined as: $E = \frac{1}{2}(y_0 - t_0)^2 + \frac{1}{2}(y_1 - t_1)^2$. Now that we have a measure of the error of the network, this error should be minimized. This is achieved by updating the weights of the network, which can be referred to as *learning*.

The concrete method to accomplish this, is called the *backpropagation algorithm*, which essentially implements a gradient-descent method to minimize the error. The objective of this procedure is to compute for each weight w_i in the network the corresponding gradient of E with respect to w_i, $\frac{\partial E}{\partial w_i}$ as well as updating the weight as follows: $w' = w - \eta \frac{\partial E}{\partial w_i}$. w'

is the new value of w and η denotes the learning rate, which determines to what extent the weight w_i is updated towards the negative gradient direction.

The steps to calculate the corresponding gradients are basically determined by the chain rule for computing derivatives of compositions of functions. If, for instance the function $z = g(f(x))$ is given and $y = f(x)$, then the corresponding derivative $\frac{\partial z}{\partial x}$ is defined by the chain rule as follows: $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$. Expressed in words, it simply examines first to which extent z changes due to changes of the value of y. In a second step it examines, to which extent y changes due to changes of the value of x. The multiplication of both values then shows to which extent z is changed by value changes of x.

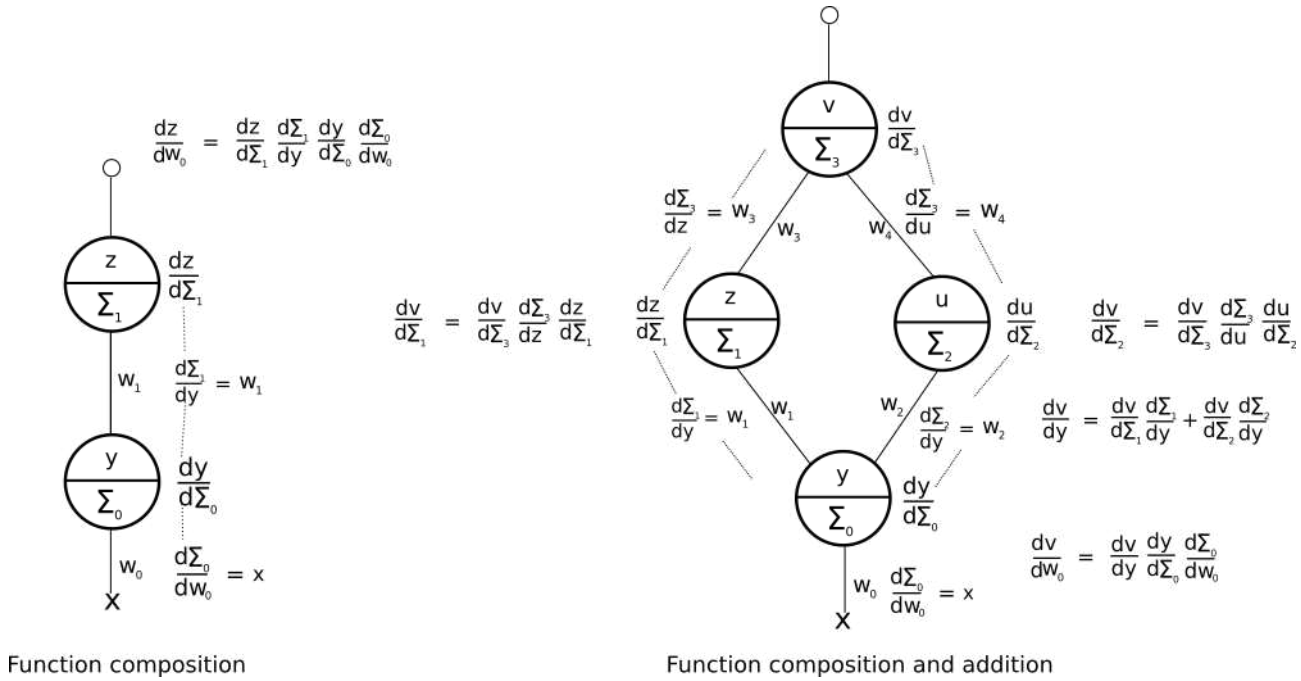


Figure 2.4: Backpropagation

Figure 2.4 outlines the general behavior of backpropagation to calculate the required gradients. The left computational graph shows an example of a composition of functions, the right one shows function compositions as well as the sum of them. Every node corresponds to an artificial neuron, consisting of the summation unit $\sum_i$ (already shown in 2.1) and the respective activation function, for example y.

The left example has the overall output z and wants to know the gradient of z with respect to the weight w_0, $\frac{\partial z}{\partial w_0}$. During the forward step, the gradients of the activation functions with respect to their input (the output of the corresponding summation unit) can already be stored and attached to the respective node. For instance, if y is the sigmoid function $sig(x)$, then $\frac{\partial y}{\partial \sum_0}$ can be computed during the forward step by

$sig'(\sum_0) = sig(\sum_0)(1 - sig(\sum_0))$. After computing the overall network output z in the forward step, the backpropagation step now traverses the graph backwards and simply multiplies the corresponding derivatives with regard to the chain rule. The final result is then the questioned partial derivative, which provides the basis for the update of w_0.

The right example has the overall output v and wants to know the gradient of v with respect to the weight w_0, $\frac{\partial v}{\partial w_0}$. The forward step is analogous to the left example. But, because y now affects the value of two following nodes, two corresponding gradients have to be computed and added in the backpropagation step. Thus, every time two edges "merge" during the backpropagation, their respective gradients have to be added. In this case the gradient of v with respect to y is computed as follows: $\frac{\partial v}{\partial y} = \frac{\partial v}{\partial \sum_1}\frac{\partial \sum_1}{\partial y} + \frac{\partial v}{\partial \sum_2}\frac{\partial \sum_2}{\partial y}$.

Both presented cases basically illustrate how the backpropagation works. One additional aspect to note is the computation of the gradient of the summation function with respect to a single corresponding edge weight w_i, $\frac{\partial \sum_i}{\partial w_i}$. This is simply the output of the preceding node. For example (see figure 2.4) $\frac{\partial \sum_0}{\partial w_0} = x$.

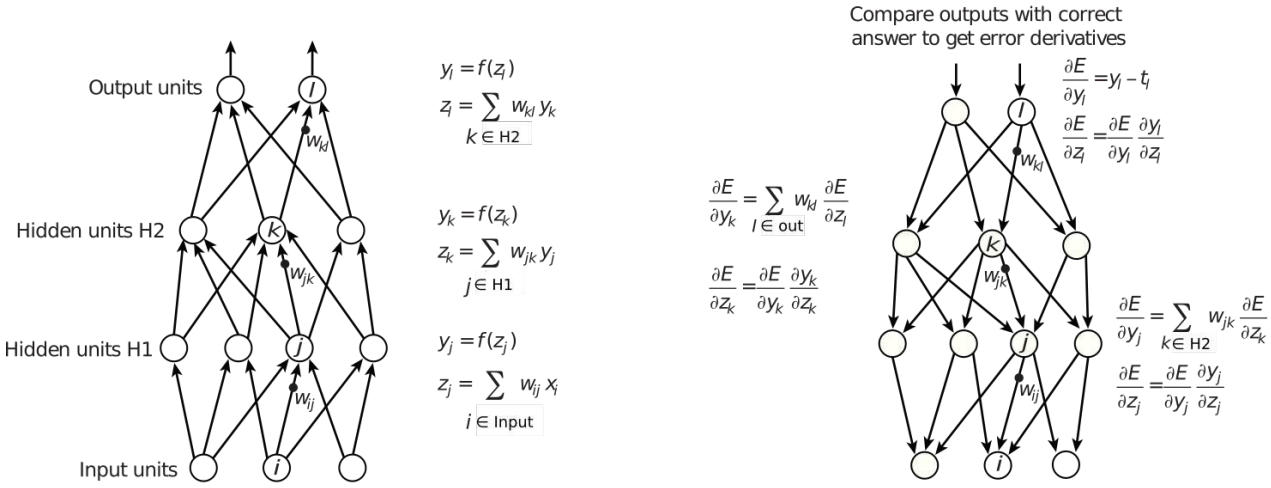


Figure 2.5: Backpropagation (relating to the squared-error cost function used above) [aLoBeHnt]

After illustrating the basic concepts of backpropagation, 2.5 further demonstrates it with respect to the square-error loss function used above.

Overall training procedure

As demonstrated above, for training purposes, several input-output patterns (x, t) are presented to the network. The most simple procedure is the stochastic gradient descent (SGD) procedure, which feeds a single input-output pattern to the network and then

updates the weights according to the gradient of this single example. Other approaches for instance, first calculate the overall gradient of the entire or a small subset (batch) of the input-output patterns. For the purposes of this work, the batch approach has been chosen. Furthermore, the training process is usually divided up into so called *epochs*. One single epoch corresponds to a full cycle of having fed the entire set of input-output patterns to the network. This cycle is repeated several times, until a desired local minimum of the loss function has been reached.

Generalization

The actual objective of training an ANN for classification is to leverage its capabilities of classifying and generalizing. After training the network with several known input-output patterns (the *training set*), it is intended to also output the correct class label for input patterns, whose correct classification output is unknown. Therefore, the network has to generalize in terms of the domain, which is represented by the *training set*.

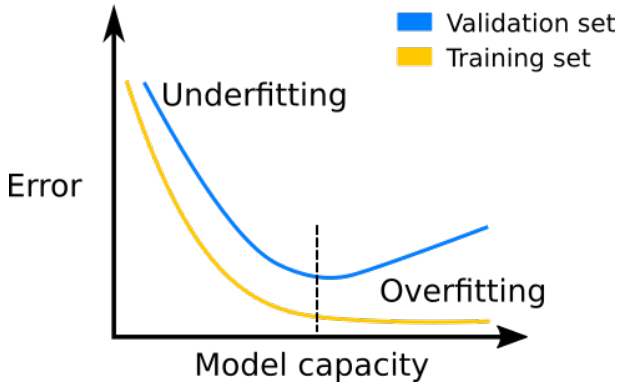


Figure 2.6: Over- and underfitting

For being able to measure the degree of generalization of a trained network, a further set of known input-output patterns called the *validation set* is created. The items of this set are not used for training the network and updating the weights. But they are periodically fed to the network (e.g. after each epoch) and an accuracy or performance metric is calculated accordingly. For classification purposes, the most simple accuracy metric is given by the number of correctly classified validation patterns, divided by the whole number of validation patterns available. Usually the same metric is also computed for the training set periodically. As figure 2.6 illustrates, both accuracy metrics can be compared afterwards. There are two extremes of missing the optimal capability of generalizing. One is called *overfitting* and another *underfitting*. In the case of overfitting, the network often posseses too much capacity in terms of too many layers or neurons.

It is characterized by high accuracy values according to the training set and relatively low accuracy values with respect to the validation set. This indicates that the network overfits the provided training data, which for instance includes the learning of irrelevant noise and details specific to only the training data items.

The second case is underfitting. Underfitting is characterized by low accuracy values with regard to the training as well as the validation set. In general, this indicates a network having too little capacity to represent the classification domain. Both concepts are useful indicators for adapting the structure of a network according to the predefined goal.

Apart from the above described ways of dealing with overfitting, there are some other well-known tools and techniques to overcome overfitting.

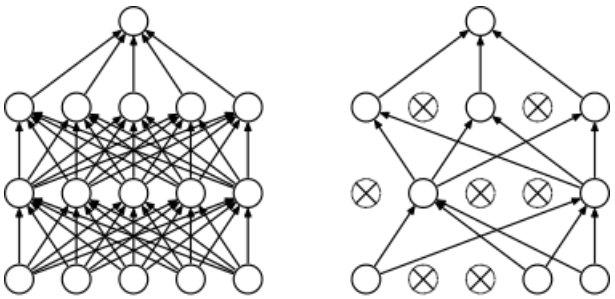


Figure 2.7: Dropout example [SHK+14]

Dropout layer One well-known technique specifically addressed to the problem of overfitting, is the use of dropout. Figure 2.7 shows a neural network before and after the application of dropout. The idea behind dropout is to randomly drop neurons and their connections from the network (while training), to avoid too strong co-adaptions of nodes in the network. This implicitly creates an exponential number of smaller sub-networks that are trained. Usually a parameter p is defined, which specifies the probabilty of dropping a single neuron or not. An often chosen value is 0.5 [SHK+14].

Regularization Regularization is another technique to deal with overfitting. The idea is essentially to add an additional penalty term to the loss function with respect to the concrete values of weights in the network. Generally it specifies a term that penalizes too large weights and thus in essence establishes a kind of weight constraint to the network [Pha]. The L_2-regularization is defined later in this work (4.3.2).

Data augmentation Finally, overfitting can also be caused by a scarce dataset, which is not large enough to represent the requested domain adequately. In these cases, there are several approaches and techniques to augment the existing dataset to artificially increase the number of training samples. In general, those techniques involve the adding of noise or various transformations to the existing training items. For the present work for instance, random horizontal flips have been applied to the training set, as a horizontal flip does not change the semantic information, which is inherent to an image.

2.2 Convolutional neural networks (CNNs)

After characterizing the fundamental basics of artificial neural networks in general, this section introduces a special type of neural networks, the so called *convolutional neural network (CNN)*.

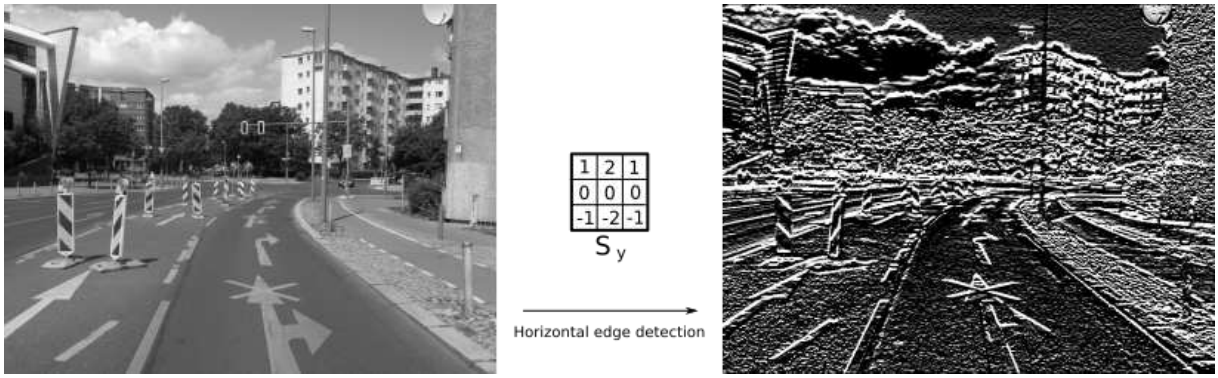


Figure 2.8: 2d Convolution Example

The term *convolutional* stems from the image processing operation named *convolution*. Figure 2.8 shows an example of applying a convolution using the sobel operator filter S_y to the left picture. The output of this operation is a picture representing specific features with reference to the original picture. In this case, horizontal edges have been filtered (extracted).

The Convolution operation for the two-dimensional case is defined as

$$I'(x,y) = I * g = \sum_{i,j} I(x-i, y-j)g(i,j).$$

I denotes the original image, g is the convolution kernel (e.g. S_y as defined in 2.8) and $I'(x,y)$ denotes the new pixel value of the resulting image I' at the position (x,y). Essentially, the kernel and its concrete values are able to specify and detect a specific

local feature of interest in the corresponding input image.

The concept of CNNs leverages this property of the convolution operation and transfers it to the field of artificial neural networks.

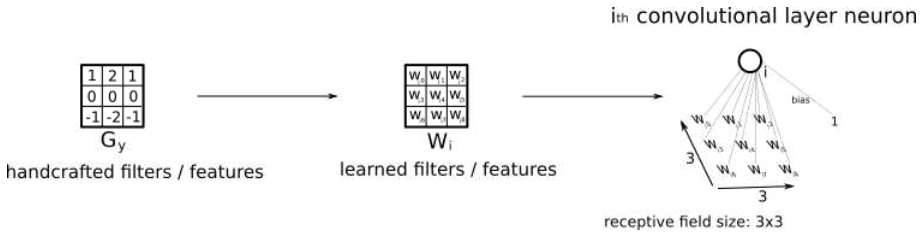


Figure 2.9: 2d Convolution Example

The basic and powerful idea behind this concept (illustrated by figure 2.9)is to replace hard wired kernel parameters by adaptable weights that can be learned using ANNs. Thus a CNN essentially defines a new type of neuron, which only contains the summation unit and is not fully connected to its input values. It only has a few weights compared to the overall input size. The local area of its concrete application regarding the input (e.g. an image) is called the *receptive field size* (see figure 2.9).

Basically, such a neuron is used in the exact same way as a convolutional kernel. First, the convolution of the input and the weights of the newly defined neuron (acting as the convolution kernel) is computed. Then additionally, the corresponding bias is added element-wise to each output value. This produces an output similar to the convolution output of figure 2.8, which is called a *feature map*. This name indicates the actual purpose of the neuron to derive local features from the input. As the neuron and its weights are slidden over the input and in effect compute local features, this is also called weight sharing as the weights are shared at each local position of the input and are not fully connected to each input element as in common ANNs. The number of pixels moved in each direction while sliding over the image is called the *stride*.

Figure 2.10 finally shows the basic structure and usage of a single convolutional layer in conjunction with the following ReLU layer and the optional subsampling layer consisting of a MaxPool layer. The top left of the figure shows the input consisting of a three channel (r,g,b) image. To the right there is a first convolutional layer specified. Essentially it is formed by n single convolutional layer neurons as illustrated in 2.9. As already mentioned, each of those convolutional layer neurons is convolved with the input and thus produces a corresponding featuremap. Given n convolutional neurons, the output is a batch of n featuremaps. It is important to notice that a convolutional neuron always has weights to cover the full depth of the input volume. In this example, the input image has three channels and therefore a depth of three. If the receptive field of the neuron

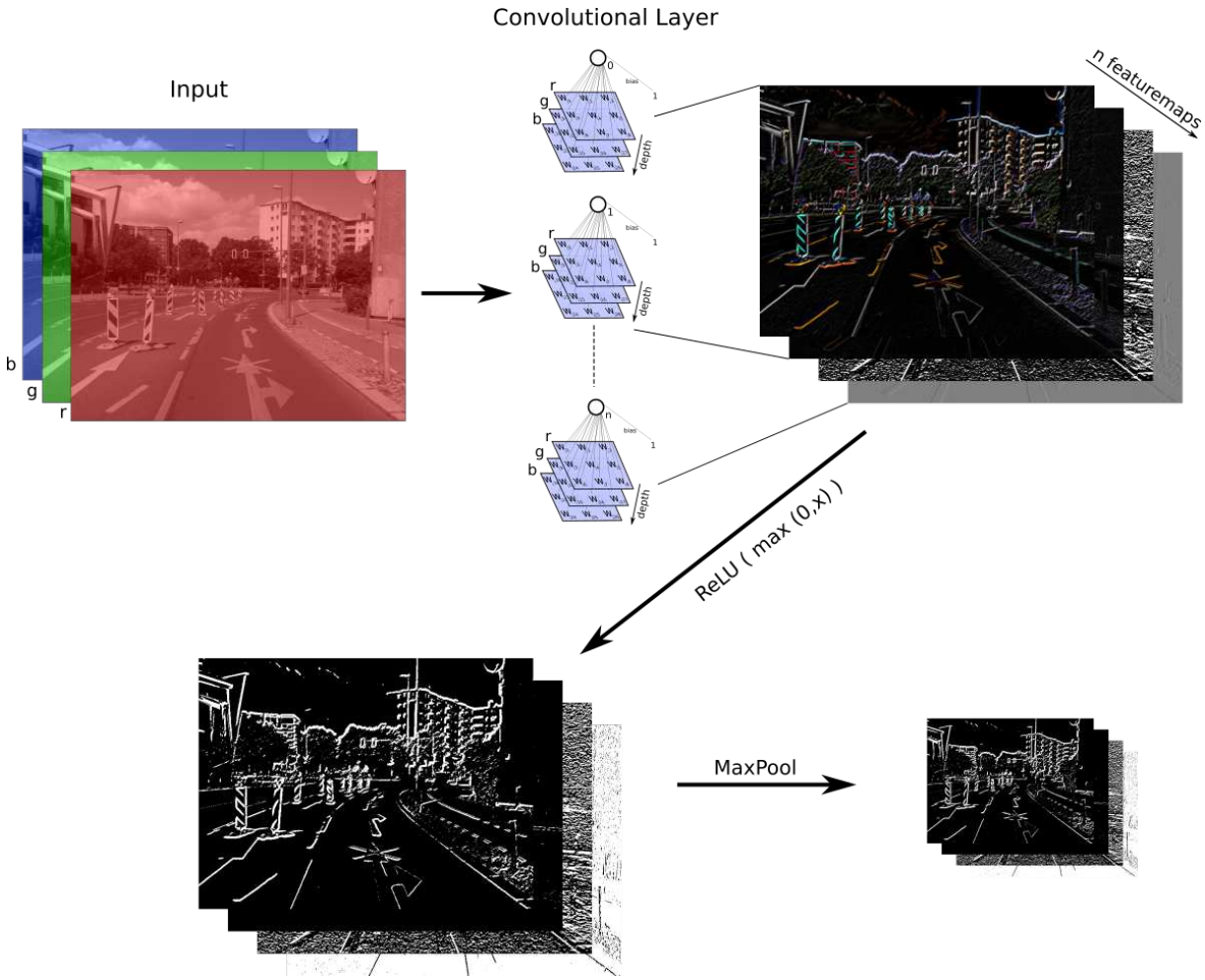


Figure 2.10: From convolution to convolutional neural networks

is of size 3x3 pixels, then it has to provide $3 * 3 * depth = 3 * 3 * 3 = 27$ corresponding weights to cover the receptive field up to the required depth. As the weights of different neurons have different values, the n featuremaps represent n different types of local features of the same input. The overall size of the output is given by the input size, the number of convolutional neurons and the specific convolution behavior. The convolution is further specified by the stride (as already mentioned), as well as optional zero padding. For instance, in order to keep the width and height dimensions of the input image, the border of the input has to be padded with zero values. For this work, always a stride of one and zero padding to keep the input width and height has been applied. Thus if width and height remain the same, the output has the dimensions width x height x n (number of output feature maps).

