Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Egil Holm

Classification of Corrosion and Coating Damages on Bridge Constructions using Deep Learning

Master's thesis in Industrial Cybernetics
June 2019

NTNU
Norwegian University of
Science and Technology

NTNU – Trondheim
Norwegian University of
Science and Technology

Classification of Corrosion and Coating Damages on Bridge Constructions using Deep Learning

Egil Holm

Master's thesis in Industrial Cybernetics

Submission date:	June 2019
Supervisor:	Annette Stahl, NTNU
Co-supervisor:	Aksel Andreas Transeth, SINTEF
Co-supervisor:	Ole Øystein Knudsen, SINTEF

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Preface

This master's thesis is the final assignment in the two-year master's degree programme Industrial Cybernetics at the Department of Engineering Cybernetics, NTNU. The work in this thesis has made it possible to combine knowledge from my B.Sc in Materials Science and Engineering with engineering cybernetics in order to investigate possibilities in automatic inspection of corrosion and coating damages. The duration of the project presented in this thesis was from January 7th to June 3rd 2019.

The aim of this thesis has been to present a comparison of image classification performances for different convolutional neural networks, in order to evaluate the potential in using deep learning as an automatic analysis method for corrosion and coating damages on bridge constructions. This thesis is a continuation of my specialization project written autumn 2018, where methods for automatic inspection of bridge constructions were investigated. The Department of Engineering Cybernetics has provided both necessary hardware and software for performing the experiments in this thesis, which includes a computer, GPU and a MATLAB license.

I would like to thank Sølvi Austnes from the Norwegian Public Roads Administration (Statens Vegvesen) for providing a data set of images from previous bridge inspections. This data set was a necessary basis in the work of categorizing relevant images of corrosion and coating damages. I would also like to thank my co-supervisor Ole Øystein Knudsen from SINTEF Industry for advice regarding corrosion damages and bridge inspections. A special thank you to my supervisor Annette Stahl from NTNU and co-supervisor Aksel Andreas Transeth from SINTEF Digital for giving me feedback on this thesis, and for regular meetings in the project period.

Egil Holm
Trondheim, 03.06.2019

Abstract

This master's thesis presents a comparison of performance for different convolutional neural networks on image classification of corrosion and coating damages on bridge constructions. A total of 9300 images, divided into four classes; *paint flaking*, *red corrosion on rebar*, *red corrosion on steel construction* and *not corrosion*, were collected through manual categorization and image augmentation. Four different convolutional neural networks called AlexNet, GoogLeNet, ResNet-50 and VGG-16 were trained and tested on the collected images in order to evaluate the potential in using neural networks for automatic analysis of damages on bridge constructions. Each neural network was trained and tested using both three and four classes for comparison of performance scores depending on the number of images used in training and testing. Evaluations and comparisons were done through the performance metrics recall, precision, accuracy and F1 score. The neural networks were also evaluated on the ability of detecting damages, meaning that the number of false positives for *not corrosion* were studied. In this thesis, VGG-16 with four classes gave the overall best performance results with average values for recall, precision, accuracy and F1 score being 95.45%, 95.61%, 97.74% and 95.53%, respectively. The highest score on damage detection was achieved with AlexNet, which was trained and tested with four classes, where a detection accuracy of 99.14% was achieved. From the results obtained in this thesis, it is concluded that convolutional neural networks have a great potential in automatic analysis of corrosion and coating damages on bridge constructions. In combination with a drone and a 3D program for localization of images taken during inspections, a data analysis system using convolutional neural networks will give a complete overview of bridge construction condition. Thus, necessary measures related to repairs and maintenance can be performed efficiently. Regarding further work, collection of more relevant images of damages for network training is recommended in order to improve classification performance for convolutional neural networks. In addition, utiliziation of regions of interest or segmentation for better labeling of relevant areas of an image is considered to be a suitable approach.

