Universiteit Leiden

Master Computer Science

Pavement Distress Classification
in 3D Pavement Measurements
using Convolutional Neural Networks

Name:	Thierry van der Spek
Student ID:	s0922714
Date:	23/04/2018
Specialisation:	Advanced Data Analytics
1st supervisor:	Dr. W.J. Kowalczyk
2nd supervisor:	Prof. Dr. T.H.W. Bäck

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Modern roads provide higher levels of mobility than ever before, and the number of vehicles on those roads has been increasing since the beginning of the 20th century. This development has led to automated road surface scanning and processing systems for the detection of damage. Recently, systems have been developed for the detection of cracks in 3D pavement surface measurements. These systems, however, often detect patterns that are not indicative of structural pavement damage. To improve the time-to-maintenance prediction for highway agencies, the number of these false-positive detections should be minimized. In this work, we, therefore, propose a convolutional neural network approach to improve the classification of pavement cracks. The proposed method has been tested on real-world pavement scans, captured by the Dutch Highway Agency (Rijkswaterstaat) and The Netherlands Organisation for Applied Scientific Research (TNO). Using this data, we show that the proposed method is a useful addition to the present maintenance pipeline, aiding the current manual labelling approach.

Contents

1 Introduction **8**

2 Problem Statement **10**

2.1 Pavement construction 10

2.2 Pavement distress 11

2.3 Problem description 13

2.4 Research questions 15

3 Related and Previous work **16**

3.1 Ravelling 16

3.2 Cracking 18

3.3 Our contribution to literature 21

4 Background **22**

4.1 Supervised learning 22

4.2 Convolutional neural networks 35

4.3 State-of-the-art neural networks 38

5 Methodology **43**

5.1 Data 43

5.2 Model definition 50

6 Experiments **56**

6.1 Experimental setup 56

6.2 Classification using features 59

6.3 Base convolutional models 61

6.4 Custom Architecture 64

6.5 Data and model ensembling 66

6.6 Human performance 71

6.7 A look into mis-classification 72

7 Discussion and Outlook **75**

List of Figures

2.1 Illustration of the intersection of porous asphalt (ZOAB) in the Netherlands. Adapted from [50] 11
2.2 Ravelling of porous asphalt in the Netherlands. Adapted from [1]. 12
2.3 Different crack types according to AASHTO. Taken from [59] 13
2.4 A range image of an actual crack (left) and a false positive (right). A pixel's intensity represents the relative measuring distance of the capturing sensor to the surface (darker is farther) . . . 14

3.1 Stoneway algorithm by Ooijen et al [82]. 17

4.1 A graphical representation of a linear model. Input $\boldsymbol{x}$ is weighted by $\boldsymbol{\theta}$ and results are summed to form y. Input x_0 is again set to a constant 1. 23
4.2 Graphical model of logistic regression. 24
4.3 Graphical model of a simple feed forward neural network as an acyclic graph. 25
4.4 Relationship between model capacity and error. Adapted from Goodfellow et al. [26] 33
4.5 AlexNet architecture. The network is split into two streams to distribute computation over two graphics processing units (GPUs). Adapted from Krizhevsky et al. [43] 38
4.6 VGG-16 architecture [7]. VGG-16 starts out with a small number of channels and doubles it after every max-pooling layer while the max-pooling halves the spatial dimension of the generated feature maps. 39
4.7 Inception module, adapted from [76]. To match up output dimensions same-convolutions are used and padding is added for max-pooling. On the left (a) is the naive version, without dimensionality reduction. On the right (b) 1×1 convolutions are used to reduce dimensionality 39
4.8 Inception modules with dimension reductions, adapted from [77]. On the left (a) convolutions are factorized into convolutions with smaller kernel sizes. On the right (b), this idea is taken further to factorize into asymmetric convolutions, ... 40
4.9 Residual block, adapted from [29]. 41
4.10 An Xception module as visualized by Chollet [13]. 41

5.1 General methodology pipeline 43
5.2 An illustration of the LCMS sensor and a vehicle with these sensors, showing how the data is captured.[2] 44

5.3 Labelling process as done by inspectors. Detected crack-like patterns (yellow) are labelled into either crack (blue) or non-crack (green). 46
5.4 Cumulative distributions of the length (a) and width (b) of cracks and non-crack patterns, taken from equally sized subsets of 10k samples. 47
5.5 Distributions of the angle of cracks (a) and non-crack patterns (b) of cracks and non-crack patterns, taken from equally sized subsets of 10k samples. 47
5.6 Distributions of the position of the crack or not-crack pattern in a lane, taken from equally sized subsets of 10k samples. 48
5.7 Pattern sampling. The red dot indicates the sampled point. Left: a full lane hectometre with 2 patterns. Middle: zoomed into the pattern on the top right. Right: sampled patch from the same pattern, where the patch is centred around the red dot. 50
5.8 General structure of a CNN, used for classification. Modified from [87] 51
5.9 Condensed architecture overview of the Inception-ResNet architecture [75]. 53
5.10 Inception-ResNet modules [75]. The blocks correspond (from left to right) with the coloured blocks from Figure 5.9 53

6.1 Exponentially increasing learning rate vs. validation loss for different patch sizes for the Inception-ResNet architecture. 57
6.2 Feature importance for training a XGBoost model 60
6.3 AP of several CNNs on the class-balanced holdout set. Pre-trained indicates the convolutional base of the model, pre-trained on ImageNet, was used. 62
6.4 AP vs. training set size. Inception-ResNet was trained on the respective fractions of the training set size and AP results are reported on the test set. 63
6.5 AP-score for different k values on different patch sizes 64
6.6 AP-score vs. pattern length (left), width (middle) and severity(right). 65
6.7 Number of patches vs AP-score 67
6.8 Average overlap fraction vs. the maximum size spanned by the crack 68
6.9 Five-patch single model ensemble 69
6.10 Multi-model, single-patch model ensemble 69
6.11 Single patch, multi-patch ensemble, multi-model ensemble and their combination 70
6.12 False negative detections. On the right is the annotated crack present in the left image. Blue is a crack, where green is a non-crack 73
6.13 False positive detections. Left is the original patch, middle is the guided backpropagation mask, and right is an overlay of the two. 74

1 Examples of different cracks (1 m^2) 87
2 Examples of different non-crack patterns (1 m^2) 88

List of Tables

5.1 Crack classes according to the Pavemetrics/TNO system for measurement year 2016. 45
5.2 Different 'classical' features of cracks and non-crack patterns 51

6.1 XGBoost AP-score on test set for different tree depths and number of trees. 60
6.2 AP of several CNNs on the test set. Pre-trained indicates the convolutional base of the model, pre-trained on ImageNet, was used. 61
6.3 Number of parameters vs. model size, given by k 65

Nomenclature and Notation

List of used symbols

x	A scalar, either real or integer
$\boldsymbol{x}$	A vector of reals or integers
$\boldsymbol{x}^T$	The transpose of a vector $\boldsymbol{x}$. Turning a column vector into a row vector
$\boldsymbol{X}$	A matrix of reals or integers
$\boldsymbol{X}^{\text{TRAIN}}$	A matrix of training samples, where a row i is represented as a vector $\boldsymbol{x^{(i)}}$
$\boldsymbol{x}^{(i)}$	The i-th example in a matrix $\boldsymbol{X}$
$x^{(i)}$	The i-th value in a vector $\boldsymbol{x}$
$\mathbb{A}$	A generic set
$\mathbb{R}$	The set of real numbers
$f : \mathbb{A} \to \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function f of $\boldsymbol{x}$, parametrized by $\boldsymbol{\theta}$
$J(\boldsymbol{x}; \boldsymbol{\theta})$ or $J(\boldsymbol{\theta})$	Cost function J of $\boldsymbol{x}$, parametrized by $\boldsymbol{\theta}$, outputting a single real number
$\log x$	The natural logarithm of x
$\arg\max f$	The points of the domain of f at which the function values are maximized

List of domain related terms

Rijkswaterstaat	The Dutch Highway Agency
pavement	General term for surface material intended to sustain vehicular movement.
porous asphalt	An open grade type of pavement, mostly used in the Netherlands
dense asphalt concrete	A closed grade type of pavement, historically served as a baseline for noise reduction
ravelling	Significant detachment of aggregate particles and the asphalt binder from the pavement surface
rutting	Significant deformation of the asphalt in the driving direction
cracking	Interconnected deep breaking of the pavement surface

List of used abbreviations

i.i.d	independent and identically distributed
ELU	Exponential Linear Unit
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
FCN	Fully Convolutional Network
MLP	Multi Layer Perceptron
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
ILSCVRC	ImageNet Large Scale Visual Recognition Competition
AASHTO	American Association of State Highway and Transportation Officials
LCMS	Laser Crack Measurement System
MSE	Mean Squared Error
AUC	Area under the Curve
ROC	Receiver Operating Characteristic
AUC-ROC	Area under the Receiver Operating Characteristic Curve
AUC-PR	Area under the Precision-Recall Curve
AP	Average Precision
GAP	Global Average Pooling
GAN	Generative Adverserial Networks
CPU	Central Processing Unit
GPU	Graphics Processing Unit

Chapter 1

Introduction

Modern roads provide higher levels of mobility than ever before, and the number of vehicles on those roads has been increasing since the beginning of the 20th century. These roads are usually made from asphalt concrete, which is a composite material that consists of a binder and aggregate material. Under the influence of factors such as weather and traffic, these materials degrade over time. This is why most agencies responsible for the pavement, whether governmental or private, have some form of maintenance division.

Historically, maintenance programs relied on time-based schedules, manual on-site inspections or reports of damage by road users. Currently, most maintenance programs are based on (semi-)automated inspections that involve scanning and processing road data. These automated inspections are safer and more cost- and time-effective. With the vast amount of roads that need to be maintained, an automated approach reduces overhead significantly.

The development of automated inspections has led to developments in processing algorithms for the captured data. Most algorithms focus on the quality of the pavement, which is often directly linked to distresses such as ravelling, rutting and cracking within the pavement.

One specific type of pavement distress is cracking. Current processes for the detection of cracks have good detection rates, but state-of-the-art methods still detect other structures within the pavement apart from cracks, such as expansion joints or jointed surfaces. These structures have to be taken into account separately to accurately plan maintenance.

The Dutch highway agency (Rijkswaterstaat), therefore, manually annotates crack detections to separate true cracks from false positives. Since these other detected structures do not indicate pavement damage, they should be filtered out. Once this has been done, the number and type of distresses can be used to schedule reliable and effective maintenance.

Recently, developments in the field of deep learning, specifically convolutional neural networks (CNNs), show high accuracy on various problems, including image recognition. CNNs have surpassed human performance on classification benchmarks [28]. This indicates that these types of models are very well adapted to

learn specific patterns, which humans may also internally use to recognize certain shapes.

Since the task of separating true cracks from other structures can be formulated as a problem of recognizing specific patterns, CNNs are a promising candidate for the correct classification of pavement cracks. Therefore, in this work, we have investigated these models to gain insight on their applicability for the improvement of crack detection. We have developed a pipeline for the assisted classification of cracks and tested its applicability on the state-maintained road network in the Netherlands.

This work is organized as follows. In Chapter 2, we expand the problem formulation and provide context within the domain of pavement maintenance. In Chapter 3, we review related work in the field of pavement distress detection and classification. In Chapter 4, we introduce the field of machine learning and neural networks, elaborate on the use of deep learning-based techniques in the field of computer vision and discuss the state of the art in image classification. Thereafter, in Chapter 5, we present our proposed methods for the classification of pavement cracks, based on these techniques, and discuss the data collection and processing methods. In Chapter 6, we describe, evaluate and discuss several experimental results arising from the proposed methods. Finally, in Chapter 7, we continue this discussion and conclude this work. Additionally, we provide suggestions for further research in this field.

Chapter 2

Problem Statement

With ever-increasing traffic, roads must be in good condition to continue providing a high-throughput capability and prevent damage to vehicles. In this chapter, we will introduce some basic principles behind the construction of pavement and discuss the different types of distresses that may appear in pavement. We elaborate on one type of distress – cracking – and formulate a research question related to this type of distress.

In this work, we often illustrate problems with the use case of Dutch state-maintained roads. The provided explanations are therefore also focused on these types of roads, but can be adapted for other countries as well.

2.1 Pavement construction

In this work, the word 'pavement' refers to the surface material laid down to support motorized vehicles. The pavement or road surface of Dutch state-maintained roads consists of asphalt concrete. Asphalt concrete is a combination of asphalt or bitumen and mineral aggregate. The asphalt binds the mineral aggregate, which may consist of different sizes of stone, sand, stone chippings and clay. In other countries, cement may be used as a binder, instead of asphalt. In this case, the surface may be referred to as 'concrete cement'. In either case, the aggregate material forms a skeleton that provides the carrying capacity of the pavement. This skeleton is not completely closed but has pores that can be filled with the asphalt (or concrete) and filler, which may consist of basal, limestone or fly ash. By heating the material, the asphalt becomes liquid and is able to flow in between the aggregate. The heated material is rolled onto the surface and, after cooling, the asphalt concrete is formed. Asphalt concrete for roads is usually laid in multiple layers. The top layer is constructed to optimize driving experience. The bottom layer transfers the forces on the road to the ground or sand laid beneath the road. These layers can consist of slightly different compositions of asphalt concrete to enhance these properties.

The composition of the materials and whether the pores are completely filled with material determines

the type of asphalt. In the Netherlands, over 80% of asphalt concrete is *porous asphalt*. In porous asphalt, the skeleton is not completely filled with asphalt and material, which causes a reduction in noise and increased water drainage[1]. A schematic illustration of porous asphalt is shown in Figure 2.1.

Figure 2.1: Illustration of the intersection of porous asphalt (ZOAB) in the Netherlands. Adapted from [50]

A disadvantage to porous asphalt is that it is more sensitive to some types of distress, which is why porous asphalt has a shorter lifespan than dense asphalt. Historically, the holes in the skeleton were 'overfilled', creating dense asphalt concrete. Dense asphalt concrete has a longer lifespan, but has less noise reduction and water drainage capabilities, which is why it is gradually being replaced by porous asphalt and now makes up less than 10% of Dutch state roads.

2.2 Pavement distress

Pavement distress is an important factor determining the lifespan of roads. The lifespan of asphalt concrete is influenced by environmental conditions, traffic, the surface beneath the road and the quality of how the asphalt was laid.

2.2.1 Ravelling

Under the influence of several of these factors, one of the dominant failure mechanisms of porous asphalt is ravelling. Vehicle vibrations, combined with the hardening of the asphalt (binder) due to age, under the influence of UV light, causes particles to break loose from their surrounding material. Ravelling is characterized as the significant detachment of aggregate particles and the asphalt binder from the pavement surface. Ravelling affects pavement longevity, noise reduction, and user safety and comfort. An image of severe ravelling on a Dutch state road is shown in Figure 2.2. The detection of ravelling has been widely studied and we will discuss the common methods in Chapter 3.

[1] ZOAB, Rijkswaterstaat, `https://www.rijkswaterstaat.nl/wegen/wegbeheer/aanleg-wegen/zoab.aspx` (Dutch)

Figure 2.2: Ravelling of porous asphalt in the Netherlands. Adapted from [1].

2.2.2 Cracking

The other main class of pavement distress is *cracking*. Cracking or fracturing of the asphalt may be caused by similar reasons to ravelling. Cracking can be caused by repeated heavy-load vehicles fatiguing the asphalt and varying weather conditions combined with poor draining. It is commonly the case that the base layer can no longer support the surface layer properly, causing the surface layer to crack, often in an interconnected manner. While in ravelling small particles break loose from the surface area, cracking is characterized by interconnected deeper areas with no material in between.

The American Association of State Highway and Transportation Officials (AASHTO) has released a standard for classification and quantification of cracks: 'PP67 - A Standard Practise for Quantifying Cracks in Asphalt Pavement Surfaces Collected Pavement Images Utilizing Automated Methods'. This standard specifies three types of cracks: a longitudinal crack, with an orientation between -10 and +10 degrees of the lane centreline; a transverse crack, with an orientation between 80 and 100 degrees to the lane centreline; and a pattern crack, a crack that might be part of a network of cracks. Pattern cracks may be defined as all cracks that are not defined as transverse or longitudinal. Examples of different types of cracks (in America) are shown in Figure 2.3.

Different crack types may have slightly different causes. Longitudinal cracks may appear close to joint structures of different lanes, whereas transverse cracks may be more influenced by shrinkage of the asphalt due to cold weather. Cracking may appear near seams, near induction loops under the asphalt to count traffic and near the edge of the road. Again, the actual breaking of the asphalt may be influenced by vibrations of vehicles driving over the pavement.

2.2.3 Rutting

Another less significant class of pavement distress is referred to as *rutting*. Rutting appears in the wheel path of vehicles and is a deformation of the asphalt in the driving direction. This can be caused by insufficient

(a) Longitudinal crack

(b) Transverse crack

(c) Pattern crack

Figure 2.3: Different crack types according to AASHTO. Taken from [59]

pavement thickness, lack of compaction or weak asphalt mixtures. Rutting may reduce water drainage in denser asphalt, increasing the risk of aquaplaning. However, the latter is less a problem in the Netherlands, where porous asphalt is more prevalent. Nonetheless, rutting is a deformation of pavement that has to be addressed; it may also be an indicator that other distresses, such as ravelling and cracking, may also be present. The detection of wheel paths and rutting is often incorporated into research on ravelling or cracking, but is less often the subject of research on its own. This is because the location of the wheel paths is where the asphalt is used most. Therefore, we will discuss research related to ravelling and cracking in Chapter 3, and leave rutting as only a small part of that discussion.

2.3 Problem description

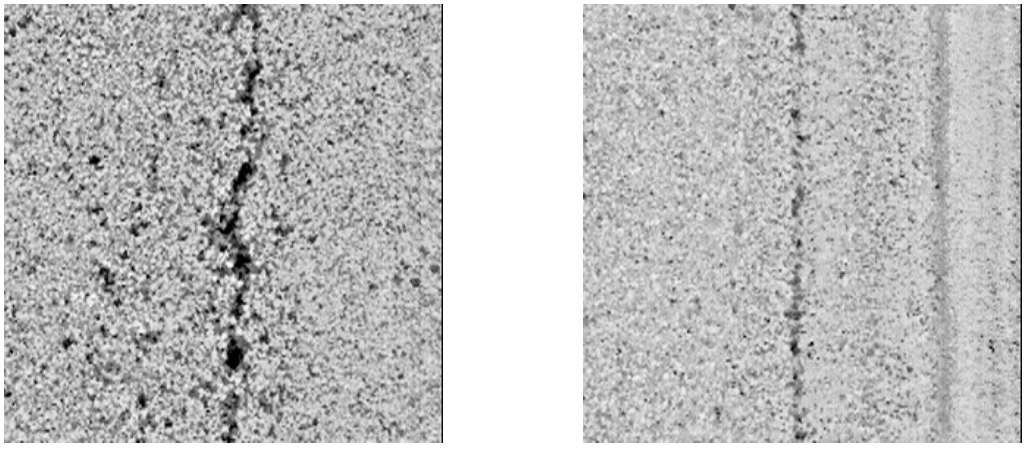
Pavement distress is the main indicator for the remaining lifespan of the pavement and therefore also the time to maintenance. As we will discuss in Chapter 3, the automated detection and classification of distresses has been the subject of considerable research. In recent years, automated road scanning systems have been developed to capture road data. This has led to large datasets, increasing the need for an automated processing system.

For example, in the Netherlands, around 5200 km of state highway is maintained by the Dutch Highway Agency (Rijkswaterstaat), which is a part of the Dutch Ministry of Infrastructure and Environment. In collaboration with The Netherlands Organisation for Applied Scientific Research (TNO), an automatic scanning and processing method for pavement distress has been developed under the *Detectie Oppervlakte Schade / Detection Surface Damage*[2] (DOS) project. Within the DOS project, the entire state maintained

[2]Monitoren en meten aan wegen, https://www.tno.nl/nl/aandachtsgebieden/bouw-infra-maritiem/roadmaps/buildings-

road network is analysed and maintenance planning is partly automated. The automated scanning system captures depth data of the pavement and produces range images (in which pixel values correspond to depth) that are used to automate inspections.

Automated ravelling detection has already been implemented in this project, and the results show a strong correlation with planned time to maintenance as evaluated by experts. Since many observations from the literature show that methods with good results exist for the detection of ravelling, we chose not to work on extending research in that direction. However, we believe that crack detection can be improved. Most methods, used to detect cracks focus on obtaining a high detection rate of cracks. While this is important, additionally, the number of false positive detections should be taken into account, which is rarely mentioned. False positives detections include, for example, jointed surfaces, rim marks, pavement joints and road shoulder drop-offs. In Figure 2.4, an example of a true crack and a false positive result is presented. Many different variations, however, exist. In Appendix A, more examples are given. The capturing method of these images is described in more detail in Chapter 5.



(a) True crack **(b)** False positive detection

Figure 2.4: A range image of an actual crack (left) and a false positive (right). A pixel's intensity represents the relative measuring distance of the capturing sensor to the surface (darker is farther)

These false positives are not signs of degrading asphalt and therefore do not need to be taken into account in maintenance planning. In fact, the time to maintenance is negatively influenced by these false positives. Since time to maintenance is determined by the amount of distresses in a specific road, false positives might cause maintenance to be scheduled earlier than needed.

It is therefore important to distinguish these artefacts from actual cracks. Currently, in the DOS project, there is a manual step where inspectors look at the detected cracks in the scans and categorize them as either an actual crack or something else. We believe that this categorization step can be supported by an automated pre-labelling process, allowing a more efficient allocation of the inspectors' working hours.

infrastructure/infrastructuur/monitoring-en-meten-aan-wegen/ (in Dutch), visited: September 10, 2018

2.4 Research questions

The amount of work in the manual step of the crack detection process could be reduced by a more precise classification of cracks that filters out more false-positive results while maintaining the detection rate for true cracks. Therefore, in this work, we aim to develop a system to achieve this objective.

Many advancements in the field of image recognition have been achieved in recent years, mainly due to developments in the field of CNNs. Since the captured 3D scans are converted to range images, the question arises of whether general image recognition methods can be applied to this problem in order to reduce the number of false-positive crack detections in pavement.

To answer this question, the following objectives are studied in this work:

(1) What systems are currently used for crack detection and classification in pavement, and what other approaches are described in the literature?

(2) What are the recent developments in general image recognition, specifically related to CNNs, and can these methods can be generalized and applied to crack classification in pavement?

(3) What are the implications of implementing such a system for the use case of the Dutch state-maintained road network?

In the following chapter, the first research question is addressed through a review of pavement distress literature.

Chapter 3

Related and Previous work

This chapter provides an overview of recent developments in the field of pavement distress detection and classification. Automated detection and classification of pavement distress have been a field of interest within both academic and industrial research. Since automated inspections are less subjective, safer and cost less, methods that promise to obtain results that are up to par with manual inspections have gained traction.

Several types of distress affect the lifespan of pavement and therefore the maintenance planning. In this chapter, we review the literature on the automatic inspection of the two aforementioned types of pavement distress: ravelling and cracking.

3.1 Ravelling

The first study describing the automated measurement of ravelling is the so-called 'Stoneway' algorithm, developed by van Ooijen et al. [82]. This algorithm uses the texture profile data from 25-line laser points to identify holes. It works by determining the height and length of the imprint of a lost stone in a transverse road sample (orthogonal to the driving direction) and thresholding them to determine the amount of ravelling. A sample is shown in Figure 3.1. The results are summed up over multiple samples, and surface disintegration by ravelling is determined per square metre. One disadvantage of this approach is that two parameters (the height and length of a gap) have to be specifically set for every pavement type.

McRobbie et al. [54] described work that was undertaken to create a surface-type independent measure to identify ravelling. The authors used a similar capturing method to van Ooijen et al, where the pavement texture profile is inspected using multiple (20) point cameras. Deviations between the local and global texture characteristics are used to identify ravelling. Measurements are taken relative to the mean profile depth (MPD). The MPD, indicated in Figure 3.1, is computed as the difference between the highest point on the profile and the average line. Measurements for which deviations are deep enough and sustained over a sufficient length indicate a loss of aggregate. Two indicators that were found to infer the presence of ravelling are entropy and edge density. High-entropy images have high structure variations, indicative of ravelling.

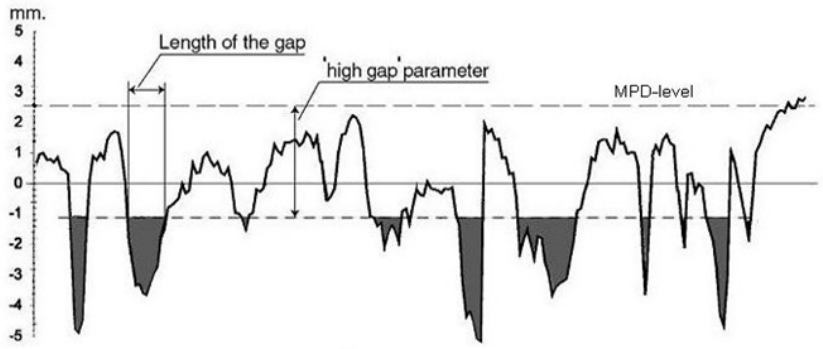


Figure 3.1: Stoneway algorithm by Ooijen et al [82].