After the n output featuremaps have been created, a $ReLU$ layer follows , which effectively calculates the rectifier activation function, already shown in 2.2. It essentially

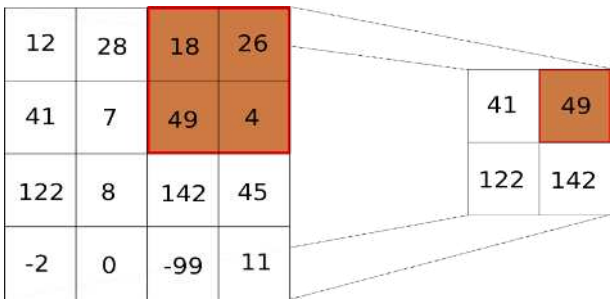


Figure 2.11: 2x2 MaxPool layer

represents the activation function for the preceding convolutional layers. The last optional layer is a subsampling layer, which reduces the size of the output of the ReLU layer. For this purpose the so called *MaxPool* layer is often used. A 2x2 MaxPool layer with a stride of 2 pixels in all directions, for instance, determines the maximum activation value of each two pixels by two pixels area (see figure 2.11).

In summary, now the basic building block of a CNN, a compound of a convolutional, a ReLU and an optional subsampling layer like MaxPool has been outlined.

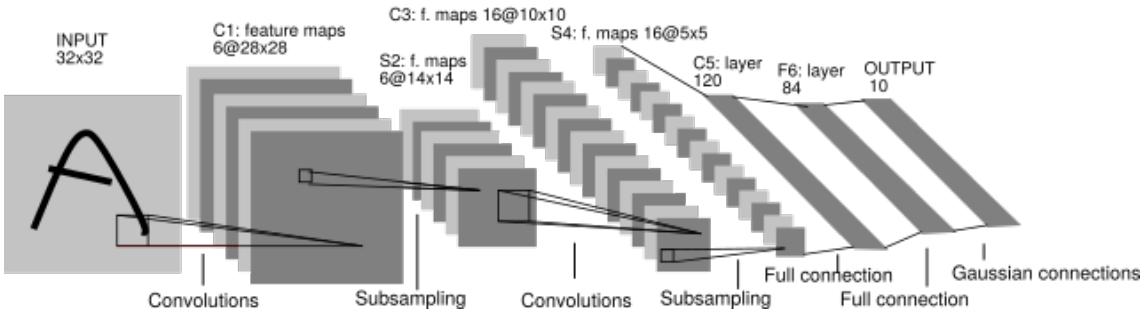


Figure 2.12: CNN overview (LeNet-5 [LBBH98])

Figure 2.12 illustrates the general overall architecture of a convolutional neural network. First, there are several consecutively arranged layer compounds, as described above (in this example only two). Finally, a few fully connected layers as known from common ANNs are added to the network. The basic idea is as follows: The first part of the network that consists only of CNN specific layers, form the feature learning part. The fully connected part at the end, forms the corresponding classification part.

CNNs are mostly used to recognize and classify visual input data. While the first convolutional layer extracts very basic features, such as edges, corners, lines, colors or other fragments, the next convolutional layer already is able to combine the features of the first layer to more complex features. This process continues with the number of

stacked convolutional layer compounds. This is the reason why CNNs are predestined for a broad range of visual tasks. As any class of visual objects basically is characterized by its visual features and its specific arrangement, a CNN is generally able to find and extract the important features.

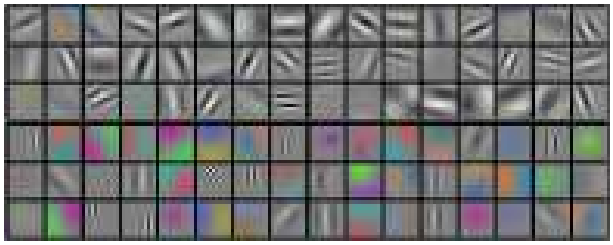


Figure 2.13: Examples of layer 1 filters [KSH12]

Figure 2.13 shows some examples of learned filter weights of the first convolutional layer and represents the variety of different basic features used to build up more complex features in the following layers.

Finally, the power of CNNs compared to handcrafted approaches especially lies in the automated learning of the relevant features, which enable a high accurate classification of input patterns and which are used in this work.

2.3 Fully convolutional neural networks (FCNs)

Another special type of CNNs applied in this work is the so called *fully convolutional neural network (FCN)*.

The common CNN architecture illustrated by 2.12 requires a convolutional part, which provides the ability of hierarchically learning features, and a classification part, which usually consists of several fully connected layers.

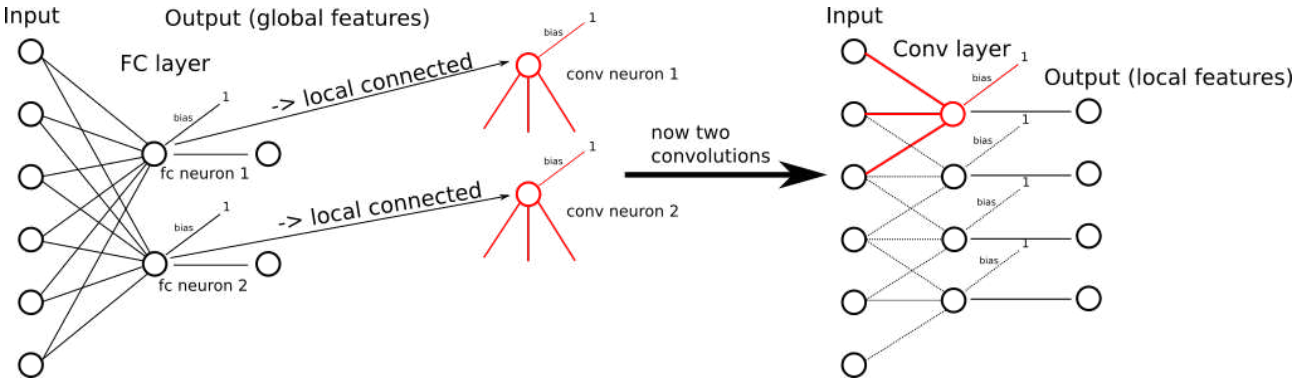


Figure 2.14: Transform fully connected into local connected convolution layers

The FCN, however, completely leaves out the use of fully connected layers. Instead, it simply uses convolutional layers, too. Figure 2.14 demonstrates, how fully connected layers can generally be replaced by their convolutional counterparts. While the neuron of a fully connected layer is fully connected to any input value, the convolutional equivalent is only connected to a small subset of the inputs and also produces an output value for every small subset, that lies within the receptive field. Thus a transformed fully connected layer can be considered to change the focus from a global point of view to a local point of view. This architectural rearrangement, for instance, allows to simply gain spatially local classification information, which is also used for the purposes of the present work.

3 Tool setup and methods

The following sections give an overview of the basic tools, frameworks, methods and execution environments utilized for the development of the desired convolutional neural network. All components have been chosen to meet the specific requirements of deep learning while focusing on flexibility, minimality and effectiveness with respect to the development process.

3.1 TensorFlow

The most important component used for the development process is the open-source deep learning framework TensorFlow [AAB+16], which has been released in November, 2015. It has been developed by the Google Brain project[1] based on the experiences with an earlier framework called DistBelief [DCM+12]. The experiences incorporated applications in the fields of object classification and detection, speech recognition, move selection for Go and other areas. It has also been successfully applied to various products of Google Inc. and Alphabet Inc. [AAB+16]. Due to its development and influences, TensorFlow offers a wide range of applicability and well-engineered methodologies to implement, train and inference from machine learning models. Besides TensorFlow there are several other frameworks also dealing with the specific requirements of deep learning, such as Theano [2] or Caffe [3] . However, TensorFlow has been chosen, as it provides an adequate combination of qualities supporting the development as well as potential usages of the final outcome.

As the development process for a particular deep learning application is still quite experimental, a comparatively fast compile time [Gol16] as well as the ability to quickly probing concepts and ideas by leveraging interactive sessions via the Python interface [4][5] serves this needs. Furthermore, the computational model as well as the range of low-level

[1] `https://research.google.com/teams/brain/` (visited on 10/18/2016)
[2] https://github.com/Theano/ (visited on 10/18/2016)
[3] https://github.com/BVLC/caffe (visited on 10/18/2016)
[4] `https://docs.python.org/2/tutorial/interpreter.html` (visited on 10/18/2016)
[5] `https://www.tensorflow.org/api_docs/python/tf/InteractiveSession` (visited on 10/18/2016)

to high-level building blocks (single weights to complete layers) allow a flexible design and implementation [AAB+16].

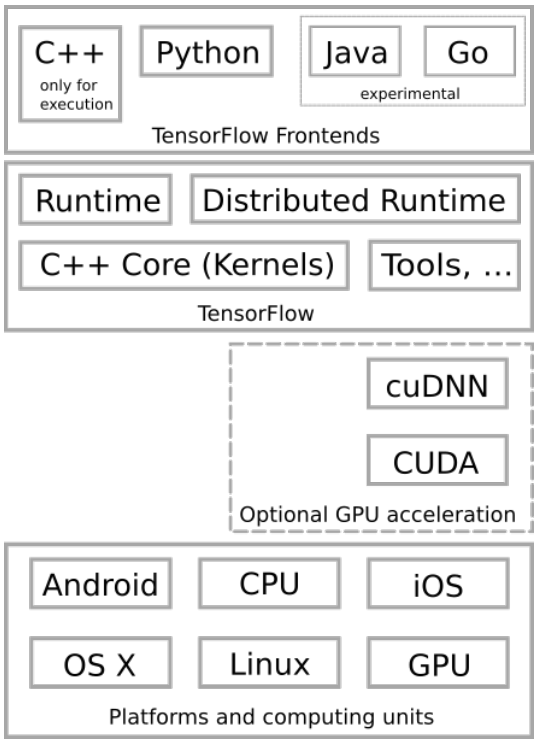


Figure 3.1: TensorFlow stack

Regarding the target platforms, it specifically focuses on a flexible (even distributed) deployment on various different platforms and devices, such as GPU, CPU, server, desktop, mobile platforms as well as embedded systems by supporting the ARM architecture [Sch16][BRSS15]. This in principle makes it possible for a later reuse of the final outcome to work on arbitrary systems, such as autonomous vehicles. Figure 3.1 illustrates possible configurations.

3.1.1 Programming Model

The core concept of TensorFlow is given by its declarative programming model, which focuses on the structure of the intended network or computation, and does not involve any statements in an imperative manner. This level of abstraction essentially enables the framework to gain a high level of flexibility [AAB+16].

Any TensorFlow computation or machine learning model is basically represented by a directed data flow graph (Figure 3.2). It consists of computational *nodes* that are connected by edges. When a model is finally being executed, so called *tensors* flow

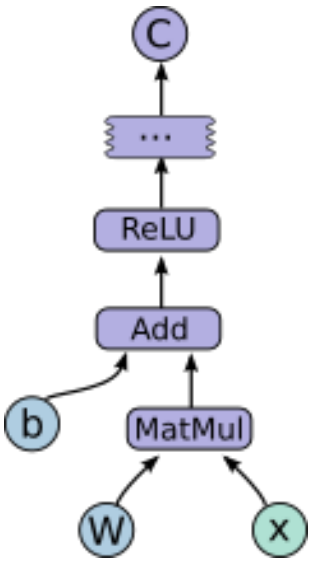


Figure 3.2: Example of a TensorFlow data flow graph. [AAB+16]

along the edges, which represent the processed data and are implemented by typed arrays of arbitrary dimension [AAB+16].

By defining the applied *nodes* as instances of so called *operations*, this computational model is fully decoupled from its concrete implementation. *Operations* are conceptual, abstract computation entities named as "Add", "MatMul". Their concrete implementation in turn is realized through entities called *kernels*. Thus an operation can be implemented by one ore more*kernels* supporting different platforms and devices. This precise architectural property not only increases the level of flexibility and cross-platform compatibility, but also allows the final model to be executed efficiently, as kernels are implemented in C++ and allow to adapt and optimize for specific devices, such as GPUs. Finally it is worth mentioning that without exception any object, calculation or provided procedure is represented as an *operation*. Even variables, constants and control flow instructions are defined as *operations* as well as placeholder *nodes* that take the required input for processing [AAB+16].

3.1.2 Partial Execution

After building the desired data flow graph using the python interface, a *session* object provides the *Run* operation, which expects parameters, such as desired output nodes as well as input data fed to input placeholder nodes [AAB+16].

Figure 3.3 illustrates this concept. In this case the client wants to get the network output of node **f** while providing input data for the placeholder node **b**. Before getting

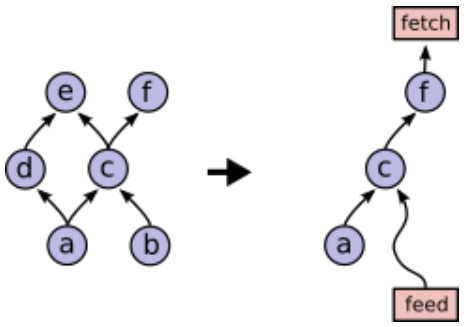


Figure 3.3: TensorFlow partial execution. [AAB+16]

outputs, the graph is transformed to the corresponding subgraph only containing those nodes which are required for computing the output. The input data is fed and the transformed data flow graph executed [AAB+16].

For simplicity this work has utilized this feature for integrating associated training and inference tasks in one data flow graph. Both tasks then only differ in the required output nodes handed to the *Run* operation [AAB+16].

3.1.3 Auto gradient computation

Apart from providing an intuitive and modular principle with the concept of a data flow model, one further advantage is the handling of gradient computation and back-propagation.

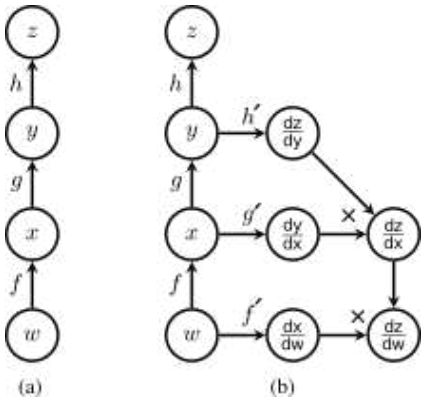


Figure 3.4: Automatic gradient computation [Gol16]

Figure 3.4 illustrates that TensorFlow does not provide a special mechanism for these types of application, but simply adds corresponding *gradient nodes* (*operations*) to the data flow graph. In this example the gradient of z with respect to w has been requested. After automatic addition of the required *gradient nodes*, and the forward step

of executing the graph from w to z, the data flow graph is traversed along the newly added *gradient nodes*, effectively computing the chain rule and thus serving for back-propagation [Gol16].

3.2 Python and TensorFlow

As already stated in the last section, Python is used to construct and execute machine learning models with TensorFlow. Furthermore, TensorFlow provides a good integration with NumPy[6], an open-source library for numeric and scientific programming in Python: The input to a placeholder node in Tensorflow can be provided by a NumPy array structure as well as output nodes are able to output NumPy arrays. [Gol16]

Since there are several additional steps besides the construction and execution of the machine learning model, such as data preparation (e.g. image cropping and scaling), logging and data visualization, some supplemental libraries or tools are required. Due to the fact that Python already supports a wide variety of optimized scientific frameworks and libraries [MA11], TensorFlow and Python are already a sufficient toolset.

One example open source framework is SciPy[7], which includes sub-packages, such as Matplotlib [8], NumPy and pandas [9] already forming a numerical computing environment similar to Matlab [MA11]. Another library specifically supporting image processing, is the Python Imaging Library (PIL) [AB09] , which also deals with NumPy arrays.

3.3 Infrastructure as a Service (IaaS)

Since the training of deep neural networks is a computational intensive task [CWV+14] that even takes up to several days for various applications [KSH12], specialized hardware, such as GPUs (optimized for vector and matrix computations) is needed.

As Figure 3.1 indicates, TensorFlow explicitly supports GPU acceleration by optimized *kernel* implementations using the NVIDIA® CUDA® Toolkit [10] together with the NVIDIA CUDA® Deep Neural Network library (cuDNN) [CWV+14] [AAB+16].

In order to leverage these capabilities and speed up the training process if needed, this work has made additional use of infrastructure as a service (IaaS). A suitable virtual

[6] `http://www.numpy.org/` (visited on 11/12/2016)
[7] `https://www.scipy.org/` (visited on 11/12/2016)
[8] `http://matplotlib.org/` (visited on 11/12/2016)
[9] `http://pandas.pydata.org/` (visited on 11/12/2016)
[10] `https://developer.nvidia.com/cuda-toolkit`, visited on (12/21/2016)

machine image [11], including the mentioned types of software, is already available for hosting at Amazon Elastic Compute Cloud (EC2) [12]. Amazon EC2 is a cloud computing service of Amazon.com, Inc.[13] that allows to request the instantiation and use of a predefined virtual machine on various hardware platforms. The virtual machine for training has been hosted as a g2.2xlarge[14] instance that has access to an NVIDIA GRID K520 GPU [15], supporting CUDA and cuDNN. The financial costs incurred have been funded by the Amazon Educate Program for students [16].

3.4 Development strategy

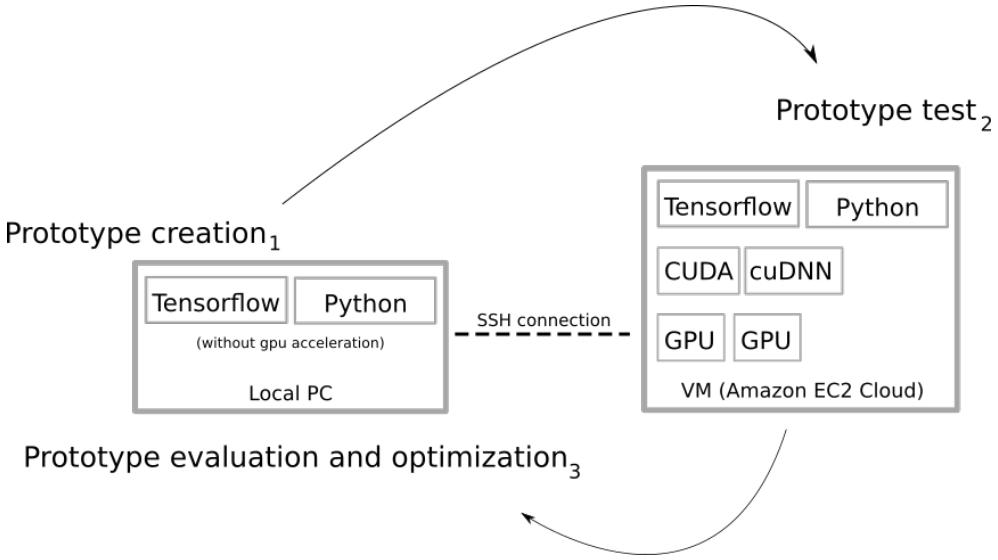


Figure 3.5: Tool setup and development strategy

Due to the high computational costs and IaaS approach mentioned in 3.3, the present work has followed an adapted, prototypal development strategy (see Figure 3.5). The single iterative steps are characterized as follows:

Prototype creation$_1$ In the first step, a data flow graph (the prototype) is constructed using the Python API interface of TensorFlow. As the local PC has only sparse computing resources (only CPU support) just the first training steps or epochs are

[11] Amazon Machine Image (AMI) ID: ami-e357bb8c
[12] `https://aws.amazon.com/ec2/` (visited on 01/14/17)
[13] `https://www.amazon.com` (visited on 01/14/17)
[14] `https://aws.amazon.com/de/ec2/instance-types/` (visited on 01/14/17)
[15] `http://www.nvidia.com/object/cloud-gaming-gpu-boards.html` (visited on 01/14/17)
[16] `https://aws.amazon.com/de/education/awseducate/` (visited on 4/23/17)

locally run to see if the network basically works and outputs reasonable values. A highly oscillating loss, for example, can already indicate that the chosen learning rate is too high and can thus further be modified locally.

Prototype test$_2$ As soon as the prototype of step one basically works, the data flow graph as well as all associated hyperparameters are transferred to the Amazon EC2 instance, which has been specified in 3.3. Now the same code is being executed with GPU support and therefore allows a full training of 100 epochs within one hour or less. During the training process, several metrics as well as loss values and other network outputs are written to disk. When the training is finished, all written log files and output data are transferred back to the local PC.

Prototype evaluation and optimization$_3$ In this step, all log information and outputs provided by step two are analyzed. For example, tendencies of over- or underfitting can be determined by comparing training and validation accuracy, or oscillating loss values in later epochs can indicate an improvable update mechanism. Based on these insights, architectural as well as behavioral modifications are specified. Treating overfitting, for instance, could involve reductions of layer sizes or an enhanced input preprocessing step. After the required changes are fixed, the development process continues with step one, building the new prototype.

4 Construction site classification

4.1 Bottom-up approach

The objective of the present work is to build and train a convolutional neural network model for the task of detecting construction sites by using camera images as training data, while the design of a working CNN architecture is by far the most challenging task. There are potentially countless design decisions encountered in building an adapted CNN model being able to represent the requested domain. The selection of hyperparameters, the number, type, size and order of layers and their individual effect on the outcome are still subjects of current research. Besides some basic principles inherent to most successful architectures there is still a "current lack of guidance on design"[ST16a, p.1] in deep learning.