Sammendrag

Denne masteroppgaven presenterer en sammenligning av ytelse for forskjellige nevrale nettverk på bildeklassifisering av korrosjon og beleggskader på brukonstruksjoner. Totalt 9300 bilder, delt inn i fire klasser; *malingsavflaking*, *rød korrosjon på armering*, *rød korrosjon på stålkonstruksjon* og *ikke korrosjon*, ble samlet gjennom manuell kategorisering og augmentasjon. Fire forskjellige nevrale nettverk, kalt AlexNet, GoogLeNet, ResNet-50 og VGG-16 ble trent og testet på de kategoriserte bildene for å evaluere potensialet i bruk av nevrale nettverk for automatisk analyse av skader på brukonstruksjoner. Hvert nettverk ble trent og testet ved å bruke både tre og fire klasser for å sammenligne ytelse avhengig av antall bilder som ble brukt til trening og testing. Evalueringer og sammenligninger ble gjort gjennom ytelsesmålinger kalt recall, presisjon, nøyaktighet og F1. De nevrale nettverkene ble også evaluert på evnen til å oppdage skader, noe som innebar at antall falske positiver for *ikke korrosjon* ble undersøkt. I denne oppgaven ga VGG-16 med fire klasser de beste resultatene med gjennomsnittlige verdier for recall, presisjon, nøyaktighet og F1 på henholdsvis 95.45%, 95.61%, 97.74% og 95.53%. Nettverket med beste resultat på evnen til å oppdage skader ble oppnådd med AlexNet, trent og testet med fire klasser, hvor deteksjonsnøyaktigheten ble 99.14%. Fra resultatene som er oppnådd i denne oppgaven, konkluderes det med at nevrale nettverk har et stort potensial i automatisk analyse av korrosjon og overflateskader på brukonstruksjoner. I kombinasjon med en drone og et 3D-program for lokalisering av bilder tatt under inspeksjon, vil et dataanalysesystem som bruker nevrale nettverk gi en fullstendig oversikt over tilstanden til en brukonstruksjon. Dermed kan nødvendige tiltak knyttet til reparasjoner og vedlikehold utføres effektivt. Når det gjelder videre arbeid, anbefales det å samle inn flere relevante bilder av skader til bruk i nettverksopplæring for å forbedre ytelsen for nevrale nettverk. I tillegg betraktes utnyttelse av såkalte “regions of interest” eller segmentering, for å bedre kunne merke relevante områder av et bilde, som en egnet tilnærming i videre arbeid.

Contents

Chapter 1

Introduction

1.1 Motivation and background

In the western part of the world, costs related to corrosion damages are estimated to be 3-4 percent of a country's Gross Domestic Product (GDP) [1] [2], and the global cost was estimated to be 3.4 percent of the global GDP in 2013 [3]. Corrosion damages are common challenges in many industries and on elements of infrastructure like bridges, tunnels and vehicles. In Norway, the GDP in 2016 was approximately 3119 billion NOK [4]. Assuming 3-4 percent of the Norwegian GDP in 2016 was related to corrosion damages, this equals a cost of 94-125 billion NOK. In this thesis, focus will be placed on corrosion and coating damages on bridge constructions in Norway. Regarding coating damages, paint flaking is the only type of coating degradation that is studied in this thesis.

In Norway there are more than 17500 bridges to be inspected and maintained by The Norwegian Public Roads Administration [5]. Corrosion damages, cracks and faults in surface treatment are examples of elements of interest during inspection of bridges. Every year there are high economic costs related to inspection of bridge constructions, and there are also safety challenges related to implementation of certain types of manual inspection methods that require use of access equipment. Manual inspection methods also have challenges in terms of subjectivity when evaluating corrosion damages, which are eliminated when using automatic analysis methods instead. Therefore, it is important to investigate the potential in automatic inspection methods [6].

The focus in previous work on detection of corrosion damages from images using deep learning approaches has mainly been on binary classification problems, with prediction of two classes; *corrosion* and *not corrosion* [7] [8]. The previous work in [7] did use images from previous bridge inspections; a total of 3500 images divided into two classes. However, in order to develop a complete automatic analysis system for damages on bridge constructions, it is necessary to be able to detect more than *corrosion* or *not corrosion*. In this thesis, a total of 9300 images from previous bridge inspections, divided into four classes, are used for training and testing of convolutional neural networks. The four classes are *paint flaking*, *red corrosion on steel construction*, *red corrosion on rebar*, and *not corrosion*.