Edge detectors tend to detect the outline of the aggregate within the binder (asphalt).

Wright et al. [86] improved upon this work. They argue that the number of point lasers and limited survey width of the camera array are insufficient to fully describe the transversal profile of the pavement. In addition, road markings (which can be up to 6 mm high) affect ravelling measurements. Their first contribution was to install a scanning laser capable of collecting over 1000 transverse points with a larger total survey width. The effect of road markings is reduced with a 'sample and hold' method. Since the markings cause a high-intensity amplitude response, the point next to the spike point is used to replace the value. This method generally improves accuracy.

Laurent et al. [45] use even denser laser data provided by their laser crack measurement system (LCMS). As this is the same data collection system that is used in this work, we will discuss its function further in Chapter 5. The LCMS system covers an entire lane transversely and is able to capture depth data. This increases the performance of ravelling detection. Since their system acquires sufficiently dense 3D data, the detection of missing aggregates can be accurately determined using a so-called digital sand patch model. This model determines the volume of the voids that would be occupied if sand was poured over the surface area. The authors propose a ravelling indicator measure, which is defined as the volume of aggregate loss per unit of surface area and is more accurate than the measure based on MPD.

Using data from the LCMS, Mathaven et al. [52] developed an algorithm to quantify ravelling on the combination of range (3D) and intensity (2D) data. A texture descriptor method called Laws' texture energy measure is used to detect texture boundaries. This measures the amount of variation within a window of finite size. In addition, a Gabor filter and other morphological operations are applied. Signal processing techniques are then used to detect and quantify ravelling.

Van Aalst et al. [1] also used data from the LCMS to determine network-wide ravelling in the Netherlands on 14,000 km of highway infrastructure. Their main contribution in this work is the preprocessing of measurements, including by removal of road markings, 'flattening' to compensate for vehicle movement and road unevenness, removal of wheel paths (lateral location of maximum damage) and removal of non-ravelling damage (rim damage). The authors also used a quadratic classifier to determine the asphalt type. Thereafter,

a 3D generalization of the Stoneway algorithm [82] is used to determine the amount of ravelling. With the automated asphalt type detection, the height and length parameters of the algorithm can be automatically set. From there on, the remaining lifespan of the asphalt and time to intervention/maintenance can be determined for the Dutch Highway Agency (Rijkswaterstaat). The authors' results were validated with manual inspections on the road.

3.2 Cracking

Another significant source of pavement distress is cracking. There are two main research directions, which are closely related: crack detection and classification. While crack detection has a straightforward interpretation, crack classification is open to multiple interpretations. We will first discuss crack detection and follow up with a review of research on the classification of detected cracks, which includes the various interpretations.

3.2.1 Detection of cracks

Extensive research has been conducted on the detection of cracks on road surfaces. Since the early 1990s, multiple approaches have been developed for the detection of cracks, starting with captured 2D intensity images and later moving toward the use of 3D depth measurements.

Since crack pixels, in many cases, can be distinguished from their surrounding area by a lower-intensity value (darker pixels) and cracks are usually continuous within a local neighbourhood, thresholding and edge detection are common approaches described in the literature.

Kirschke and Velinsky [40] were one of the first to describe an approach to crack detection. They proposed a histogram-based approach using 2D intensity images. The image of the pavement is divided into a grid, and a histogram is built for each tile. The statistical moment of each histogram is then computed and compared to the local values of statistical moments to identify cracks. The authors do not report quantitative results; however, this work is relevant as it is one of the first methods described for detecting cracks. This work does mention difficulties in 'noisy' environments, such as an image with oil spots, which may be identified as cracks.

Koutsopoulis et al. [42] conducted a comparative study of different thresholding techniques in images. The authors compared methods such as Otsu's method [63], Kittler's method [41] and a segmentation by regression-based thresholding method. The regression method was shown to outperform the others, but again no quantitative measures were mentioned. However, the reported downside to this method is that its performance quickly degrades when lighting conditions are poor.

To address this issue, Oh et al. [60] proposed an iterated clipping method to deal with 'noisy' images. Their method iteratively uses the mean and variance to eliminate noise while preserving crack pixels.

Cheng et al. [12] described a system based on fuzzy logic and morphological operations. Similar to Kirschke and Velinsky [40], they divide the images into a grid and build a histogram of pixel intensities for

each tile. However, the authors argue that the parameters that define the shape of a peak in the histogram (which is indicative of a crack in the image) may have different thresholds depending on the brightness and contrast. They reformulate the problem as finding a combination of parameters to ensure the corresponding event has maximum entropy. This method shows improved results compared with earlier intensity-based methods. However, performance is shown by giving example images, but no metrics are defined to measure the quality of the produced segmentations.

The core logic of these thresholding methods is to compare pixel intensity to its local neighbourhood. Since the calculation is based on comparison with a small direct neighbourhood, these methods are not particularly computationally expensive. However, noise in intensity-based images limit these approaches, and low contrast of the cracks with the background may introduce many false-positive detections. In particular, different lighting conditions, rim marks and oil stains adversely affect detections.

One other class of methods that is frequently used is based on edge detection in the image. Ayenu-Prah and Attoh-Okine [4] compared a Canny edge detector [10] with a Sobel edge detector [70] and reported results on two small sets. One set contained 9 images of asphalt concrete and the other set 6 images of cement concrete. The authors showed that Sobel filters are affected by noise on images with high irregularity; the Canny edge detector performed better on these types of images, achieving what they refer to as 67% 'good detections' (6 images) on asphalt concrete and 33% (2 good images) on cement concrete.

Cuhadar et al. [16] used a wavelet transform to de-noise the images. They argue that small noise can be removed by converting the data into a smoother waveform, which is more suitable for segmentation. Their three-stage process involves computing the wavelet coefficients on the pavement data, shrinking the coefficients that are smaller than a given threshold and reconstructing the signal with the inverse wavelet transform. The singularities of the smoothed data are then marked and used to segment the data into regions that have similar characteristics.

Wavelet transformations have also been used by Nejad and Zakera [56]; these authors also used a Radon transform. According to the authors, the Radon transform is based on the parametrization of straight lines and the evaluation of integrals along these lines. They use these aspects to capture the directional features of an image to improve segmentation.

Zalama et al. [88] use a Gabor filter to detect cracks in specific directions on 2D images. While crack detection is the first step in their work, classification of crack orientation is an additional focus. Interestingly, this works present quantitative results on a a set of around 3000 images, of which 10% has some form of cracking. They report balanced accuracy, which is the average of sensitivity and specificity. The reported balanced accuracy for transverse cracks is .91 and .88 for longitudinal cracks.

Edge detection is similar to thresholding methods, but instead of thresholding which uses the pixel values in the image domain, the image is converted into a different domain, such as the gradient or frequency domain. Thresholding is then applied in this domain.

Other approaches include the use of local probability modelling and global probabilistic dynamic opti-

mization. Tsai et al. [81] showed that this approach outperformed most other algorithms, but was more computationally expensive. They show results based on their own metric, which is based on the difference between a manually labelled image and the predicted crack segmentation.

Furthermore, minimal path approaches have been proposed by Amhaz et al. [2] and Avila et al. [3] based on Dijkstra's algorithm [21]. The minimal path approaches combine intensity and geometry aspects for segmentation. Despite their advantages in accuracy, these algorithms are affected by prior user input beginning and endpoints to extract open curves. In addition, these studies enumerated over every pixel across the image to calculate the minimal path, which is quite computationally expensive. Kaul et al. [37] improved the minimal path algorithm, removing the endpoint to be set. Their algorithm works on closed curves, such as circles, allowing more complex topologies to be detected.

A deep learning approach has been introduced by Zhang et al. [94]. Crack detection is conducted as a patch-wise classification using images taken from smartphones. Image patches are classified as being either a crack or clear pavement. The authors provide a comparison with a support vector machine (SVM) and boosting methods, and show that their method outperforms these methods. The SVM achieved an F_1-score of .736 and their method .897. Unfortunately, the authors only used a small manually annotated data set (500 images); as previously mentioned, 2D images are affected by inherent noise and interference from other pavement surface objects such as oil stains, which might not be overcome in this type of data.

Recent approaches in crack detection utilize similar techniques as earlier research, but work with 3D data [84, 91]. The introduction of 3D laser imaging has recently emerged as a data collection approach. 3D data are less affected by lighting conditions and noise.

In a recent study, Zhang et al. [90] demonstrated the use of a deep learning algorithm on 3D pavement data. They combined a custom feature extractor with a convolutional neural network, called CrackNet, and showed promising results on a data set of 5000 3D pavement images as it out-performed pixel-wise SVMs and other traditional approaches. They also report classification results on images, with the F_1-score used as a metric. The obtained F_1-score for their approach is reported to be 88.86 vs 63.36 for a SVM based approach. This number is, however, hard to compare to other works, such as the work by Zhang et al. [94], due to completely different data (2D smartphone images vs 3D pavement scans).

Apart from academic research, the industry has developed systems for the collection of pavement data. One of the most commonly used systems [34] for gathering 3D pavement data is the aforementioned LCMS by Pavemetrics. Their system not only gathers the data but also provides software for the detection of cracks. Unfortunately, Pavemetrics do not provide insight into their algorithms; however, Jiang [34] provided a quantitative analysis on the performance of their algorithms at that time (2015), which showed that the Pavemetrics software could compete (and sometimes outperform) methods from academic research. Since this system is under constant development, we assume that newer versions can compete with state-of-the-art research, such as the aformentioned work by Zhang et al. [90].

3.2.2 Classification of cracks

Whereas detection of cracks is typically the first step in automated pavement distress systems, many studies focus on classification as a subsequent step in the same work. There are multiple approaches to the classification of cracks. The first group of studies focuses mainly on the orientation of cracks: whether they are longitudinal, transverse or diagonal to the driving direction. The most common approach is to classify cracks based on their orientation, i.e., whether they are longitudinal, transverse or diagonal to the driving direction. For example, Cheng et al. [12], Bray et al. [8] and Saar and Talvik [66] used a projection of the images in a direction (rows, columns or diagonal) and determined the crack class by analyzing the corresponding projections onto a vector. Lee and Lee [46] used the same idea, but divided the images into a grid and counted the number of crack cells to reduce the influence of noisy pixels. The disadvantage of this method is the choice of tile size. In addition, this work includes two other classes: alligator and block cracks, which under the AASHTO would be classified as pattern cracks.

Other studies use their own metrics for the classification of cracks. In addition to the type of crack, Zhou et al. [95] provided a severity level ranging from 1 to 100. Oliveira and Correia [61] labelled and classified images according to the Portuguese Distress Catalog.

Overall, different approaches show similar results on the classification into longitudinal and transverse classes [34]. Some studies also add subclasses of pattern cracks, which makes comparison difficult. The lack of a publicly available image database of pavement distress makes the comparison between different approaches cumbersome. Moreover, these classes may be too simple and not useful for the prediction of needed maintenance [34].

3.3 Our contribution to literature

In this work, we branch out to a different type of classification, which is also related to the correct detection of cracks. As mentioned in our research objectives, the detection of cracks often includes false positives. These false positives are rarely mentioned by other research, but actually make up most of the detected cracks in the data of the Dutch highways.

The false detection have to be reduced for a correct detection algorithm. Using 3D data captured by the LCMS system by Pavemetrics and manual annotation of detected cracks for the Dutch road network, we extend on previous research by classifying incorrect detections as such. This classification is limited to crack detections from the discussed Pavemetrics software only, so some cracks (false negatives) within the pavement will not appear in our data.

We develop the classification system by using state-of-the-art convolutional neural networks. In the following chapter, we introduce terminology and techniques that will be useful for comprehension of the applied methods.

Chapter 4

Background

In this chapter, we provide an introduction to artificial neural networks, how they are positioned in the field of machine learning and recent applications in the field of computer vision, specifically image classification. As developments in the field of computer vision are rapidly evolving, this review is inherently outdated. However, this context can still serve to enable a better understanding of the research area. The main aim of this chapter is to familiarize the reader with some principles and terminology related to the field of machine learning to form a basis to understand the mentioned terms and concepts in the following chapters.

4.1 Supervised learning

Machine learning is the field of computer science where computers are learning to do tasks without being explicitly programmed to do so. A class of machine learning is *supervised learning*. Often in supervised learning, a machine is given an input $\boldsymbol{X}$ and a vector of labels $\boldsymbol{y}$, where $\boldsymbol{X}$ is a design matrix in which each row contains a different example $\boldsymbol{x^{(i)}}$ of the same length. If samples have different lengths, the input can also be described as a set instead of a matrix. The objective in supervised learning is to learn the optimal function mapping $f : \boldsymbol{X} \rightarrow \boldsymbol{y}$. In most cases, the exact underlying relationship between $\boldsymbol{X}$ and $\boldsymbol{y}$ is not known, but we are given a set of training examples $\boldsymbol{X}^{\text{TRAIN}}$ with a set of associated target labels $\boldsymbol{y}^{\text{TRAIN}}$. We then denote $\boldsymbol{x^{(i)}}$ as the i-th training example and $y^{(i)}$ as the target associated with $\boldsymbol{x^{(i)}}$. Such an example is typically represented as a vector $\boldsymbol{x} \in \mathbb{R}^n$, in which each entry $x^{(i)}$ of the vector is a feature. From our training examples, we try to reconstruct an inferred function mapping f, which generalizes well to new examples, e.g. from a different set $\boldsymbol{X}^{\text{TEST}}$. When scalars y_i from $\boldsymbol{y}$ take on discrete values, we refer to the objective as *classification*. In the case of continuous output values, we call it *regression*.

In this section, we start by introducing simple supervised learning algorithms, starting with linear and logistic regression, which allow simple linear function mappings. We then delve deeper into a group of more complex supervised learning methods: neural networks, which are able to perform non-linear mappings.

4.1.1 Linear regression

Linear regression can be seen as a simple machine learning algorithm to solve regression problems, in which the form of the model appears as follows:

$$y = \boldsymbol{\theta}^{\boldsymbol{T}} \boldsymbol{x} \equiv \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n, \tag{4.1}$$

where $\boldsymbol{\theta} \in \mathbb{R}^n$ is a vector of parameters of the model.

Parameters control how the system behaves. In this case, $\boldsymbol{\theta}$ is the set of weights or parameters (we will use both terms interchangeably), which weigh our inputs $\boldsymbol{x}$. The input vector $\boldsymbol{x}$, additionally, has a bias term x_0, set to 1. The bias shifts the decision boundary from the origin and is independent of the input $\boldsymbol{x}$.

This equation can be expressed as a directed graphical model as seen in Figure 4.1.

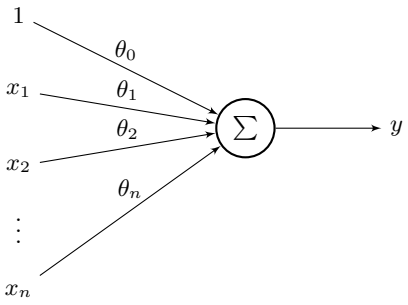


Figure 4.1: A graphical representation of a linear model. Input $\boldsymbol{x}$ is weighted by $\boldsymbol{\theta}$ and results are summed to form y. Input x_0 is again set to a constant 1.

The goal of linear regression is to find values for $\boldsymbol{\theta}$ to minimize some cost function $J(\boldsymbol{x}; \boldsymbol{\theta})$. This cost function penalizes the difference between the model's output and the target variable. A commonly used cost function for linear regression is the mean squared error (MSE). Cost functions are discussed in Section 4.1.7.

4.1.2 Logistic regression

In the case that our dependent variable y takes on discrete values instead, such as 0 or 1, we could use linear regression and set a threshold to separate the binary classes as giving an output on either side of the threshold. However, since linear regression assumes that y is continuous, our inferences may be invalid in the discrete case.

A better approach is to apply a nonlinear function to transform our linear predictor into a value between 0 and 1, which can be interpreted as a probability. In a model, referred to as logistic regression, this function is the so-called sigmoid function $\sigma(z) = 1/(1 + e^{-z})$, where $z = \boldsymbol{\theta}^{\boldsymbol{T}} \boldsymbol{x}$. The sigmoid function is discussed in more detail in Section 4.1.7. We use the resulting probability to identify which binary class should be

predicted. Note that logistic regression is still a (generalized) linear model, as the predicted class label depends only on the weighted sum of its inputs and not on their product or quotient.

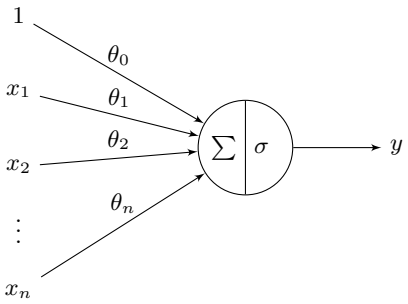


Figure 4.2: Graphical model of logistic regression.

In Figure 4.2, inputs $\boldsymbol{x}$ are being weighted by weights $\boldsymbol{\theta}$. Results are summed and a nonlinear function σ is applied. The result is computed as $y = \sigma(\sum_{i=0}^{n} x_i\theta_i)$, where σ is the logistic sigmoid function and $x_0 = 1$.

4.1.3 Single layer perceptron

In 1958, Frank Rosenblatt, developed what was essentially the first neural network [65], based on the McCulloch-Pitts neuron [53] and the findings of Hebb [30]. Similar to logistic regression, the perceptron performs a binary classification, where the function maps $\boldsymbol{x}$ to a single binary value using a threshold of 0 as seen in Equation 4.2.

$$y = \begin{cases} 1 & \text{if } \boldsymbol{\theta x} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

This threshold mapping, in the context of neural networks, is called a step activation function. If this step function is replaced by the logistic sigmoid, we obtain our logistic regression model seen before.

4.1.4 Multi layer perceptrons

In 1969, Minsky and Papert stated in their book Perceptrons [55] that single layer perceptrons are not able to learn simple functions, such as the exclusive or (XOR) function. The XOR function uses two binary inputs and produces a binary output, for which no single linear function exists, that is capable of separating all outputs.

This insight halted the field of neural networks for a while until the idea of the *multilayer perceptron* (MLP) emerged. If we take several single layer perceptrons and connect the outputs of each neuron to the inputs in the subsequent layer of neurons, using parametrized connections, we create a MLP, or *feedforward neural network*. Given a non-linear activation function, such as the sigmoid, this model is able to perform non-linear input-output mappings, which is required to solve the XOR problem. A graphical representation of a simple MLP is provided in Figure 4.3.

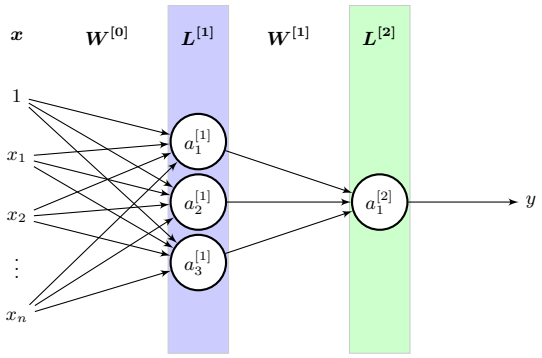


Figure 4.3: Graphical model of a simple feed forward neural network as an acyclic graph.

We can also describe a MLP as a series of matrix-vector multiplications, containing non-linear functions in between. The vector of inputs $\boldsymbol{x}$ of length n is being weighted by a matrix $\boldsymbol{W}^{[0]}$ of size 3×5, resulting in a vector of length 3. In layer $\boldsymbol{L}^{[1]}$ (in blue), each element of this vector is then passed element-wise through an activation function, such as the sigmoid activation function. The resulting vector (still of length 3) is weighted by $\boldsymbol{W}^{[1]}$ of size 1×3, which results in a single number. Following that operation, $\boldsymbol{L}^{[2]}$ (in green) applies another non-linearity, forming y. MLPs can perform both regression and classification tasks, depending on the output layer.

The idea behind neural networks, similarly to other parametrized models, is that a cost function should be optimized such that the function approximation f matches the function f^*. This is done by providing samples $\boldsymbol{x}$ to the network with a label $\boldsymbol{y} \approx f^*(\boldsymbol{x})$ to the network and updating the network parameters to better fit the mapping, according to some cost function.

In MLPs, we have a model that performs a mapping f between $\boldsymbol{x}$ and $\boldsymbol{y}$ by applying a composition of functions. Every *layer* (as illustrated in Figure 4.3) computes a function where the input is determined by the output of the previous layer. The number of layers is referred to as the *depth* of the network. Often the depth is referred to as only the number of hidden layers, where a hidden layer is a layer which is not an input or output layer. The number of nodes in a layer is referred to as the *width* of the layer. Since different layers can have different numbers of nodes, the width of a network is sometimes referred to as the width of the layer with the most nodes, however, the terminology regarding this is not consistent.

MLPs are also referred to as feedforward neural networks because information flows forward from input $\boldsymbol{x}$ through the layers to produce $\boldsymbol{y}$. There are no feedback connections in these networks. Networks in which these feedback connections exist are called *recurrent neural networks* (RNNs); however those will not be discussed in this work. The term 'neural' comes from the idea that these networks resemble and might be loosely inspired by biological neurons.

Recent findings show that deeper networks can perform complex function mappings, which has been shown to be useful in different domains. This gave rise to the term 'deep-learning', which refers to neural networks with many layers. The use of wider instead of deeper networks has also been subject of research.

The universal approximation theorem [17] states that two-layer networks with any non-polynomial activation function, can approximate any continuous function $f : [0, 1]^d \to \mathbb{R}$. However, it has been shown, that to compute the same function as a deeper network, these networks must be exponentially wider [78], which is often not computationally feasible.

4.1.5 Cost functions

The cost function measures the quality of a set of parameters, based on how well the predicted values agree with the ground truth labels. An important aspect of the design of neural networks is this cost function. When designing a cost function for a neural network, one has to keep several requirements in mind. The first requirement is that the cost function can be written as the average of cost functions for individual training examples [58]. We will see why this aspect is useful, when discussing gradient-based optimization methods, later on in this chapter. The second requirement is that is can be written as a function of the outputs of the network. This means we can alter the parameters of the network to optimize the cost function.

Neural networks, just like linear models, are parametric models. For these models, the principle of maximum likelihood can be used to determine a cost function. Maximum likelihood estimation minimizes the difference between the empirical distribution of the data $\hat{p}_{\text{data}}$ and the model distribution p_{model}, measured by the Kullback-Leiber (KL) divergence [44]. Minimizing the KL divergence corresponds to minimizing the cross-entropy between the two distributions. Deriving a cost function from maximum likelihood estimation helps to find the optimal cost function for each model. There is a relation between the choosing of the cost function and the choice of output units. An in-depth analysis of this relationship, and an extensive derivation of optimal cost functions for certain output units is given in [26].

We will quickly discuss the commonly used cost functions for both regression and classification below.

Cost function for regression

As stated earlier, one common cost function for regression problems is the MSE.

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 \tag{4.3}$$

Here we clearly see that the cost function is written as an average over the individual training examples, which was the first requirement of a cost function. Also, it is a function of the network outputs $\mathbf{y}$, fulfilling the second requirement.

In the case of linear regression, maximizing the log-likelihood (or minimizing the negative log-likelihood), with respect to the parameters of the model, yields the same estimate as minimizing the MSE [26].

In fact, this equivalence between maximum likelihood estimation and minimization of the MSE holds for any parametrized linear model.

Cost function(s) for classification

For regression problems, the MSE is a reasonable cost function. However, in the case of classification, the MSE is usually not a good choice. The problem is that some output units can saturate when using gradient-based optimization. The result is that the gradients will be very small when combined with the MSE. This is explained in more detail in Section 4.1.8.

We have seen logistic regression as a classifier before, outputting a probability between 0 and 1. For logistic regression, assuming that our examples are i.i.d., we can formulate the maximum likelihood as follows:

$$\begin{aligned} \boldsymbol{\theta}_{\mathrm{ML}} &= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log\, p(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \\ &= \sum_{i=1}^{m} \log\, [\sigma(\boldsymbol{x};\boldsymbol{\theta})^{y^{(i)}} + (1-\sigma(\boldsymbol{x};\boldsymbol{\theta}))^{1-y^{(i)}}] \end{aligned} \tag{4.4}$$

where $\sigma(\boldsymbol{x};\boldsymbol{\theta})$ is the logistic sigmoid (defined later in Equation 4.10) over the inputs $\boldsymbol{x}$, weighed by parameters $\boldsymbol{\theta}$.