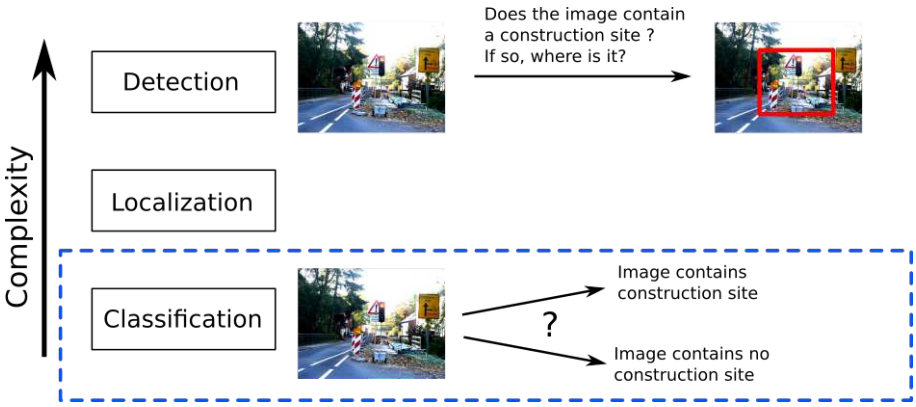


Figure 4.1: Computer vision tasks classification, localization and detection with increasing order of complexity

Facing this complexity of design, a *bottom-up* approach has been chosen for successively developing the final detection architecture (see figure 4.1). Since the task of detection can be decomposed into the less complex sub-tasks localization and classification [SEZ+13, p.3] it is reasonable to solve the least complex task first.

Therefore the following sections will first address the development of a CNN model for the task of construction site classification. Chapter 5 will then extend this model to

solve the detection task.

4.2 Classification task

The classification task is characterized by the question "does the image contain any instances of a particular object class?"[EGW+10, p.1]. Transferred to the domain of construction sites the question basically is "does the image contain a construction site?". Hence, to solve this task using a CNN model, there has to be a "positive" image set containing construction sites and a "negative" ("background") image set containing ordinary, everyday road scenes without construction sites. The trained model should finally be able to classify a corresponding image by assigning a score $s \in [0, 1]$. An image with a score $s \geq 0.5$ is then assumed to contain a construction site, while a value of $s < 0.5$ implies that there is no construction site visible.

The following sections describe the required steps to build up an adequate dataset and explains how the corresponding CNN archicture was designed and implemented.

4.2.1 Data acquisition

In order to build a robust classifier, there has to be a suitable number of training images representing the domain of road construction sites as well as images representing the "background" class. The "background" or "negative" class acts as the corresponding counterpart for the binary classification task. It is to be made up of images representing "everyday, ordinary road and traffic scenes" strictly excepting any construction site related content. The next section defines the desired properties for data sets of both classes, which have been taken into account for data collection and acquisition.

Requirements

A "suitable" number of images per dataset

While there is no explicit definition of "suitable" or "sufficient" amount of data required for building a robust classifier, the needed number of training samples can at least be considered in terms of density and diversity [ZLX+14]. To represent its corresponding domain well, a dataset has to be diverse and dense. On the one hand the diversity ensures that a dataset is able to represent the variety inherent to the specific domain and avoids too one-sided information. On the other hand the density aims at preventing too rigid concepts of the related domain.

Apart from these qualitative indicators this thesis roughly is oriented to the average number of training samples per category of 600 images of the ImageNet[RDS+15a] dataset, which acts as a lower quantitative bound.

Camera angle and point of view

As the desired construction site detection is built for autonomous vehicles, the training images should roughly reflect the typical point of view of a vehicle located on the road and the driver's view of the road.

Lighting conditions

The dataset is limited to daylight conditions.

Focus on german roads and construction sites

The dataset focuses on german roads and construction sites.

Diversity

The "positive" dataset should represent the variety of different possible road construction sites. Thus, the focus lies on different configurations of traffic signs, traffic lights, vertical panels and different types of roads like urban roads and highways.

The "negative" dataset should include everyday road scenes of different types of roads. For a more robust classifier it is important to particularly focus on potential boundary cases. This applies to images having properties typically inherent to construction sites such as loud colors (especially shades of red and yellow), stripe patterns, rough textures or construction vehicles (that currently not belong to a construction site).

Density

Both datasets should also provide images with great similiarities and minor differences. For instance this can be an image of the same or very similar construction site but with a different camera angle, point of view or traffic volumes.

Applying existing datasets

Due to the new rise of deep learning and its wide spectrum of applications, many open source datasets for various computer vision tasks in various domains have evolved. Generally they allow reusing big amounts of categorized and annotated data while at the same time being useful for benchmarking, comparing and evaluating state-of-the-art deep learning approaches in research. For the specific domain of road construction sites, several existing datasets have been reviewed with respect to the criteria mentioned above.

Among these are: ImageNet[RDS+15a], PASCAL VOC[EGW+10], Places2[RDS+15b], SUN[XHE+10], CIFAR[KH09], Cityscapes[COR+16] and Microsoft COCO[LMB+14]. None of them contains a corresponding category with images satisfying the defined requirements. Only nearby categories like construction sites in terms of building sites exist. The domain of road construction sites is still too specific.

Manual data collection

Figure 4.2: Sample images with construction site (positive class)

As existing datasets do not cover the domain of road construction sites yet, the needed dataset was initially constructed by utilizing the image search engines Google Images and Bing Images, yielding about 600 images per class. After removing duplicates using fdupes[Ló16], this dataset has been additionally supplemented by several manually taken pictures in Berlin, Germany. In total the collected dataset includes about 800 pictures per class, almost reaching the goal of 1,000 pictures per class. Figures 4.2 and 4.3 illustrate some examples.

Figure 4.3: Sample images without construction site (negative class)

Data preparation

The collected dataset contains pictures with an individual size of [width x height x depth=3]. However, training common CNN architectures requires input items of uniform size. This is due to the fully connected part. As any neuron of the first FC layer is connected to any output of the preceding Conv layer, the required number of learnable weights is originally determined by the input size. For a trainable CNN model this number and thus the input size has to be fixed. That is why the collected images have been processed to fit uniform, fixed sizes. For a greater flexibility of constructing an adequate model, every picture has been arranged to fit the different input size categories [32x32x3], [64x64x3], [128x128x3] and [256x256x3]. Figure 4.4 illustrates the process of data preparation. In a first step, every image was cropped into a left and a right (top and bottom with respect to the aspect ratio) part each having a square size. Images with a large aspect ratio of > 1.75 additionally were cropped at the center, yielding three new training examples in total. In a second step every image was resized to the specified dataset size. Afterwards, the generated datasets were reviewed manually to exclude incorrect classified images, as original images with a construction site only visible on

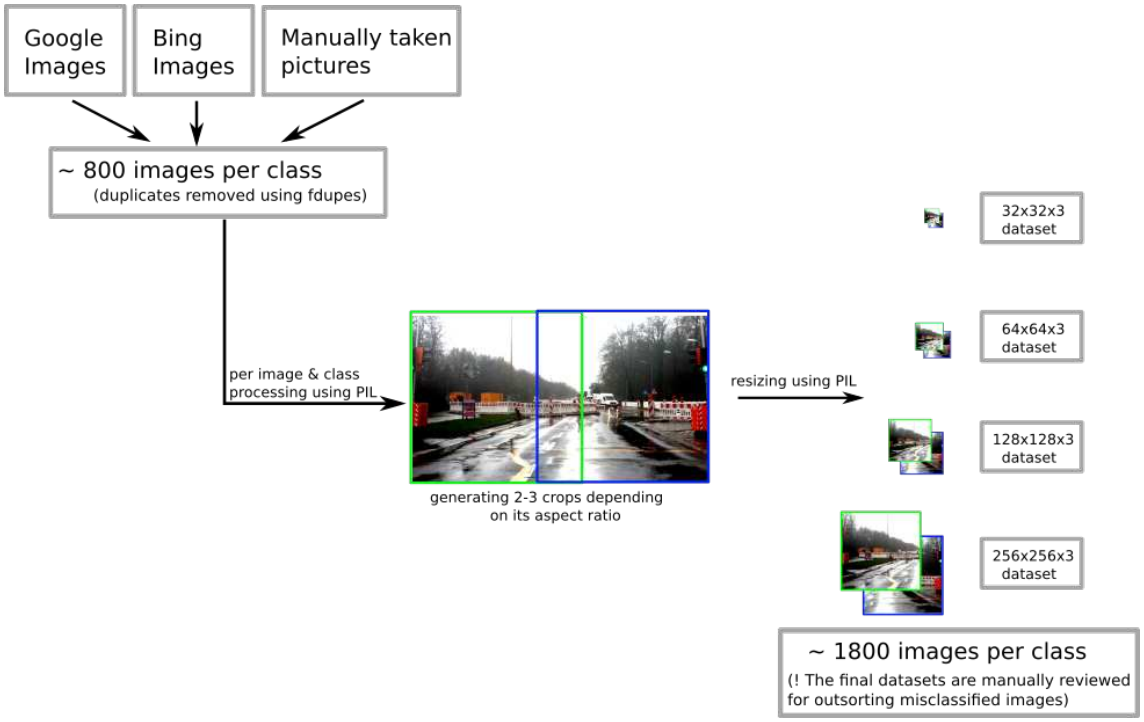


Figure 4.4: Data preparation using the Python Imaging Library (PIL) [AB09]

the right or the left could produce a resulting image without a visible construction site. Overall, the specified procedure not only preparated the collected dataset, but also increased the dataset size from about 800 examples per class to about 1800 examples per class.

4.3 Implementation

As already mentioned in section 4.1 the process of creating an adequate implementation is still quite experimental. That is why the development strategy defined in 3.4 was applied throughout the implementation phase to successively approach the desired architecture for the construction site classification task.

The following sections will first give an overview of the concrete scope of implementation decisions to be made. Then they explain initial assumptions and considerations that served as starting points for prototyping 3.4. Finally the completed CNN model implementation will be presented and illustrated.

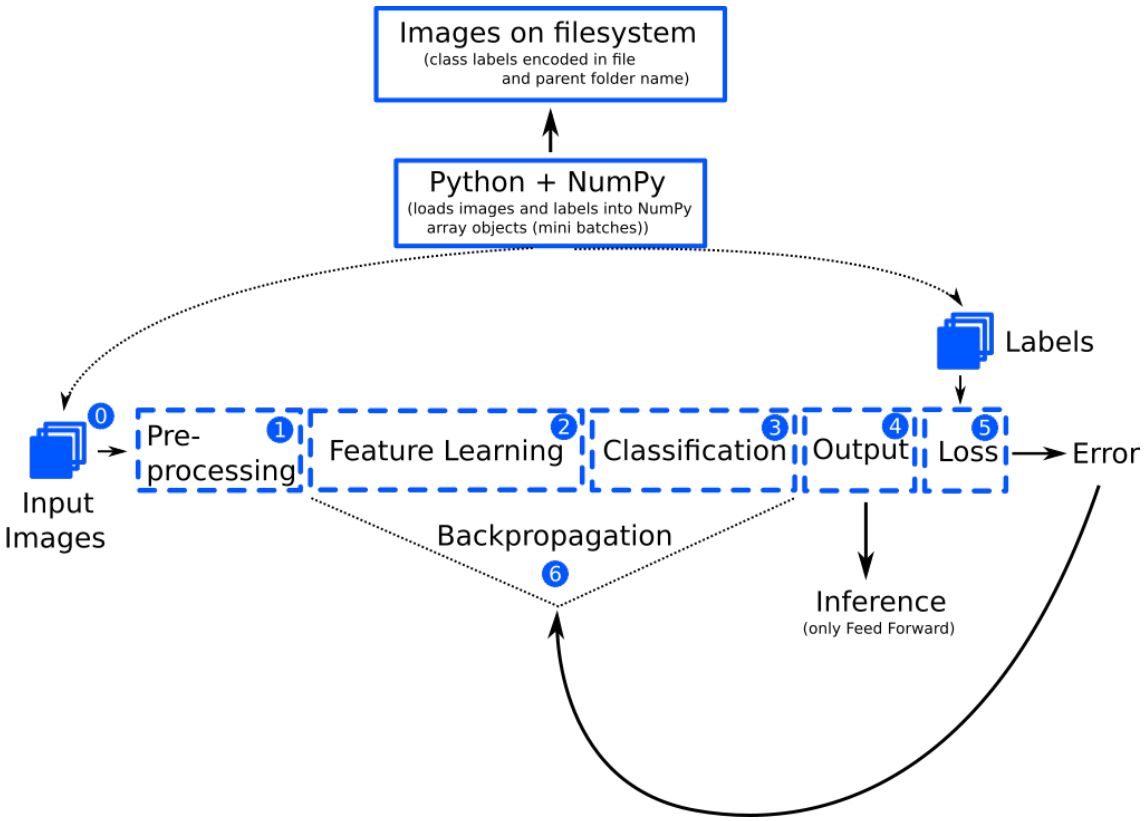
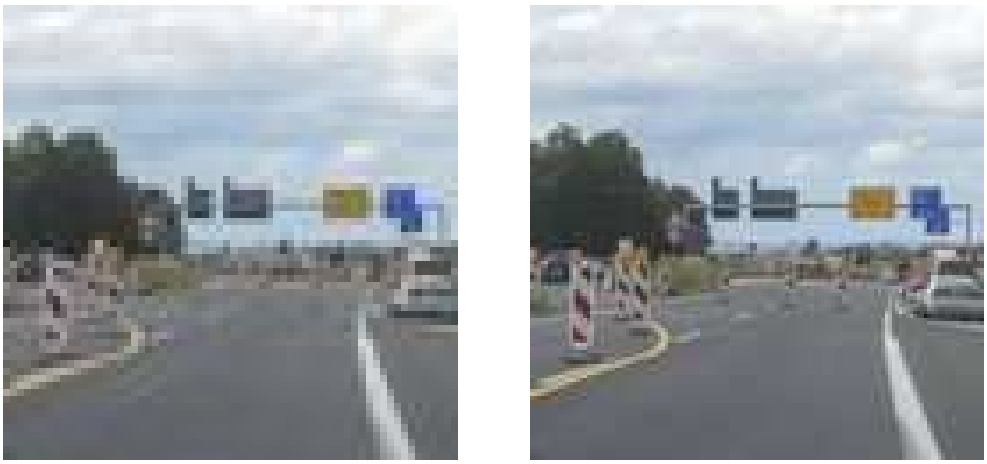


Figure 4.5: Illustration of the "Deep Learning Pipeline" implemented using Python and TensorFlow

4.3.1 Scope of implementation

Figure 4.5 roughly outlines the concrete "Deep Learning Pipeline" realized using Python, TensorFlow and the dataset collected in 4.2.1. As the illustration shows, a Python programmed component loads the corresponding training image set, including the corresponding class labels ("construction site" or "no construction site") from disk. Since the training is done via Mini-batch gradient descent, it also acts as a provider for mini-batches of training images and their labels. One pipeline cycle is described as follows (numbering refers to figure 4.5):

Each cycle (or training step) starts with a new mini-batch (0) fed into the "preprocessing pipeline" of TensorFlow (1). The preprocessed batch is then passed through the feed forward of the actual CNN model (2-3). The output segment (4) takes the results and translates them to the desired output format. This output can either be used for inference after training was completed or for validation purposes after each epoch of training. If the pipeline is not interrupted at this point, the next segment (5) takes the corresponding labels and calculates a loss value which is to be optimized by the back-



(a) 64x64 training image [1]

(b) 128x128 training image 1

Figure 4.6: 64x64 images lose important structural information

propagation algorithm. Finally, there is a specific update rule (6) that is used to update the weights of the CNN model during the backpropagation step. Unless the pipeline was not interrupted before, the next cycle starts.

Thus every numbered segment in figure 4.5 serves as a placeholder to be replaced by some concrete design decision or implementation including behavioral elements as well as structural elements.

Input images (0)

Section 4.2.1 dealed with the preparation of image sets in sizes 32x32, 64x64, 128x128 and 256x256. In terms of computational costs of the resulting network, the image resolution should be as low as possible [WYS+15, p.2], while at the same time it has to be high enough for not losing important features. As stated earlier, there are several specific visual properties inherent to construction sites.

Figure 4.6 shows a representative training image in resolutions 64x64 and 128x128. Any clear structures or patterns still visible in the 128x128 picture (e.g. the striped patterns of the vertical panels) are not visible anymore in the 64x64 picture. The 128x128 picture has a low resolution but important objects like vertical panels and yellow lines are still recognizable. That is why the dataset with a resolution of 128x128 has been selected as input data.

[1] `https://img1.osthessen-news.de/images/14/08/140818-2-dsc-9395.jpg` (visited on 3/6/2016)

Preprocessing (1)

The preprocessing step allows to normalize the input data. Furthermore it allows to artificially expand the dataset by several different operations without affecting the corresponding label data. This can be a useful, low-cost way to prevent overfitting without having to collect new expensive training data. One well-known technique is to simply flip images horizontally generating a new training sample. This way a picture of a construction site is still a picture of a construction site, even if it was flipped horizontally. Finally there are several further operations provided by the TensorFlow framework like random cropping, random contrast or random brightness.

Feature Learning and Classification (2-3)

While feature learning corresponds to the "convolutional part" of the model, the classification corresponds to the "fully connected part" of a standard CNN architecture. Together they form the actual core of the model to be implemented. As there are no fixed rules for designing an adequate model architecture, the following paragraph takes a quick look at existing architectures that could serve as a kind of template. Important characteristics like fast execution requirements are taken into account.

Utilizing existing CNN architectures There are a variety of well-known deep learning architectures available such as GoogLeNet[SLJ+15], ResNet[HZRS15], AlexNet[KSH12] or VGGNet[SZ14], each of them designed to deal with complex object recognition, localization and detection tasks. As they are intended to handle hundreds or even thousands of different object types and categories, these architectures have a high capacity in terms of learnable parameters and overall model size. For example, the AlexNet architecture (see Figure 4.7) has about 60 million parameters and requires 5 to 6 days to train it on two GTX 580 3GB GPUs [KSH12].

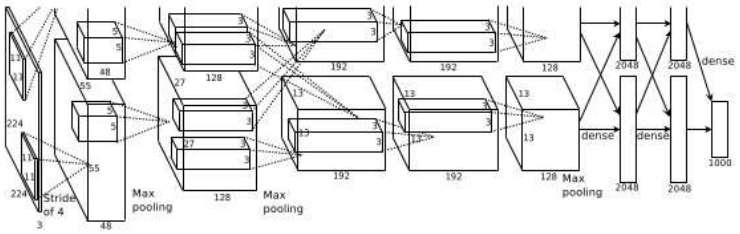


Figure 4.7: An illustration of the AlexNet architecture. [KSH12]

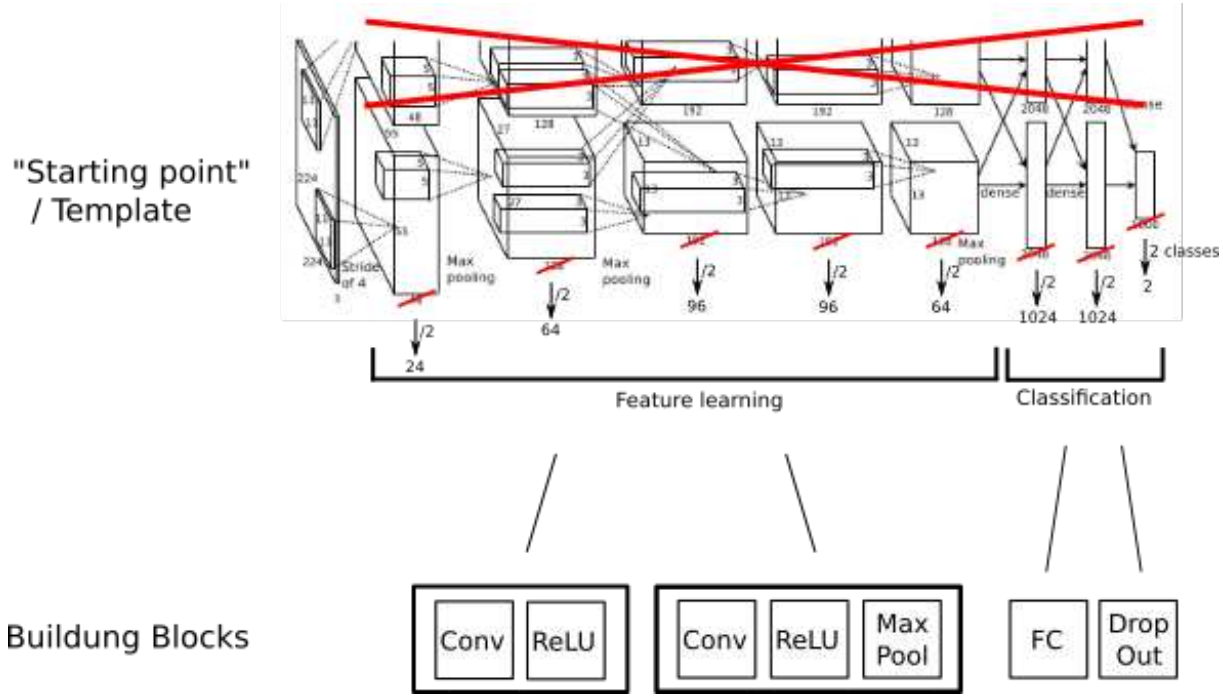


Figure 4.8: Deriving an initial prototype from the AlexNet architecture. [KSH12]

AlexNet derived prototype and building blocks Despite its large capacity, the AlexNet4.7 architecture is less complex compared to the other mentioned architectures and is quite similar to simple CNN models like the early LeNet-5 [LBBH98]. Due to its simplicity it has been selected to serve as a starting point for the implementation. As Figure 4.7 illustrates, the architecture was constructed using two lines of convolutional and fully connected layers that "communicate" with each other at certain points. This way the architecture is able to explicitly support parallel computing using GPUs.