A specialization project [6] was introduced in the preface. In this specialization project, the student presented existing methods, sensors and software for automatic inspection of bridge constructions, and a design of a robotic inspection system was suggested. The designed system consists of three main modules; data collection, handling and analysis. This master thesis is a continuation of the project, where emphasis is put on data analysis. Figure 1 shows a simple, self made illustration of the system concept presented in the specialization project. The suggested system consists of a drone, also known as an Unmanned Aerial Vehicle (UAV), with sensors for both inspection and autonomy, and minimum one storage device. The module for data handling uses one, or more, of three layers called cloud, fog and edge, depending on the conditions for data transferring. The last module of the robotic inspection system is data analysis, using computer vision techniques or deep learning approaches.

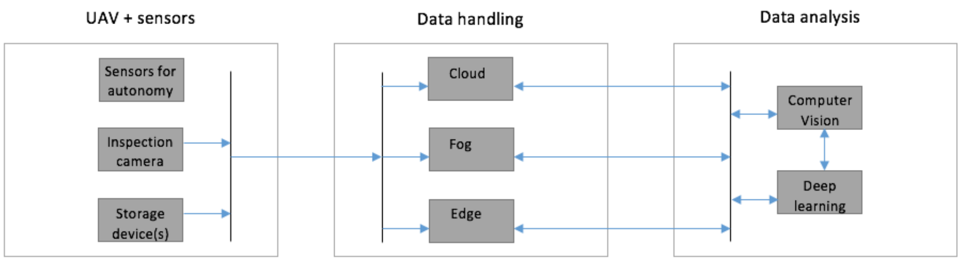


Figure 1: A simple illustration of a robotic inspection system concept. Images are passed from the inspection camera or from a storage device on an UAV, to a cloud or a local computer. Next, images are analyzed using computer vision (CV) techniques, deep learning, or a combination of these approaches [6].

Deep learning is the chosen approach for image analysis in this thesis, and figure 2 illustrates the relevant concept for automatic image analysis using what is called convolutional neural networks. The concept of data analysis suggested in figure 2 can, in combination with a localization system for images taken by a drone, give a complete overview of a bridge construction and automatically tell what type of damage one is dealing with. A system based on, for example, Visual Simultaneous Localization and Mapping (VSLAM) and 3D models of bridges, used in combination with the analysis system proposed in this thesis, would provide information on exactly where detected damages are located. Thus, damage development can be observed over time in order to find the best suited measures related to maintenance and repair.

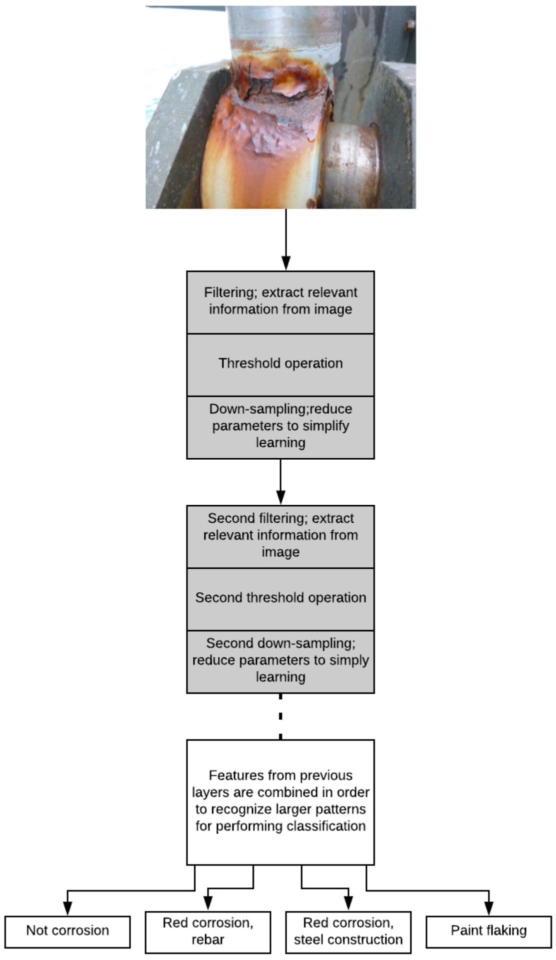

Filtering; extract relevant information from image
Threshold operation
Down-sampling;reduce parameters to simplify learning
Second filtering; extract relevant information from image
Second threshold operation
Second down-sampling; reduce parameters to simply learning
Features from previous layers are combined in order to recognize larger patterns for performing classification
Not corrosion
Red corrosion, rebar
Red corrosion, steel construction
Paint flaking