The cost (or loss) function we derive from Equation 4.4 is as follows:

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{m}(y^{(i)} \log\, \sigma(\boldsymbol{x};\boldsymbol{\theta})) + ((1-y^{(i)}) \log\, (1-\sigma(\boldsymbol{x};\boldsymbol{\theta}))) \tag{4.5}$$

Here we minimize the so-called negative log-likelihood or cross-entropy. We can use this approach for any parametrized model that outputs a probability between 0 and 1.

In case of multi-class classification the cost function that should be optimized, can be derived from the maximum likelihood in a similar way:

$$\begin{aligned} \boldsymbol{\theta}_{\mathrm{ML}} &= \arg\max_{\boldsymbol{\theta}} \sum_{k=1}^{c}\sum_{i=1}^{m} \log\, p(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta})^{y_k^{(i)}} \\ &\arg\max_{\boldsymbol{\theta}} \sum_{k=1}^{c}\sum_{i=1}^{m} \log\, \sigma(\boldsymbol{x}^{(i)};\boldsymbol{\theta})^{y_k^{(i)}} \end{aligned} \tag{4.6}$$

where $\sigma(\boldsymbol{x};\boldsymbol{\theta})$ now is the softmax (defined in Equation 4.11) and c the number of output units, corresponding to the classes.

$$J(\boldsymbol{\theta}) = -\sum_{k=1}^{c}\sum_{i=1}^{m} y_k^{(i)} \log\, \sigma(\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \tag{4.7}$$

4.1.6 Parameter learning

As in linear regression, neural networks are parametrized models and we want to find the values for the parameters $\boldsymbol{\theta}$ that minimize the chosen cost function. Assuming we use the MSE as a cost function for linear regression the optimal value for the parameters can by found by minimizing the error over the training set. Because the MSE is convex with respect to the parameters, a closed form solution can be found by solving for where the gradient is zero.

Unfortunately, the optimization surface as a function of the parameters in a feedforward neural network with non-linearities is non-convex and has no closed form solution. This is why we use optimization algorithms to find good values for $\boldsymbol{\theta}$. Most methods used in the context of neural networks are iterative first-order gradient-based and drive the cost function to a low value.

The most commonly used algorithm to train neural networks is called *gradient descent*, which works as follows: We start with our training set and forward propagate it through the network. We assume the network parameters are initialized randomly for now. We then calculate the loss, using the associated training labels. Given some loss function $J(\theta_0, \ldots, \theta_n)$, we then want to know which parameter settings minimize the loss on our training set:

$$\min_{\theta_0,\ldots,\theta_n} J(\theta_0, \ldots, \theta_n) \tag{4.8}$$

We differentiate the loss function with respect to the parameters of the network. The derivative of the loss function is useful because it tells us how to scale a small change in the input to obtain a change in the output. In other words, it tells us how to reduce the cost function by making a small change in the opposite sign of the derivative. With gradient descent, we take small steps in the direction of steepest descent of the gradient and update the parameters accordingly:

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla J(\boldsymbol{\theta}) \tag{4.9}$$

Here $\nabla J(\boldsymbol{\theta})$ is the vector of partial derivatives (gradient) of the cost function and η is referred to as the *learning rate*. The learning rate is a hyperparameter that controls the speed of adjusting the parameters of our network with respect to the loss gradient. This is one of the more important hyperparameters to tune in order to train deeper networks [5].

The loss can then be recalculated with the updated parameters. The training set is then again forward propagated through the updated network and the steps are repeated. This is referred to as training a neural network. Convergence of the loss or a fixed number of passes through the network are commonly used indicators to stop the process.

One problem with gradient descent, when dealing with larger training sets, is that it can still become computationally expensive to calculate the cost function (as it is a sum of a per-example loss). To address this problem, random subsets of the training set can be used instead to create an estimate of the gradient. Such subsets are called mini-batches, and mini-batches of 16–512 data points are common [5]. The extreme case, where a mini-batch of a single data point is used is called *stochastic gradient descent* (SGD). Using mini-batches is a necessity with large modern training sets and is, therefore, commonly used in the literature. Moreover, there is some empirical proof that larger batches converge to sharper minimums on the training set and can hurt generalization of the model [38].

Different adaptations to gradient descent have been proposed. One method is gradient descent with momentum. Gradient descent with momentum works by computing an exponentially weighted average of

the gradient at every step and using the momentum of the gradient to update the parameters instead of the actual gradient. This speeds up learning by accelerating learning in the previous opposite gradient direction.

A variation to momentum is called Nesterov momentum [57]. With Nesterov momentum, first a step is taken in the direction of the previous momentum, and the gradient is then calculated at that point. We then return to the previous point but use the gradient from the 'lookahead' point. This gradient is combined with direction of the current momentum to calculate a new update step. This approach has a faster convergence rate than plain SGD with momentum [74]. Another approach to optimization is called RMSProp [79]. RMSProp uses an exponentially weighted average of the squares of derivatives and divides the gradient by this squared average. This has the effect of slowing down updates of parameters with large derivatives, thus dampening out the oscillations in the loss landscape. This allows for a larger learning rate, speeding up learning. Combinations of these methods have led to other optimization methods, such as Adam [39], which can be seen as the combination of momentum and RMSProp. Another method is called Nadam [22], and uses the Adam algorithm combined with Nesterov momentum. Currently, Adam is one of the most commonly used optimization algorithms in the field of neural networks, but RMSProp and Nadam are also used frequently. We will discuss some of the state-of-the-art networks that were trained with one of these algorithms at the end of this chapter.

An advantage to gradient-based methods is that they are relatively fast optimization methods and scale better to larger examples than numerically exact optimization techniques. However, the loss landscape of deep neural networks is non-convex and has been shown to have many points where the gradient is small or zero. These points can be either local optimums or saddle points. We therefore usually settle for finding a value of the cost function that is low, but not necessarily a global minimum. It is still a subject of research whether large neural networks have high-cost local minima; however, it is suspected that many local minima exist that have low costs, which differ little from the global minimum [27].

There are also some second-order optimization algorithms based on Newton's method have been applied to training deeper feedforward networks. Due to the computational complexity, these methods can become intractable for larger networks as the inverse Hessian would have to be computed. Some methods, such as L-BGFS [49] attempt to approximate this inverse Hessian; however, the use of these methods remains uncommon.

4.1.7 Activation functions

Neural networks create a non-linear function mapping. To do this, some form of non-linearity in the network is needed. In fact, deeper networks only make sense with non-linear functions in between. Without non-linearity, the feedforward function is just a linear function. Suppose $f^{(1)}(\boldsymbol{x}) = \boldsymbol{W}^T\boldsymbol{x}$ and $f^{(2)}(\boldsymbol{h}) = \boldsymbol{h}^T\boldsymbol{w}$ then $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{W}\boldsymbol{w}$, so $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{w}'$ where $\boldsymbol{w}' = \boldsymbol{W}\boldsymbol{w}$. The network would simply be equal to a single layer network.

Activation functions create the needed non-linearity. In neural networks, each node commonly applies

some non-linear function to its inputs. Historically, the sigmoid function (Equation 4.10) was a commonly used non-linearity.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4.10}$$

The sigmoid activation function maps the input z to an output between 0 and 1. A problem with the sigmoid activation function is that large numbers will become close to one and large negative numbers will become close to zero. This causes a problem because the gradient of these regions is close to zero, which is problematic for our gradient-based optimization methods discussed in the previous section. Another problem with the sigmoid activation function is that it is not zero-centred. This means that after this activation, subsequent layers will receive non-zero-centred data, which might be undesirable for optimization algorithms.

Sigmoid activation functions have become fallen out of fashion in is recent research; however, they can still be used in the output layer of the network to give a posterior probability for a class in the case of a binary classification problem. A single node with a sigmoid activation function might provide a mapping to the range $[0, 1]$, where a threshold like 0.5 can be used as the decision point between the two classes. The sigmoid function is a special case of the so-called *softmax* function. The softmax activation function as seen in Equation 4.11 is a smooth, differential approximation of the arg max function, where we take the maximum of all node activations.

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}} \text{, for } i = 1, \ldots, c \tag{4.11}$$

This is useful in for example in multi-class classification problems, with mutually exclusive classes. In that this case we predict one of c classes. Therefore, it the softmax activation is merely used in the final layer of a feedforward network.

One way to tackle the problem of non-zero-centred activation outputs was tackled by the *hyperbolic tangent* function (Equation 4.12)

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{4.12}$$

The hyperbolic tangent or *tanh* activation function squashes the input between -1 and 1, which causes the output to be zero-centred. The tanh activation function is a scaled sigmoid neuron and is preferred over the sigmoid activation function, however, it does not solve the problem of saturated gradients.

In recent years, the *rectified linear unit* (ReLU) activation function has become the dominant non-linearity in feed-forward networks. The ReLU (Equation 4.13) is a simple function to compute, which sped up computation [67] and is almost linear.

$$\text{ReLU}(z) = \max(0, z) \tag{4.13}$$

The ReLU is actually piece-wise linear, but it has been shown that, as with other non-linear activation functions, feedforward networks with a ReLU activation function are universal approximators [71]. The

ReLU has no problem with saturating gradients as the gradient is constant for the positive region of the input. One problem that can occur is that the negative input area has a gradient of zero. This might cause a weight update that will cause it to never activate on any data point again. This is referred to as the dying ReLU problem. In addition, the mean of the ReLU is no longer zero, causing the same problems as described earlier.

Small modifications to the ReLU that have been proposed to deal with the dying ReLU problem include the Leaky ReLU [51], which has a small negative slope, and the Exponential Linear Unit (ELU) [15], which has the following form for $x < 0$: $\alpha(e^x - 1)$, where α is an additional hyperparameter. Another advantage to the ELU is that it has a mean close to zero. However, the ELU is slightly more computationally expensive.

4.1.8 Weight initialization

As mentioned, it is undesirable for neural networks to have non-zero-centred data, either in the input or in subsequent layers. One problem this may cause is ending up in the area of activation functions where the gradient is saturated. Another problem encountered when using gradient-based optimization is that gradients are calculated by means of backpropagation. Centring the data removes possible bias in the gradient. Otherwise, non-zero-centred data could cause the gradients to have a bias in one direction, due to a large eigenvalue. This could slow down learning and lead to worse solutions.

For our input layers, we can easily zero centre the data by subtracting the mean of the data. Often, we also divide by the standard deviation of the data to create unit variance. This makes training less sensitive to the scale of features. When working with images, this is less of an issue, as pixels are already in the same range of values.

Having zero-centred data throughout the network is also dependent on the weights of the network. We note that it is important to initialize weights differently to break symmetry. Equal weights cause each neuron to compute the same function and the same gradients, reducing the network's capacity to that of a single neuron.

One option is to initialize the network's weights with small random values. However, multiplying the input data with small random weights causes the data to become progessivly smaller. As a consequence, the gradient becomes smaller and smaller in earlier layers because during backpropagation, the gradient is proportionally scaled by the weights. This may cause earlier layers to learn much more slowly or stop learning at all. This problem is referred to as the vanishing gradient problem [6]. Although less common in feedforward networks, large random weights could have the opposite effect, referred to as the exploding gradient problem. This may cause the model to be unstable and unable to learn, and cause numerical issues such as overflow. These effects are somewhat alleviated with activation functions such as ReLU, which only saturate in one direction. Another method to prevent saturating or exploding gradients is to simply clip the gradients. It remains important to properly initialize the weights. A proposed method for good weight initialization is called Xavier initialization [25].

This initialization states that we should initialize weights $\boldsymbol{W}^{[i]}$ drawn from a random uniform distribution, where the variance is $\text{Var}(\boldsymbol{W}^{[i]}) = \dfrac{2}{(n_{\text{in}}^{[i]} + n_{\text{out}}^{[i]})}$, where $n_{\text{in}}^{[i]}$ and $n_{\text{out}}^{[i]}$ are the number of neurons in the previous and next layer respectively. This tries to make the variance of the outputs of a layer, equal to the variance of the inputs. This work, however, was mostly based on the sigmoid activation function. In a follow-up paper by He et al. [28] the Xavier methods was revised to work better with ReLU activation functions. Instead, the weights are drawn from a random Gaussian with variance: $\text{Var}(\boldsymbol{W}^{[i]}) = \dfrac{2}{n_{\text{in}}^{[i]}}$. This is currently the most common method, for the weight initialization of deeper feedforward networks and is the current default option in several deep learning libraries, such as Keras [14].

A recent development by Ioffe and Szegedy called batch normalization [33] also reduces problems with properly initializing neural networks. The key observation this work makes is that the distribution of the layer inputs are constantly shifting slightly when training with mini-batches. This is because the distribution of each mini-batch is slightly different than that of the full training set. This problem is amplified throughout the network by weight multiplications; thus, small changes in previous layers can lead to larger changes in later layers. They refer to this problem as internal covariate shift. The layers have to constantly adapt to the newly seen distribution, which slows down learning.

The idea of batch normalization is to normalize layer outputs. This will ensure that the output of a layer has a fixed mean and variance. What it does is standardize the mini batch by subtracting the mini-batch mean and dividing the mini batch by its standard deviation. The output of the layer will then have zero mean and unit variance. Batch normalization then adds two learnable parameters β and γ for each layer, on which it is applied. These parameters can shift and scale that mean and variance. These parameters are updated similarly to the network weights, so the network can learn an optimal mean and variance for a layer output. Algorithm 1 shows batch normalization.

Algorithm 1 Batch normalization over a mini batch. Adapted from Ioffe and Szegedy [33]

Input: Values of x over a mini-batch $\mathcal{B} = \{x_1, \ldots, x_m\}$; parameters to be learned γ, β.

Output: $y_i = \text{BN}_{\gamma,\beta}(x_i)$:

$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ ▷ mini batch mean

$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$ ▷ mini-batch variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$ ▷ normalize

$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$ ▷ scale and shift

Batch normalization weakens the coupling between earlier and later layers, reducing the effects of internal covariate shift. Since data is standardized after every layer, the effect of improperly initialized weights is reduced. In addition, batch normalization has a slight regularization effect. We will briefly return to this adventitious effect at the end of the following section.

4.1.9 Generalization

In supervised learning, we are not only interested in how well we can fit the model to our training set; if we only try to fit our model on the training set, we would simply be doing optimization. We want to find specific regularities in the data that generalize well to new data. The ability to perform well on unknown data is called generalization. This is why we often work with a separate test set, and in some cases also an additional validation set, in supervised learning. The goal is to optimize the generalization error or test error. During learning, however, we only see the training set. This means that we have to assume that the data from the two sets is *independent and identically distributed* (i.i.d). With these assumptions, we train a supervised algorithm by minimizing the error on the training set and reducing the gap between the training error and the test error.

The capacity of a model is loosely defined as the complexity of the function it can model. The higher a model's capacity, the more complex the function it can represent. There is a trade-off between a model's *bias* and *variance*. Bias is an error arising from incorrect assumptions in the model. The variance is an error from the sensitivity to small fluctuations in the training set. Models with low capacity might not be able to fit the training set, which is called *underfitting*. In this case, the model shows low variance, but high bias. The inverse is when the model capacity is so high that it memorizes certain properties of the training set that do not generalize to the test set, which is called *overfitting*. In this case, the model shows low bias, but high variance. Figure 4.4 shows the relationship between a model's capacity and error.

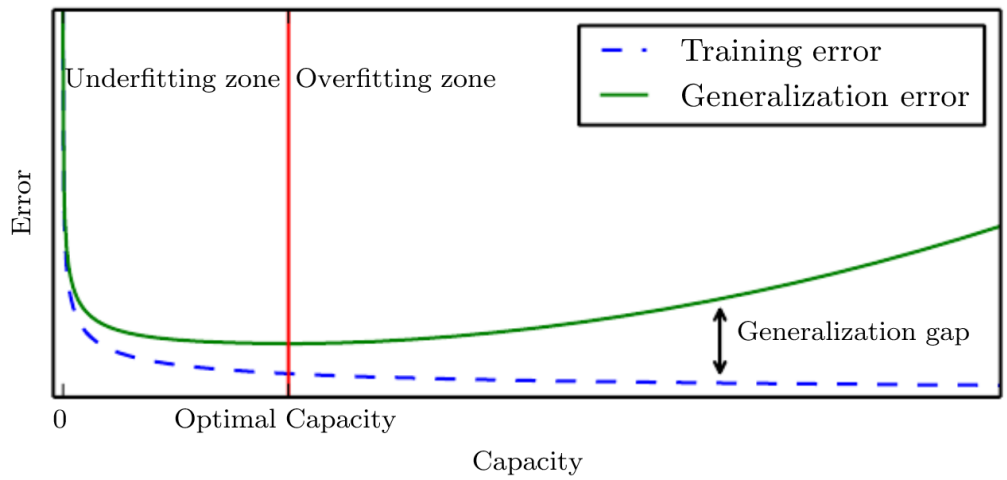


Figure 4.4: Relationship between model capacity and error. Adapted from Goodfellow et al. [26]

When creating a model, we must keep these trade-offs in mind. Often, when training a model, we use a separate validation set to monitor the generalization error and adjust the model (e.g., hyperparameters) to minimize this error. In this way, our test set can remain an unbiased evaluator of our model's performance.

layers and millions of parameters. We will discuss some commonly used methods to reduce the effect of

overfitting and close the generalization gap. We refer to these methods as *regularization*.

One common form of regularization is norm regularization. Norm regularization is not specific to neural networks but can be used in them as well. With norm regularization, we define a cost function as follows:

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda\Omega \tag{4.14}$$

where $\lambda \in [0, \infty)$ is a hyperparameter weighing Ω, which penalizes the parameters in some way. Large parameter values may cause the network to create a large response to small changes in the input. This can cause the network to learn noise in the training data, which does not generalize to the test set. By penalizing the parameter values of the network, we attempt to prevent this from happening. One common form of norm regularization sets $\Omega = \frac{1}{2}\|\boldsymbol{\theta}\|_2$, which is known as *$L^2$-regularization*, *ridge regression* or *Tikhonov regularization*. In L^2-regularization we shrink the weights proportional to $\boldsymbol{w}$.

Another commonly used method is *L^1-regularization*, which sets $\Omega = \|\boldsymbol{w}\|_1$. The difference between L^1-regularization and L^2-regularization is that L^1-regularization shrinks a parameter value much less when it has a large magnitude and much more if a value has a small magnitude. This will cause the network to concentrate more on a small number of important connections, making the parameter vector sparse.

Another recently described regularization technique, now commonly used in deep neural networks, is called *dropout* [73]. The idea of dropout is as follows: while training a neural network, at an iteration, we randomly disable a fraction p of the neurons in the network. We forward- and backpropagate through the network and update the parameters, then enable the disabled neurons. For the next iteration, we repeat this process. We keep doing this during the training process. We compensate for the fact that a portion of the neurons was disabled by either scaling the activation at test time by the dropout ratio p or inverting the dropout during the training process by scaling the dropped out activations by $\frac{1}{p}$.

An earlier work by Krizhevsky et al. [43] gives a heuristic explanation of why dropout works. Their work states: 'This technique reduces complex co-adaptations of neurons since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.'

Another interpretation of why dropout works is that we can view it as an ensemble of classifiers where each step in the training batch creates a smaller network, which are all ensembled in a similar manner to bagging [85]. Wager et al. [83] show that dropout works by implicitly imposing a L^2-regularizer on the weights of the network. Dropout has been a popular regularization technique, mainly due to good results in deeper networks. Some more recent works [35, 24, 80] have adapted dropout to work well with convolutional neural networks. As we will discuss in the section hereafter, convolutions are spatially correlated, which causes information to flow through, despite dropout.

To conclude regularization, we return to batch normalization. Recall batch normalization normalizes a layer output by the mean and variance of the mini-batch. Since this mini-batch is a random subset of the training set, the mean and variance of these batches fluctuate. This creates some additive and multiplicative

noise and each layer has to learn to be robust against this. However, this effect is reduced with larger batch sizes because the mean and variance of different batches approach one another. In some recent works, which are discussed in Section 4.3, batch normalization is the only form of applied regularization.

This concludes the general introduction to neural networks within the context of supervised learning. In the next section, we discuss convolutional neural networks, a specific type of neural network that has shown successes in many machine-learning tasks in recent years.

4.2 Convolutional neural networks

In this section, we discuss the *convolutional neural network* (CNN). CNNs are neural networks that are suited to deal with data in a grid-like topology, where there is a statistical dependency between neighbouring elements. The functional form of CNNs is identical to *fully connected network*s (FCs), but CNNs exhibit some additional constraints. A CNN applies *convolution* or *cross-correlation* in one of its layers in place of general matrix multiplication. CNNs sometimes also incorporate so-called pooling layers that down-sample the input. We will discuss both the convolution and pooling operation in this section.

4.2.1 Convolution

Convolution can be seen as matrix multiplication, where some matrix entries are constrained to be equal to others. Because this work is focused on images, we describe the discrete convolution operator over a two-dimensional input:

$$Y(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{4.15}$$

We call I the input (image) to the convolution and K the kernel or filter. We will use both these terms interchangeably. The output of this convolution is called a *feature map* or *activation map*.

Since convolution is commutative, we can also write:

$$Y(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{4.16}$$

This form is easier to implement since there is less variation in the range of valid values for m and n, but note that we have flipped the kernel relative to the input. This is done to retain the commutative property of the operation. A form that is more often implemented in machine learning libraries, but does not retain this property by flipping the kernel is called cross-correlation:

$$Y(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{4.17}$$

Since this commutative property is not important for our neural networks, we will use the term convolution where we mean cross-correlation. The convolution operator can be interpreted as sliding the kernel over the

input, where we create a weighted average of the input at that position (when K is a valid probability density function). Convolution can replace general matrix multiplication in one or more layers of the neural network.

Convolution provides several useful properties for neural networks. One property of CNNs is that the network has sparse connectivity. Typically, the kernel size is smaller than that of the input. With images, kernel sizes of 3,5 and 7 are often used [32, 67, 75, 43]. This means that, opposed to FCs, we enforce only a local connectivity pattern. However, this sparse connectivity is not limiting in deeper networks, as the *receptive field* of nodes in deeper layers may span a large input region.

Another advantage of using convolutional layers is parameter sharing. By using the same kernel that 'slides' over the image, the weights or parameters applied to one input is tied to the value of the parameter applied elsewhere. Each element of the kernel is used at every position of the input. This causes the layer to be equivariant to translation: when we move an object in the input, its representation will move by the same amount in the output. Parameter sharing causes CNNs to be far more efficient than FCs using matrix multiplications. The reduced number of parameters helps the network train faster and reduces the risk of overfitting, although the model capacity is theoretically less than that of a fully connected network. This is because for every convolutional layer there is a fully connected layer that implements the same forward function by using a weight matrix with mostly zero entries. Conversely, fully connected layers can be implemented into convolutional layers by setting the filter size to exactly that of the input volume.

Activation maps are composed of the responses of a filter for every spatial position of the input. When (successfully) training a network, the parameters of a filter will be updated in such a way that the activation map that is produced provides useful patterns for subsequent layers. The responses of filters in earlier layers often include simple visual features, such as edges in different orientations, whereas deeper-layer filters often respond to more complex visual patterns [89] when applied to images. In most cases, a single visual feature per layer might not result in a model that is sufficiently discriminative. Different filters, when initialized differently, will likely produce different feature maps, driven by the decrease of the cost function.

There are several factors that determine the output volume of the convolution operation on an image. As we are sliding a filter over the image, we can determine the filter size F (we assume a square filter for now), number of filters K, *stride* S and the amount of *zero-padding* P. These parameters are set beforehand as hyper-parameters. When working with finite data such as images, we have to take special care around the boundaries of the input. Zero-padding is a way of dealing with these border cases. By padding the input with zeros, we preserve the input size and retain information at the border of the input. When the input is not padded, the convolution operation is often called *valid convolution*. When we restrict the centre of the kernel to overlap with the total input by using zero-padding, we call the operation *same convolution*. The size of the output volume $W_2 \times H_2 \times D_2$ is determined by the shape of the input volume, the number of

filters K and the aforementioned hyperparameters as follows [36]:

$$W_2 = \frac{(W_1 - F + 2P)}{S} + 1 \tag{4.18}$$

$$H_2 = \frac{(H_1 - F + 2P)}{S} + 1 \tag{4.19}$$

$$D_2 = K \tag{4.20}$$

If we have an RGB image, we have three separate input channels. Since patterns in different channels are not similar per se, we generally use different filters for each input channel. The generated feature maps are stacked. Although we have three channels, we still call this 2D convolution. We can consider the filter as being fully connected in the channel dimension and convolutional in the spatial dimensions of the input.