To act as an initial prototype for the task of construction site classification, this architecture has been greatly reduced (see figure 4.8). One of the two "GPU lines" has been fully cut off. Furthermore, the number of feature layers as well as the number of neurons in the fully connected layers have been halved. An exception is the last fully connected layer. Since the original AlexNet architecture aims to distinguish between 1000 classes of objects and the construction site classification task is a binary classification task, the number of neurons has been reduced to two. These changes reduced the total number of parameters by a factor of 30 to about 2 million parameters.

After creating this first prototype, its initial structure essentially has been modified by removing or adding coarse building blocks (see figure 4.8). While "Conv + ReLU" building blocks derive new features, "Conv + ReLU + MaxPool" blocks also downsample the data. The classification part is modified by adding or removing fully connected layers.

Moreover the insertion of dropout layers helps to avoid overfitting.

Apart from these major changes, modifications of parameters like receptive field sizes, stride lengths, number of feature maps, zero padding and concrete numbers of fully connected neurons have been extensively tested.

Output (4)

This section primarily deals with the transformation of the output of the last fully connected layer to a more use- and meaningful output.

For these purposes a softmax output layer has been used. The softmax function maps the output values into the range $(0, 1)$, whereas all mapped values add up to 1 similar to a probability distribution. One single softmax value x_i is calculated as follows from the ingoing output s_i:

$$x_i = \frac{e^{s_i}}{\sum_c^{nclass} e^{s_c}} \text{[Sad16]}$$

Transferred to the task of construction site classification, x_0 represents the probability of being a construction site, while $x_1 = 1 - x_0$ represents the complementary probability of being no construction site.

Loss function (5)

x_0 and x_1 are handed to the loss function. It takes the correct class labels l_i and quantifies the total error.

The loss function has been implemented by the cross entropy error function, which is calculated as follows:

$$E = -\sum_i^{nclass} l_i \log(x_i) \text{[Sad16]}$$

Update rule (6)

For simplicity, no special update rule has been selected initially. The learning rate γ has been set to 0.1 and the update of a parameter p is given by:

$$p' = p - \gamma \cdot \nabla_p E(current\_mini\_batch) \text{ , according to [Rud16, p.3]}$$

$\nabla_p E(current\_mini\_batch)$ is the gradient of the network (loss) function with respect to p and the the current processed mini-batch.

However, there are several gradient descent optimization algorithms available that mitigate drawbacks of gradient descent such as Momentum, Adagrad, Adadelta, Adam and others.

4.3.2 Final architecture

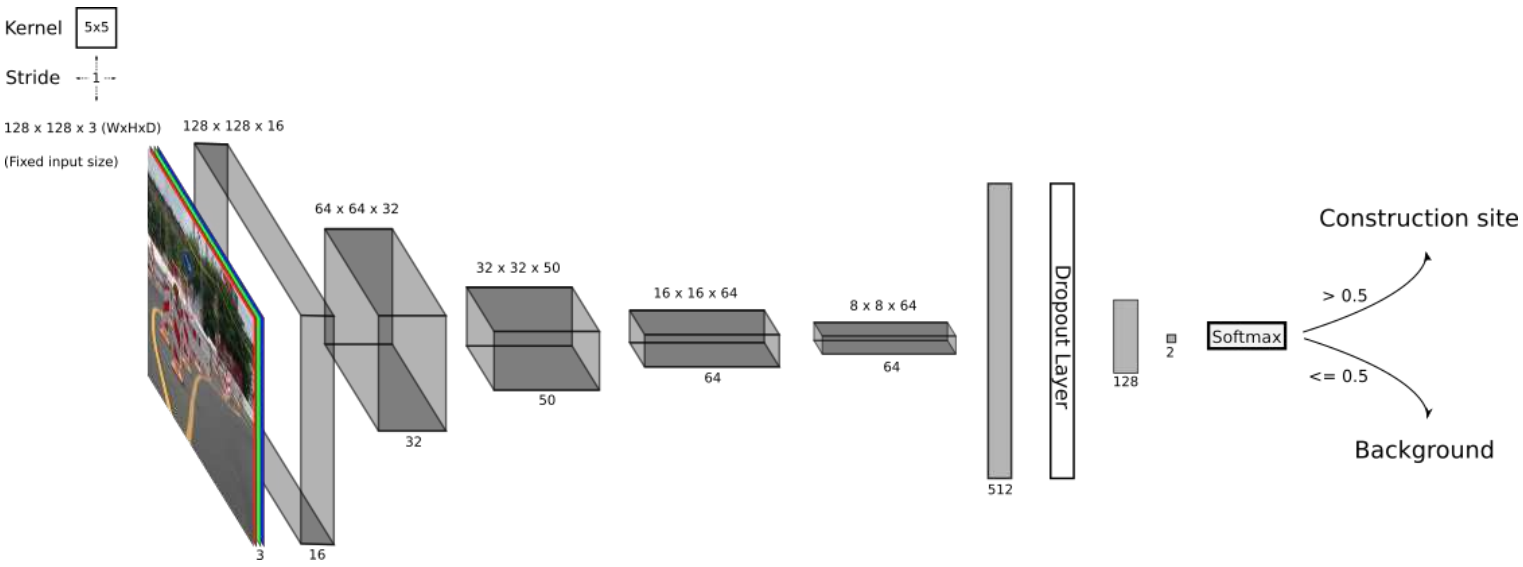


Figure 4.9: Deep learning architecture for classification task

After the considerations and preparatory work of the preceding sections a final working architecture, 4.9, has been developed through prototyping techniques defined in 3.4. The corresponding details of the implementation are presented and discussed below.

Structural improvements

As argued in 4.3, the binary classification task to be solved is much less complex than the classification of the original AlexNet architecture, which deals with at least 1000 different categories of objects. Therefore, the capacity of the derived prototype defined in 4.3, already has been reduced by a factor of 30. The development process revealed that this first prototype still had too much capacity indicated by high overfitting. Thus, several network elements have been further downsized and a dropout layer has been added. Figure 4.9 illustrates the final architecture.

The major changes have been made to the number of feature maps and fully connected neurons. Compared to the initial prototype, the number of feature maps has been rougly reduced by a factor of two. The number of neurons in the first fully connected layer

has been reduced by a factor of two and the number of neurons in the second even by a factor of eight.

One important difference compared to AlexNet, is the input image resolution which is four times lower. The AlexNet model uses a first layer receptive field size of 11x11 with a striding length of four, which reduces the height as well as the width of the first Conv-layer output by a factor of four. The final model (4.9) applies a smaller filter size of 5x5 with a stride of 1 and zero padding to compensate the smaller image resolution, as a smaller filter is able to grasp more detailed features. This modification however results in larger output dimensions of each Conv-layer output as well as an increase of weights needed for each neuron of the first fully connected layer. To compensate this, each Conv-layer has been generally succeeded by a MaxPool layer. Finally, the number of parameters has been reduced from about 2 million to only $827,348$ parameters.

A detailed structural view of the final model is shown in the table 6.2 and can be found in the appendix.

Behavioral improvements

Besides structural changes, some behavioral elements have been improved. For instance, the cross entropy loss has been extended by an L_2-regularization term to further reduce overfitting by penalizing too large weight values. This term is simply defined by the sum-of-squares of all weight parameters (W):

$$L_2(W) = \frac{1}{2} W^T W \tag{4.1}$$

and is added to the original loss E weigthed by a factor λ:

$$E_{total} = E + \lambda L_2(W) \tag{4.2}$$

[Pha].

Preprocessing steps of TensorFlow have been leveraged, such as random brightness and random contrast added to each input image.

Finally, the learning rate γ has been adapted by applying an exponential decay:

$$\gamma = \gamma_{initial} \cdot r^{\frac{\#epoch}{s}} \tag{4.3}$$

, where r denotes the decay rate, s the number of decay steps and $\#epoch$ the current epoch number. Several training runs have showed that after already reaching a low error rate, the error suddenly increased extremely again. This is due to the fixed learning rate,

which works well for the first epochs, but can be too high for later epochs as it can cause the network to "jump out of the current local minimum". By successively reducing the learning rate with respect to the number of epochs the training has been converged more smoothly.

A detailed overview of concrete hyperparameters of the final model is provided by the table 6.1 and can be found in the appendix.

4.4 Results

To train the network, the dataset created in 4.2.1 has been divided into a training and a validation set. Due to the sparse amount of images concerning the broad application domain, the validation set only has been made up of about eight to nine percent of all images.

The following charts and figures demonstrate the results of the final classification architecture 4.9 and basically verify that the designed architecture is able to solve the classification task.

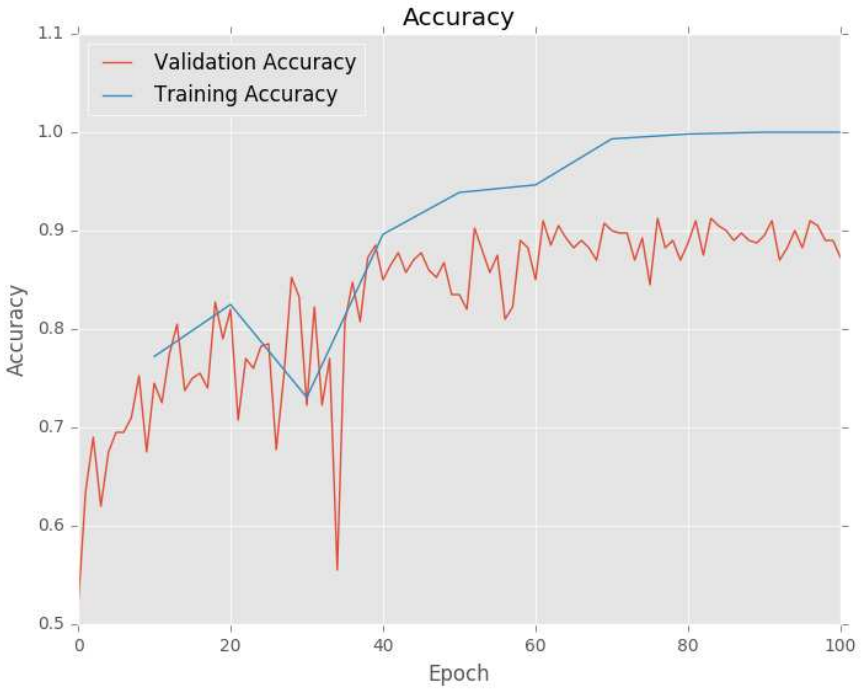


Figure 4.10: Accuracy of the training and validation sets

	Construction site	**Background**
Construction site	132	19
Background	11	136

Table 4.1: Confusion matrix for the validation set of 298 images (using weights of the 83th epoch).

Figure 4.10 shows the development of both the validation set accuracy [2] as well as the training set accuracy relating to the particular epoch [3]

Due to the high computational and temporal expenditure of accuracy calculation [4], the training set accuracy has only been calculated every tenth epoch. Since the validation set is quite small, its accuracy has been evaluated after each epoch.

As the chart in 4.10 shows, both accuracy values develop quite similar up to the 40th epoch. From then on, only the training accuracy significantly improves and thus indicates a slight overfitting. In epoch 80, the training set reaches a value of 100, meaning that each picture of the training set has been classified correctly. The maximum validation accuracy of 0.9125 is reached in the 83th epoch.

Table 4.1 shows the corresponding confusion matrix.

[2] The accuracy is determined by $\frac{\text{Number of correctly classified images}}{\text{Number of all images}}$

[3] One epoch corresponds to a full training cycle, incorporating the entire training set.

[4] Inference (Feed-forward) step has taken about 90 milliseconds on average per image (with respect to the IaaS solution (3.3))

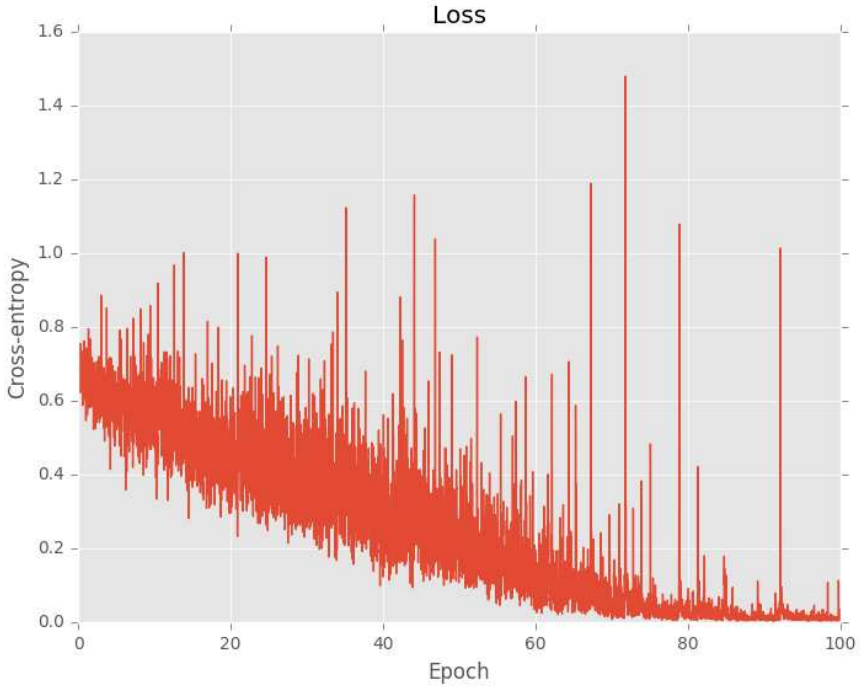


Figure 4.11: Cross-entropy loss

Figure 4.11 shows the development of the cross-entropy loss. The loss successively decreases and reaches a value near zero after 70 to 80 epochs.

In the following several image compilations show concrete classification results of the trained network.

Figure 4.12: Selection of best classified construction site images with their corresponding softmax values.

Figure 4.12 illustrates construction site images with a high classification score. The first four images include typical construction site elements like yellow lines, construction vehicles and vertical panels that are clearly visible. The first three images of the second row also contain those elements, but they are less recognizable due to the lighting conditions or longer distance. The last picture only contains rubble and construction vehicles and is also definitely classified as construction site.

Figure 4.13: Selection of best classified background images with their corresponding softmax values.

Analogous to the preceding figure, 4.13 illustrates background images with a low score. The images of this selection basically show that the network has learned successfully to distinguish between typical properties inherent to construction site scenes, such as loud colors, red and white striped patterns, and quite similar properties and features of everyday road scenes. Moreover, the first image shows a construction site truck, which could indicate the presence of a construction site, but the image is still correctly classified as background.

Figure 4.14: Selection of false positives with their corresponding softmax values.

Figure 4.14 shows a selection of false positives. These images have been labeled as background in the dataset but were classified as construction site. The first background image however could also have been labeled as a construction site, as it indeed shows a construction site (even if it is no road construction site). The classification network thus even revealed that one background image initially has been labeled in a wrong way. The other background images show similar objects and features as images in 4.13, but in this case are not classified correctly. This could be due to the sparseness of the training set, as the image set only covers a tiny subset of ordinary road scenes.

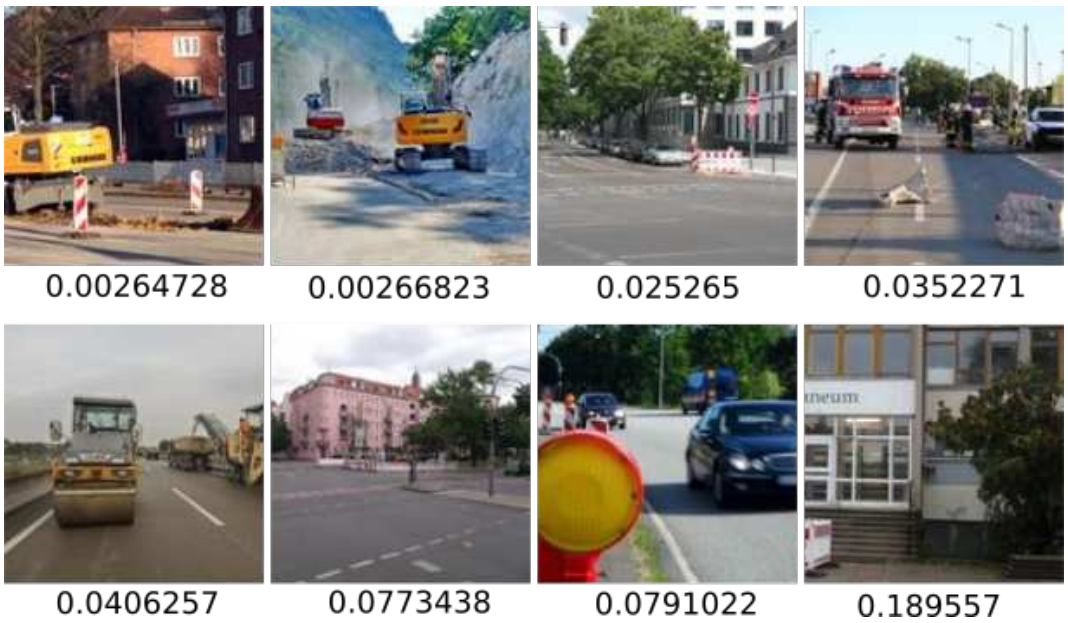


Figure 4.15: Selection of false negatives with their corresponding softmax values.

Figure 4.15 shows a selection of false negatives. As can be seen, three of the construction sites are hardly recognizable, as the barricades and vertical panels are hardly visible. This could be the reason of this misclassification. Four further pictures show construction sites, but from an unusual point of view. The last image shows a fire department operation, which is not representative for a construction site in terms of this work.

Figure 4.16: Boundary cases of construction site images with a softmax value near 0.5.

Figure 4.17: Boundary cases of background images with a softmax value near 0.5.

Figures 4.16 and 4.17 further illustrate some boundary cases that have just been classified correctly or already incorrectly classified. This could be explained by similiar reasons as above. The first image in 4.16 shows a construction site in the background, but no prominent features on the road indicate this construction site. Therefore it is classified correctly, but not very clearly. The two first images in 4.17 are also classified correctly as background but with a higher softmax value, which could be due to its red-white transitions of the cycle path on the first and red-white transitions of the diagonal level crossing sign. The remaining pictures are assumed to be better classifiable using larger training sets.

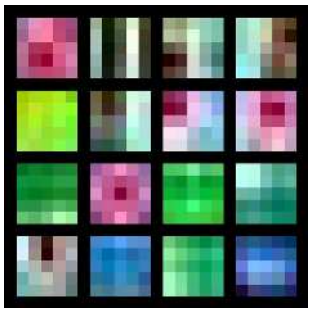


Figure 4.18: Layer-1 Filters (83th epoch)

Finally, figure 4.18 illustrates the learned Layer-1 filter weights with a receptive field size of 5x5. Any filter corresponds to a specific color and pattern, and no dead filters are present.

In summary, all results indicate a basically working classification architecture, but also indicate still a slight overfitting and reveal several boundary cases that are assumed to be solvable by a larger and even more diverse training set.

5 Construction site detection

5.1 Bottom-up approach

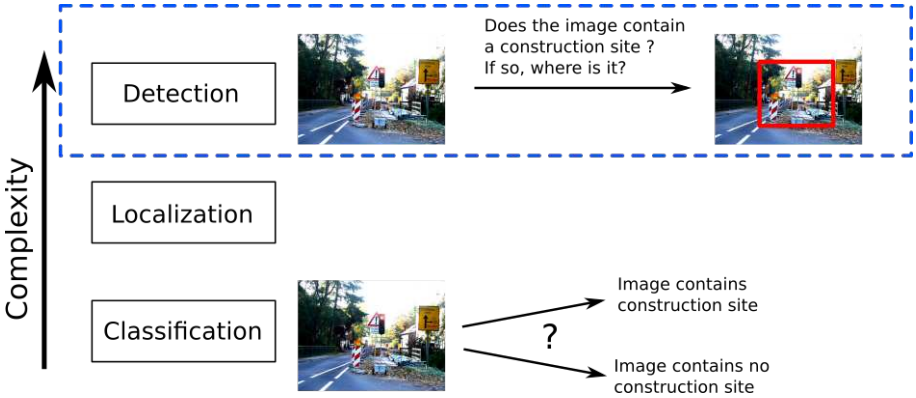


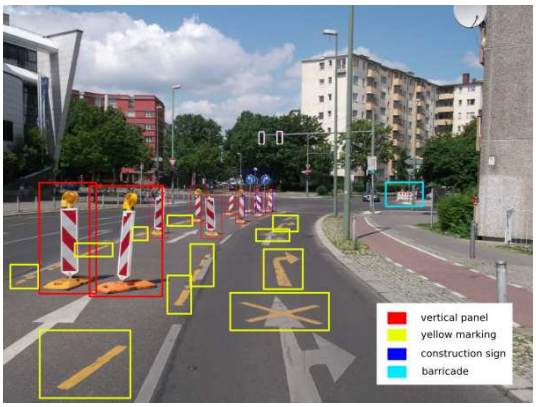
Figure 5.1: Computer vision tasks classification, localization and detection with increasing order of complexity

The last section presented the development of a CNN model capable of solving the classification task for construction sites. As the bottom-up approach in 4.1 specified, this was the first step towards the more complex tasks localization and detection (5.1). Accordingly, the next logical step can be to solve the localization task before finally solving the detection task. However, the following sections demonstrate that only a single architectural rearrangement is able to solve both tasks at once. For this, the task of detection (including its localization sub-task) is reformulated as a semantic segmentation problem more suitable to the domain of construction sites.