Figure 2: Illustration of the concept of automatic image analysis that will be further investigated in this thesis. The illustration is self made, and shows how an image from bridge inspection is to be analyzed and classified by convolutional neural networks into one of the four classes *paint flaking*, *red corrosion on steel construction*, *red corrosion on rebar*, and *not corrosion*. The three elements, or *layers*, in the grey modules; filtering, threshold and down-sampling, are called *convolution*, *activation function* and *pooling*, respectively. A convolutional neural network typically has several such modules. The last layers where classification is performed are called *fully-connected layer* and *output layer*. The different layers will be further explained in the theory chapter of this thesis.

1.2 The aim of the thesis

The aim of this thesis is to compare performance of different convolutional neural networks on classification of corrosion and coating damages on bridge constructions, in order to evaluate the potential for creating a system for automatic image analysis to be used in bridge inspections. A relevant data set of images must be collected, and suitable software tools for image pre-processing, training and testing of neural networks, is to be found. The thesis consists of the following subtasks:

- Search for, and write about previous work/methods on corrosion detection and classification of images using deep learning. Write theory on artificial intelligence, machine learning, deep learning and corrosion.
- Find a relevant data set containing images of corrosion damages. Images from previous bridge inspections in Brutus are relevant. Label and pre-process images using appropriate software tools.
- Choose software and networks for image classification. Modify existing neural networks, add/remove layers and choose training options. Perform training of neural networks and comparison of these through testing.
- Present results in a suitable way using graphs and tables. Find and express metrics of performance for the trained neural networks.
- Discuss results from testing of the neural networks. Explain what the results show and how this can be used, also what could have been done differently. Suggest elements for further work.

1.3 Limitations

The limitations related to this master's thesis are listed below:

- The available time for implementation of this master's thesis was between January 7th and June 3rd 2019.
- In this thesis, the studied corrosion and coating damages are damages on elements of bridge constructions *above* water only. Also, the majority of images used in training of convolutional neural networks are taken in daylight.
- The available hardware for training of convolutional neural networks was Intel Core i7-8700 CPU 3.20 GHz, Intel Core i7-6850K CPU 3.60 GHz and NVIDIA Titan X (Pascal) Single GPU.

1.4 Contributions

The aim of this thesis is to compare convolutional neural networks in order to evaluate the potential for creating an automatic analysis system for damages on bridge constructions. Because the work performed in this master's thesis is considered highly relevant for further work within automatic inspection and maintenance, it was decided to write a scientific paper based on the obtained results in this thesis. The scientific paper will be submitted to the 12th International Conference on Machine Vision (ICMV 2019)[9], and is added in the appendix in section B.

The main contributions in this thesis are listed below:

1. Development of a resource for further work within image classification through collection and categorization of relevant images in a data set. Such a data set for classification of corrosion and coating damages does not exist from before, to the best of my knowledge. The images that have been categorized in this thesis form a basis for further work on training of convolutional neural networks to be used in automatic inspection of bridge constructions in general. Over 7500 images of corrosion and surface damages from bridges in Norway were manually categorized. Through the use of data augmentation techniques, the data set was increased to a total of approximately 9300 images. The collected data set is the basis for work in this thesis, but it is also highly relevant for future work on automatic analysis of corrosion and coating damages in general.

2. Extension of previous work by classification of four categories of corrosion and coating damages, compared to only two in previous work. In this thesis, images of surface and corrosion damages were divided into four different classes to be used for training of convolutional neural networks. In previous work on automatic corrosion detection on bridge constructions [7], the focus has been on studying two classes; *corrosion* and *not corrosion*. Using more than two classes gives a better overview of a bridge construction's condition, thus eases the work of finding relevant measures related to maintenance and repair. Approximately 240 000 images received from The Norwegian Public Roads Administration were manually evaluated in order to find relevant images to be used in this thesis. The images were categorized in folders into the classes; *paint flaking*, *red corrosion on steel construction*, *red corrosion on rebar*, and *not corrosion*. From the respective folder names the images were labeled to serve as a data set for training and testing of convolutional neural networks in this master thesis.