4.2.2 Pooling

CNNs, in addition to convolutional layers, sometimes contain pooling layers. Pooling layers down-sample the volume spatially, while volume depth is retained. The pooling layer reduces the number of parameters in the network and therefore reduces the risk of overfitting. A common form of pooling is max-pooling, where a filter of a certain size is stridden along the input in both height and width (for each depth slice independently) and the maximum value of the input area is retained. Similar to the convolution operator, two hyperparameters can be set: the spatial extent of the filter F (again assuming a square filter) and the stride S. The output volume is then calculated as follows [36]:

$$W_2 = \frac{(W_1 - F)}{S} + 1 \tag{4.21}$$

$$H_2 = \frac{(H_1 - F)}{S} + 1 \tag{4.22}$$

$$D_2 = D_1 \tag{4.23}$$

The difference here is that zero-padding is commonly not used with pooling. Pooling adds no additional learnable parameters to the network, but the hyperparameters have to be set. Common values are $F = 2$ and $S = 2$ when using max-pooling. Other methods, such as average pooling and L^2-pooling work in a similar manner as max-pooling, but are less prevalent in the literature. Pooling helps to make the representation somewhat invariant to small translations of the input: if the input is translated slightly, the output will not change. Pooling does not introduce rotation or scale invariance, so data augmentation techniques might still be useful in some cases, where objects can appear in different scales or rotations (except self-similar objects or objects with radial symmetry). Pooling is useful in, for example, image classification, where we only care if a feature is present, but not in the exact location of the feature. However, a downside to pooling is a loss of spatial information. For other tasks in which the exact spatial location of features is more important, such as in image segmentation, pooling may result in the loss of too much information.

4.3 State-of-the-art neural networks

The popularity of neural networks in recent years has been mainly driven by their remarkable performance in various domains. In this section, we discuss some of the key contributions and recent development in the field and their performance on a commonly known benchmarking data set for image classification called ImageNet [19]. ImageNet contains over 15 million images with approximately 22,000 categories. Since 2010, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which evaluates algorithms for object detection and image classification, has taken place annually. The classification task of the ILSVRC involves a subset of labelled images from ImageNet; the challenge is to obtain the best top-1 and top-5 score. Top-1 is whether the prediction in the same as the image label and top-5 is whether the image label is in the algorithm's top-5 predictions.

The first highly successful deep learning approach to ILSVRC was published by Krizhevsky et al. [43] in 2012. Their network, referred to as 'AlexNet', outperformed all previous methods, with a more than 10.8 per-cent point decrease in error to a top-5 error rate of 15.3%. A key contribution of this work was the idea that deeper networks increase performance, even though they are more computationally expensive. The architecture used five convolutional layers and three fully connected layers. The layers are followed by a ReLU nonlinearity and max-pooling is applied after each convolutional layer. In addition, the authors used Dropout as a regularization technique. Figure 4.5 presents a visualization of the network from the original paper by Krizhevsky et al.

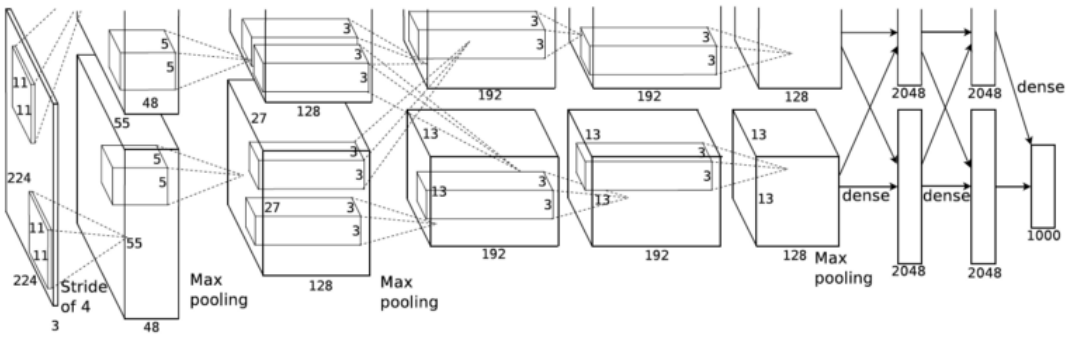


Figure 4.5: AlexNet architecture. The network is split into two streams to distribute computation over two graphics processing units (GPUs). Adapted from Krizhevsky et al. [43]

The next milestone in image classification came from a study by Simonyan and Zisserman [67] in 2014. In their work, they introduced the VGG-16 model, which composed of sixteen convolutional layers and three fully connected layers, where the convolutional layers are again followed by ReLU activations and 2×2 max-pooling with a stride of 2. They used 3×3 convolutional filters instead of 11×11 used in AlexNet. Using smaller filters they reduced the number of parameters but showed that they could recognize the same useful patterns. Additionally, this work introduced the VGG-19 model, which had 3 more convolutional

layers. This increased accuracy with 0.1 per-cent point, but also increased the number of parameters and therefore also memory requirements and computation time. The VGG-16 model obtained a 7,3% top-5 error, reducing the AlexNet error by a factor of two. Figure 4.6 shows the VGG-16 architecture.

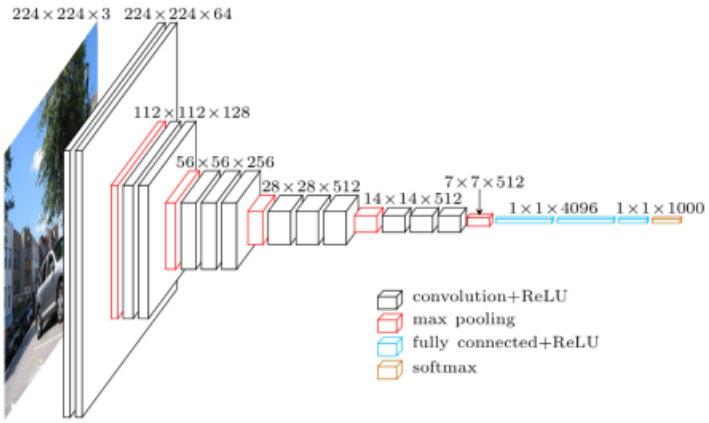
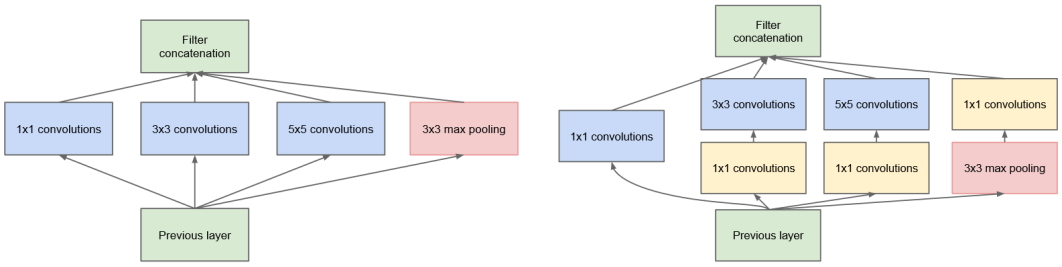


Figure 4.6: VGG-16 architecture [7]. VGG-16 starts out with a small number of channels and doubles it after every max-pooling layer while the max-pooling halves the spatial dimension of the generated feature maps.

Also in 2014, Google researchers Lin et al. [76] developed the idea of Inception modules. Their idea was to train multiple heterogeneous (differently sized) convolutional kernels and a max pooling at the same time and stack their generated feature maps. Different kernels can capture features on different scales in the input, which turns out to be useful. The network can then learn which size or combination of sizes is most useful. The initial idea of this *split-transform-merge* strategy is shown in Figure 4.7.



(a) Inception module, naïve version (b) Inception module with dimension reductions

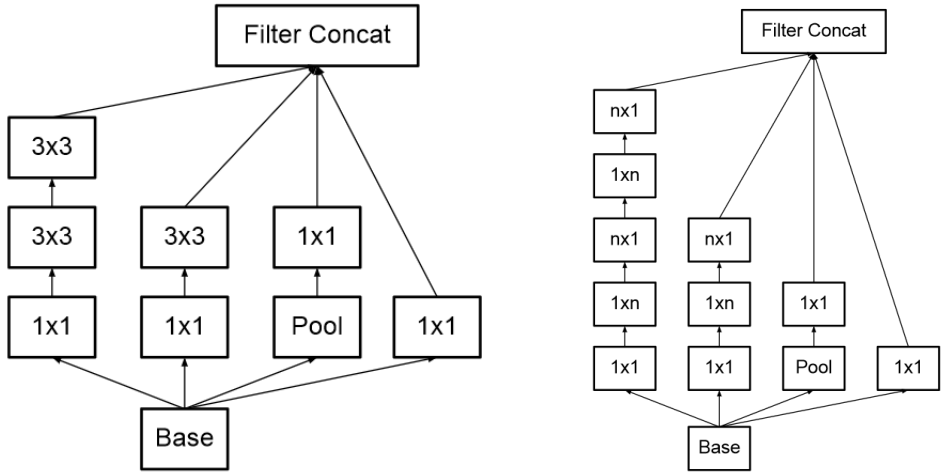
Figure 4.7: Inception module, adapted from [76]. To match up output dimensions same-convolutions are used and padding is added for max-pooling. On the left (a) is the naive version, without dimensionality reduction. On the right (b) 1×1 convolutions are used to reduce dimensionality

To reduce the computational cost, 1×1 convolutions are used. 1×1 convolutions reduce dimensionality in the channel dimensions but not the spatial dimensions. By applying these convolutions, the input is

mapped to separate lower-dimensional embeddings, in which they are transformed by regular convolutions. Their idea stated is that cross-channel and spatial correlations are sufficiently decoupled to allow factoring of these operations [13]. Lin et al. refer to the 1×1 convolutions as cross-channel parametric pooling. Their work effectively reduces the computational complexity while maintaining representational power.

The authors' network GoogLeNet/InceptionV1 used this idea to create a network with over 50 convolutional layers, reducing the top-5 error to 6,7%. Compared to VGG-16, which has 138 million parameters, GoogLeNet is small, with only 6.7 million parameters.

A follow-up study by the same authors [77] presented improvements to the original InceptionV1. The model describes new types of Inception blocks. The first change is to replace the 5×5 convolutions by two 3×3 convolutions. This gives similar representational power as the receptive field of the neuron in the second layer is still 5×5, but is approximately 2.8 times more efficient. Stacking two 3×3 kernels counter-intuitively uses fewer parameters and more non-linearity (when adding non-linearities between the convolutions). Another idea the authors propose is to factorize any $n \times n$ convolutions into $1 \times n$ and $n \times 1$ asymmetric convolutions. This factorization worked well in later layers of the network and is again computationally cheaper. Figure 4.8 illustrates these factorizations.



(a) Factorization of convolution kernels into multiple smaller (symmetric) kernels

(b) Factorization of $n \times n$ convolutions into $1 \times n$ and $n \times 1$ convolutions

Figure 4.8: Inception modules with dimension reductions, adapted from [77]. On the left (a) convolutions are factorized into convolutions with smaller kernel sizes. On the right (b), this idea is taken further to factorize into asymmetric convolutions,

Their work discusses different variants with inception blocks based on these ideas. The authors refer to InceptionV2 as InceptionV1 with batch normalization and test different combinations of inception blocks based on the mentioned ideas. The best-performing network is then trained with RMSProp and custom regularization is added to the loss function. The resulting network is called InceptionV3.

Another variant, which takes the idea of Inception to the extreme is called Xception [13]. Xception proposes a network architecture of solely depth-wise separable convolutions, as shown in Figure 4.10 (channel-wise spatial convolutions followed by 1×1 pointwise convolution). Whereas in Inception networks, the pointwise convolutions are followed by a nonlinearity before the spatial convolutions, the Xception model omits this nonlinearity. Moreover, Chollet only uses 3×3 spatial convolutions. Nonetheless, the model resulted in higher performance compared to InceptionV3 with the same amount of parameters and the author argues that this is due to more efficient use of the parameters.

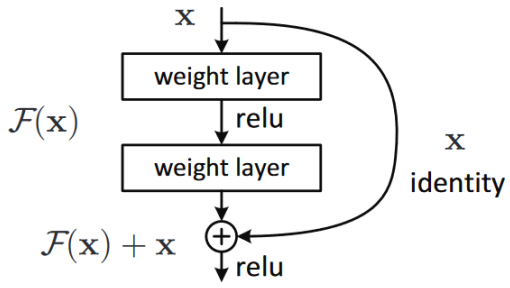


Figure 4.9: Residual block, adapted from [29].

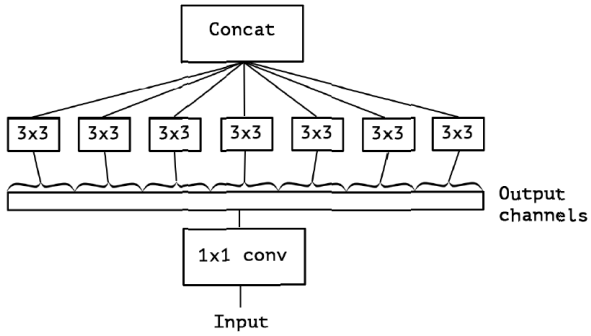


Figure 4.10: An Xception module as visualized by Chollet [13].

One common trait of these state-of-the-art networks is their increasing depth. However, He et al. [29] have obsevered that increasing depth eventually causes the training error to go up. The authors state that this is an optimization problem, due to the difficulties in training such deep models and that it is not a cause of overfitting. In this work, they showed that the addition of so-called residual blocks helped the learning process. Residual blocks contain a shortcut connection between earlier and later layers in the network as seen in Figure 4.9.

These shortcut connections do not contain weights or non-linearities. The idea behind these shortcut connections is that the network can easily learn the identity function, by pushing parameters in the block to zero, which helps to preserve the gradient. This reduces the vanishing gradient problem; therefore, the addition of more layers will theoretically not decrease performance. It has also been shown that these shortcut connections prevent the increase in non-convexity of the loss landscape when networks get deeper [47], which makes gradient-based learning easier. The presented network 'ResNet' is a network which contains many of these residual blocks stacked. This network is a 152-layer network, which obtained a 3,57% top-5 error.

Other research has followed up on the idea of residual blocks. Some researchers have combined the idea of inception and residual blocks, in a network called Inception-ResNet [75]. We will discuss this architecture in more depth in Chapter 5 as it forms the basis for our custom architecture. In addition, work has been done on creating additional skip connections between all layers in a network. This network is called DenseNet [31].

This concludes our review of the current state-of-the-art models for general image classification. We conclude this chapter by noting that convolutional neural networks show promising results for the large scale recognition of images. In the following chapter, we describe the methodology for the large-scale recognition of pavement cracks, in which we use some of the most promising models from the literature and make adaptations to one particular model.

Chapter 5

Methodology

In this chapter, we describe the used methods for the classification of pavement cracks and introduce the data used to test these models. The general classification pipeline in this work follows the structure shown in Figure 5.1. Using these components as a framework, we will describe our rationale on our choices for each.

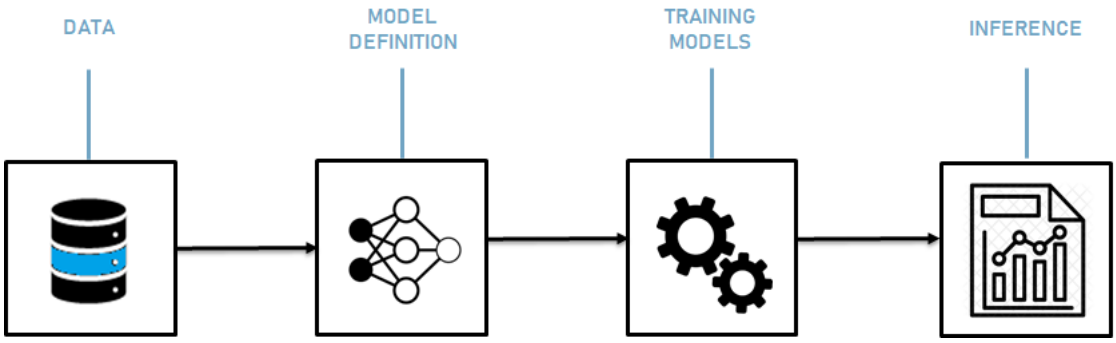


Figure 5.1: General methodology pipeline

In this chapter, we elaborate on the introduction and sampling of the data and the definition of the models. In the chapter hereafter, we elaborate on training the models and results that follow from inference.

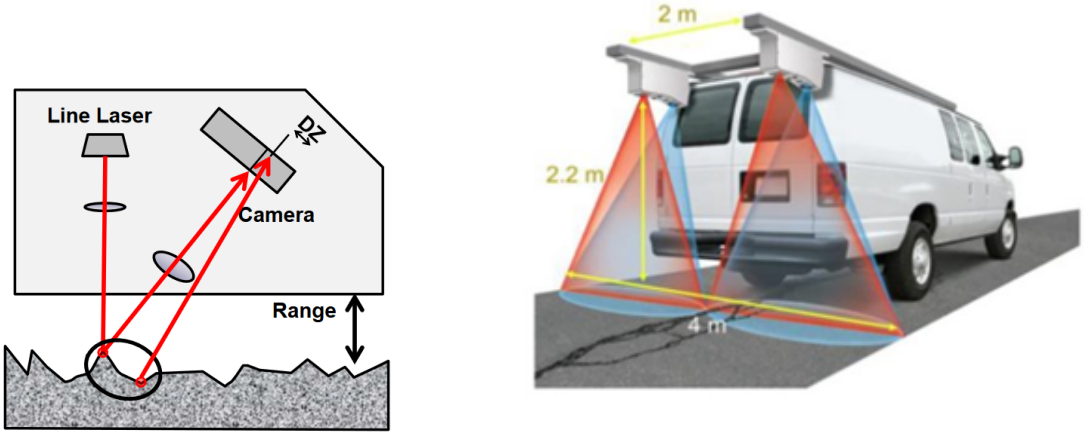
5.1 Data

In this section, we describe the collection and processing of the data, including a detailed discussion of the capturing process by the sensor system used. Properties and sampling of the data are additionally discussed. This will give more insight into the difference between cracks and non-crack patterns and how we can exploit these aspects in the used models.

5.1.1 Data collection

In recent years, the road management industry has witnessed a trend toward the use of high-speed scanning lasers. The most commonly used system for scanning road surfaces is the aforementioned LCMS by Pavemetrics[1]. This system is also used by the Dutch Highway Agency (Rijkswaterstaat). Data collection is performed by a Digital Highway Data Vehicle with the LCMS installed.

The vehicle is equipped with two line scanning lasers, each with a field of view of half a lane-width; the combined measurement therefore covers a full lane (~ 4 m). While driving, the sensors can measure intensity and range using LIDAR. Using two cameras, measurements are taken on dry road surfaces in daylight to reduce the influence of weather. In this manner, each year, the entire state-maintained Dutch road network is scanned and processed. Line measurements are combined to form 3D height profiles of road surfaces. An illustration of the LCMS sensor and vehicle is shown in Figure 5.2



(a) LCMS sensor **(b)** Vehicle with LCMS sensors

Figure 5.2: An illustration of the LCMS sensor and a vehicle with these sensors, showing how the data is captured.[2]

Since 2011, the Dutch Highway Agency has collected and processed data using this methodology. To address vehicle movement while using the LCMS sensors, Inertial Measurement Units (IMUs) are employed to measure inertia, which is later processed to compensate for linear and angular motion. The resolution of the measurements is 1.0 mm (transversal) by 4.7 mm (longitudinal). Single point depth measurements are accurate up to 0.25 mm.

Since there are some changes in resolution and post-processing of the data, as well as keeping the size of the data manageable, in this work we use data from 2016.

[1]Pavemetrics, http://www.pavemetrics.com

5.1.2 Processing of the data

The data processed by the LCMS software after it is collected. Processing is done by the Pavemetrics[2] software including several additions by TNO. Measurements are processed to form 100-meter blocks for each measured lane. These hectometre blocks correspond to the markers that are present every 100 meters alongside highways to indicate position. A full hectometre is roughly 21.000 by 4.000 pixels. The pixel intensities represent the relative depth from the sensor, where points further away from the sensor are represented by lower pixel values.

The algorithms further compensate for vehicle movement using the IMU data and detect ravelling of the road surface as discussed in Chapter 3. Additionally, the algorithms compensate for wheel paths (rutting) and measure crack-like patterns.

These range images are further processed to detect crack-like patterns. This detection is similar to the AAHSTO PP67 standards. This means the software detects the location, orientation, length and width of a pattern. These crack-like patterns are additionally grouped together into a single crack if they have both the same direction (longitudinal or transverse) and are less than 50 cm apart. Longitudinal patterns have an angle (orientation) of less than $\pi/8 = 22.5$ degrees with the driving direction. Patterns with angles higher than 22.5 degrees are considered transverse cracks. Block or "other" cracking from the AAHSTO PP67 standard is not registered, but small crack segments (< 10 cm) are added to larger cracks independent of direction and therefore block cracking can be classified as transverse or longitudinal cracks. As a final step, the created crack is given a label, according to Table 5.1.

Table 5.1: Crack classes according to the Pavemetrics/TNO system for measurement year 2016.

Crack class	Crack type	Description
0	repair	repair of other damage
1	crack	an actual crack
2	filled crack	a historical crack that has been repaired with a filler
9	other	other damages

Currently, this classification step of cracks involves a manual annotation. The difference between a crack (class 1) and other (class 9) is not automatically performed. Thus, every year, inspectors of the Dutch Highway Agency manually inspect the generated 3D image data and the new crack-like patterns that appear on them. These cracks are then labelled by an inspector to be either an actual crack (class 1) or other (class 9). As mentioned in Chapter 2, the other class includes detections of jointed surfaces, rim marks, pavement joints, road shoulder drop-offs and other damages that are not referred to as cracks. It is open to each inspector to assess and determine whether these damages are in fact cracks based on guidelines.

[2]Pavemetric 3D vision systems, `http://www.pavemetrics.com/`. Last visited: 15 October 2018

Some differences in interpretation have therefore been observed, and the data is thus not labelled with 100% accuracy. An overview of this process is shown in Figure 5.3.

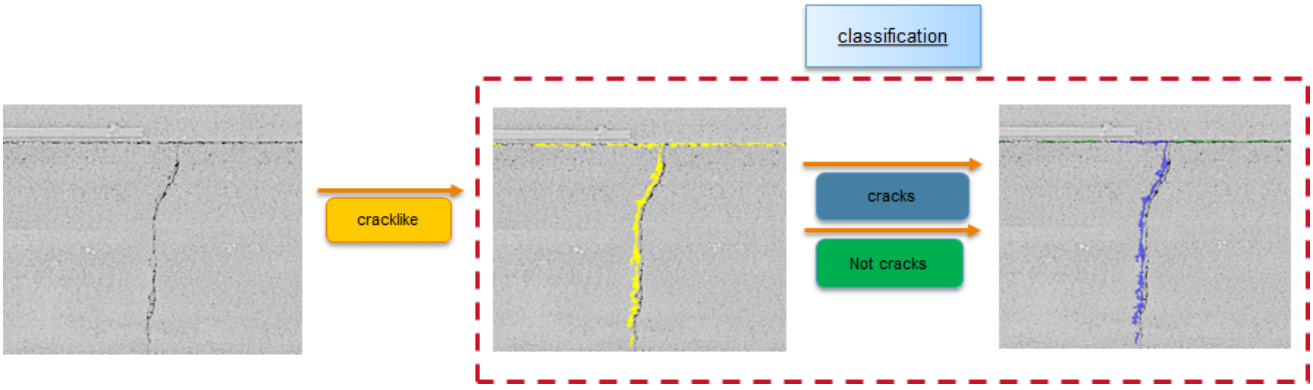


Figure 5.3: Labelling process as done by inspectors. Detected crack-like patterns (yellow) are labelled into either crack (blue) or non-crack (green).

The image shows the original range measurement on the left. The LCMS software detects the crack-like patterns, which are shown as an overlay in the middle picture. The inspector then classifies the different parts as shown in the overlay on the right.

5.1.3 Properties of the data

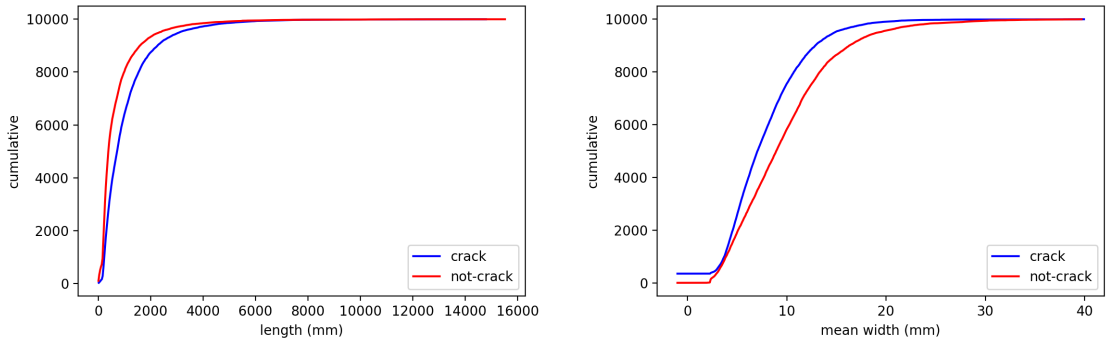
We carried out a preliminary data analysis to understand the differences between cracks and other crack-like (non-crack) damages. The data was in 2016 and covers approximately 15,000 lane kilometres of road spanning 16 administrative districts. The data consists of 52 state roads (Rijkswegen). Each state road consists of multiple lanes (in both directions) and each lane can be divided into 100 m blocks (hectometres), forming a total of around 106,000 lane hectometres. Approximately 83% of these lane hectometres contain one or more cracks or crack-like patterns.