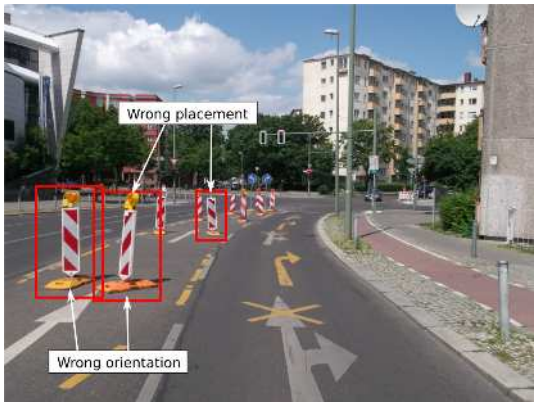
5.2 Detection Task

The detection task is characterized by the question "where are the instances of a particular object class in the image (if any)?" [EGW+10, p.1]. Since a construction site consists of many different types of objects, such as vertical panels, barricades, traffic cones, construction vehicles, construction workers, construction signs, lights, yellow lines, rubble,

(a) Original image



(b) Classical detection ground-truth



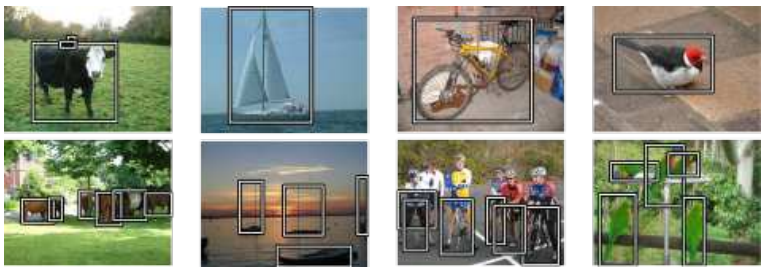
(c) Pitfalls of the classical detection approach

Figure 5.2: Reviewing the classical detection approach

fresh asphalt and other ones, the detection can be implemented by creating a complete index of all object classes potentially belonging to a construction site and solving the detection task for each of them.

Figure 5.2 illustrates how this process can look like. While 5.2 (a) shows the original image of a construction site, 5.2 (b) illustrates the desired output with a bounding box located at each detected object.

As argued in the introduction, the most important two types of information for an autonomous vehicle passing a construction site are the size and location of the construction site as well as the size and location of the remaining passable road. If the way described above is used, all single objects belonging to a construction site can be detected. However, both mentioned types of information are not provided by this approach. Since the majority of objects as well as their spatial configurations are standardized by law, one can argue that an additional algorithmic step could solve this task. It could infer the two types of spatial information from the individual location and orientation of each



(a) Example outputs for detection [EGW+10]

(b) Example inputs with corresponding ground-truth for semantic segmentation [**?**]

Figure 5.3: Comparison of semantic segmentation and detection

detected object. Anyway, it is quite difficult when it comes to reality. As there are not only perfectly secured construction sites that respect all legal regulations, this approach has its pitfalls.

Figure 5.2 (c) illustrates this aspect. There are three bounding boxes marking vertical panels. The german road traffic law and its regulations[1] stipulate that the diagonal striped pattern of vertical panels have to fall from the secured area to the direction of the traffic. The "Wrong orientation" labeled panels do not fulfill this requirement. Additionally, two panels labeled with "Wrong placement" are placed too central and are not reflecting the spatial extent of the blocked turn lane. To summarize figure 5.2 (c) only shows possible imperfections concerning real road construction site environments.

For the above reasons, the present work has chosen an alternative approach, which does not take individual objects into account, but focuses directly on the required information. This is accomplished by interpreting the detection task as a 3-class semantic segmentation task recognizing the classes "construction site", "passable road" and a background class for detection. The basic intention is to leave it to the concept of deep learning to incorporate all necessary visual information and detect the required information based on the generalization of a variety of real world examples.

(a) Example ground-truth (blue: background, red: construction site)

(b) Example ground-truth (blue: background, red: construction site, green: passable road)

Figure 5.4: Construction site detection interpreted as semantic segmentation task

5.2.1 Interpretation as semantic segmentation task

Figure 5.3 demonstrates the differences of the computer vision tasks detection and semantic segmentation. While 5.3 (a) shows examples of detection outputs, where each detected object instance is marked by a surrounding bounding box, 5.3 (b) illustrates example outputs of semantic segmentation, which are much more detailed as not coarse regions, but even single pixels are classified. On the other hand the semantic segmentation approach has the disadvantage of not being able to distinguish between single object instances of the same class. For example the second sample in 5.3 (b) detects and highlights areas with a chair on it, however, without actually recognizing how many instances of chair objects are visible and which specific area belongs to which chair.

This disadvantage of the semantic segmentation approach presents an advantage in view of the objectives of this work, since the information on specific areas are far more important than the type and placement of individual objects.

Figure 5.4 illustrates this idea by showing two images specifying ground truth information for the original image shown in 5.2 (a). 5.4 (a) demonstrates a two-class annotation. Each part of the construction site is highlighted in red and the remaining areas are classified as background and are colored in blue. 5.4 (b) additionally annotates the "passable road" area of the image, colored in green. Some construction site elements, like the yellow markings are covered by the green colored area to prioritize the passable road area as the most important information type.

[1] 3.1.2 (2), RSA - Richtlinien für die Sicherung von Arbeitsstellen an Straßen, February, 1995 edition

5.2.2 Data annotation

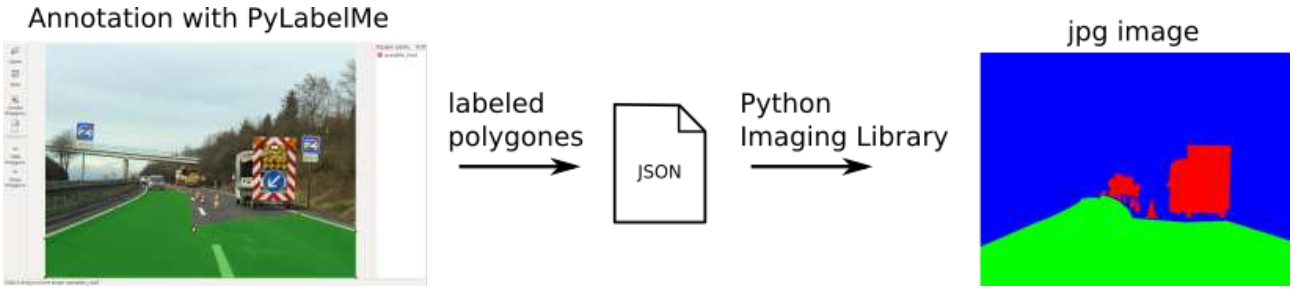


Figure 5.5: Annotation process overview

According to the data acquisition and preparation steps for the classification task, this section deals with the generation and preparation of ground truth data for the detection task. While the classification task only required image data with the two corresponding labels "construction site" or "no construction site", the detection task in terms of semantic segmentation needs pixelwise labels referring to their respective classes "construction site", "passable road" and "background". This is achieved by manually annotated images of the original dataset collected and described in 4.2.1. To do so, the suitable open source application PyLabelMe [Pit11] has been used. Essentially it is a small Python program that allows to load images and annotate it by creating, modifying and labeling polygons. Moreover the application allows to store all annotations formatted in JSON on disk. As outlined in figure 5.5, the generated JSON output is then further processed and finally converted to a JPEG formatted image using the Python Imaging Library (PIL) [AB09]. To accomplish this all labeled polygons defined in the JSON output have been read in and drawn onto a virtual canvas including three channels for red, green and blue. Polygons labeled with "construction site" have been colored in red, polygons labeled with "passable road" green and polygons labeled with "background" in blue. This three channel JPEG image has been converted to a three channel and not a one channel (grayscale) JPEG image, as this format is both useful in terms of ground truth data (pixelwise label simply corresponds to the maximum channel value), and also acts as a visual reference of the generated ground truth data.

Requirements

Besides the required technical basis of the annotation process, some basic requirements have been established to ensure overall consistency and to define the annotation process in detail. This only concerns images of the positive dataset including a construction site.

All images of the background dataset without a construction site have been replaced by an entirely blue image matrix at training time.

The following enumeration sums up the annotation rules, which have been taken into account. This only concerns the annotation of construction site and passable road elements. All omitted areas are automatically considered as background.

1. All elements of construction sites (coarse and fine) visible on the image have to be covered by respectively labeled poylgons. If those annotated elements are cut out of the image, no remaining element would indicate the presense of a construction site anymore. Therefore, visual elements like torn streets, fresh asphalt and rubble also have to be included.

2. In general these annotations have to be as detailed as possible to focus on specific visual properties inherent to construction sites. Regarding the high time expenditure of manually annotating detailed structures, there has to be a trade-off with respect to the level of detail. For instance, if there are conglomerations of lots of finely structured and detailed objects these conglomerations can be coarsely annotated, if the surrounded area, which does not belong to the construction site does not make up the major part.

3. The "passable road" annotated areas have to cover those parts of the visible road that are still passable by the autonomous vehicle and are not blocked by the construction site. More precisely, the "passable road" can also include areas that have not been part of the road before, but have been made available by the construction site.

4. Due to the sparse visual consistency of the training images regarding the point of view and the camera angle, there are only a few images representing approximately real visual characteristics with respect to an autonomous vehicle. If there are multiple lanes or roads visible on these images, only the one related to the first persion view of this virtual vehicle has been annotated.

Annotation results

Following this data annotation process with respect to the defined requirements, resulted in a new dataset containing 311 original size images and their generated ground truth data. Due to the time-consuming task of manual annotation (20 to 30 minutes on average per picture), only about half of all available images have been annotated. Figure 5.6

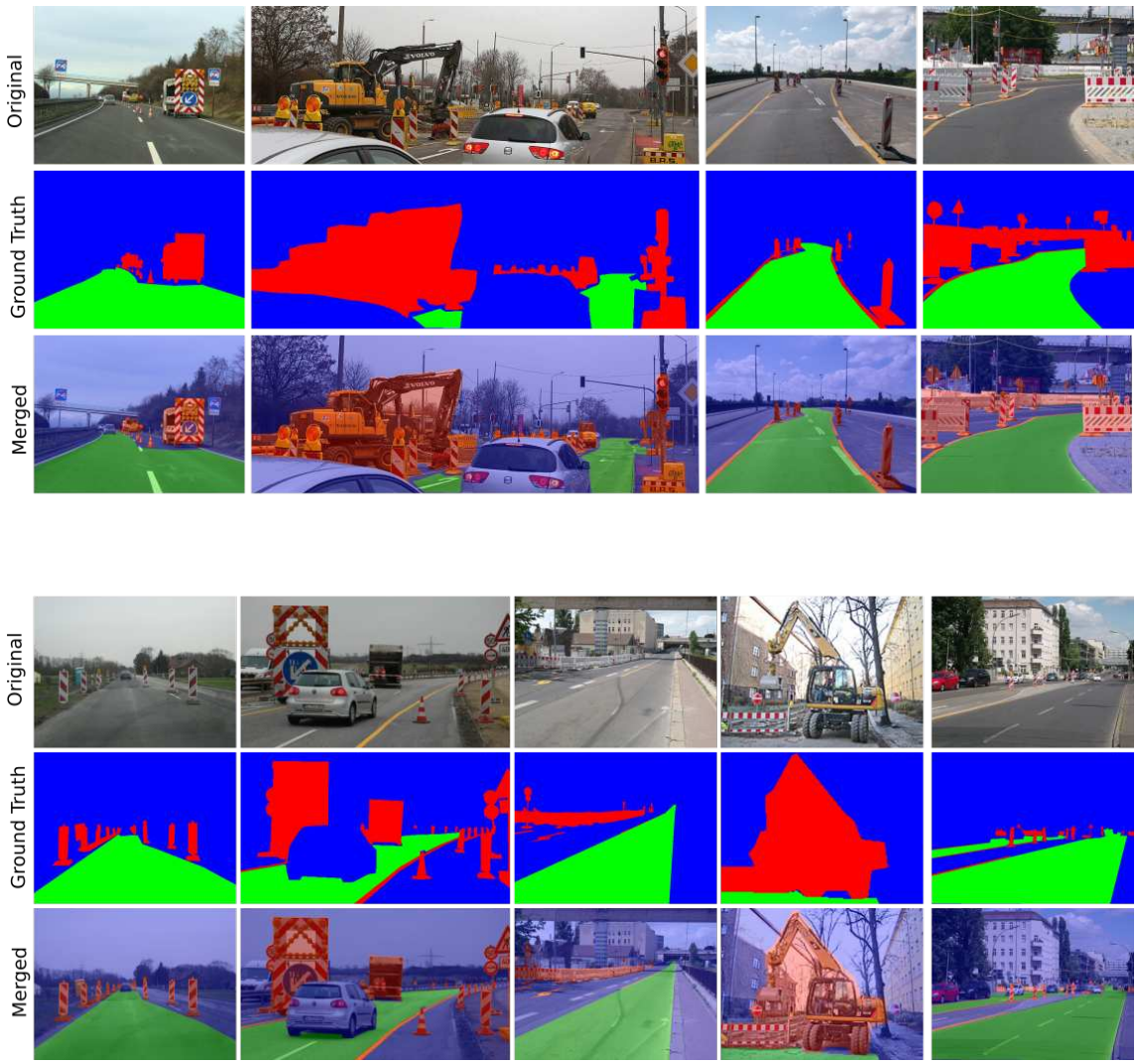


Figure 5.6: Sample annotations

shows a selection of these, including the original image, the corresponding ground truth and a merged picture of the original image and the ground truth information.

Eventually the existing data preparation mechanism illustrated in 4.4 has been reused to resize and crop these images. The final dataset includes 678 images with their corresponding ground truth data as well as 1600 background images with the implicit ground truth represented by an image matrix with the blue channel overall set to the maximum value.

5.3 Implementation

After annotating and generating the semantic ground truth data, the required structural and behavioral changes to the classification task network are discussed in the following sections.

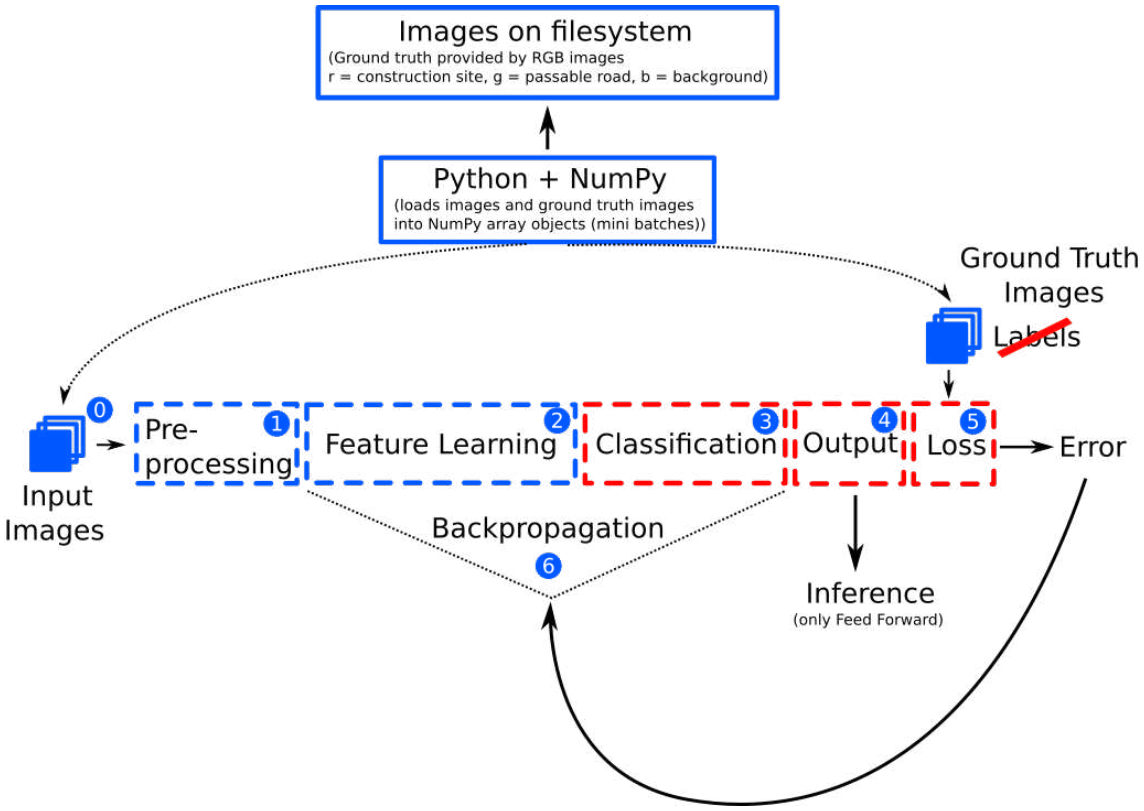


Figure 5.7: "Deep Learning Pipeline" and parts to be modified for the detection task

Figure 5.7 gives an overview of those parts of the data flow graph , which essentially have to be adjusted to support the intended semantic segmentation task. The classification as well as the output section (3-4) now have to make pixelwise predictions. Therefore, the loss function (5) also has to deal with pixelwise loss information. Finally, the architecture now also has to support three instead of two classes of information.

5.3.1 From convolutional to fully convolutional (3-4)

To support pixelwise classification and thus semantic segmentation, the classification network 4.9 has been reorganized as a fully convolutional neural network (FCN). Referring to 2.3, a fully convolutional neural network differs from a conventional CNN in the way that the classification part of the network is constructed. In a conventional CNN

the classification is done by several fully connected layers. In a FCN it is achieved by using convolutional layers as well. Neurons that had been connected to each output value of a convolutional layer in a CNN, are now only connected to a local subset of this output in a FCN, and thus providing local classification outputs.

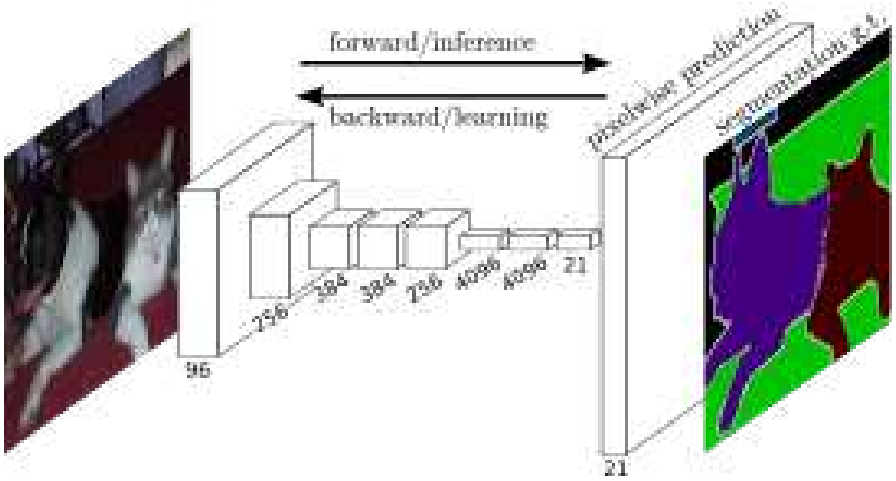


Figure 5.8: Semantic segmentation using a fully convolutional neural network and bilinear upsampling [SLD16]

As the information processed by the classification network 4.9 gets downsampled to a lower resolution, a simple replacement of fully connected layers with convolutional layers is not sufficient. Because the final output is intended to have the same resolution as the input image and to provide pixelwise classification information, this information has to be upsampled to fit the original image size as figure 5.8 demonstrates.

Concerning the classification network 4.9, the output of the last convolutional layer with a feature map size of 8x8 pixels has to be upsampled to 128x128 pixels.

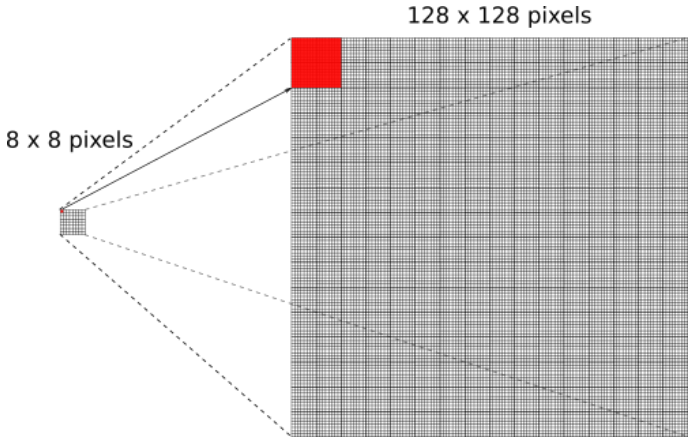


Figure 5.9: Illustration of the upsampling dimensions

Figure 5.9 demonstrates the proportions of aggregated classification information and finally upsampled output values. As it implies, the precision of the final output is limited. But in terms of a proof of concept, this solution has been already sufficient. Several further approaches to refine the final classification outputs can be to combine the output of different convolutional layers with different feature map sizes ([SLD16]) , or to mirror the entire convolution network by a corresponding deconvolution network ([NHH15]). For the final detection architecture, the upsampling is implemented as a bilinear upsampling.

The last important change to the classification task model is to extend the number of softmax channels from two to three to support the classes "construction site", "passable road" and "background". For the same reason, the last convolutional layer of the new detection architecture is extended from two to three feature maps.

5.3.2 Loss (5)

Besides the structural adjustments, the loss function of the classification architecture has to be replaced with a loss that supports three channel pixelwise errors by comparing the overall output with the corresponding ground truth image.

In order to produce an accurate output with regard to the three semantic classes, loss functions with different prioritizations of these classes have been tested.

They have been derived from the cross-entropy, which has already been defined in 4.3.1. While the classification task only needed to compare two single softmax outputs to a two-class label ("0" or "1"), the detection architecture has to compute the loss per pixel involving a three-class label.