3. The results in thesis show that convolutional neural networks have a great potential in classifying images from bridge inspections automatically, thus providing a condition

monitoring of the entire bridge construction. Images taken by a drone can be passed directly to a trained neural network, removing challenges with subjectivity related to manual evaluation of images performed by an engineer, while significantly increasing efficiency of inspections.

4. This master's thesis creates a basis for development of a data analysis system for automatic detection and classification of damages on bridge constructions. All convolutional neural networks trained and tested in this thesis obtained results showing an average classification accuracy of over 95 %.

1.5 Outline of the thesis

First, an introduction to the thesis is given in chapter 1, including the background and aim of the thesis, as well as contributions. Next, chapter 2 explains relevant background theory on corrosion and machine learning. Chapter 3 introduces previous work on classification of corrosion damages using deep learning, as well as existing neural networks and relevant software. Chapter 4 explains the process of data collection and implementation of a classification system, including categorization and labeling of images, and how the chosen CNNs were trained and tested. Training and tests of networks performed in this thesis, along with obtained results and discussions, are presented in chapter 5. A more general discussion of tests, results and choices made in this thesis is made in chapter 6. Finally, a conclusion of the thesis and a summary of suggestions for further work are given in chapter 7.

1.6 Abbreviations

AI	Artificial intelligence
CNN	Convolutional neural network
CV	Computer vision
MSE	Mean square error
ReLU	Rectified linear unit
SGD	Stochastic gradient descent
UAV	Unmanned aerial vehicle
VSLAM	Visual Simultaneous Localization and Mapping

Chapter 2

Background theory

This chapter includes relevant theory on principles of corrosion, types of damages, artificial intelligence and machine learning. In addition, theory on corrosion protection and inspection procedures for bridge constructions are included in this chapter. First, theory on the basic principles of corrosion is introduced, and different types of corrosion damages are explained and illustrated. Next, corrosion protection and inspection procedures for bridge constructions are introduced. Principles of artificial intelligence and machine learning are presented to create a basis for understanding the work in this thesis and the main elements of previous work. The theory on machine learning includes several topics such as defining a learning problem, types of machine learning, deep learning and artificial neural networks. Some of the paragraphs regarding neural networks are taken directly from the specialization project [6]. This is made clear through citation of the project and explanations in the text.

2.1 Corrosion and corrosion damages

In this section, corrosion and types of corrosion damages will be introduced. First, the general principles of corrosion are explained, including the electrochemical process and relevant chemical reactions. Section 2.1.1 is included to give a brief introduction to the basics of corrosion, for understanding of why and how it occurs. Next, forms of corrosion damages and appearances are introduced and illustrated. This is important for knowledge about how corrosion damages can be categorized and included in formulation of a machine learning problem.

2.1.1 General principles of corrosion

Corrosion can be defined as "an attack on a metallic material through reaction with its surrounding medium" [10]. For metals, the most common type of corrosion is *wet corrosion*, where the surrounding medium typically is water containing different dissolved substances. Water becomes an *electrolyte* in what is called an *electrochemical* reaction, typically on the surface of the metal. An electrochemical reaction is a chemical reaction including transferring of electrons through a common medium, an electrolyte, between the materials and substances involved. For metals, this reaction is called an *oxidation* reaction [10] [11]. A typical oxidation reaction for some theoretical metal, M, is shown in (1), where it is observed that the metal "loses" electrons, e^-. In this reaction, n is the number of electrons and $n+$ indicates that the metal, M, becomes a positively charged ion after oxidation [11].

$$M \rightarrow M^{n+} + ne^- \tag{1}$$

Examples of typical oxidation reactions for iron (Fe), zinc (Zn) and aluminium (Al) are shown in (2).

$$Fe \rightarrow Fe^{2+} + 2e^- \tag{2a}$$
$$Zn \rightarrow Zn^{2+} + 2e^- \tag{2b}$$
$$Al \rightarrow Al^{3+} + 3e^- \tag{2c}$$

The electrons from an oxidation reaction are transferred to what is called a *reduction* reaction. The locations where oxidation and reduction occur are also called *anode* and *cathode*, respectively [11]. Examples of typical reduction reactions for corrosion in acid, neutral or basic solutions, are shown in (3).

$$2H^+ + 2e^- \rightarrow H_2 \tag{3a}$$
$$O_2 + 4H^+ + 4e^- \rightarrow 2H_2O \tag{3b}$$
$$O_2 + 2H_2O + 4e^- \rightarrow 4(OH^-) \tag{3c}$$