Each pattern is described by a vector of interconnected points $(x, y$ pairs). The total length of the crack is the sum of lengths of the Euclidean distance between these points:

$$\text{cracklength} = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{5.1}$$

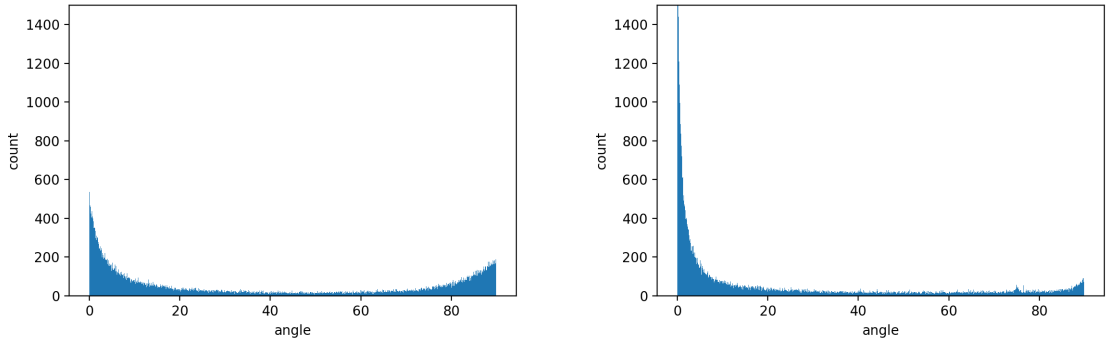
In addition, each of the $n-1$ crack segments has a width, which is the widest part of that crack segment. The cumulative distributions of the length and width of cracks are shown in Figure 5.4.

Figure 5.4 demonstrates that the average width of a crack is somewhat smaller than that of a non-crack pattern, but that a crack is on average longer than a non-crack pattern (larger median). However, we also see that much longer non-crack patterns do exist.



(a) Length **(b)** Width

Figure 5.4: Cumulative distributions of the length (a) and width (b) of cracks and non-crack patterns, taken from equally sized subsets of 10k samples.

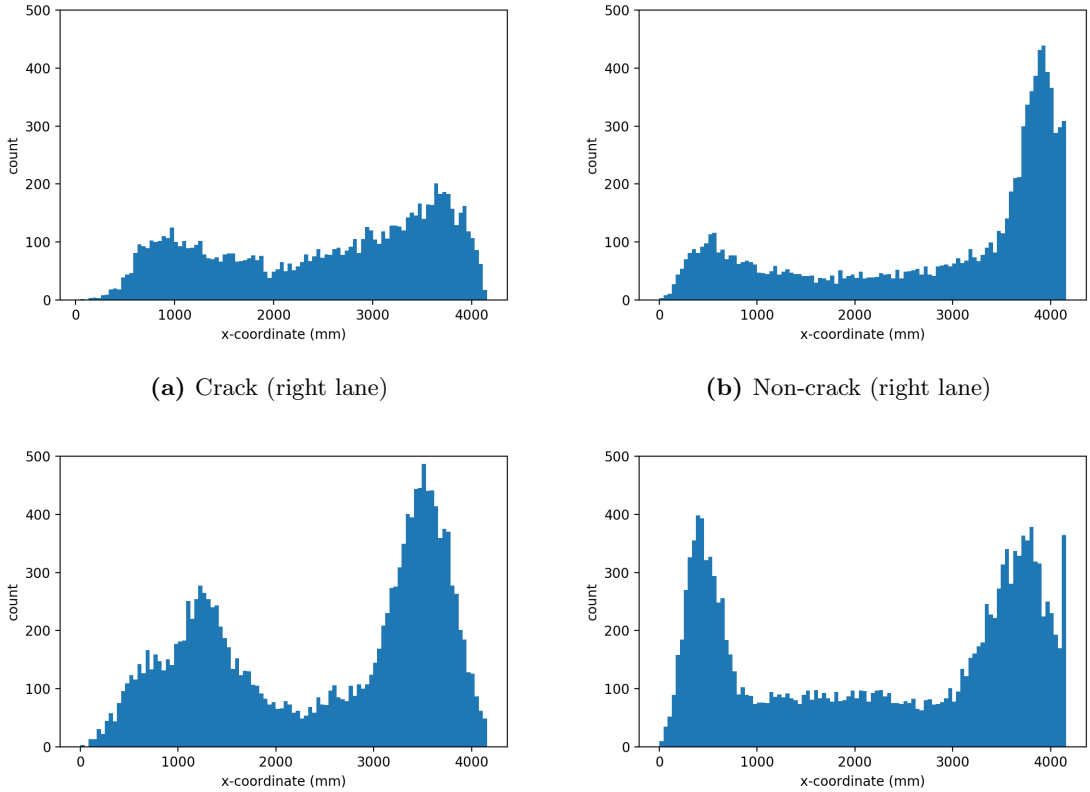


(a) Crack **(b)** Non-crack

Figure 5.5: Distributions of the angle of cracks (a) and non-crack patterns (b) of cracks and non-crack patterns, taken from equally sized subsets of 10k samples.

Examining these properties of the data, we see that there are some differences between cracks and crack-like patterns. Since inspectors of the Dutch Highway Agency base the classification solely on the range data and not from the captured statistics, we expect to gain all discriminative features from the range data as well. These are not limited to the shown features per se. We will therefore also solely use the range data in our main method. We, however, return to the mentioned features in an alternative feature-based approach, which we will use as a baseline classifier.

In order to classify cracks based on the range data, the data has to be subsampled; this process is discussed in the following section.



(a) Crack (right lane)

(b) Non-crack (right lane)

(c) Crack (other lanes)

(d) not-crack (other lanes)

Figure 5.6: Distributions of the position of the crack or not-crack pattern in a lane, taken from equally sized subsets of 10k samples.

5.1.4 Sampling of the data

In this section, we will describe our method of selecting the data that we use to test our methods. The main sampling method is a patch-based approach, by taking crops of crack-like patterns from the range data. These crops are used for the training and testing of several CNNs.

As an alternative, we briefly discuss a feature-based sampling method. This method is used for the training and testing of a feature based classifier, XGBoost. This classifier is introduced in Section 5.2.5

Sampling as a patch based approach

CNNs, as discussed in Chapter 4, are often computationally limited to a fixed input size (in the presence of fully connected layers for classification). To create a balanced training set of both cracks and non-cracks, we applied a stratified sampling method, which is outlined in Algorithm 2.

This method guarantees a 50/50 balance between cracks and non-crack patterns in our set. Picking a random lane hectometre samples cracks and non-crack patterns from different regions, which are labelled by

Algorithm 2 Sampling of random cracks and non-crack patterns

```
# num_samples: number of cracks and cracklike patterns to be sampled in total
# patch_sizes: set of different patch sizes to use
# samples_per_crack: number of samples to take for each crack (per patch size)
procedure SAMPLE(num_samples, patch_sizes, samples_per_crack)
    for n=0;n<num_samples;n++ do
        for c ∈ {crack, cracklike} do
            for sp=0;n<samples_per_crack;sp++ do
                # random hectometre rh, containing one or more patterns of type c
                rh ← RANDOMHECT(c)
                # random crack/cracklike pattern rc from the random hect
                rc ← RANDOMCRACK(rh)
                for ps ∈ {patch_sizes} do
                    # samples_per_crack random points in the crack/cracklike pattern rc
                    rs ← RANDOMPATCH(rc, ps, samples_per_crack)
                end for
            end for
        end for
    end for
end procedure
```

different inspectors. This way, the data is not biased by a single inspector's labelling, or by the particular traffic conditions of certain regions and roads.

The random section procedure takes a random point from the cracklike pattern. The vector that defines this pattern consists of (x, y) pairs, relative to the start of the hectometre. The lines between these pairs determine the crack. A random point can be sampled anywhere on these lines. This point then serves as the middle point of the patches. Patches that overlap with the border of the lane hectometre are zero-padded in those regions. Figure 5.7 illustrates the sampling of these patches.

We note that we explicitly do not sample asphalt segments without any cracks/non-crack patterns.

To have sufficient data for the training of deeper models and a rigorous analysis afterwards, we sampled 100,000 cracks and non-crack patterns using square patch sizes with lengths 250 mm, 500 mm, 750 mm, 1000 mm and 2500 mm. This gives a total set of 1 million samples (200,000 per patch size).

The reason for using this patch-based sampling method was to obtain fixed size inputs. As described earlier in this chapter, cracks can have various dimensions. Taking the pattern as a whole and re-sizing it to fit a fixed input may cause different distortions, as some patterns will have to be up-sampled and others down-sampled significantly. A patch-based approach avoids this problem, and even when re-sizing

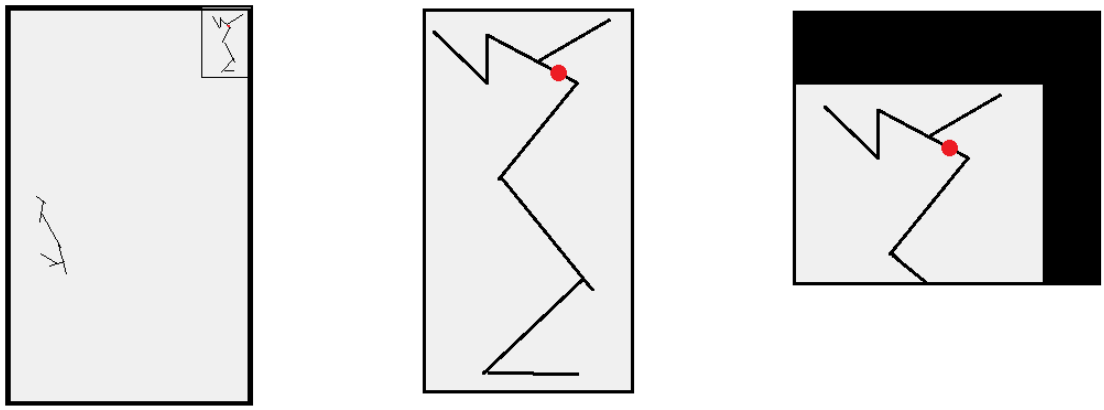


Figure 5.7: Pattern sampling. The red dot indicates the sampled point. Left: a full lane hectometre with 2 patterns. Middle: zoomed into the pattern on the top right. Right: sampled patch from the same pattern, where the patch is centred around the red dot.

the patches, all cracks will be distorted evenly (as we only train on a single patch size per model).

Cracks and non-crack patterns are split in a 80/20 train/validation split. Each of these sets has a 50/50 ratio of cracks and non-crack patterns. Patterns sampled from the same lane hectometre can only appear in either the training split or validation split to prevent accidental data leakage between the sets. Splitting on patterns, instead of individual patches, also prevents leakage in the same way. Having the same patches in both the training and validation set might cause the network to simply memorize the data completely instead of extracting patterns that generalize [92]. Patches of the same crack and patch sizes of the same crack all appear in the same sets. For the test set, another 50k patches are sampled. However, in that case, the 50/50 ratio is not held, but we use the original distribution. This is to get a realistic set for actual real-world performance. We make sure again that there is no overlap with the training set.

Sampling of features

As mentioned, to establish a baseline performance, based on simple features, we derive several features for each pattern. Using a simple feature-based classifier confirms our hypothesis that mere simple features are not sufficient to classify cracks and more complex features, derived from the images is needed.

The features are intuitively chosen to be associated with patterns being either a crack or non-crack pattern. The features are shown in Table 5.2

5.2 Model definition

In this section, we propose the main method for the classification of cracks. First basic CNN architectures and the manner in which they are used are discussed. Thereafter, the choices made to create a custom architecture are examined.

Table 5.2: Different 'classical' features of cracks and non-crack patterns

Feature	Type	Description
right lane	bool	the pattern is in the rightmost lane
# segments	int	the number of crack segments
angle	float	the angle of the crack to the driving direction
length	int	the total pattern's length in mm (sum of segments)
mean width	float	the average pattern segment's width in mm
width var	float	the variance of the width of the pattern
max width, height	int	the maximum of the pattern's width and height

5.2.1 Base models and feature extraction

We test some of the promising earlier described models from (Section 4.3) on the patches. These models are selected for their performance on general image classification. The selected models are as follows: ResNet, DenseNet, Xception, InceptionV3 and Inception-ResNet. These models are selected for their performance on general image classification. They consist of two parts, which can be separated into: the feature extraction part, which consists mainly of convolutional layers; and the classifier block, which, in the case of classification, forms a class prediction from the extracted features. This is illustrated in Figure 5.8.

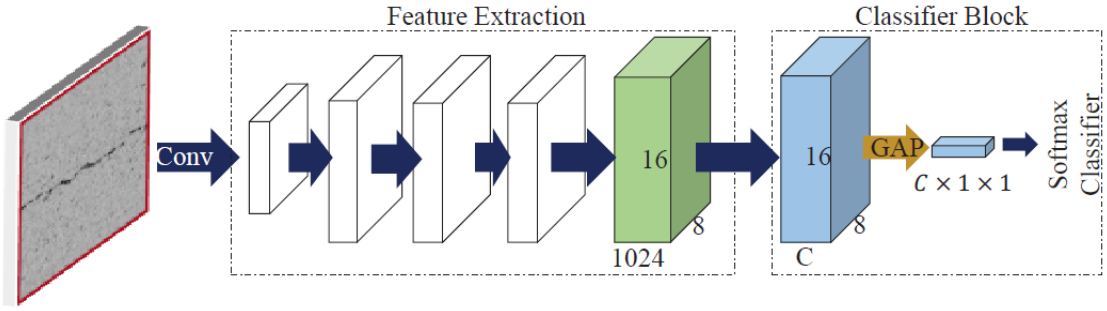


Figure 5.8: General structure of a CNN, used for classification. Modified from [87]

.

In Figure 5.8, C is the number of classes. The resulting feature maps in these networks are passed through a Global Average Pooling (GAP) layer [48]. GAP is a technique to spatially downsample feature maps to a single value, turning feature maps of size $H \times W$ to 1×1. It forces the feature maps to correspond to categories. The resulting vector of length C is fed into a softmax function in order to find probabilities for each class. GAP is applied in many modern networks in favour of several fully connected layers, which contain more parameters.

In our networks, however, we use the feature extraction base of the mentioned networks, but replace the classifier block architecture with our own. One aspect of GAP is that it removes spatial information from the feature maps, making the network spatially invariant, acting as a structural regularizer. However, our cracks are sampled in such a way, that the crack or non-crack is centred within the image. This information should be retained. This is why we omit the GAP layer and simply connect each value from the feature maps in the green block to an output sigmoid node. One downside to this approach is the number of parameters. The final layer now consists of $H \times W \times C$ parameters.

5.2.2 Input of the network

The standard spatial input dimensions of these networks are either 224 by 224 (ResNet and DenseNet) pixels or 299 by 299 pixels (Inception and Xception networks), both having three input channels (RGB). Since we only use the convolutional base of these networks, we are not restricted to this standard input shape. However, we chose to maintain the standard input shape and resize the patches to this fixed input shape. If the input patches were not resized, the amount of memory on the graphics processing unit (GPU) was not sufficient for larger patch sizes.

Resizing of larger patches is done using min-pooling, which retains the minimum value in a square area of pixels. This way, patterns that are darker than their surrounding pixels are preserved. The 25 mm patches, which are the only ones that are smaller than the input shape, are upsized using bilinear interpolation. Several resized images were visually analysed, and distortions that may appear due to the resizing process are not prohibitive in distinguishing cracks from non-cracks.

5.2.3 Transfer learning

An advantage to using these common architectures is pre-training or transfer learning. In transfer learning, we reuse a model that has been adapted to a specific task (in this case, general image classification on ImageNet) for a new problem (crack classification in range scans). Transfer learning is often used in machine learning and can improve generalization in the new setting as well as reducing the training time of models.

In our case, we use the weights of the convolutional base of our networks, which have been pre-trained on ImageNet, and add the final fully connected layer on top of it. The fully connected layer is not pre-trained, as this forms the final classification on ImageNet and uses a GAP step. This final layer is initialized using the technique proposed by He et al. [28], discussed in Chapter 4. In Chapter 6, we describe the effect of pre-training and investigate the generalization abilities and convergence of pre-trained networks.

5.2.4 Custom architecture

In addition to using standard networks, we developed a custom architecture. We hypothesize that crack classification requires fewer complex features than a 1000-class ImageNet classification. Since convolutional

layers learn increasingly complex features, our own custom network could be less deep and still obtain similar performance. A less deep network has fewer parameters, which reduces the risk of overfitting and is faster to train.

The state-of-the-art architectures shown before have been constructed over many iterations of experimentation, which has led to insights (such as residual connections and inception blocks) that we do not want to ignore. Therefore, we based our own architecture on the Inception-ResNet architecture shown in Figure 5.9.

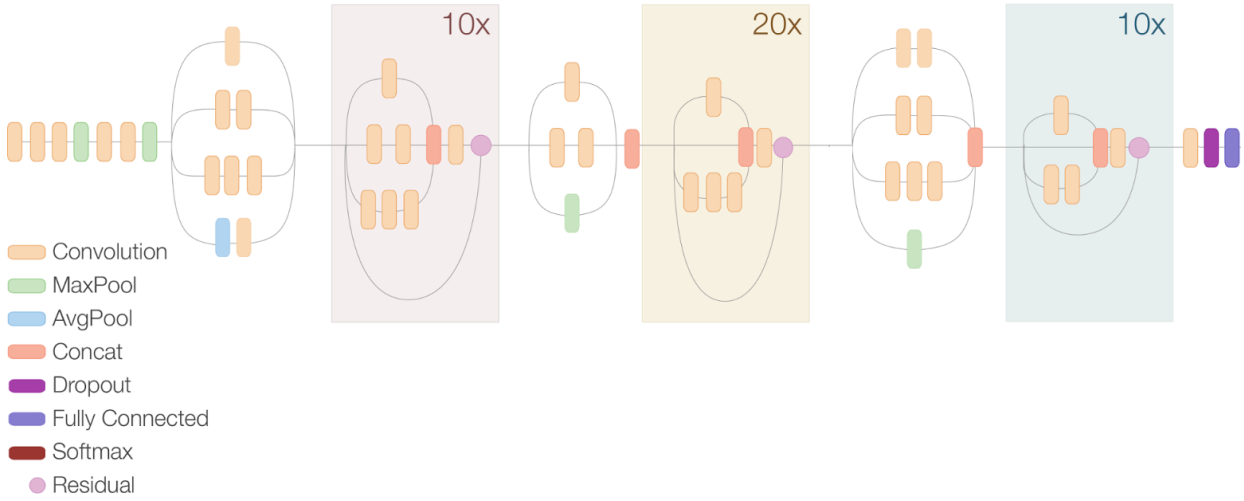


Figure 5.9: Condensed architecture overview of the Inception-ResNet architecture [75].

This architecture is very well suited for customization as blocks of layers are repeated multiple times, as shown within the three coloured blocks. Although the architecture has been discussed before in Chapter 4, we focus in on the blocks to show details in Figure 5.10.

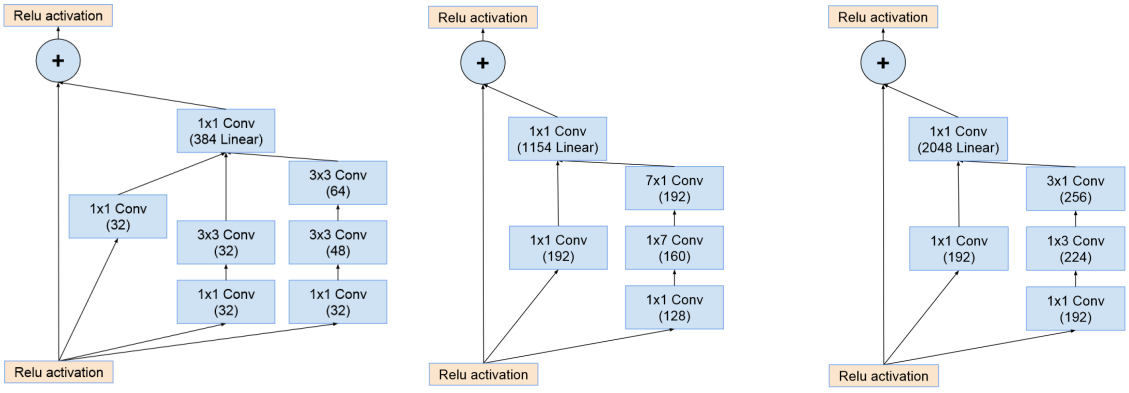


Figure 5.10: Inception-ResNet modules [75]. The blocks correspond (from left to right) with the coloured blocks from Figure 5.9

Figure 5.10 shows that these blocks (inception modules) contain a residual connection and several parallel convolutions of different scales. The convolutions are same-convolutions, which preserve the spatial input

dimensions. Output feature maps are stacked. Within the round brackets, the number of convolutions of that type are shown. 1×1 convolutions are used to reduce the number of output feature maps, which acts as a coordinate dependent transformation in the kernel space [48]. By using 1×1 convolutions, we project the input to a lower dimensional embedding, which reduces the amount of compute.

The layers between the shown blocks perform spatial downsampling of the data in order to reduce the number of parameters. We create our own architecture by experimenting with the number of layers (depth) of the network, aiming to reduce the number of parameters of the network.

5.2.5 An alternative approach: Classification using features

We built a classifier for the mentioned features using a gradient boosted tree algorithm [23]. Gradient boosted trees focus on reducing bias in a supervised setting by combining weak learners into a stronger one. The idea is that these decision trees complement one another, so ensembling leads to a more accurate model. These models can be written in the following form:

$$\hat{y} = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F} \tag{5.2}$$

where K is the number of trees and f_k is a function in $\mathcal{F}$. These functions f contain the tree structure and the leaf scores.

Trees can be trained differently, leading to different models, such as random forests and gradient boosted tree models (such as the one used).

In tree boosting, we additively (sequentially) add new trees to the model, fixing what is learned [20]:

$$\hat{y}_i^t = \sum_{k=1}^{K} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \tag{5.3}$$

In this way, the subsequent trees seek to predict how the original predictions different from the ground truth y_i, optimizing a loss function $\mathcal{L}$ by using the residuals of the prior trees. Regularization is also needed to prevent overfitting. The general form of the loss function becomes:

$$\mathcal{L}^{(t)} = \sum_i l(y_i, \hat{y}_i^t) + \sum_k \gamma T + \frac{1}{2}\lambda \left\| w \right\|^2 \tag{5.4}$$

Here l measures the difference between $\hat{y}_i$ and y_i, T represents the numbers of leafs in the tree and w represents the leaf weights.

In gradient tree boosting, we greedily add the function f_k that optimizes the loss function. To do so, all possible splits on all features must be computed. This can be quite computationally expensive, which is why we use a specific gradient boosted tree implementation: XGBoost [11]. This implementation incorporates an approximate split algorithm, among other optimizations (for example on the amount of disk I/O and CPU caching). This makes the XGBoost implementation a highly efficient one. We use the classifier on the

selected features in Table 5.2 and report on the results in Chapter 6. By doing this, we establish a fast baseline algorithm on pre-computed features, which is quicker to train than CNNs, which have to load image data.

In this chapter, we have introduced our data sampling techniques and model definitions (steps 1 and 2 in Figure 5.1). In the following chapter, we describe how these models are trained, including several experiments on the effect of pre-training and model architecture (step 3 in Figure 5.1). Furthermore, we discuss inference and the metrics used to evaluate performance (step 4 in Figure 5.1).

Chapter 6

Experiments

In this chapter, we assess the performance of the models by training them on the sampled patches and evaluate them on a separate holdout set. We first describe the experimental setup and introduce the evaluation metrics. The alternative XGBoost approach is evaluated to show the performance of a feature-based classifier. We then examine the experimental results of standard networks and the effect of pre-training. Furthermore, we describe an experimental evaluation performed on the depth of the Inception-ResNet architecture. We conclude this chapter with a comparison to human performance and briefly look into model visualization.

6.1 Experimental setup

6.1.1 Software

Our implementation is created in Python 3.6. We use Keras 2.2 as a deep-learning framework with Tensorflow 1.9 as a backend. CUDA 9.0 with cuDNN 7 is employed to provide GPU accelerated functionality.

6.1.2 Hardware

Training and testing of our models is done using a GeForce GTX 1080 TI 12GB graphics card. Since our training set does not directly fit in RAM, we wrote a custom data generator to pre-load batches of images into memory using 2 Intel(R) Core i7-6800K CPUs @ 3.40GHz. This is done to prevent data loading being a bottleneck for training.