Unweighted cross-entropy The pixelwise and overall cross-entropy based on 4.3.1 are defined as follows:

$$e_p = -\sum_{i=0}^{nclass=3} l_i \log(x_i)[\text{Sad16}] \tag{5.1}$$

$$E = (\frac{1}{128 \cdot 128}) \sum_{p=0}^{128 \cdot 128} e_p \tag{5.2}$$

e_p denotes the pixelwise cross-entropy and the overall loss E is simply the mean of the sum of all pixelwise cross-entropy values of the network output regarding the corresponding ground truth image. x_i denotes the softmax output for class i for pixel

p and $l_i \in \{0, 1\}$ represents the ground truth label for pixel p. As only input images with a size of 128x128 pixels have been tested, the number of pixels in the equation has already been set to $128 \cdot 128$.

As there is no factor that weights the pixelwise cross-entropy values in terms of the corresponding ground truth class l_i, for the purposes of this work, this loss is called the *unweighted cross-entropy*.

Global weighted cross-entropy Because the unweighted cross-entropy implies that all misclassifications are equally important, and the majority of all ground truth pixels are classified as background, the background pontentially gains "more interest" by the network than construction site or passable road pixels. To prevent this, a further weighted version of the unweighted cross-entropy has been applied. The global weights per class have been determined by counting all per class pixels of the entire dataset and setting them in relation.

The dataset contains $cs_{count} = 3235966$ construction site, $pr_{count} = 1849996$ passable road, and $bg_{count} = 32269558$ background pixels in total. The corresponding weights:

$$w_{cs} = \frac{bg_{count}}{cs_{count}} = 9.972156073333279 \tag{5.3}$$

$$w_{pr} = \frac{bg_{count}}{pr_{count}} = 17.443042039009814 \tag{5.4}$$

$$w_{bg} = \frac{bg_{count}}{bg_{count}} = 1 \tag{5.5}$$

are then used to weight the pixelwise cross-entropy with respect to the corresponding class i:

$$e_p = -\sum_{i=0}^{nclass=3} w_i l_i \log(x_i)[\text{Sad16}] \tag{5.6}$$

As the calculated weights refer to the global pixel statistics this version is called the *Global weighted cross-entropy*.

Batch local weighted cross-entropy Finally, for testing purposes, a local weighted cross-entropy has also been applied. The weights are computed exactly the same way as the global weighted cross-entropy does. However, the underlying pixel statistics and hence also the weights are computed for each processed image batch independently.

5.3.3 Final detection architecture

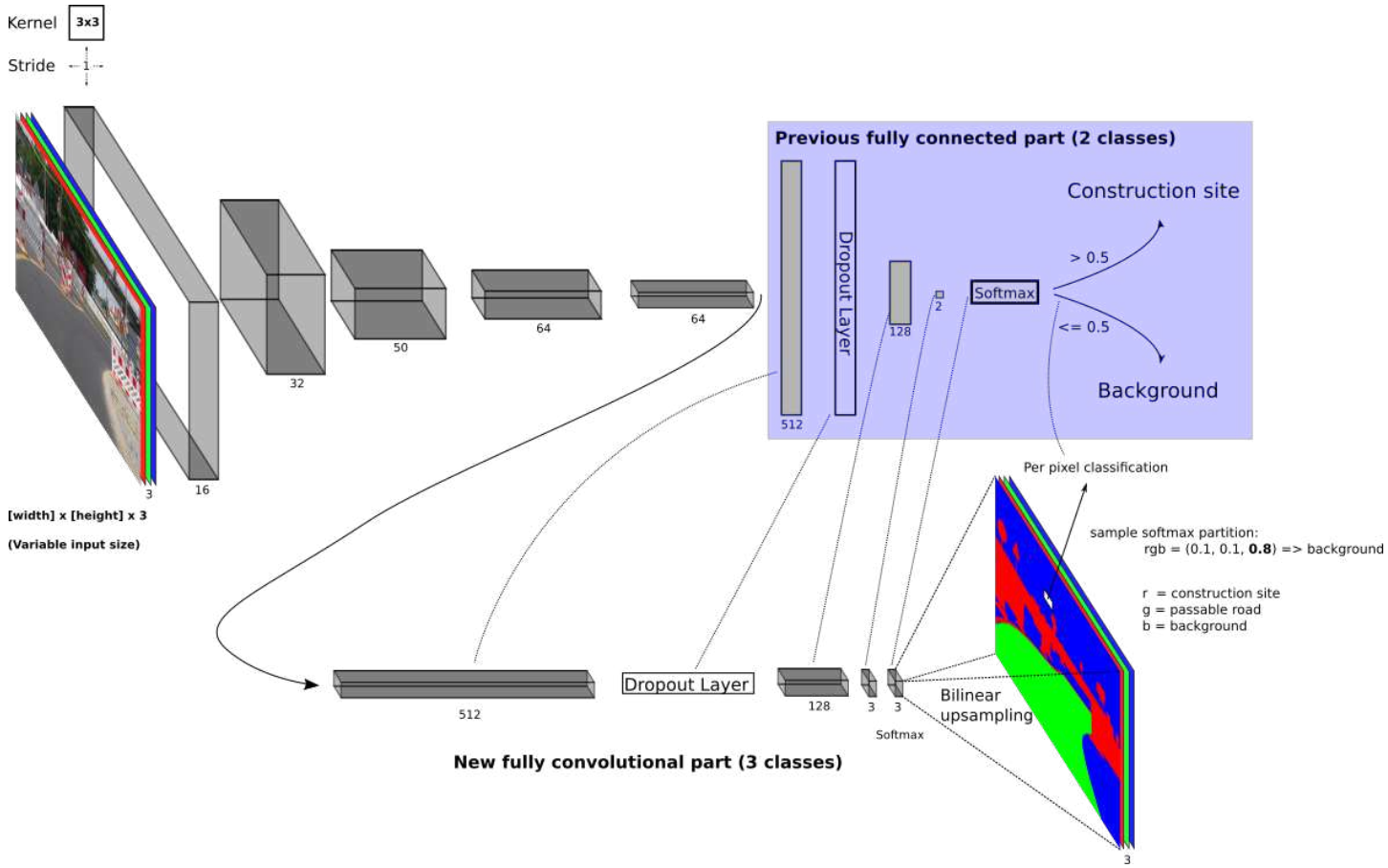


Figure 5.10: Deep learning architecture for detection task

Figure 5.10 illustrates the final detection architecture. As it shows, no further profound structural changes to the classification network than the above described have been applied for a final working architecture. The convolutional layers have remained the same and each fully connected layer with n neurons has been converted to a convolutional layer containing n output feature maps. However, the receptive field size of the kernel has been reduced from 5x5 to 3x3. The receptive field size of the last converted fully connected layer has even been reduced to 1x1. - The positive effect of this fine tuning is assumed to relate to the increased locality supporting the pixelwise classification score. Finally, each layer's output has been scaled by a specific factor to avoid floating point underflows that initially occurred due to small weight multiplications.

A more detailed listing of the structure as well as other hyperparameters is given by 6.4 and 6.3 can be found in appendix 6.

5.4 Results

5.4.1 Accuracy metrics

The accuracy metrics for evaluating the detection model and its performance have been adopted from [SLD16]. Four types of different metrics have been defined. n_{ij} denotes "the number of pixels of class i predicted to belong to class j, where there are n_{cl} different classes, and"[SLD16] $t_i = \sum_j n_{ij}$ is the total number of pixels of class i.

Pixel accuracy

$$\sum_i n_{ii} / \sum_i t_i \tag{5.7}$$

The pixel accuracy measures how much percent of all pixels have been classified correctly. There is no statement regarding the different classes. If all training images have a relative proportion of 80 percent background and the model always predicts a result with only background, the pixel accuracy has a value of 80 percent. Thus this metric is of limited use.

Mean accuracy

$$(1/n_{cl}) \sum_i n_{ii}/t_i \tag{5.8}$$

The mean accuracy measures a pixel accuracy per class and calculates the mean value. This metric is slightly more meaningful than the pixel accuracy, because it incorporates the number of different classes by using the arithmetic mean.

Mean IU (Intersection over Union)

$$(1/n_{cl}) \sum_i n_{ii}/(t_i + \underbrace{\sum_j nji - n_{ii}}_{\text{false positives}}) \tag{5.9}$$

The mean IU makes a more precise statement about the accuracy of the semantic output. Compared to the mean accuracy it also takes the number of false positives into account. The term "Intersection over Union" (IU) refers to the intersection of the predicted pixels of class i with the actual ground truth pixels of class i and the corresponding union.

Frequency weighted IU

$$(\sum_k t_k)^{-1} \sum_i t_i n_{ii} / (t_i + \sum_j n_{ji} - n_{ii}) \tag{5.10}$$

This metric is the same as the mean IU. However, the class specific terms are not weighted by using the arithmetic mean, but are weighted by the relative frequency of the particular classes.

5.4.2 Additional mean IU derived loss function

For testing purposes, besides the cross-entropy based loss functions defined in 5.3.2, simply the reciprocal of the *mean IU* (5.4.1) has also been used as an additional loss function:

$$(1/n_{cl}) \sum_i \frac{t_i + \sum_j n_{ji} - n_{ii}}{n_{ii}} \tag{5.11}$$

It is important to mention that accuracy metrics deal with interpreted output values. For instance, regarding the classification architecture a picture classified with a softmax value 0.69 has been clearly assigned to the "construction site" class and counted as such for the computation of the accuracy metric.

The loss function presented above, however, does not deal with clear "integer class assignments". Instead, it replaces a counted pixel in n_{ii} or n_{ji} with its concrete softmax output value $\in (0, 1)$. For example, if another pixel is correctly classified as i with a probability of 0.75, n_{ii} is not increased by 1, but only increased by 0.75. Without this refinement, the gradient computed with regard to this loss would not have much significance.

5.4.3 Accuracy results

The following charts present the results according to the accuracy metrics defined in 5.4.1.

The first chart (5.11) shows the pixel accuracy for each loss function relating to the corresponding epoch. The first aspect revealed by this chart is the clear underperformance of the mean IU derived loss function. All cross-entropy based loss functions, however, eventually reach a high accuracy value. The second aspect revealed is the correspondence between the global weighted and batch local weighted loss function. Both

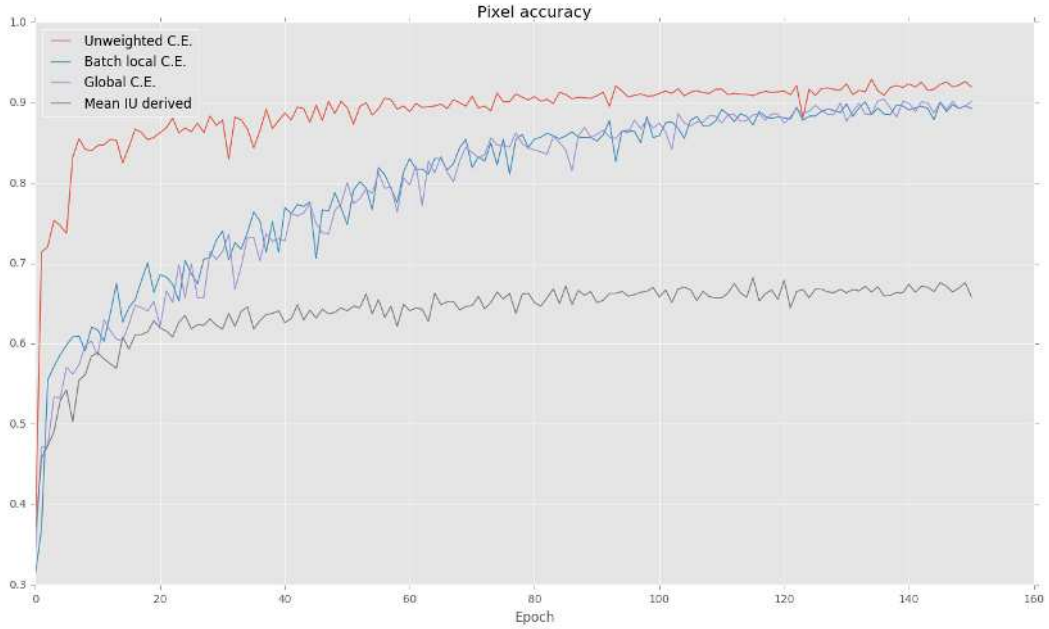


Figure 5.11: Pixel accuracy of detection task training

graphs are nearly identical. This is assumed to be due to the fact that the relatively high and fixed weights used with the global weighted approach, on average have exactly the same effect as the local per batch computed weight factors. The third aspect is given by the rapidly increasing graph of the unweighted loss. This has been expected, since the unweighted cross-entropy loss does not apply with any priorities regarding the three pixel classes and thus first maximizes the pixel acccuracy by simply classifying lots of pixels as background.

The second chart showing the mean accuracy (5.12), confirms the conclusions from the first graph. The global and local weighted cross-entropy losses show an identical course. They outperform both the unweighted cross-entropy loss as well as the mean IU derived loss, since they have prioritized the different pixel classes with respect to their number of occurrences. The unweighted cross-entropy loss only increases slowly, because it initially focuses on the background pixels as stated above. The mean IU derived loss however shows a better performance compared to the overall pixel accuracy, which is due to its class respective focus on correctly and not correctly classified pixels.

The third chart illustrates the course of the mean IU accuracy (5.13). The most interesting aspect to see here is the good performance of the unweighted cross-entropy loss. Although it is unweighted, it reaches a good mean IU accuracy, which in essence is a class respective accuracy. The mean IU derived loss shows again a clear underperformance.

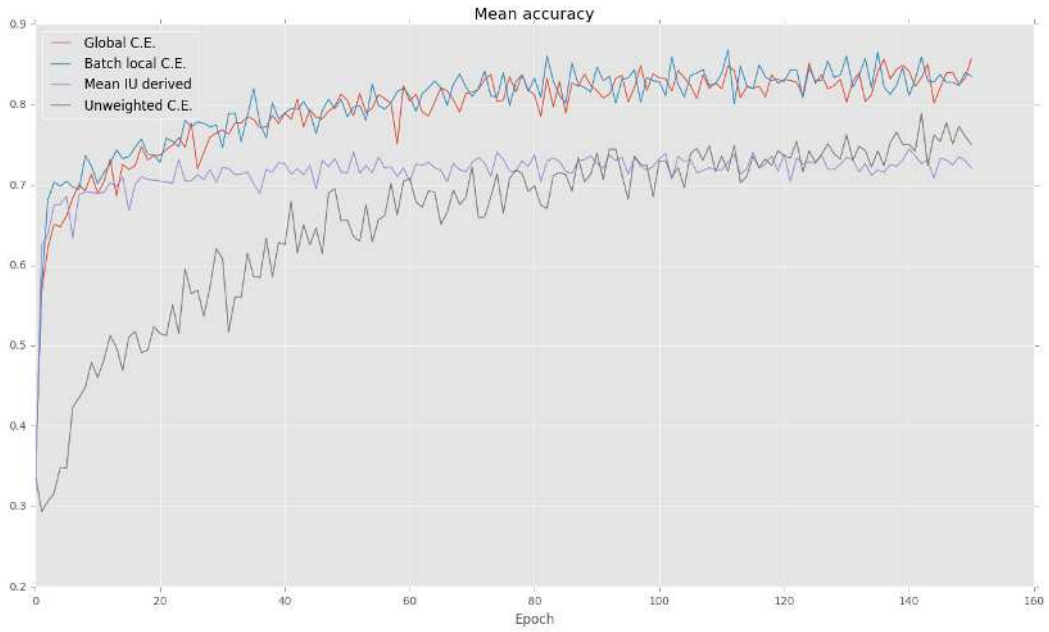


Figure 5.12: Mean accuracy of detection task training

All cross-entropy based losses eventually reach about the same level of accuracy.

The last accuracy metric is the frequency weighted IU (5.14). The results are comparable to the mean IU accuracy metric. Due to the focus on the most frequented classes (in this case the detection of background pixels), the accuracy of the unweighted cross-entropy graph again rapidly increases at the beginning.

Summary of accuracy metrics results In summary, three important observations have been made:

1. The batch local and the global weighted cross-entropy behave identically
2. The mean IU derived loss is clearly outperformed by the cross-entropy approaches
3. The unweighted cross-entropy loss suprisingly shows a consistent and high accuracy.

5.4.4 Visual results

The following image compilations (5.15, 5.16, 5.17) showcase the semantic output of the final detection architecture and compare the outputs regarding the different loss functions applied.

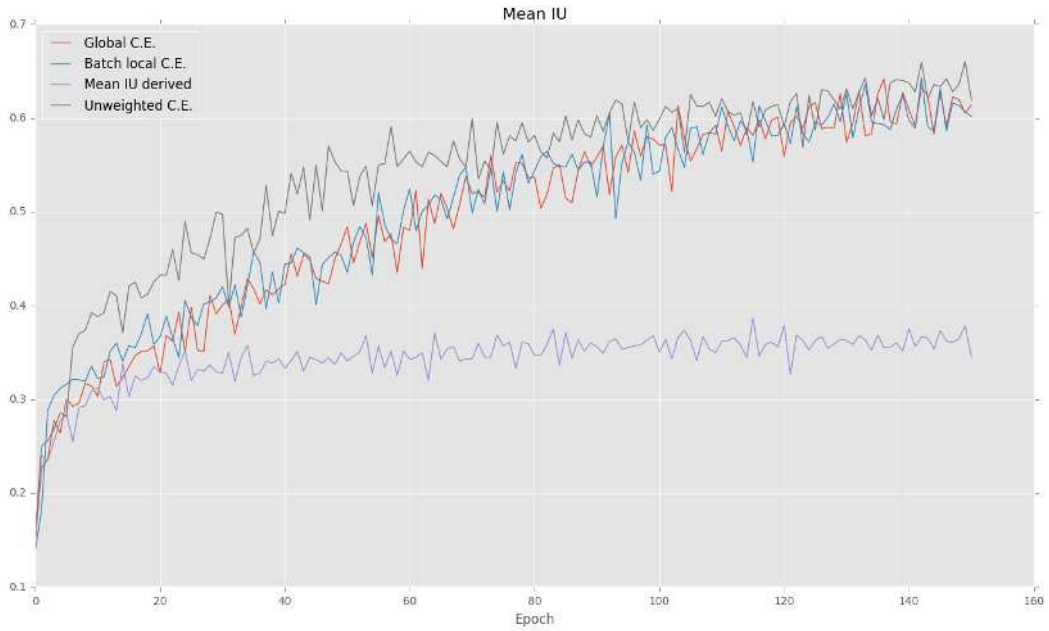


Figure 5.13: Mean IU of detection task training

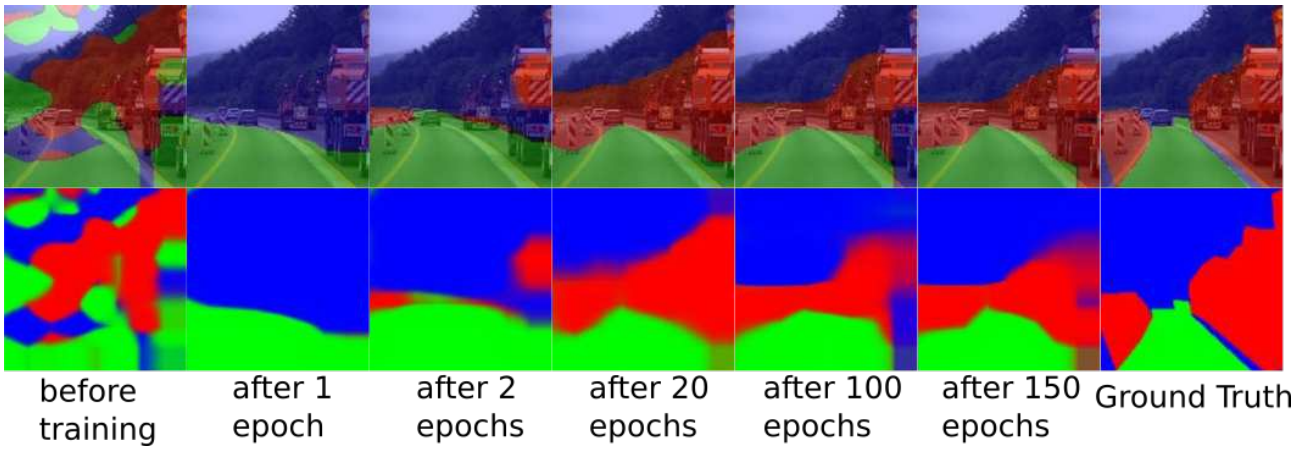


Figure 5.15: Example development of semantic segmentation output over time

At first, figure 5.15 illustrates the general form and type of the semantic network output. While the second row of the figure corresponds to the original network output with respect to the specified epoch, the first row shows a blended image of this output and the original input image. The last two images in both rows show the respective ground truth data. Referred to the small softmax dimensions of 8x8 pixels, 5.9 and thus only 8x8 classification scores upsampled 16-fold to 128x128 pixels the final semantic output of epoch 150 is almost comparable to the actual manually annoted ground truth

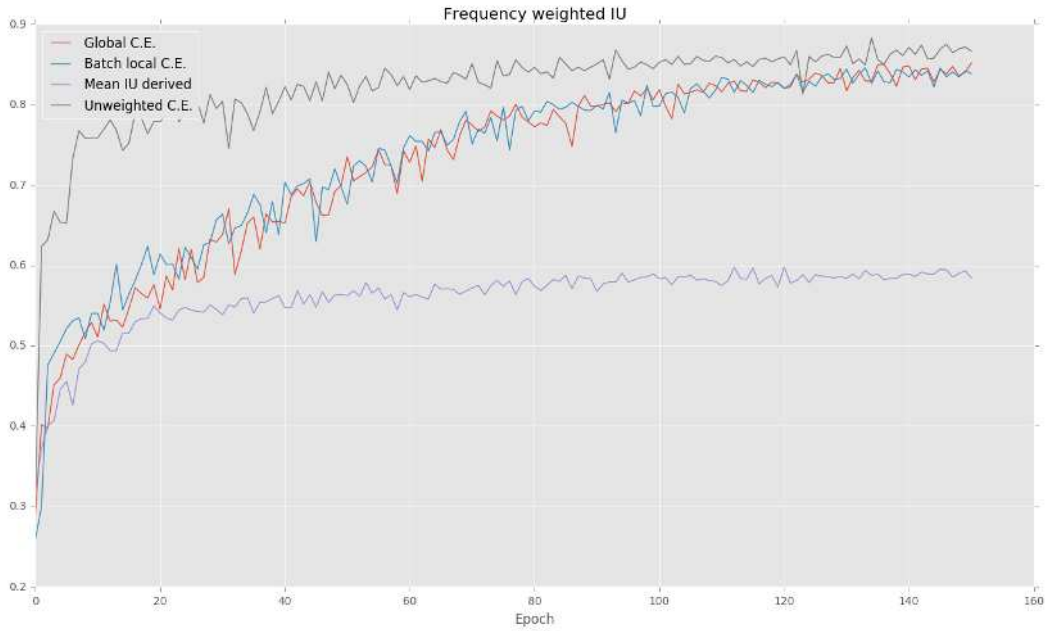


Figure 5.14: Frequency weighted IU of detection task training

image.