Reactions shown in (2) and (3) are termed *half-reactions* since they show only half of the total electrochemical reaction that occurs. In (4), the total reaction for corrosion of zinc in an acid solution is given by the sum of the reactions in (2b) and (3a) [11]. This reaction is also illustrated in figure 3.

$$Zn + 2H^+ + 2e^- \rightarrow Zn^{2+} + 2e^- + H_2 \tag{4a}$$
$$\Rightarrow Zn + 2H^+ \rightarrow Zn^{2+} + H_2 \tag{4b}$$

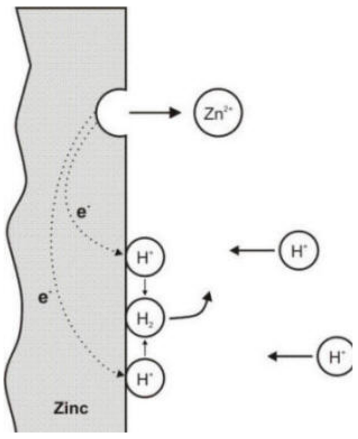


Figure 3: Corrosion of zinc in an acid solution [12].

2.1.2 Forms of corrosion damages

In this section the following forms of corrosion damages will be explained and illustrated:

- General/uniform corrosion
- Pitting

- Crevice corrosion
- Galvanic corrosion

General/uniform corrosion

General, or uniform, corrosion is a form of corrosion damage that is quite evenly distributed over the exposed surface, leading to a uniform thickness reduction of the material. This is the most common form of corrosion damage, but at the same time it is predictable, so necessary measures and design choices can be made early to prevent or reduce damages. For example, if the corrosion rate for a specific material in a construction is known, it can be designed with extra thickness to make sure strength requirements are met even after a severe corrosion damage [10] [11]. Figure 4 and 5 both illustrate a typical uniform corrosion damage.

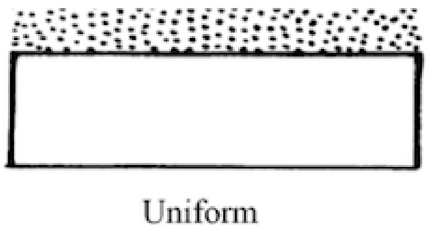


Figure 4: Illustration of a uniform corrosion damage on a surface [13]. The dots represent the corroded part of the surface, which is uniform.

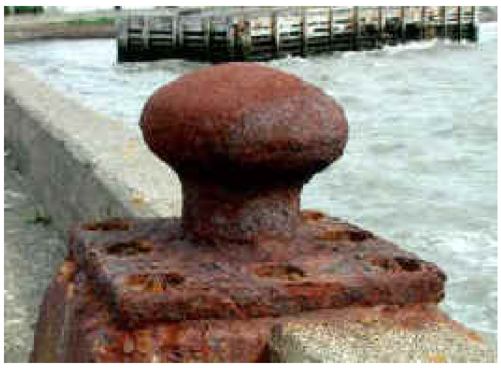


Figure 5: A typical uniform corrosion damage where the damage is evenly distributed over the surface [14].

Pitting

Pitting is a *local* corrosion damage, meaning that corrosion occurs as local pits or holes in the surface of the material. This form of corrosion damage is common on *passivated* metals and alloys in mediums containing, for example, chlorine, bromine or iodine. *Passivity* is a result of the formation of a thin, protective oxide film on the surface of a metal, which happens under specific environmental conditions. This is very common for metals like chromium, iron, nickel and titanium, including their alloys. Pitting is a form of corrosion damage that, in general, is difficult to detect and predict, and it is therefore considered dangerous [10] [11]. Examples of different holes and shapes common for pitting is illustrated in figure 6. Figure 7 shows an example of pitting on steel.