6.1.3 Hyperparameters

For each patch size (250 mm, 500 mm, 750 mm, 1000 mm, 2500 mm), we train our models separately so that the effect of patch size can be shown. Our models are trained with the Nadam optimizer (Adam + Nesterov momentum), which is less sensitive to the initial learning rate (updates are estimated using a

running average of first and second momentum of the gradient). For batch size and initial learning rate, we follow the guidelines outlined in a recent work by L. Smith [69]. The recommendation from this work is to set the batch size as large as the GPU memory permits for the particular model. Larger batch sizes allow more stable learning with higher learning rates. For our largest model, this is 32, which we adopt for all models. Furthermore, larger batch sizes, decrease the number of copies from CPU to GPU memory (compared to smaller batch sizes), reducing total training time. As mentioned, the learning rate is one of the most important hyper-parameters to set. Setting the initial learning rate wrong results in a lower effective model capacity due to optimization failure [5]. Setting the learning rate too low might slow down learning with a factor proportional to the learning rate reduction and might even cause overfitting [68], but setting it too high will cause the training process to diverge. The initial learning rate is determined for each model by training the model for a few hundred iterations, while exponentially increasing the learning rate. We plot the learning rate vs the validation loss and look for the point just before the loss starts to become ragged or increase. Smith [68] recommends that the optimal initial learning rate, in that case, is found just before that point at around $\frac{1}{3}$-th or $\frac{1}{4}$-th of the learning rate found. The idea behind this is that the loss, early on in training, already provides clues about the full training process.

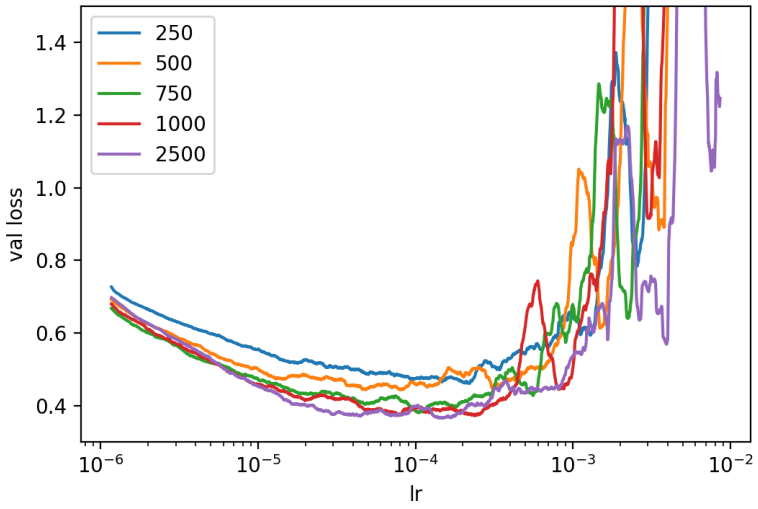


Figure 6.1: Exponentially increasing learning rate vs. validation loss for different patch sizes for the Inception-ResNet architecture.

Figure 6.1 shows an example of the learning rate - validation loss plots on the non-pre-trained Inception-ResNet model for different patch sizes. Here, a value between 10^{-4} and 10^{-5} would be a reasonable choice for an initial learning rate for all patch sizes. Most networks on this data set are trained at around this same learning rate. The optimal learning rate for the pre-trained networks is found to be around one order of magnitude smaller. We do not want the learning rate of pre-trained networks to be too high as the pre-

trained convolutional filter weights are most likely already relatively good; thus, we do not want to distort them too much.

6.1.4 Evaluation metrics

Our test set contains labelled cracks and non-cracks, randomly sampled across lane hectometres which are not contained within the training or validation set. While accuracy may give an intuitive interpretation of performance, it is too restrictive in this case.

Where our training and validation sets contain a balanced ratio between cracks and non-crack patterns, the real-life set is skewed towards non-crack patterns, leaving cracks as the minority class. Accuracy, which gives equal importance to both classes, does not represent our minority class (cracks) well. Instead, we would like to know the precision and the recall of the models before implementing an operational model. For clarity, we show the four metrics that contribute to our performance measure along with their definition in this specific case:

- **True positive (TP)**: a detected crack by the LCMS software, labelled by an inspector as crack and predicted by our model as crack
- **False positive (FP)**: a detected crack by the LCMS software, labelled by an inspector as non-crack and predicted by our model as crack
- **True negative (TN)**: a detected crack by the LCMS software, labelled by an inspector as non-crack and predicted by our model as non-crack
- **False negative (FN)**: a detected crack by the LCMS software, labelled by an inspector as crack and predicted by our model as non-crack

What is noticed is that all these metrics include only the detected cracks by the LCMS software. Cracks that might be present in the pavement, but are not detected by the LCMS data, are not labelled by the inspectors. From the metrics we determine precision P and recall R as follows:

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{6.1}$$

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{6.2}$$

Precision measures how many of the predicted cracks are actually cracks. Recall measures how many of the actual cracks our model captures. In most cases, there is a trade-off between precision and recall, which can be made by shifting the probability threshold of our models.

Since we do not know the preferred trade-off between the two metrics, models are compared over all thresholds by means of the average precision (AP). The AP-score [93] computes the mean of precisions

achieved at each probability threshold n. The increase in recall from the previous threshold $n-1$ is used to weigh the mean:

$$\text{AP} = \sum_n (R_n - R_{n-1}) P_n \tag{6.3}$$

AP is used instead of the area under the curve (AUC) since points on the Precision Recall (PR) curve should not be linearly interpolated, as the precision does not necessarily change linearly with recall.

In addition, one might argue that the trade-off between the false-positive rate and true-positive rate is informative. Such trade-offs are generally shown in a Receiver Operator Characteristic (ROC) curve. The ROC curve, however, similar to accuracy, can give an overly optimistic view of performance in the case of imbalanced data. This is because it remains the same, regardless of the prior probability of the positive class. Interestingly, it has been shown that if a curve dominates another in PR space, it also does so in ROC space, so one might think the AUC of the ROC curve could be optimized. Optimizing the AUC-ROC, however, does not optimize the AUC-PR (or AP) per se [18].

The choice of higher precision or recall depends on the cost of false positives and false negatives. At this moment, these costs are not known. If both recall and precision are equally important, the mean between them should be optimized; thus, for example, an F_1 score (harmonic mean between recall and precision) should be looked at. We leave this as an implementation choice for the decision makers at RWS and focus on optimizing AP.

Although the AP-score does not show a model strictly dominates another in PR-space, it is a good single evaluation metric of model performance along different threshold values.

6.2 Classification using features

Before the CNNs are tested, we start by testing the performance of a feature based approach. If this approach has similar or better performance at a lower computational cost, it is preferred over CNNs. Using the XGBoost model described in Chapter 5 on the features from Table 5.2 the same cracks and non-cracks as prepared for the CNN models are used. However, only the extracted features are used and not the range/image data itself. We trained several models, experimenting with a maximum tree depth between 1 and 5 and vary the number of trees between 2 and 4096 using exponential increments. Each model was trained with a default learning rate of 10^{-1} adapted from the sklearn Python library [64]. The obtained APs on the test set are shown in Table 6.1.

The AP-score seems to be quite low and as we will see later, considerably lower than that of the proposed CNNs. The results indicate that the combination of features do have some predictive power in classifying cracks vs. non-cracks, but they are not powerful enough to create a good prediction. Nonetheless, we want to see whether the chosen features are relevant to the classification. A reasonable setting, in this case, would be to choose a maximum tree depth of 3 and an ensemble of approximately 128 trees. Using this setting, we explore the feature importance. The feature importance is computed by calculating the amount that each

Table 6.1: XGBoost AP-score on test set for different tree depths and number of trees.

		number of trees											
		2	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**
max tree depth	**1**	.294	0.294	0.294	0.294	0.355	0.382	0.424	0.438	0.433	0.432	0.433	0.425
	2	.360	0.368	0.365	0.373	0.399	0.434	0.444	0.453	0.447	0.439	0.426	0.429
	3	.378	0.391	0.412	0.423	0.435	0.450	**0.455**	0.447	0.428	0.432	0.422	0.418
	4	.411	0.416	0.429	0.443	0.441	0.453	0.444	0.433	0.429	0.426	0.428	0.418
	5	.412	0.422	0.435	0.442	0.45	0.438	0.446	0.445	0.433	0.429	0.417	0.420

attribute split point improves the performance measure, weighted by the number of observations the node is responsible for [9]. We use the Gini index to select the split points. The feature importance is averaged across the 128 decision trees. The results are shown in Figure 6.2.

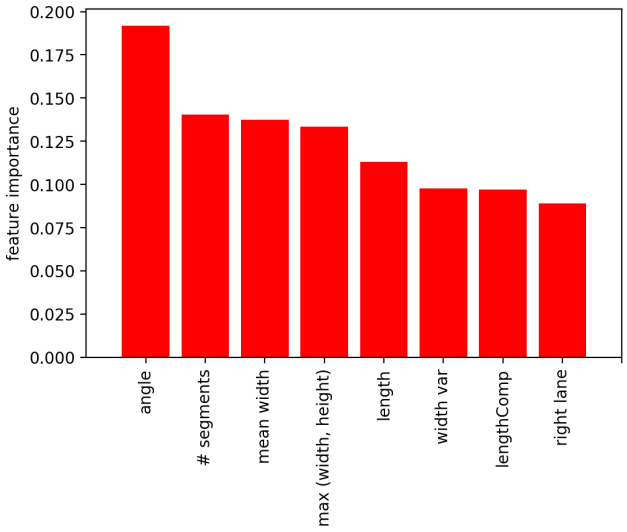


Figure 6.2: Feature importance for training a XGBoost model

In Figure 6.2, we see that most chosen attributes contain some distinctive power in combination with the other features i.e. their rank shows the importance in combination with the other features. Like seen earlier, in Figure 6.6, the length and width of the crack-like patterns are factors related to classification. However, the angle with the driving direction and the 'smoothness', indicated by how many independent segments the pattern is segmented as, appear to be even more relevant. We assume that a significantly powerful CNN might capture similar features and more relevant patterns.

6.3 Base convolutional models

We start by showing results on our base CNN models. The pre-trained CNN models are first trained for 10 epochs with a "frozen" convolutional base. This means that the weights of the convolutional base are fixed and only the fully connected layers are trained. If this is not done, the error signal, propagated through the entire network, will be too large and we will lose the previously learned features in the convolutional base. After these 10 epochs, the models are trained for 40 epochs, which allows some features to be fine-tuned to the problem specific task. The non-pre-trained networks are given the same number of epochs (10 + 40), which allowed all of the models to converge. The validation error did seem to go up at some point, indicating of overfitting. This is why, during training, after every epoch, we save the intermediate models. We then use the model, saved just before the generalization error starts to increase, for testing. This technique can be referred to as 'early stopping'. The resulting networks are evaluated on single patches of the test set. The performance on this test set is given in Table 6.2 and Figure 6.3.

Table 6.2: AP of several CNNs on the test set. Pre-trained indicates the convolutional base of the model, pre-trained on ImageNet, was used.

Model	**Pre-trained**	**Patch size**					**Parameters (millions)**
		250	500	750	1000	2500	
Densenet	yes	0.872	0.896	0.907	0.917	0.930	20,2
	no	0.852	0.893	0.906	0.918	0.928	
InceptionV3	yes	0.860	0.902	0.918	0.920	0.932	23,9
	no	0.861	0.903	0.899	0.915	0.927	
Inception-Resnet	yes	0.864	0.905	0.904	0.914	0.934	55,9
	no	0.855	0.892	0.898	0.905	0.919	
ResNet	yes	0.860	0.896	0.917	0.922	0.929	44,7
	no	0.844	0.886	0.903	0.909	0.926	
Xception	yes	0.854	0.896	0.911	0.918	0.929	22,9
	no	0.858	0.896	0.906	0.915	0.925	

There are several effects that can be observed from these results. When interpreting these results one must keep in mind that each experiment is only conducted once, and training the same network twice might result in a slightly different outcome due to the different initialization of weights (on non-pre-trained networks) and the stochasticity of training gradient-based algorithms on random mini-batches.

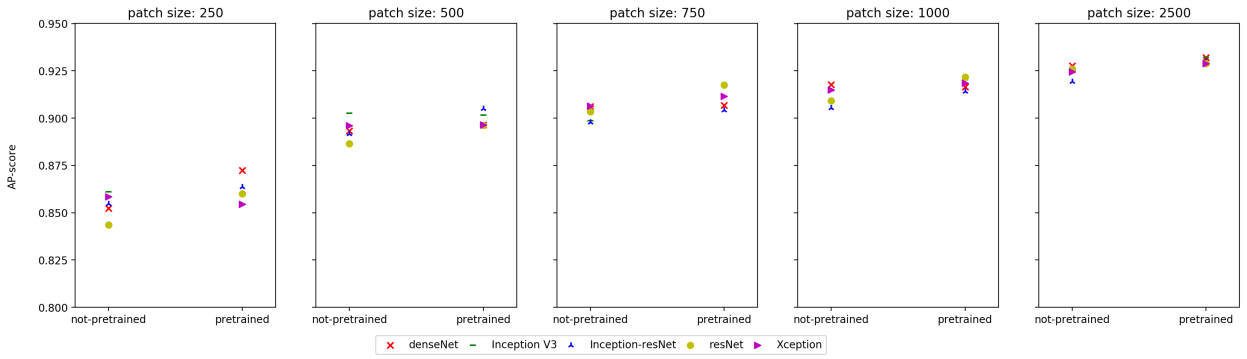


Figure 6.3: AP of several CNNs on the class-balanced holdout set. Pre-trained indicates the convolutional base of the model, pre-trained on ImageNet, was used.

6.3.1 The effect of pre-training

The first effect seen is higher performance when using a pre-trained network. Most pre-trained networks perform slightly better than their counterparts and converge to a lower loss. Here, the positive effect of initializing the convolutional kernel weights to those trained on ImageNet is seen. The networks are already trained to recognize several patterns and these, with some additional training, quickly generalize to the problem of crack classification. The pre-trained networks perform better than the non-pre-trained networks. This indicates that there might be features learned from ImageNet that cannot sufficiently be learned from our data set. A common statement in transfer learning is that the more similar two tasks are, the easier it becomes to transfer knowledge between them. Following that statement, a larger data set of the original problem should perform at least as well as a pre-trained (on another problem) network, even though it might take longer to train. Since this is not the case here, we believe that training on more data could close this gap and the current size of our training set is limitative.

To explore this hypothesis, we train the Inception-ResNet architecture on subsets of our total training set and show the effect of training set size in Figure 6.4. This shows a large increase in the non-pre-trained network in the first step where the subset size increases from approximately 10% to 20% of the training set. Fractions lower than 0.1 have not been tested, but will eventually produce results close to random. With increasing training set size, there are diminishing returns, but performance is shown to be approaching the pre-trained performance. Although these are single experiments, this does strengthen our hypothesis that more data might close the gap with the pre-trained network. If the figure is extrapolated to a larger fraction, the network that is trained from scratch should reach the same AP as the pre-trained network. For pre-trained network performance, we see that a basic amount of data is needed to fine-tune the network, but this amount need not be as large as training from scratch.

These results show that it is beneficial to use a pre-trained network, which is readily available. This will decrease the need for data and reduce the number of training iterations. The main limitation of using a pre-trained network is the use of fixed architectures, which could be overly complex for the task at hand.

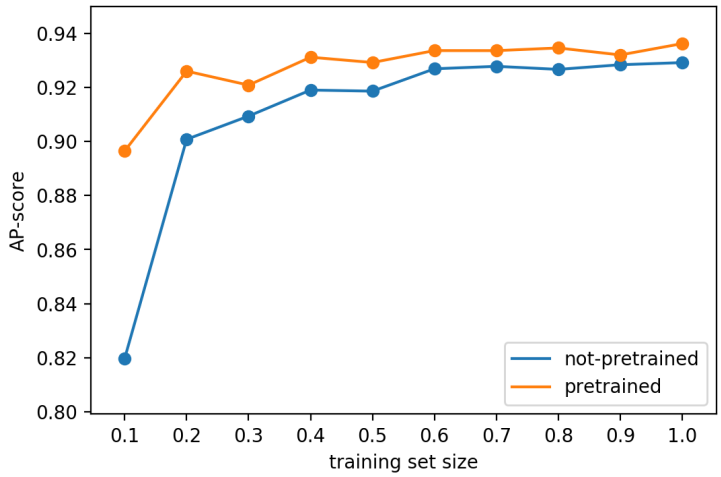


Figure 6.4: AP vs. training set size. Inception-ResNet was trained on the respective fractions of the training set size and AP results are reported on the test set.

6.3.2 The effect of patch size

Another interesting effect that can be seen is that of patch size. Although not very apparent in all models, there appears to be an increase in performance with patch size for both the pre-trained and non-pre-trained networks. Increasing the patch size increases the amount of information that is given to the network. More surrounding pavement might give the network more information on which to base the classification. Cracks (or non-crack patterns) might be close to other cracks (or non-crack patterns), which might also show into the patch. Surrounding pavement might also be indicative of these patterns. A structural failure of the surrounding pavement might already show indicators of cracks.

There are, however, also downsides to an increase in patch size. Larger patches might include patterns of the other class, in which case the convolutional filters (equivariant to translation), recognizing the other class, may respond highly. Fortunately, the equivariant translation does not directly lead to invariance of translation as we omit from using GAP and centre the main pattern in the patches. Not all invariance can be avoided, due to down-sampling inside the used networks. It will generally hold that, as long as the opposite class patterns do not appear too often in the patches, the fully connected layers will not respond too strongly to the non-centre regions of the feature maps.

Another downside to larger patches is the distortion caused be resizing the patches. We decided to work with the standard spatial input size of the networks, which is either 224×224 or 299×299. We visually inspected resized patches and confirmed that it is still possible to distinguish cracks from non-cracks, but information is nonetheless lost. Resizing a 2500×2500 mm patch to a 299×299 input shape keeps only around 1.5% of the pixels.

The main effect that increases the performance with larger patches is the portion of the crack-like patterns contained within the patch.

6.4 Custom Architecture

In Section 5.2.4, we introduced our custom architecture. A network with fewer parameters, based on the Inception-ResNet architecture. The main architecture consists of three sets of inception-residual blocks, separated by dimensionality reduction layers. The inception-residual blocks inside the sets are repeated as can be seen in Figure 5.9. The number of repeats are 10, 20 and 10 for the first, second and last set respectively. We devise an experiment on the reduction of depth in this architecture, by setting the value of repeats for the first set to k and keep the $1:2:1$ ratio of the architecture. For this original network $k = 10$. We experiment with smaller values for k to investigate how much the network depth can be reduced. The AP for different patch sizes and k values can be seen in Figure 6.5.

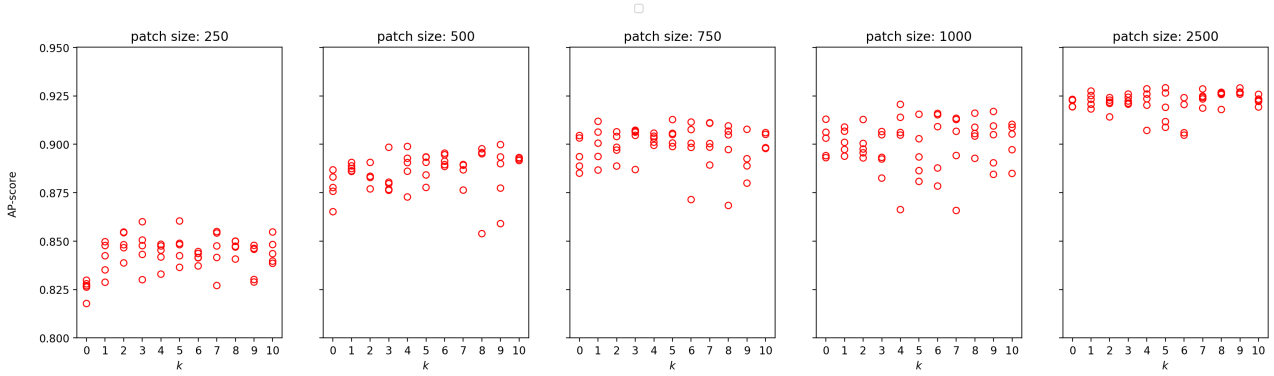


Figure 6.5: AP-score for different k values on different patch sizes

For every value of $k \in \{0, 10\}$ the model was trained five times and the model was tested on the same test set as previously discussed.

Since each model is trained five times, differences occurring from random initialization and stochastic gradient-based descent is seen. This effect similarly affects the architectures from Figure 6.3 as previously used.

Keeping this variation in mind, we observe that the value of k generally does not seem to affect the AP-score in this experiment. Even the $k = 0$ case, where all inception-residual blocks are removed, performs similarly to the (non-pre-trained) full architecture. In that case, only the dimensionality reduction blocks are left. This confirms our hypothesis that the network capacity of the state-of-the-art Inception-ResNet ($k = 10$) is actually not needed for a problem as such. For a smaller network residual connections are less needed as the gradient has less problems propagating through te network or vanishing. The AP-score of the smaller network also explains why all different SOTA models perform similarly well on this task; as they all have enough capacity to capture the complexity of this problem. When choosing a model, given the same performance, the smallest one that is sufficient should be used. This reduces the risk of overfitting and training time. The smaller models, however, do not appear to outperform the state-of-the-art models, which could indicate that the larger models overfit. We believe that the larger models are not overfitting

much because of regularization (batch norm effects and early stopping). The number of parameters for each k value architecture are shown in Table 6.3

Table 6.3: Number of parameters vs. model size, given by k

k	0	1	2	3	4	5	6	7	8	9	10
# parameters (m)	12,4	13,5	17,9	22,3	26,8	31,2	35,6	40,0	44,5	48,8	53,2

A logical follow-up question would be whether we could reduce the number of parameters even further. Given the current architecture, this would mean either removing the dimensionality reduction layers or altering the width of the network, defined by the number of filters per layer. One must be cautious, however, when removing dimensionality reduction layers. Counterintuitively, this could actually increase the memory footprint and number of floating point operations of the model. Even though removing these layers reduces the number of parameters, the resulting feature maps in subsequent layers are larger. Therefore, we leave the development of a custom smaller architecture specifically attuned to this problem to be a subject of further research. Furthermore, we suggest that it might be a reasonable approach to start a new small architecture bottom-up, starting with a very simple network and increasingly adding, for example, Inception-ResNet blocks.

For this work, we advise using the smallest tested model ($k = 0$) with the largest patch size as the operational model as this provides the highest AP. For an operational model, the performance per crack is not the only important factor; a combination of factors should determine whether maintenance should be prioritized. More severe cracks should be given more attention, as they prove to have the most risk for vehicular damage and further degradation of the asphalt. Crack severity is determined by the accumulated crack segment lengths and respective segment widths, normalized by the variance of the first principal component.

Figure 6.6 shows the relationship between AP-score and the total length, average width and severity of our operational model.

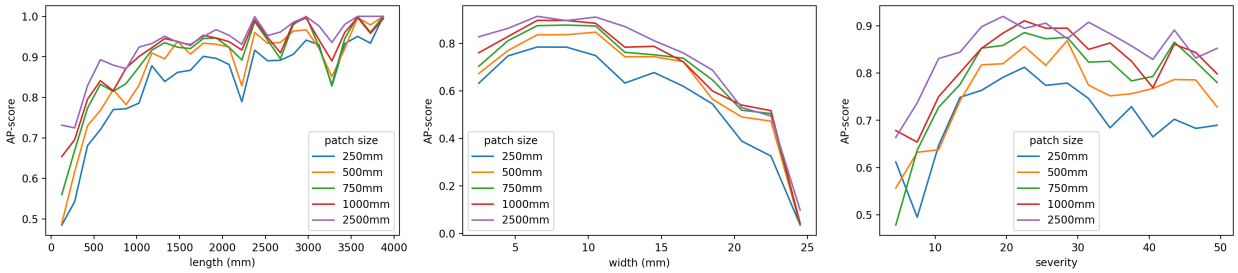


Figure 6.6: AP-score vs. pattern length (left), width (middle) and severity(right).

Generally, improved performance is seen on larger-length patterns. Longer cracks mean that a larger

fraction of the crack would be inside the patch, which might make them easier to classify. Somewhat smaller cracks, however, might be harder to classify for this reason. Interestingly enough, large patch sizes seem to perform better than smaller patch sizes on small patterns as well. We might think that smaller patches might capture finer details, but for all lengths, larger patch sizes generally seem to perform better. What might be the reason is that smaller crack-like patterns are often surrounded by other small crack-like patterns. These surrounding patterns are annotated individually, so they have a unique individual length, but are captured by the larger patch sizes.