The following two selections compare the detection output with respect to the different applied loss functions.

The first compilation focuses on detection outputs for positive input images (images with a construction site). The second one compares detection outputs of negative input images (images with only background).

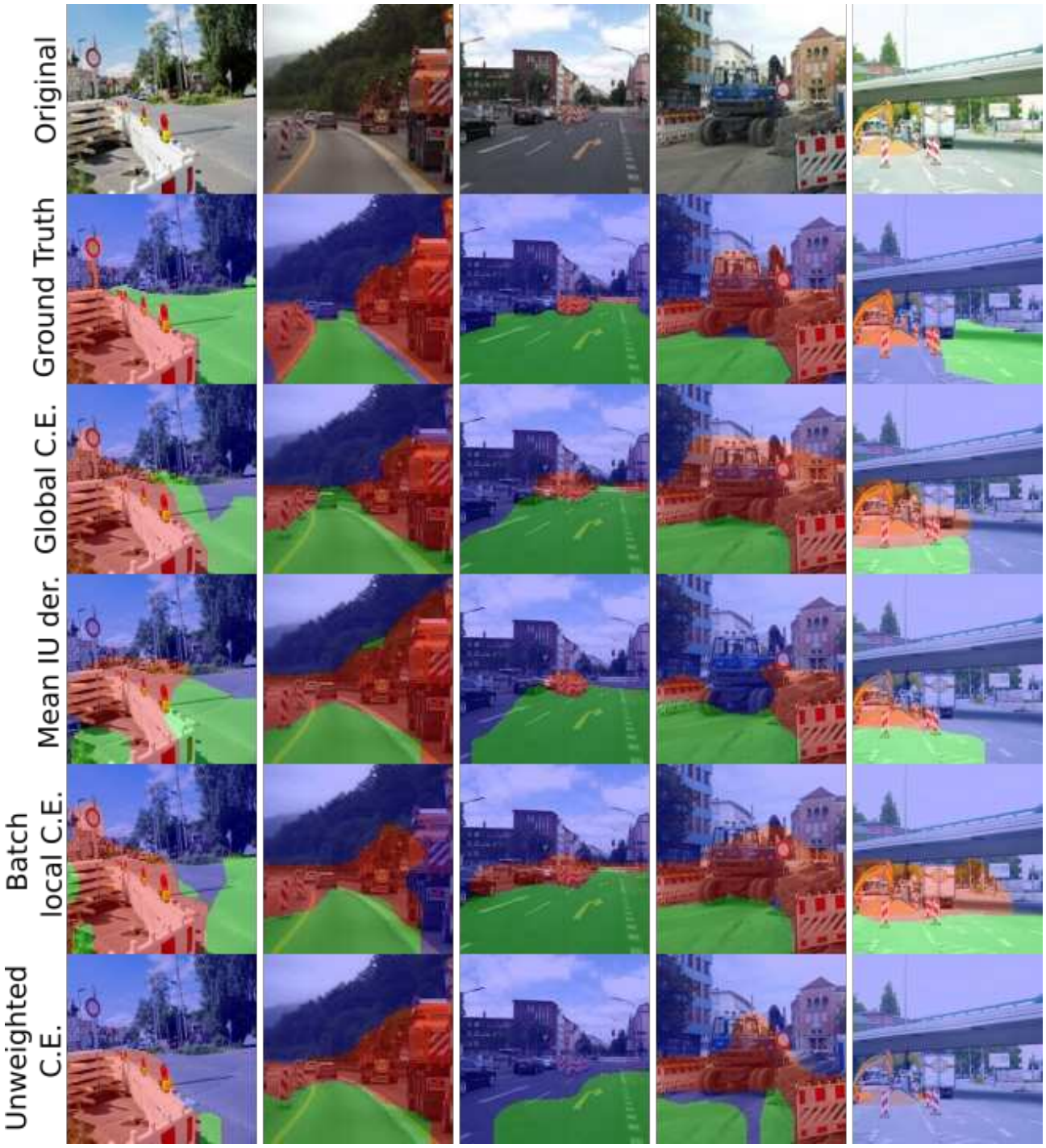


Figure 5.16: Comparison of ground truth and outputs for construction site images

Comparing the semantic outputs of figure 5.16 in terms of visual accuracy, first of all the last row is noticeable. Regarding the other three loss functions, the semantic output is less continuous and indicates that background pixels clearly are preferred compared to passable road and construction site pixels. For example, the first image hardly shows passable road detections and the round, red-white construction site sign is not included in the construction site area. The third image shows only detections of roughly the half of the passable road area. Further the construction site elements like vertical panels are not even noticed on this image. The fourth image shows a detection of the passable road area, however, the detection is discontinuous and broken by pixels that have been detected as background. The construction site that is clearly visible on the last image is even not detected at all.

Looking at the fourth row shows, that despite of its improvable accuracy metrics, the output of the mean IU derived loss is even slightly better than the output of the unweighted cross-entropy. There are only some smaller inaccuracies.

The semantic output of the global as well as the batch local cross-entropy are quite similar as expected. Especially the third column is remarkable. Compared to the ground truth and regarding the small resolution of the classification outputs (8x8 pixels) of the network, both global and batch local approaches accurately detect the construction site as well as passable road areas. The right and more distant part of the construction site, which is hardly recognizable even for a human being (having an image with a resolution of only 128x128 pixels), has precisely been detected.

The last column shows a further interesting example. The ground truth of the image only has highlighted the right part of the road to be a passable road. This annotation has been made due to the point of view of the image and the direction arrow of the left turning lane, which indicates that the current lane points in the opposite direction and thus should not be used by the autonomous vehicle. However, the image set for training is still very sparse related to the broad domain of construction sites, especially for these types of situations. Thus the detections shown in the last column, except the one of the unweighted cross-entropy approach, are in fact quite accurate and reasonable.

To sum up the observations of this first selection, the outputs of the global and batch local approaches are very accurate. The outputs of the unweighted cross-entropy approach are less accurate, since the background is prioritized over construction site and passable road pixels. The mean IU approach produced a reasonable output even more accurate than the unweighted cross-entropy approach.

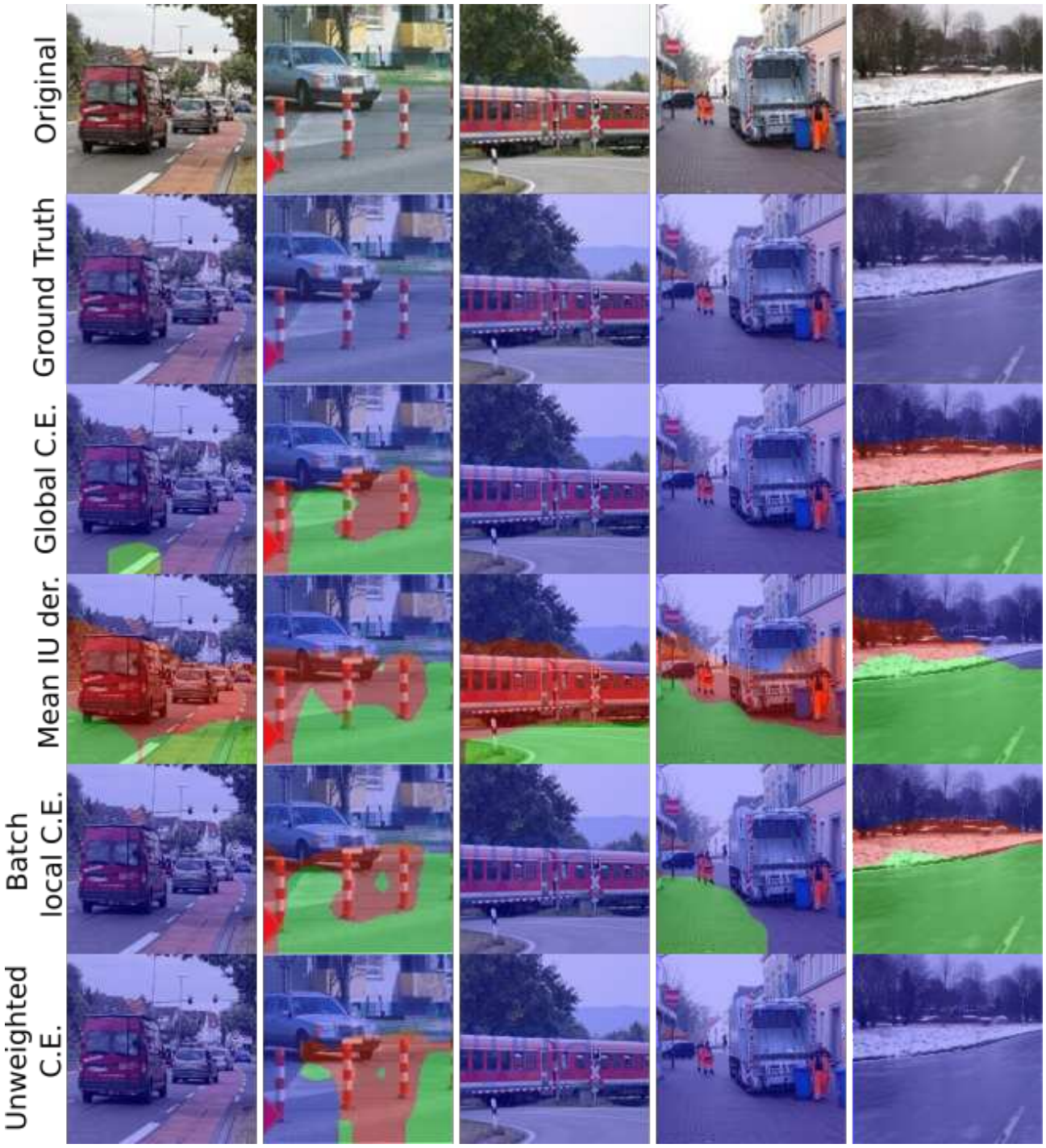


Figure 5.17: Comparison of ground truth and outputs for background images

The next selection of outputs (5.16) shows semantic outputs for negative input images. The desired output of any background input image is simply a blue image without any other colors.

The first noticeable aspect is the fourth row, which in all cases detects areas of construction sites and passable roads, although there is only background. This basically reflects the low accuracy of the mean IU derived loss reported in 5.4.3. The unweighted cross-entropy approach, however, obviously has a high accuracy, as it has prioritized the background pixels. The batch local and global approaches are comparable as expected. While the global weighted approach has still difficulties to classify red-white transitions of cycle paths as background, the batch local approach incorrectly classifies passable road sections, although no construction site is visible.

The second column shows an image containing white-red bollards not belonging to a construction site. It is misclassified by each approach as construction site. This is assumed to be due to the sparsity of the training set. Finally, the same applies to the last column. It shows a road during the winter season with snow in the background. As the training set contains almost no images of the winter season, this type of misclassification is assumed to be preventable by a broader range and diversity of the training set of images.

Summary of visual results In summary, four important observations have been made:

1. The global weighted cross-entropy as well as the batch local weighted approach provide highly accurate results.

2. The unweighted cross-entropy approach is less accurate despite of its high accuracy metrics, reported in 5.4.3, which is due to its prioritization of background pixels.

3. The mean IU derived approach clearly underperforms compared to the cross-entropy based loss function approaches.

4. The applied training set that originates from less than thousand images is not sufficient to cover the broad domain of road construction sites adequately.

Losses and filters Detailed loss function charts (6.9) and learned first layer filters (6.10) according to the detection task can be found in the appendix.

6 Conclusion

To return to the introductory question of how to enable self-driving vehicles to deal with the chaotic reality, this work gives further reason to understand deep learning as an indispensable part of future self-driving systems.

The goal has been to examine how deep learning is able to support self-driving vehicles to deal with unexpected situations in the field of road construction sites. The basic idea behind using deep learning has been the visual complexity and feature related uniqueness inherent to road construction sites compared to everyday road scenes. Therefore, a deep learning approach has been evident.

As there are no specific rules or guidelines of designing a suitable deep learning architecture, the development of the final detection architecture has been divided up into a two step process. In a first step a deep learning architecture as well as a suitable training data set has been created to first solve the corresponding classification problem, which is a sub-task of the final detection task. Therefore, the training images have been labeled with the two classes "construction site" and "no construction site". As stated in the results (4.4), the developed classification architecture finally reached a peak accuracy of 0.9125 on the validation set.

In a second step this architecture has been rearranged by converting the overall convolutional neural network into a fully convolutional neural network, which supports a pixel level classification as used in semantic segmentation. The three considered pixel classes are "construction site", "passable road" and "background". Essentially, the final detection task has been reinterpreted as a semantic segmentation task with a supporting background class for detection. For being able to train this network on a pixel level, all training images that contain construction sites had to be manually annotated to produce the corresponding ground truth data for every single pixel. This has been very time-consuming and took 20 to 30 minutes on average per image. For the final training process, different loss functions have been tested in order to determine, which loss definition achieves the best accuracy regarding to the desired output. This design aspect has been more challenging for the pixelwise detection architecture, since any of the three considered classes has a different number of pixels in total, which in effect causes

a different prioritization of the different pixel classes. As stated in the results (5.4), the best accuracy has finally been achieved by a pixelwise cross-entropy loss function that weights the different pixel classes according to their occurence in relation to the pixel class that owns the most pixels (the background).

The deep neural network model that has evolved throughout this work has finally shown surprisingly good performance with regard to a relatively scarce and inconsistent dataset. Moreover, it shows a fast execution time as well as a good cross-platform capability due to the use of TensorFlow[AAB+16] and thus essentially fits the requirements stated in the introduction.

That is why the final outcome of this work represents a suitable proof of concept that can be applied to future self-driving systems.

Bibliography

[AAB+16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[AB09] Secret Labs AB. Python imaging library (version 1.1.7). `http://www.pythonware.com/products/pil/`, 2009.

[aLoBeHnt] Y. ann LeCun, Y. oshua Bengio, and G. eoffrey Hinton. - Deep learning. - 521(- 7553):– – 444, - 2015/05/28/print.

[BH00] I.A Basheer and M Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3 – 31, 2000. Neural Computting in Micrbiology.

[BRSS15] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. Comparative study of caffe, neon, theano, and torch for deep learning. *CoRR*, abs/1511.06435, 2015.

[COR+16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[CWV+14] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014.

[DCM+12] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[eG16] Pierre e Gougelet. Xnconvert (version 1.73 - un*x x64 (jun 17 2016) - libformat version 6.90). `http://www.xnview.com/de/xnconvert/`, 2016.

[EGW+10] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010.

[Gol16] Peter Goldsborough. A tour of tensorflow. *CoRR*, abs/1610.01178, 2016.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[JMM96] A. K. Jain, Jianchang Mao, and K. M. Mohiuddin. Artificial neural networks: a tutorial. *Computer*, 29(3):31–44, Mar 1996.

[KH09] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[KKvdSS93] Ben Kröse, Ben Krose, Patrick van der Smagt, and Patrick Smagt. An introduction to neural networks, 1993.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LBBH98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[LSD14] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[Ló16] Adrián López. fdupes (version: 1.51). `https://github.com/adrianlopezroche/fdupes`, 2016.

[MA11] K Jarrod Millman and Michael Aivazis. Python for scientists and engineers. *Computing in Science & Engineering*, 13(2):9–12, 2011.

[MP43] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[NHH15] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

[Pha] Ekachai Phaisangittisagul. An analysis of the regularization between l2 and dropout in single hidden layer neural network.

[Pit11] Michael Pitidis. Pylabelme (git commit: af1e8896ddf6b029377be78f3e4b360997466cf3). `https://github.com/mpitid/pylabelme`, 2011.

[RDS+15a] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[RDS+15b] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[Sad16] Peter Sadowski. Notes on backpropagation, 2016.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015.

[Sch16] Martin Schrimpf. Should I use tensorflow. *CoRR*, abs/1611.08903, 2016.

[SEZ+13] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[SK13] Priyanka Sharma and Manavjeet Kaur. Classification in pattern recognition: A review. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 3(4), 2013.

[SLD16] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1605.06211, 2016.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[ST16a] Leslie N. Smith and Nicholay Topin. Deep convolutional neural network design patterns. *CoRR*, abs/1611.00847, 2016.

[ST16b] Leslie N. Smith and Nicholay Topin. Deep convolutional neural network design patterns. *CoRR*, abs/1611.00847, 2016.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[Tho16] Martin Thoma. A survey of semantic segmentation. *CoRR*, abs/1602.06541, 2016.

[WYS+15] Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. Deep image: Scaling up image recognition. *arXiv preprint arXiv:1501.02876*, 7(8), 2015.

[XHE+10] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.

[ZLX+14] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 487–495. Curran Associates, Inc., 2014.

List of Tables

4.1 Confusion matrix for the validation set of 298 images (using weights of the 83th epoch). 41

6.1 Hyperparameters and properties of the final construction site classification architecture 82
6.2 Construction site classification architecture in detail 83
6.3 Hyperparameters and properties of the final construction site detection architecture 84
6.4 Construction site detection architecture 85

List of Figures

1.1 Example scenarios 2
1.2 Example of construction site specific rules and regulations 3
1.3 Construction sites are characterized by specific features 4

2.1 Artificial neuron (Perceptron) 7
2.2 Activation functions - examples 8
2.3 Artificial neural network (Multi-layer perceptron (MLP)) 9
2.4 Backpropagation 10
2.5 Backpropagation (relating to the squared-error cost function used above) [aLoBeHnt] 11
2.6 Over- and underfitting 12
2.7 Dropout example [SHK+14] 13
2.8 2d Convolution Example 14
2.9 2d Convolution Example 15
2.10 From convolution to convolutional neural networks 16
2.11 2x2 MaxPool layer 17
2.12 CNN overview (LeNet-5 [LBBH98]) 17
2.13 Examples of layer 1 filters [KSH12] 18
2.14 Transform fully connected into local connected convolution layers 18

3.1 TensorFlow stack 21
3.2 Example of a TensorFlow data flow graph. [AAB+16] 22
3.3 TensorFlow partial execution. [AAB+16] 23
3.4 Automatic gradient computation [Gol16] 23
3.5 Tool setup and development strategy 25

4.1 Computer vision tasks classification, localization and detection with increasing order of complexity 27
4.2 Sample images with construction site (positive class) 30
4.3 Sample images without construction site (negative class) 31
4.4 Data preparation using the Python Imaging Library (PIL) [AB09] 32

4.5 Illustration of the "Deep Learning Pipeline" implemented using Python and TensorFlow . . . 33
4.6 64x64 images lose important structural information . . . 34
4.7 An illustration of the AlexNet architecture. [KSH12] . . . 35
4.8 Deriving an initial prototype from the AlexNet architecture. [KSH12] . . . 36
4.9 Deep learning architecture for classification task . . . 38
4.10 Accuracy of the training and validation sets . . . 40
4.11 Cross-entropy loss . . . 42
4.12 Selection of best classified construction site images with their corresponding softmax values. . . . 43
4.13 Selection of best classified background images with their corresponding softmax values. . . . 44
4.14 Selection of false positives with their corresponding softmax values. . . . 45
4.15 Selection of false negatives with their corresponding softmax values. . . . 46
4.16 Boundary cases of construction site images with a softmax value near 0.5. 46
4.17 Boundary cases of background images with a softmax value near 0.5. . . . 47
4.18 Layer-1 Filters (83th epoch) . . . 47

5.1 Computer vision tasks classification, localization and detection with increasing order of complexity . . . 48
5.2 Reviewing the classical detection approach . . . 49
5.3 Comparison of semantic segmentation and detection . . . 50
5.4 Construction site detection interpreted as semantic segmentation task . . . 51
5.5 Annotation process overview . . . 52
5.6 Sample annotations . . . 54
5.7 "Deep Learning Pipeline" and parts to be modified for the detection task 55
5.8 Semantic segmentation using a fully convolutional neural network and bilinear upsampling [SLD16] . . . 56
5.9 Illustration of the upsampling dimensions . . . 56
5.10 Deep learning architecture for detection task . . . 59
5.11 Pixel accuracy of detection task training . . . 62
5.12 Mean accuracy of detection task training . . . 63
5.13 Mean IU of detection task training . . . 64
5.15 Example development of semantic segmentation output over time . . . 64
5.14 Frequency weighted IU of detection task training . . . 65
5.16 Comparison of ground truth and outputs for construction site images . . . 66
5.17 Comparison of ground truth and outputs for background images . . . 68

6.1 Validation accuracy for globally weighted cross-entropy loss 86
6.2 Training accuracy for globally weighted cross-entropy loss 86
6.3 Validation accuracy for local batch weighted cross-entropy loss 87
6.4 Training accuracy for local batch weighted cross-entropy loss 87
6.5 Validation accuracy for mean IU derived loss 88
6.6 Training accuracy for mean IU derived loss 88
6.7 Validation accuracy for unweighted cross-entropy loss 89
6.8 Training accuracy for unweighted cross-entropy loss 89
6.9 Loss values for detection task training 90
6.10 Layer-1 filters of detection task training (150th epoch) 90