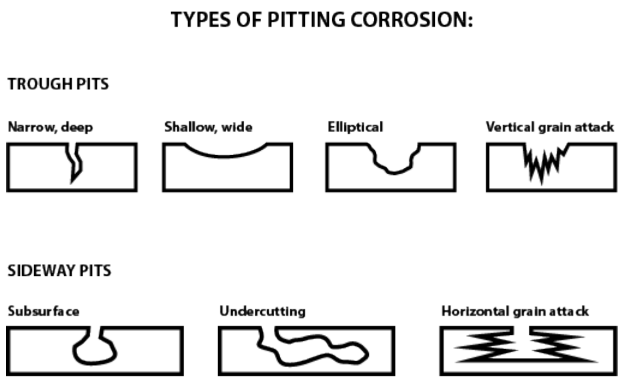


Figure 6: Illustration of different types of pitting corrosion damages. The illustration shows that some pitting corrosion damages are difficult to detect, and it can also be a challenge to evaluate the extent of the damage from just looking at the surface of the metal [15].

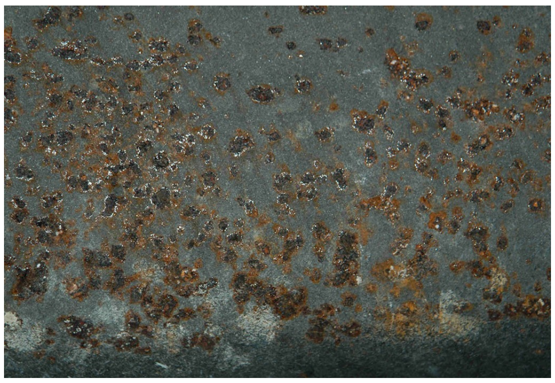


Figure 7: Example of pitting corrosion on steel, with the characteristic local pits or holes in the surface [16].

Crevice corrosion

Corrosion that occurs in occluded regions, like in the crevice illustrated in figure 8, is called crevice corrosion [17]. This form of corrosion can also occur under deposits of dirt or corrosion products. In general one can say that crevice corrosion is a local form of corrosion where "the opening of the crevice is wide enough for a liquid to enter, and at the same time tight enough for the liquid to be stagnant inside the crevice" [10]. The most distinct crevice corrosion occurs on materials that easily passivates, like stainless steels and aluminium, when the materials are exposed to substances like chlorides that can destroy the passivity locally [10]. Figure 9 shows an example of crevice corrosion between two bolted plates.

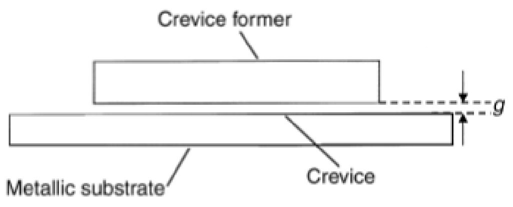


Figure 8: Illustration of a typical situation causing crevice corrosion. The gap, g, is the distance between the metallic substrate and the crevice former [17].

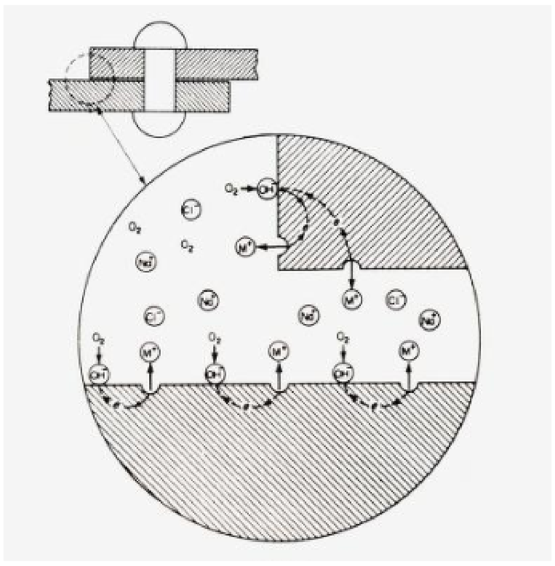


Figure 9: An example of crevice corrosion occurring between two bolted plates [10].

Galvanic corrosion

Galvanic corrosion is a form of corrosion that can occur when two metallic materials of different composition are connected, both physically and through a common electrolyte. The more *reactive* material of the two will experience more corrosion than the less reactive, or more *inert*, which gets a reduced corrosion rate [10] [11]. In other words, the more reactive material (anode) will corrode more than it would without connection to the less reactive, while the less reactive (cathode) experiences less corrosion than it would by itself [18]. Figure 10 shows an overview of a variety of metals and alloys, and their activity, or reactivity, in seawater. An example of galvanic corrosion on fasteners of carbon steel is shown in figure 11.