Cracks with an average width lower than 2.5 mm were omitted. This is because the transversal scanning resolution is 4.7 mm. The minimal width in that direction is, therefore, 4.7 mm, wherein the driving direction it is 1 mm. This means that cracks with an average width smaller than roughly $(4.7 + 1)/2$ mm are very scarce. Where the length of the crack improves the performance, the average width of the crack seems to be related differently. There appears to be an optimum between around 6 mm and 12 mm. From the cumulative width distribution seen earlier in Figure 5.4, we see that this is the same range in which most crack widths in the total set lie. Here we seem to see the effect of the number of samples in the training set. The algorithm could be learning features related to the width of the cracks and does so from the given distribution.

The combined metric of length and width is shown in the severity metric. The AP-score appears to improve with a larger severity, but degrades slowly again for more severe cracks. We assume that this is the effect of less training examples for larger patterns, as is seen with the crack width. Since severity is linearly dependent on width and length, the positive effect of having a larger part of the pattern in the patch helps with classifying the pattern. The AP-score does not degrade as quickly as for the width, which gives us confidence that a good part of the more severe cracks can still be classified by our operational model.

6.5 Data and model ensembling

Up to this point, results have been based on predicting the label for a single patch, taken from a single crack, using a single model. In Chapter 5, the patch-sampling approach describes the sampling methodology for patches. It describes that any part of the crack-like pattern can be centred in the patch, thus sampling the same pattern different times, which may result in different crops. Especially for larger patterns and a smaller patch size, this may result in completely different images. These images, however, have the same label, i.e. the one that the inspector has given to the pattern as a whole. In this section, the effects of creating an ensemble of multiple crops per pattern are studied. In addition, the effect of applying multiple models is analyzed.

6.5.1 Single model, multiple patches

We start by using a single CNN (per patch size) and determine whether adding more patches per pattern is beneficial. For each pattern in the previously used test set, we sample an additional four patches (crops),

centring a random part of the pattern. A simple average over the predicted sigmoid scores is used to compute the AP-score. The effects of adding more patches on the AP-score is shown in Figure 6.7.

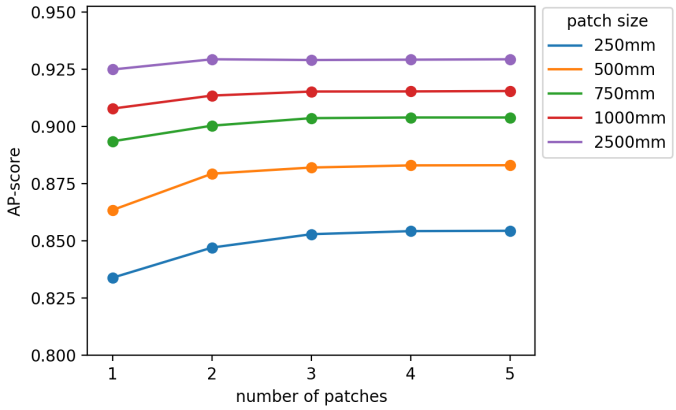


Figure 6.7: Number of patches vs AP-score

For this experiment, we used the smallest models ($k = 0$) from earlier. In the case where the number of patches is equal to 1, we have the same AP-score as seen earlier. What can be observed is that, for all patch sizes, an improvement in AP-score is seen when adding more patches.

This effect is mainly apparent in the smaller patch sizes. There, a clear improvement is seen when going from a single patch to two patches. For the larger patch sizes ($\geq$ 750 mm), a small improvement is noted. In the case of smaller patch sizes, adding more patches, on average, adds more information. The average pattern span is approximately 400 mm and almost all patterns are smaller than 2000 mm (recall Figure 5.4).

Since the sampling process centres a part of the pattern in the middle of the patch, any crack-like pattern smaller than half the patch size will be captured fully within the patch. Using the simple assumption, that most information used for classification is contained within the pattern and not the surrounding pavement or nearby patterns, adding more patches, in this case, will not improve the AP-score. The larger the patch size, the more crack-like patterns will be contained fully or almost fully within the patch. Different crops, in that case, will look very similar.

Patterns that are larger than half the patch size might only be party captured by the patch. For the smaller patch sizes ($\leq$ 500 mm), a second patch adds more information in many cases. Given that the average crack-like pattern is 400 mm, a 250 mm patch size, in the best case, will only capture 250/400 mm of the crack-like pattern, the worst case 125/400 mm and on average approximately 211 mm (see Equation 6.4).

$$\text{mean\_overlap}(c, p) = \begin{cases} 1 & , \text{if } c \leq \frac{1}{2} \cdot p \\ \left(\frac{3p}{4c} + \frac{c-p}{c} \right) \frac{p}{c}, & , \text{if } c > \frac{1}{2} \cdot p \end{cases} \tag{6.4}$$

where p is the patch size and c is the size spanned by the crack.

We can also plot this mean average overlap of the pattern covered by the patch (Figure 6.8).

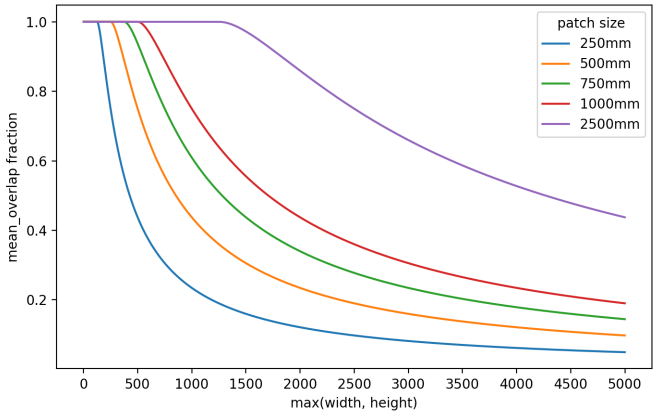


Figure 6.8: Average overlap fraction vs. the maximum size spanned by the crack

Figure 6.8 illustrates that the average fraction, covered by the patch, quickly declines for smaller patch sizes. In that case, adding more patches gives the model more parts of the crack-like pattern.

For the larger patch sizes, especially 2500 mm, adding more patches only starts to make sense for patterns larger than 1500 mm. Since only a small fraction of patterns is larger than that, the increased AP-score on multi-patch ensembling is small.

An interesting observation is that, even though the ensemble of multiple smaller patches might cover the same area of the pattern as a large patch, the performance of the ensemble never outperforms even a single larger patch size. This enforces our idea that cracks might be in the vicinity of other cracks, which are also captured by a larger patch size, strengthening the prediction, whereas these patterns might not be included in a smaller patch size.

Instead of applying the mean over probabilities, the maximum predicted probability could be used. A higher probability of a class might mean that the network is more confident in that prediction, so the highest probability could be a better predictor than the single probabilities. We have compared mean averaging and max averaging for all patch sizes versus single patch performance, and used the mean and max of five patches per pattern.

Figure 6.9 shows an increase in AP-score for both mean and max probability ensembling. Mean ensembling appears to perform slightly better than max ensembling in all cases.

We conclude from this section that there is added value in using multiple patches per pattern, specifically for smaller patches. However, the resulting AP-score from their ensemble does not surpass the AP-score obtained by larger patch sizes. Our advice for an operational model remains to use the largest patch size. Depending on the overhead of sampling a patch, a single patch, in that case, would suffice.

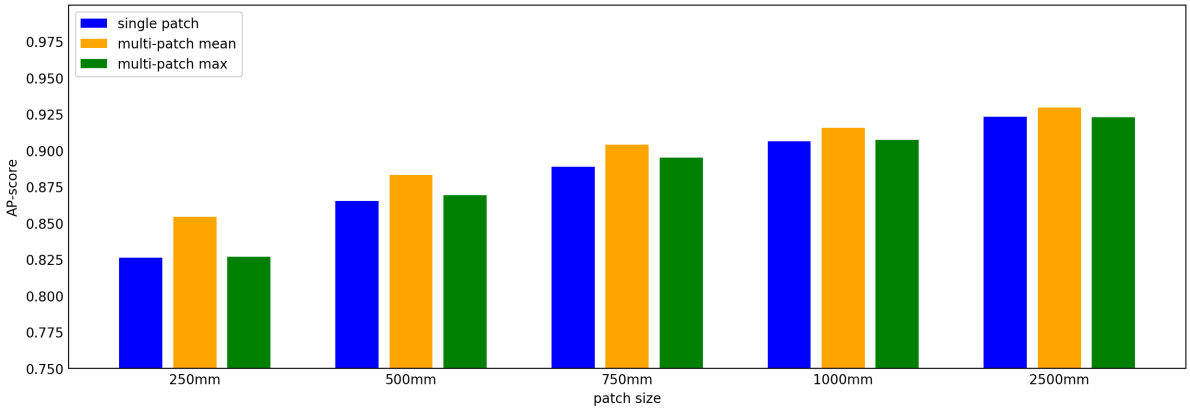


Figure 6.9: Five-patch single model ensemble

6.5.2 Multiple models, single patch

Instead of averaging different crops of the same pattern, the same crop can also be shown to different models. Using different models on the same data may improve results. Using multiple models, in general, can obtain better predictive performance than single models in the ensemble [62]. Since different classifiers may make errors on different parts of the input space, the resulting ensemble reduces the variance (specifically in models with low correlation).

Since each custom network architecture was trained five times, each individual model can be used to create a prediction on the patches and the respective predictions ensembled in a similar manner to the patch ensemble. These results are shown in Figure 6.10.

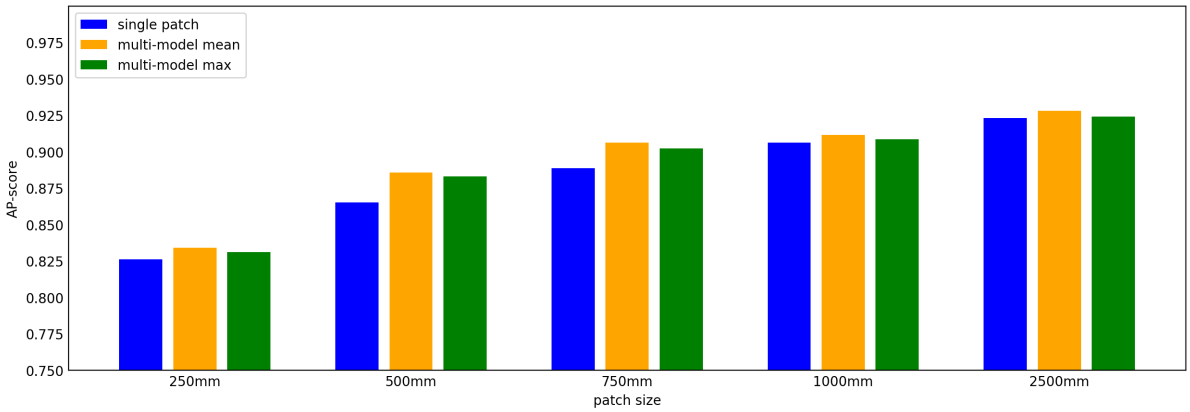


Figure 6.10: Multi-model, single-patch model ensemble

Similar to the patch ensembling approach, the mean of predicted probabilities performs slightly better than their max. One explanation of why max probability might not be as good as the mean probability is that a high probability might be caused by overfitting of the data point. In that case, the model is overconfident (represented by a high probability), but might actually be wrong.

For the operational usage of the system, we would not advise using multiple models. The benefit of their ensembles is small, and the overhead of loading each model and creating predictions is significant.

6.5.3 Multiple models, multiple patches

As a final test, we combine the two ensembling methods. Because the mean probability approach works best for both patch and model ensembling, we test their combined approach. First, each individual model predicts five patches for each pattern in the created test set. We then use the mean of these predicted probabilities for each patch. Having an average probability for each patch from five models, the mean is then again taken over

the five patches to form a combined probability for a crack-like pattern. The combined results are shown in Figure 6.11.

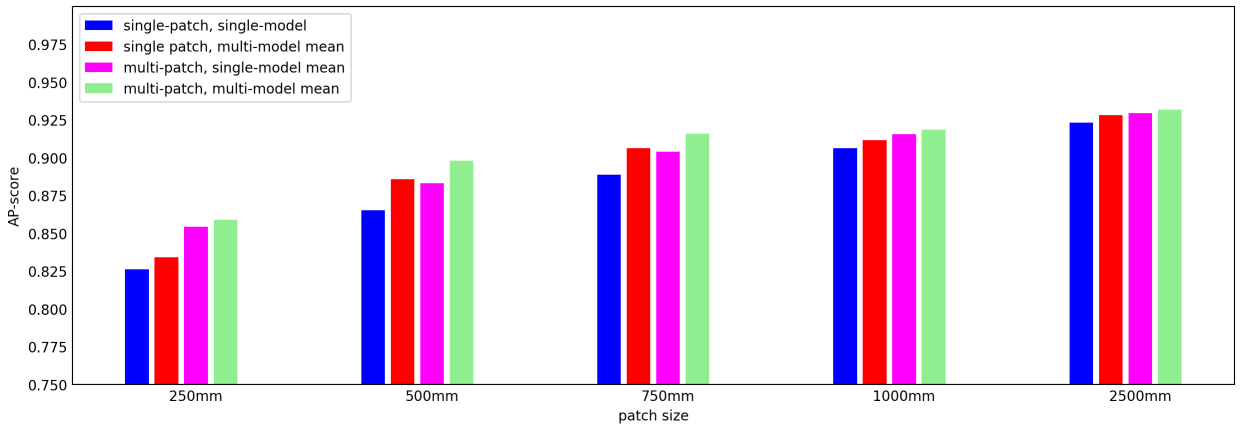


Figure 6.11: Single patch, multi-patch ensemble, multi-model ensemble and their combination

In this figure, the different ensembling methods are shown side by side. We cannot say that one of the two ensembling methods is better than the other over all patch sizes. Ensembling both models and patches, however, does give another small improvement in performance, compared to multi-model ensembles or multi-patch single-model ensembles. The improvement is, however, modest.

More advanced ensembles of models could also be used, but we refrain from doing so here. A more advanced ensembling method could add weights to different inputs by using meta-learning, also referred to as model stacking. Since our results show only moderate improvements, ensembling models would most likely not weigh up to the cost involved. Therefore, we maintain our earlier recommendations that a single model with a larger patch size is preferred.

6.6 Human performance

During our experiments, we noticed a different ratio between cracks and non-cracks for different inspectors. While there is a difference between the amount of traffic on different roads and some inspectors might have been assigned busier roads (which may have more cracks), there might also be a different interpretation of cracks between inspectors.

To further investigate this issue, we trained our model on a separate subset with a stratified sampling approach, keeping the ratio of inspectors uniformly distributed. We then tested the accuracy of a model on the labels given by different inspectors. Since the model is trained on the ensemble of inspectors, we can see that some inspectors deviate more from the ensemble. As we do not know the ground-truth without road inspections, we do not wish to state whether this is undesirable performance, but simply observe that differences were present.

Like any other image classification task, a model can only be as good as its labels. The irreducible error arising from the misinterpretation of inspectors on certain labels weighs on the lowest attainable error. Different interpretations and labelling techniques cause an upper limit to the attainable accuracy.

To gain insight into this maximal attainable performance, a subset of patches containing 100 labelled cracks and 100 labelled non-cracks was created using the 2500 patch size. Using a team of three experts, we verified the original label by inspecting the patches. The subset was labelled using the consensus of these experts. This consensus should be closer to the ground-truth than the individual inspector's labels. Using the new labels, several differences with the original label were observed. We refer to these different interpretations as 'debatable' cases. We tested all standard non-pre-trained models and our custom architecture with $k = 0$ on these subsets as well. The following results were obtained:

- Out of 100 labelled as crack:
  - 29 cases were debatable
  - avg. original accuracy on labels: 88/100
  - avg. accuracy on corrected labels: 92/100
- Out of 100 labelled as non-crack patterns:
  - 17 cases were debatable
  - avg. original accuracy on labels: 78/100
  - avg. accuracy on corrected labels: 88/100

We averaged accuracies of the different tested models, but note that they did not vary greatly in their predictions ($\pm 2\%$ was observed). While this is a very small subset of the actual data, this does raise some questions on the validity of the original labels. Different inspectors might have varying interpretations and

assign other labels to the patterns. This means that our models inherently have a limit on the maximum achievable performance.

What we can conclude, even from this small set, is that the models' performance are at least close or up-to-par with the inspector's labelling approach. The CNNs performed well on obvious cases, whereas in some cases, the inspector label was most likely wrong by mistake. Less obvious cases, such as severe ravelling that starts to crack, are more subject to debate. The individual inspector, in that case, can use his or her domain knowledge and make a better decision.

6.7 A look into mis-classification

To gain a better interpretation on which mistakes the model made, the mis-classifications were inspected. The mis-classifications are either false positives or false negatives. In Figure 6.12, we show some of the false negatives, which are predicted as not a crack, but are in fact a crack.

What is observed and also seen in this figure is that roughly three cases of false negatives exist. The first case (top) is actually severe ravelling that started to crack. Some inspectors might label this as not a crack, but at some point this raveling may start to form some cracking. The model did not find enough evidence to predict it as a crack. The second case is very fine cracking. To detect these cracks by simply looking at the range scan is difficult and to the model, looking at the entire patch, there might be more evidence against a crack. The third case is an interesting one and shows a limitation to our approach. The patches are sampled, with any random part of a pattern in the centre of the patch. In this case, the crack (blue) was sampled, and the patch was labelled as crack. However, close to the centre, a non-crack is present (green). The model decides this is not a crack as there is more evidence for a non-crack. This is inherently related to spatial invariance in the network, even though the GAP layer was replaced by a FC one. The spatial invariance, in this case, is most likely caused by downsampling and using this approach, can not be avoided.

Typically, decisions of a neural network are not as intuitive or understandable as other types of models. Some visualization techniques, however, do exist to create some better understanding. The technique we employ is referred to as guided backpropagation [72]. The idea behind guided backpropagation is that neurons detect specific image features. If we are interested in what image features the neuron detects, we can find it by forward passing an image through the network and then set all negative gradients to zero when backpropagating the gradient. Since we are using a sigmoid output unit, the returned mask only contains features that lead to a positive decision for a crack in our case. Using this approach, the false positive detections are inspected. Some common mistakes are shown in Figure 6.13.

Here the top image shows a similar picture to the top false negative detection. Severe ravelling is a difficult case and here there the model finds some features that start to resemble a crack. This time it outweighs the negative evidence. For an operational model, the guided backpropagation maps, might actually be a guidance to inspectors to further look into these cases and decide whether parts of these ravelled parts of

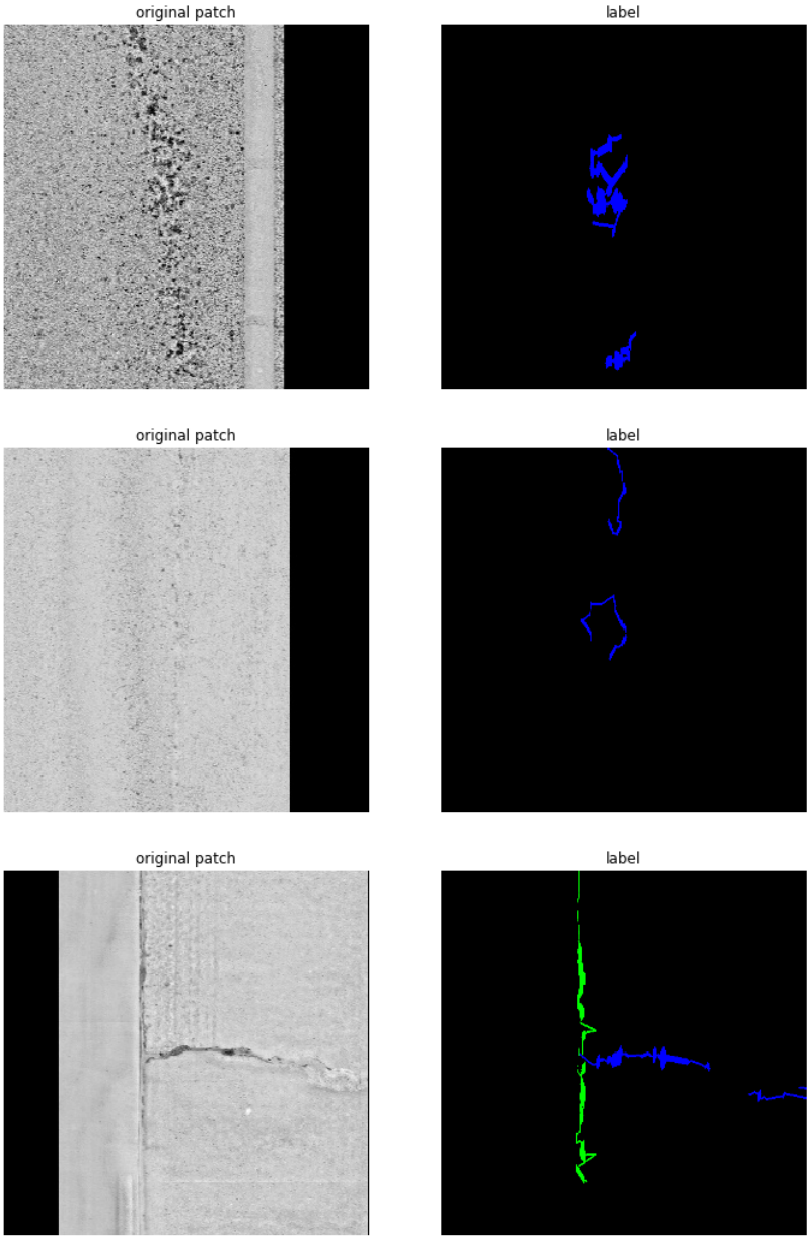


Figure 6.12: False negative detections. On the right is the annotated crack present in the left image. Blue is a crack, where green is a non-crack

asphalt start to form cracks. The bottom image shown actually is part of a counting loop, which is seamed with a sealer. The sealer is labeled as not a crack. However, some cracks actually start to appear in the sealant here. One could argue that this patch should be labelled as a crack.

We see that the algorithm correctly predicted most cases, but some mistakes inherently follow from the patch based approach. A per-pixel classification would be more accurate, but was not possible in this case. The guided backpropagation, does however highlight important parts of the image, used in the prediction of

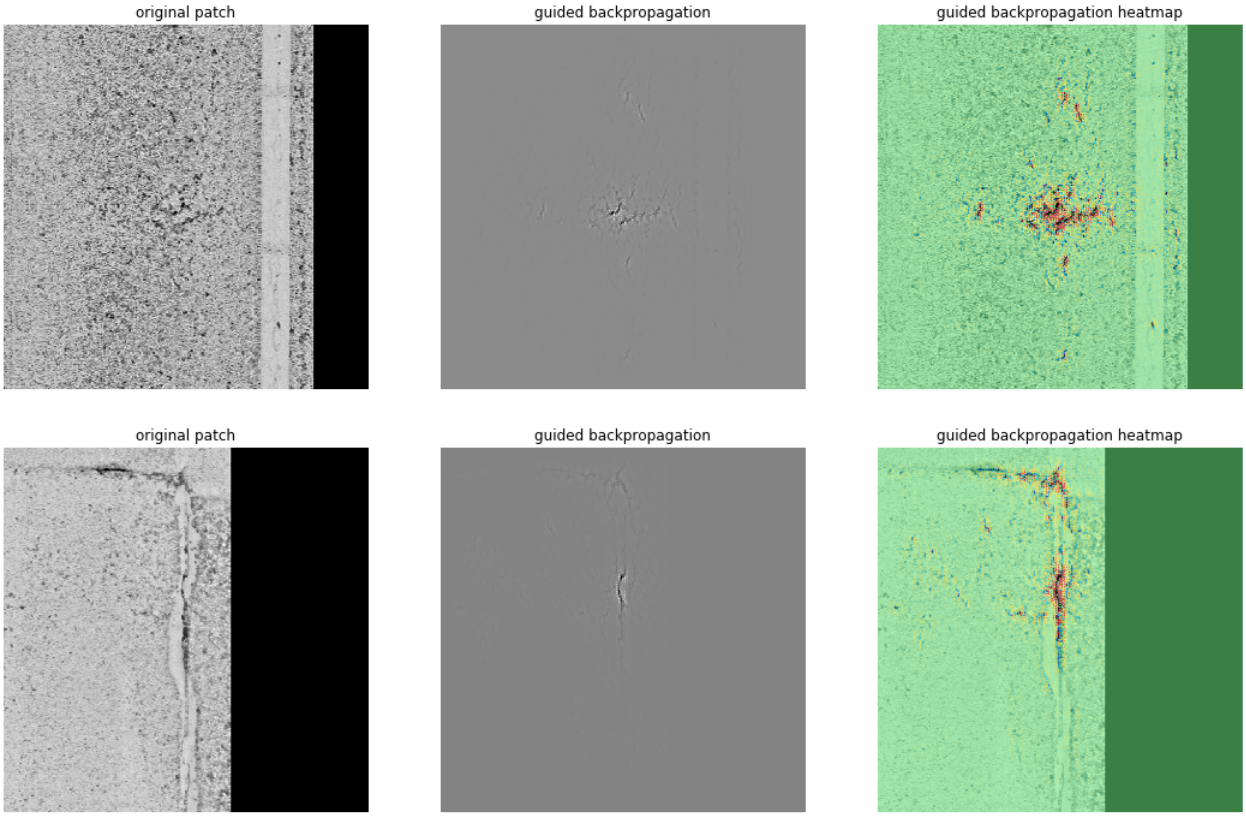


Figure 6.13: False positive detections. Left is the original patch, middle is the guided backpropagation mask, and right is an overlay of the two.

the positive class. Extending on this, by using a softmax with logits for both classes, such a gradient map could be made for both classes.