Appendices

Property	**Value / Implementation**
Initializations	• Learning rate: 0.1 • Weights: $\mathcal{N}(\mu = 0, \sigma^2 = 0.05)$ • Biases: 0
Input (0)	• Input size: 128x128 • Mini-batch size: 50
Preprocessing (1)	1. Random Horizontal Flip 2. Random Brightness 3. Random Contrast 4. Normalization (zero mean and unit variance)
Output (4)	Softmax
Loss (5)	• cross entropy combined with • L2 regularization (regularization strength: 0.01)
Update rule (6)	• Exponential decay of the learning rate with decay rate(0.8) and decay steps (10) • Mini-batch gradient descent without optimization

Table 6.1: Hyperparameters and properties of the final construction site classification architecture

layer index	layer type	output size	receptive field size	zero padding	stride	#params
#0	input image of size 128x128x3					
#1.1	Conv	128x128x16	5x5	SAME	1	1216
#1.2	ReLU	128x128x16	-	-	-	-
#1.3	MaxPool	64x64x16	2x2	SAME	2	-
#2.1	Conv	64x64x32	5x5	SAME	1	12832
#2.2	ReLU	64x64x32	-	-	-	-
#2.3	MaxPool	32x32x32	2x2	SAME	2	-
#3.1	Conv	32x32x50	5x5	SAME	1	40050
#3.2	ReLU	32x32x50	-	-	-	-
#3.3	MaxPool	16x16x50	2x2	SAME	2	-
#4.1	Conv	16x16x64	5x5	SAME	1	80064
#4.2	ReLU	16x16x64	-	-	-	-
#4.3	MaxPool	8x8x64	2x2	SAME	2	-
#5.1	Conv	8x8x64	5x5	SAME	1	102464
#5.2	ReLU	8x8x64	-	-	-	-
#5.3	MaxPool	4x4x64	2x2	SAME	2	-
#6	FC	1x1x512	-	-	-	524800
#7	Dropout (50%)	1x1x512	-	-	-	-
#8	FC	1x1x128	-	-	-	65664
#9	FC	1x1x2	-	-	-	258
#10	Softmax	1x1x2	-	-	-	-
#11	2 softmaxed output values (background and construction site score)					

Table 6.2: Construction site classification architecture in detail

Property	Value / Implementation
Initializations	• Learning rate: 0.01 for cross-entropy based loss functions, 0.00001 for mean IU derived loss • Weights: $\mathcal{N}(\mu = 0, \sigma^2 = 0.01)$ • Biases: 0
Input (0)	• Input size: 128x128 • Mini-batch size: 50
Preprocessing (1)	1. Random Horizontal Flip 2. Random Contrast
Output (4)	Softmax (size: 8x8x3) upsampled to 128x128x3
Loss (5)	• Pixelwise cross-entropy (unweighted, globally weighted, batch local weighted) and mean IU derived loss defined in 5.4.2 • L2 regularization (regularization strength: 0.01)
Update rule (6)	• Exponential decay of the learning rate with decay rate(0.8) and decay steps (10) • Mini-batch gradient descent without optimization

Table 6.3: Hyperparameters and properties of the final construction site detection architecture

layer index	layer type	output size	receptive field size	zero padding	stride	#params
#0	input image of size 128x128x3					
#1.1	Conv	128x128x16	3x3	SAME	1	448
#1.2	ReLU	128x128x16	-	-	-	-
#1.3	MaxPool	64x64x16	2x2	SAME	2	-
#1.4	BatchNorm (scaling)	64x64x16	-	-	-	factor = 100
#2.1	Conv	64x64x32	3x3	SAME	1	4640
#2.2	ReLU	64x64x32	-	-	-	-
#2.3	MaxPool	32x32x32	2x2	SAME	2	-
#2.4	BatchNorm (scaling)	32x32x32	-	-	-	factor = 10
#3.1	Conv	32x32x50	3x3	SAME	1	14450
#3.2	ReLU	32x32x50	-	-	-	-
#3.3	MaxPool	16x16x50	2x2	SAME	2	-
#3.4	BatchNorm (scaling)	16x16x50	-	-	-	factor = 10
#4.1	Conv	16x16x64	3x3	SAME	1	28864
#4.2	ReLU	16x16x64	-	-	-	-
#4.3	MaxPool	8x8x64	2x2	SAME	2	-
#4.4	BatchNorm (scaling)	8x8x64	-	-	-	factor = 10
#5.1	Conv	8x8x64	3x3	SAME	1	36928
#5.2	ReLU	8x8x64	-	-	-	-
#5.3	BatchNorm (scaling)	8x8x64	-	-	-	factor = 10
#6.1	Conv	8x8x512	3x3	SAME	1	295424
#6.2	ReLU	8x8x512	-	-	-	-
#6.3	BatchNorm (scaling)	8x8x512	-	-	-	factor = 10
#7	Dropout (50%)	8x8x512	-	-	-	-
#8.1	Conv	8x8x128	3x3	SAME	1	589952
#8.2	ReLU	8x8x128	-	-	-	-
#8.3	BatchNorm (scaling)	8x8x128	-	-	-	factor = 10
#9.1	Conv	8x8x3	1x1	SAME	1	3459
#9.2	BatchNorm (scaling)	8x8x3	-	-	-	factor = 10
#10.1	Softmax	8x8x3	-	-	-	-
#10.2	Bilinear Upsampling	128x128x3	-	-	-	-
#11	output image of size 128x128x3 (semantic segmentation)					

Table 6.4: Construction site detection architecture

Detailed accuracy values for detection training

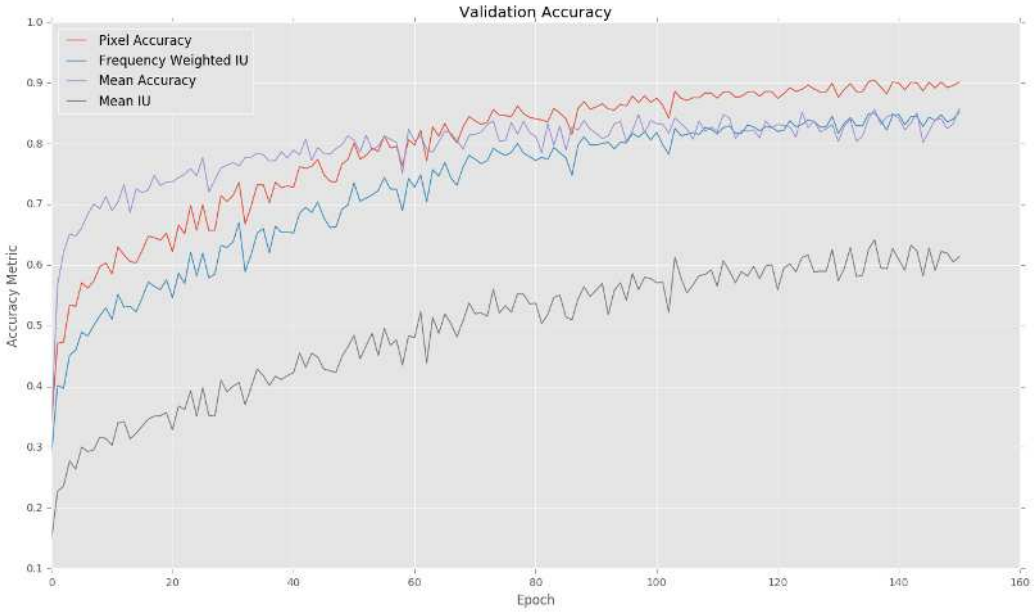


Figure 6.1: Validation accuracy for globally weighted cross-entropy loss

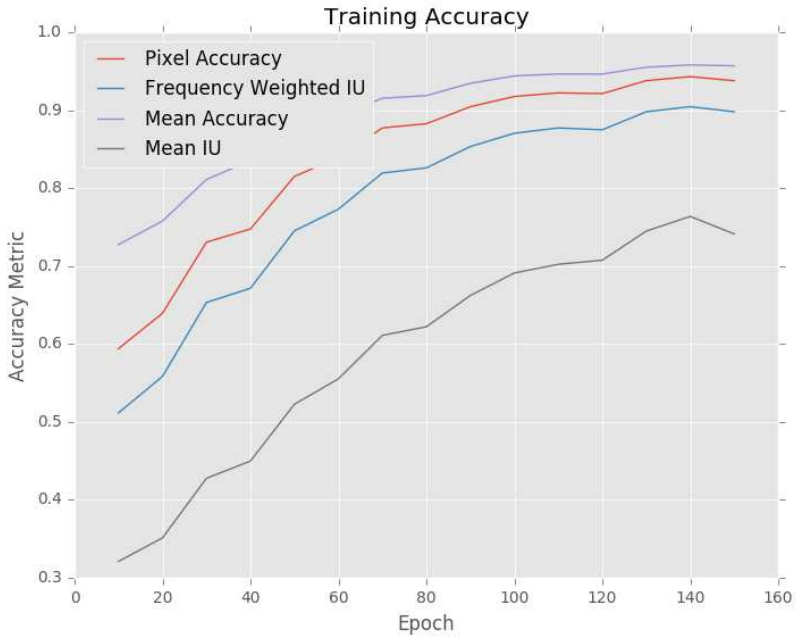


Figure 6.2: Training accuracy for globally weighted cross-entropy loss

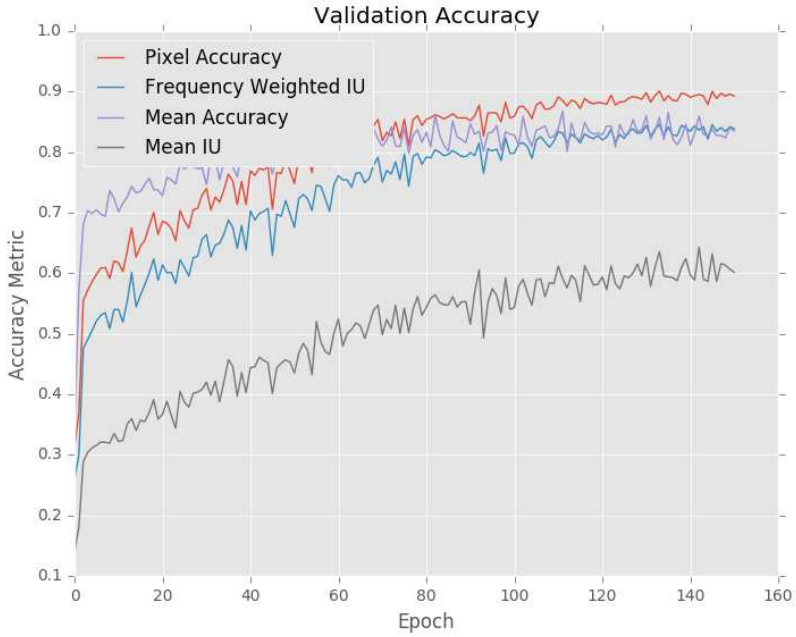


Figure 6.3: Validation accuracy for local batch weighted cross-entropy loss

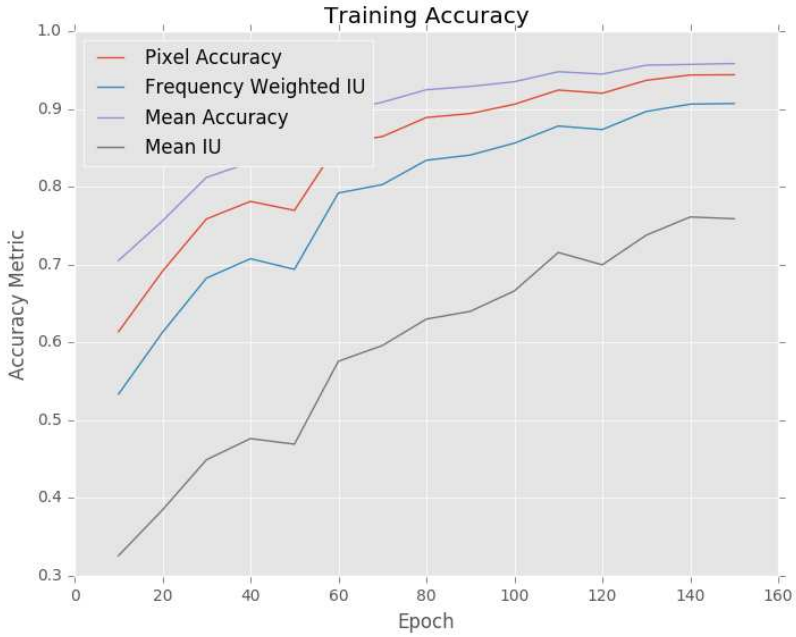


Figure 6.4: Training accuracy for local batch weighted cross-entropy loss

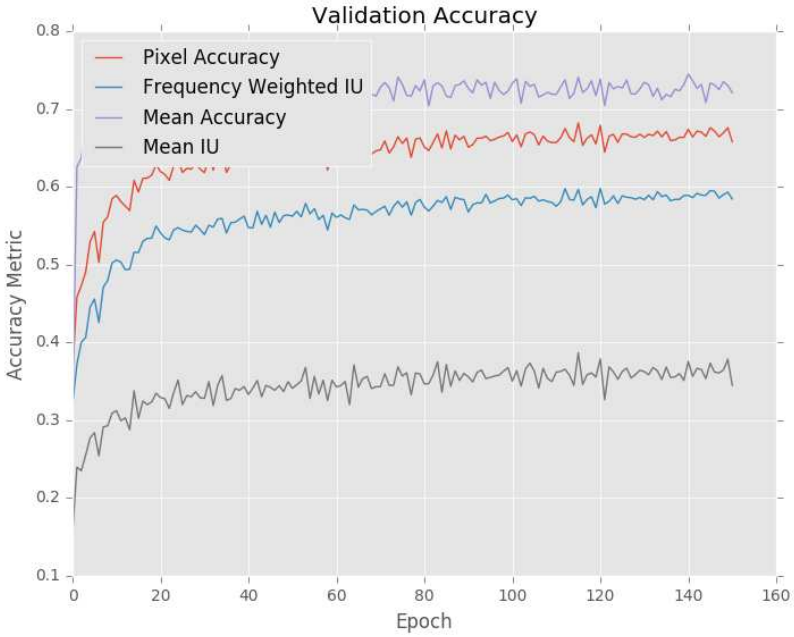


Figure 6.5: Validation accuracy for mean IU derived loss

Figure 6.6: Training accuracy for mean IU derived loss

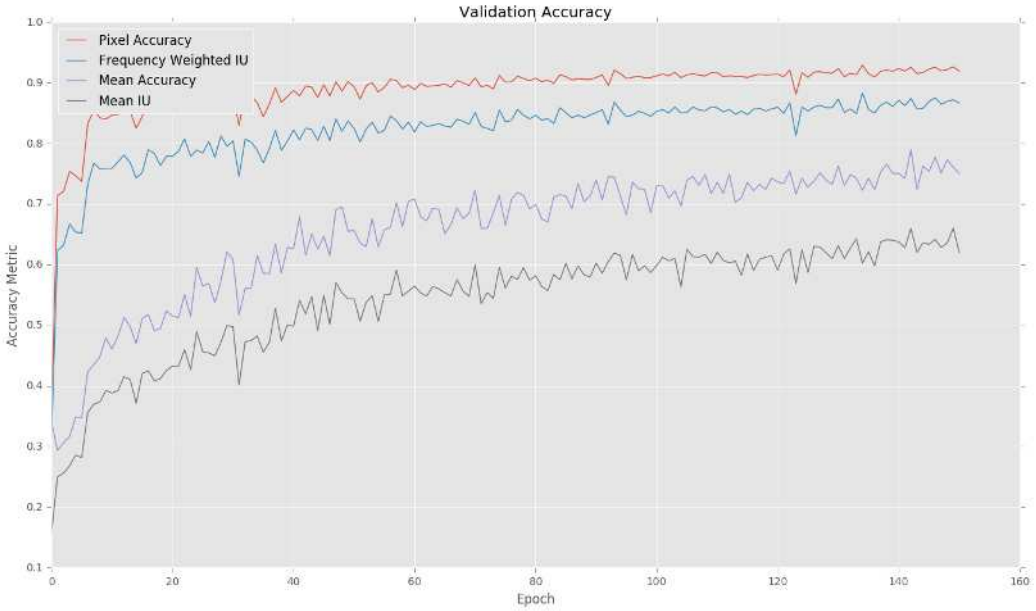


Figure 6.7: Validation accuracy for unweighted cross-entropy loss

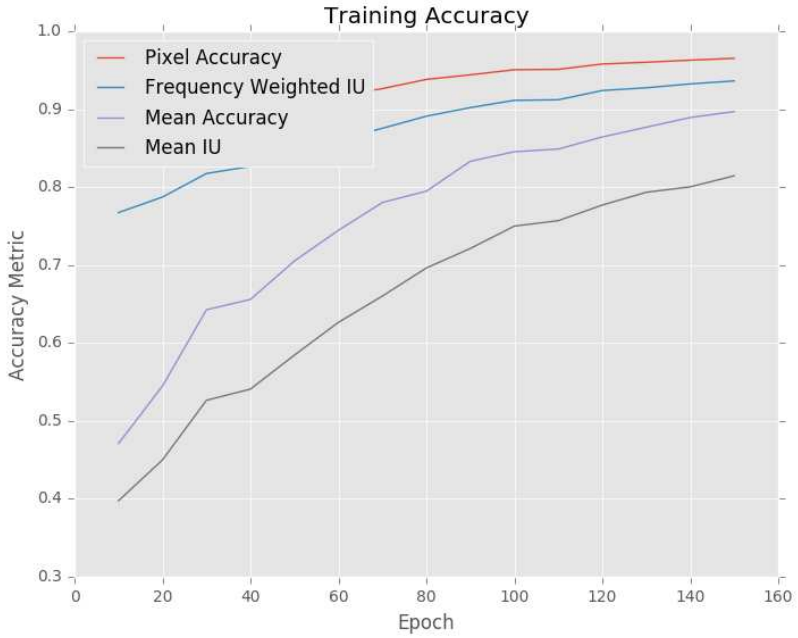
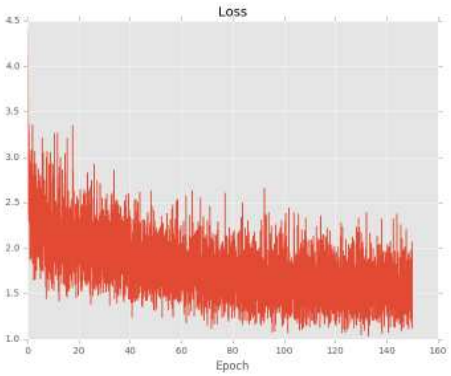
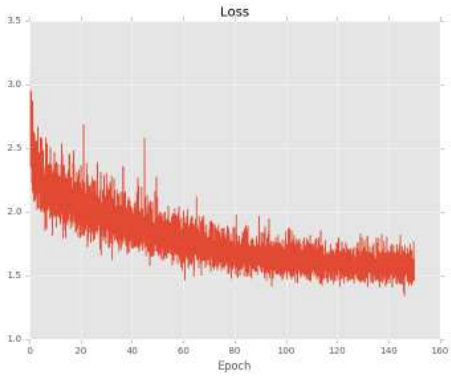


Figure 6.8: Training accuracy for unweighted cross-entropy loss

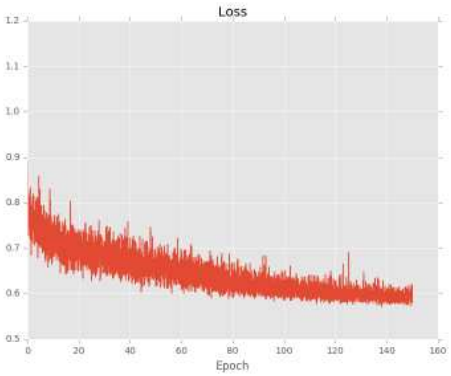
Detailed loss values and filters for detection training



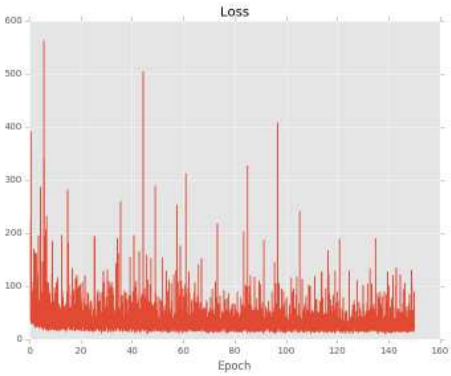
(a) Global weighted C.E.



(b) Batch local weighted C.E.



(c) Unweighted C.E.



(d) Mean IU derived loss

Figure 6.9: Loss values for detection task training

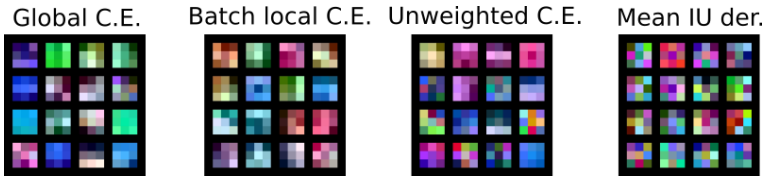


Figure 6.10: Layer-1 filters of detection task training (150th epoch)

Attached media

The attached disk contains following additional data:

- Source code [1]
- Code manual
- Datasets
  - Annotated dataset (detection task)
  - Originals
  - Preprocessed training datasets
- Detailed HTML documentation of the classification task
- Training outputs for the classification and detection tasks

[1] packaged as a Python package according to the guidelines of `https://packaging.python.org/`