Chapter 7

Discussion and Outlook

In this thesis, the recognition of pavement distresses in 3D surface measurements is studied. A literature review of the field of pavement distress detection showed that many works focus on recognition of pavement distress but we noticed no further classification of distresses. One of the main distresses studied are cracks; however, the detection methods that are currently used also identify other distresses, such as rim marks, counting loops and road expansion slits. In addition, a literature study on convolutional neural networks showed these models to be a viable approach for an improved classification of cracks.

The primary contribution of this work is the extension of the pavement maintenance pipeline employed by many road maintenance agencies, by classifying detected cracks into either actual cracks or crack-like patterns. Using hand-labelled patterns by Rijkswaterstaat, a supervised CNN was trained to recognize cracks with an average precision of up to 0.93. To achieve this, inspiration was taken from recent developments in general image classification using neural networks. We recognized that these state-of-the-art networks could be reduced in complexity to improve training and inference time. This makes the proposed model usable to analyze data from the entire Dutch road network within the current maintenance pipeline. In addition, pre-training neural networks on general image classification data was shown to be useful in reducing training time, and also showed improved performance compared with networks that were not pre-trained. Furthermore, a feature-based classification method was briefly explored; however, the lower performance illustrated the supremacy of a CNNs for this specific task. When comparing the performance to human labelling, using a small verification set, we noticed that some cases are difficult to classify and open to interpretation. However, in this specific set, the proposed model had similar performance to the inspector.

7.0.1 Limitations

There are several limitations to this work. Real-life data is inherently noisy. By using a moving capturing device for the 3D surface measurements, processing algorithms to remove measurement artefacts and compensate for road unevenness, and algorithms to map the data to images, some information is lost. However,

this is not necessarily negative, as this process also removes unneeded information.

This study was also limited by the performance of the pattern detection algorithm. Since this is an external algorithm, we could only guess at the inner workings. We therefore could not say anything about the accuracy of the detector and the number of missed cracks. In quite some cases, however, we found incomplete or missing detections.

Unfortunately, also the annotations were only a course description of the location of the pattern in the image, since they were described by a vector of interconnected coordinates instead of the actual pixels. Moreover, some of the detections were misaligned with their respective patterns, being above or below them.

These limitations prevented us from developing a pixel-wise classification or a so-called detection algorithm. Moreover, a neural network trained on this data might have an upper limit corresponding to the performance of the detection software.

The data was also limited by the quality of the labels given by inspectors. By inspecting 200 cases, we found that incorrect labels were given in some cases. Given enough data, the models could become robust against this. However, on more debatable cases, a large inconsistency between given labels for similar patterns might cause problems.

7.0.2 Future work

Using the proposed model, a follow-up study will focus on the implementation implications for the complete Dutch road network. Whether and when the model should be used as an assisting tool for inspectors will be researched. To improve the model's performance, new data will be captured at a higher resolution. The detection step that precedes classification has been improved in another study parallel to this work, and its effects on classification will be the subject of further research as well.

Another direction of research could be focused on completely different architectures of neural networks. Since nearby parts of pavement have a strong correlation, the data is somewhat sequential. This would mean that a recurrent architecture, which subsequently processes rows of pavement measurements, could be a promising architecture. Recurrent architectures might predict in which direction and how a crack might expand, something that could be added as a measure of severity and time-to-maintenance prediction.

Finally, since creating ground-truth data is a laboursome task, developments in the field of generative adversarial networks (GANs) are promising as well. GANs could generate artificial but authentic patterns, which may be used to create accurately labelled synthetic data.

7.0.3 Recommendations

With these findings at hand, we advise to use the developed custom model as a pre-labelling approach in the current maintenance planning pipeline. This may speed up the time spent on the classification of cracks. By shifting the probability threshold of the model, emphasis can be put on either higher precision or recall

by the inspector. This is dependent on the cost of both false negatives and false positives, which might be different for different roads or pavement types.

Another recommendation is to provide feedback by relating detections from within the range scans to the actual pavement. Although very labour intensive, this may provide with actual ground truth labels, which could improve classification of the models.

We advise against using the system as a complete replacement for inspector labelling at this time. Whereas in most cases the model will perform well, debatable cases should still be reviewed by a domain expert. Furthermore, the exact performance of the domain experts is not determined by a large scale test. We can therefore not assume that the model outperforms the domain expert for now, although our smaller test indicated that results are up to par.

Acknowledgements

I would like to thank the Intelligent Imaging group from TNO. Thomas van Koppen, with whom I had many fruitful discussions, guided me throughout my internship and kept me from straying away too far from the project goals. Additionally, the weekly discussions with the deep learning team inspired me to dive further into literature and obtain a broad view of the research area.

Furthermore, I would like to thank Wojtek Kowalczyk and Thomas Bäck from LIACS for supervising me throughout this thesis and earlier projects during my masters. They have inspired me to work on interesting practical problems, whilst keeping a research oriented view. Their constructive feedback has always motivated me to continue improving my work.

Bibliography

[1] W. v. Aalst, G. Derksen, P.-P. Schackmann, P. Paffen, F. Bouman, and W. v. Ooijen. Automated ravelling inspection and maintenance planning on porous asphalt in the Netherlands. In *International Symposium Non-Destructive Testing in Civil Engineering (NDT-CE)*, 2015.

[2] R. Amhaz, S. Chambon, J. Idier, and V. Baltazart. A new minimal path selection algorithm for automatic crack detection on pavement images. In *International Conference on Image Processing (ICIP)*, pages 788–792. IEEE, 2014.

[3] M. Avila, S. Begot, F. Duculty, and T. S. Nguyen. 2d image based road pavement crack detection by calculating minimal paths and dynamic programming. In *International Conference on Image Processing (ICIP)*, pages 783–787. IEEE, 2014.

[4] A. Ayenu-Prah and N. Attoh-Okine. Evaluating pavement cracks with bidimensional empirical mode decomposition. *EURASIP Journal on Advances in Signal Processing*, 2008(1):861701, 2008.

[5] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[6] Y. Bengio, P. Frasconi, and P. Simard. The problem of learning long-term dependencies in recurrent networks. In *International Conference on Neural Networks (ICNN)*, pages 1183–1188. IEEE, 1993.

[7] L. Blier. A brief report of the heuritech deep learning meetup. `https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/`. Accessed: 2019-01-10.

[8] J. Bray, B. Verma, X. Li, and W. He. A neural network based technique for automatic classification of road cracks. In *International Joint Conference on Neural Network (IJCNN)*. IEEE, July 2006.

[9] J. Brownlee. Feature importance and feature selection with XGBoost in Python. `https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/`, 2016. Accessed: 2018-03-02.

[10] J. Canny. A computational approach to edge detection. *Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986.

[11] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 785–794, New York, NY, USA, 2016. ACM.

[12] H. Cheng, X. Jiang, J. Li, and C. Glazier. Automated real-time pavement distress analysis. *Transportation Research Record*, 1655:55–64, Jan 1999.

[13] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807. IEEE, 2017.

[14] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015. Accessed: 2018-11-01.

[15] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units. In *International Conference on Learning Representations (ICLR)*, 2016.

[16] A. Cuhadar, K. Shalaby, and S. Tasdoken. Automatic segmentation of pavement condition data using wavelet transform. In *Canadian Conference on Electrical and Computer Engineering (CCECE)*, volume 2, pages 1009–1014. IEEE, 2002.

[17] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[18] J. Davis and M. Goadrich. The relationship between precision-recall and ROC curves. In *International conference on Machine learning (ICML*, pages 233–240. ACM, 2006.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[20] X. Developers. Introduction to boosted trees. `https://xgboost.readthedocs.io/en/latest/tutorials/model.htm`, 2018. Accessed: 2019-03-02.

[21] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[22] T. Dozat. Incorporating Nesterov momentum into adam. In *International Conference on Learning Representations (ICLR)*, 2016.

[23] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Ann. Statist.*, 28(2):337–407, 04 2000.

[24] G. Ghiasi, T.-Y. Lin, and Q. V. Le. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 10727–10737. Curran Associates, Inc., 2018.

[25] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics (AISTATS)*, pages 249–256, 2010.

[26] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016.

[27] I. J. Goodfellow and O. Vinyals. Qualitatively characterizing neural network optimization problems. *CoRR*, abs/1412.6544, 2015.

[28] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on Imagenet classification. In *International conference on computer vision (ICCV)*, pages 1026–1034. IEEE, 2015.

[29] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision (ECCV)*, pages 630–645. Springer, 2016.

[30] Hebb and D. O. *The organization of behavior; a neuropsychological theory.* Wiley, 1949.

[31] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269. IEEE, July 2017.

[32] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[33] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[34] C. Jiang. *A crack detection and diagnosis methodology for automated pavement condition evaluation.* PhD thesis, Georgia Institute of Technology, 2015.

[35] G. Kang, J. Li, and D. Tao. Shakeout: A new regularized deep neural network training scheme. In *Conference on Artificial Intelligence (AAAI)*, pages 1751–1757, 2016.

[36] A. Karparthy. Convolutional neural networks (cs231n). `http://cs231n.github.io/convolutional-networks/`. Accessed: 2019-02-01.

[37] V. Kaul, A. Yezzi, and Y. Tsai. Detecting curves with unknown endpoints and arbitrary topology using minimal paths. *Transactions on Pattern Analysis and Machine Intelligence*, 34(10):1952–1965, Oct 2012.

[38] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*, 2017.

[39] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2014.

[40] K. Kirschke and S. Velinsky. Histogram-based approach for automated pavement-crack sensing. *Journal of Transportation Engineering*, 118(5):700–710, 1992.

[41] J. Kittler, J. Illingworth, and J. Föglein. Threshold selection based on a simple image statistic. *Computer vision, graphics, and image processing*, 30(2):125–147, 1985.

[42] H. N. Koutsopoulos, I. E. Sanhouri, and A. B. Downey. Analysis of segmentation algorithms for pavement distress images. *Journal of transportation engineering*, 119(6):868–888, 1993.

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105. Curran Associates, Inc., 2012.

[44] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[45] J. Laurent, J. F. Hébert, D. Lefebvre, and Y. Savard. High-speed network level road texture evaluation using 1mm resolution transverse 3d profiling sensors using a digital sand patch model. In *International Conference on Maintenance and Rehabilitation of Pavements and Technological Control*, 2012.

[46] B. J. Lee and H. D. Lee. Position-invariant neural network for digital pavement crack analysis. *Computer-Aided Civil and Infrastructure Engineering*, 19(2):105–118, 2004.

[47] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 6389–6399, 2018.

[48] M. Lin, Q. Chen, and S. Yan. Network in network. In *International Conference on Learning Representations (ICLR)*, 2014.

[49] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[50] M. Loos. Automatic pavement type classication using fully convolutional neural networks. Master's thesis, Artificial Intelligence, University of Amsterdam, 2016.

[51] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International conference on machine learning (ICML)*, 2013.

[52] S. Mathavan, M. Rahman, M. Stonecliffe-Jones, and K. Kamal. Pavement raveling detection and measurement from synchronized intensity and range images. *Transportation Research Record*, 2457(1):3–11, 2014.

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

[54] S. McRobbie, A. Wright, P. Iaquinta, J.and Scott, C. Christie, and D. James. Developing new methods for the automatic measurement of ravelling at traffic speed. In *International Conference on Asphalt Pavements*, 2010.

[55] M. Minsky and S. Papert. *Perceptrons*. MIT press, 1969.

[56] F. M. Nejad and H. Zakeri. An optimum feature extraction method based on wavelet–Radon transform and dynamic neural network for pavement distress classification. *Expert Systems with Applications*, 38(8):9442–9460, 2011.

[57] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate o $(1/\hat{k2})$. In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547, 1983.

[58] M. A. Nielsen. Neural networks and deep learning, 2018.

[59] U. of Washington. Pavement interactive. `https://www.pavementinteractive.org`. Accessed: 2018-10-21.

[60] H. Oh, N. W. Garrick, and L. E. Achenie. Segmentation algorithm using iterative clipping for processing noisy pavement images. In *Imaging Technologies: Techniques and Applications in Civil Engineering*, 1998.

[61] H. Oliveira and P. L. Correia. Automatic road crack detection and characterization. *Transactions on Intelligent Transportation Systems*, 14(1):155–168, March 2013.

[62] D. W. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research (JAIR)*, 11:169–198, 1999.

[63] N. Otsu. A threshold selection method from gray-level histograms. *Transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[64] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[65] F. Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[66] T. Saar and O. Talvik. Automatic asphalt pavement crack detection and classification using neural networks. In *Biennial Baltic Electronics Conference (BEC)*, pages 345–348. IEEE, 2010.

[67] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[68] L. N. Smith. Cyclical learning rates for training neural networks. *Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

[69] L. N. Smith. A disciplined approach to neural network hyper-parameters. *CoRR*, abs/1803.09820, 2018.

[70] I. Sobel and G. Feldman. A 3x3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project in*, pages 271–272, 1968.

[71] S. Sonoda and N. Murata. Neural network with unbounded activation functions is universal approximator. In *International Conference on Learning Representations (ICLR)*, 2014.

[72] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *International Conference on Learning Representations (ICLR)*, 2015.

[73] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[74] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning (ICML)*, pages 1139–1147, 2013.

[75] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, Inception-ResNet and the impact of residual connections on learning. In *Advancement of Artificial Intelligence (AAAI)*, volume 4, page 12, 2017.

[76] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Conference on computer vision and pattern recognition (CVPR)*, pages 1–9. IEEE, 2015.

[77] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception architecture for computer vision. In *Conference on computer vision and pattern recognition (CVPR)*, pages 2818–2826. IEEE, 2016.

[78] M. Telgarsky. benefits of depth in neural networks. In V. Feldman, A. Rakhlin, and O. Shamir, editors, *Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 1517–1539, Columbia University, New York, New York, USA, Jun 2016. PMLR.

[79] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[80] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. Efficient object localization using convolutional networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 648–656, 2015.

[81] Y.-C. Tsai, V. Kaul, and R. M. Mersereau. Critical assessment of pavement distress segmentation methods. *Journal of transportation engineering*, 136(1):11–19, 2009.

[82] W. Van Ooijen, M. Van den Bol, and F. Bouman. High-speed measurement of ravelling on porous asphalt. In *5th Symposium on Pavement Surface Characteristics of Roads and Airports*, 2004.

[83] S. Wager, S. Wang, and P. S. Liang. Dropout training as adaptive regularization. In *Advances in neural information processing systems (NIPS)*, pages 351–359, 2013.

[84] K. C. Wang. Elements of automated survey of pavements and a 3d methodology. *Journal of Modern Transportation*, 19(1):51–57, 2011.

[85] D. Warde-Farley, I. J. Goodfellow, A. Courville, and Y. Bengio. An empirical analysis of dropout in piecewise linear networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[86] A. Wright, N. Dhillon, S. McRobbie, C. Christie, et al. New methods for network level automated assessment of surface condition in the uk. In *Symposium on Pavement Surface Characteristics: SURF*, 2012.

[87] H. Yao, S. Zhang, D. Zhang, Y. Zhang, J. Li, Y. Wang, and Q. Tian. Large-scale person re-identification as retrieval. In *International Conference on Multimedia and Expo (ICME)*, pages 1440–1445. IEEE, July 2017.

[88] E. Zalama, J. Gómez-García-Bermejo, R. Medina, and J. Llamas. Road crack detection using visual features extracted by Gabor filters. *Computer-Aided Civil and Infrastructure Engineering*, 29(5):342–358, 2014.

[89] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision (ECCV)*, pages 818–833. Springer, 2014.

[90] A. Zhang, K. Wang, B. Li, E. Yang, X. Dai, P. yi, Y. Fei, Y. Liu, J. Q. Li, and C. Chen. Automated pixel-level pavement crack detection on 3d asphalt surfaces using a deep-learning network: Pixel-level pavement crack detection on 3d asphalt surfaces. *Computer-Aided Civil and Infrastructure Engineering*, 32, 08 2017.

[91] A. Zhang, K. C. Wang, and C. Ai. 3d shadow modeling for detection of descended patterns on 3d pavement surface. *Journal of Computing in Civil Engineering*, 31(4):04017019, 2017.

[92] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *CoRR*, abs/1611.03530, 2016.

[93] E. Zhang and Y. Zhang. *Average Precision*. Springer US, Boston, MA, 2009.

[94] L. Zhang, F. Yang, Y. Daniel Zhang, and Y. J. Zhu. Road crack detection using deep convolutional neural network. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3708–3712, Sep. 2016.

[95] J. Zhou, P. Huang, and F.-P. Chiang. Wavelet-based pavement distress classification. *Transportation Research Record: Journal of the Transportation Research Board*, 1(1940):89–98, 2005.

Appendix A

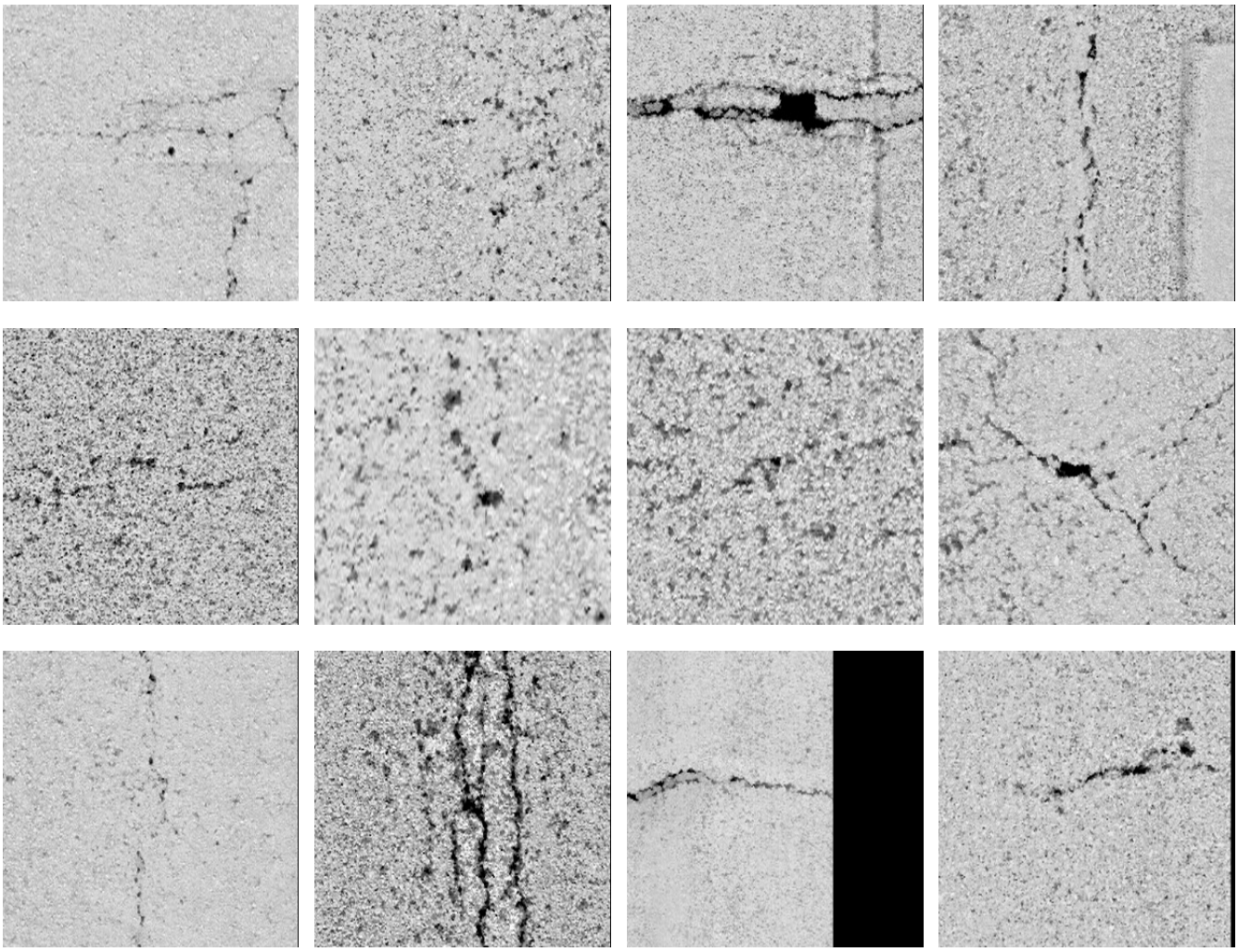


Figure 1: Examples of different cracks (1 m^2)

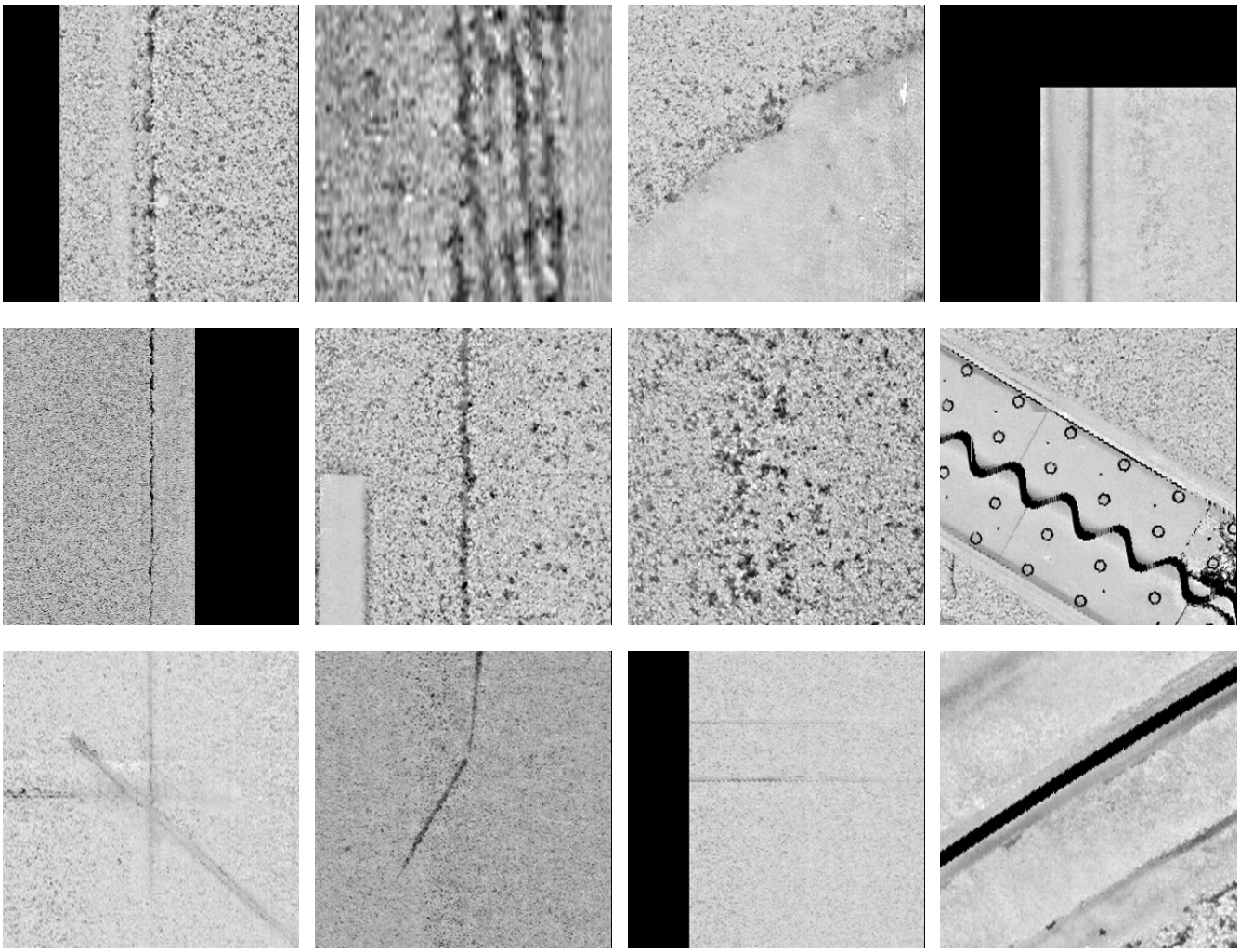


Figure 2: Examples of different non-crack patterns (1 m^2)