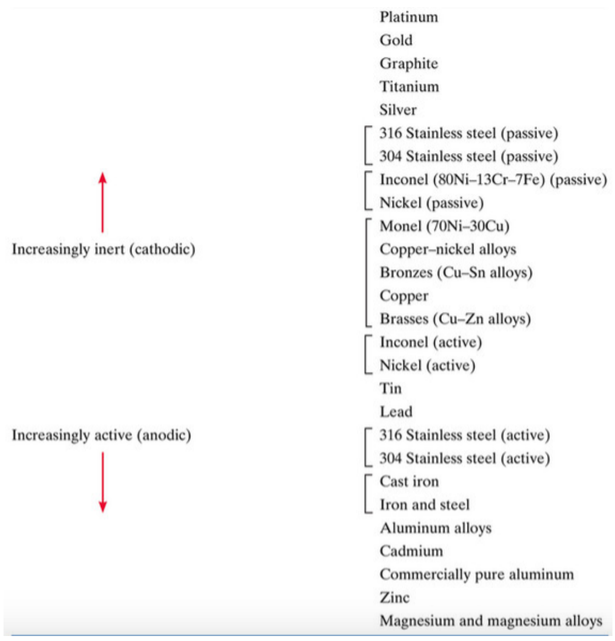

Platinum
Gold
Graphite
Titanium
Silver
316 Stainless steel (passive)
304 Stainless steel (passive)
Inconel (80Ni–13Cr–7Fe) (passive)
Nickel (passive)
Increasingly inert (cathodic)
Monel (70Ni–30Cu)
Copper–nickel alloys
Bronzes (Cu–Sn alloys)
Copper
Brasses (Cu–Zn alloys)
Inconel (active)
Nickel (active)
Tin
Lead
Increasingly active (anodic)
316 Stainless steel (active)
304 Stainless steel (active)
Cast iron
Iron and steel
Aluminum alloys
Cadmium
Commercially pure aluminum
Zinc
Magnesium and magnesium alloys

Figure 10: The galvanic series of a number of metals and alloys. This overview shows relative reactivity of some metals and alloys in seawater [19]. The red arrows illustrate which ones are increasingly cathodic or anodic. If, for example, components of 316 stainless steel and zinc are connected in a construction surrounded by seawater, the zinc component will experience most corrosion.

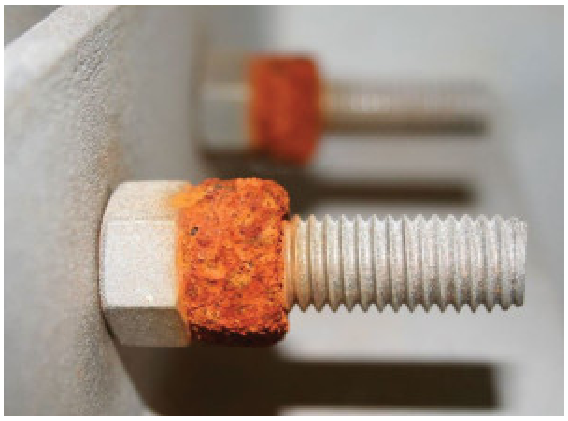


Figure 11: Galvanic corrosion on fasteners of carbon steel used on bolts of stainless steel [20]. In this case, one can compare with the overview in figure 10 and see that cast iron and steel in general are more anodic than stainless steel (316 and 304), and therefore the carbon steel fasteners are more exposed to corrosion than the bolts of stainless steel.

2.1.3 Specific metals and corrosion appearance

The appearance of corrosion depends on the form, as shown in the previous section, but it also depends on the type of metal corroding. Examples of specific metals and corrosion appearance on these are shown in this section.

The most well known corrosion appearance is probably “red rust”, which is common on iron and steel alloys. The reddish or brown colour is characteristic for this corrosion appearance, and two examples from corrosion damages on a bridge’s steel construction are given in figure 12 and 13.

“White rust” is a term used primarily for corrosion on zinc and zinc alloys. Corrosion on aluminium also results in a white-like colour. The steel surface on a bridge construction is often applied a zinc coating for corrosion protection. This is explained further in the next section. Figure 14 and 15 shows examples of typical corrosion appearances when the zinc coating on a bridge construction is damaged.

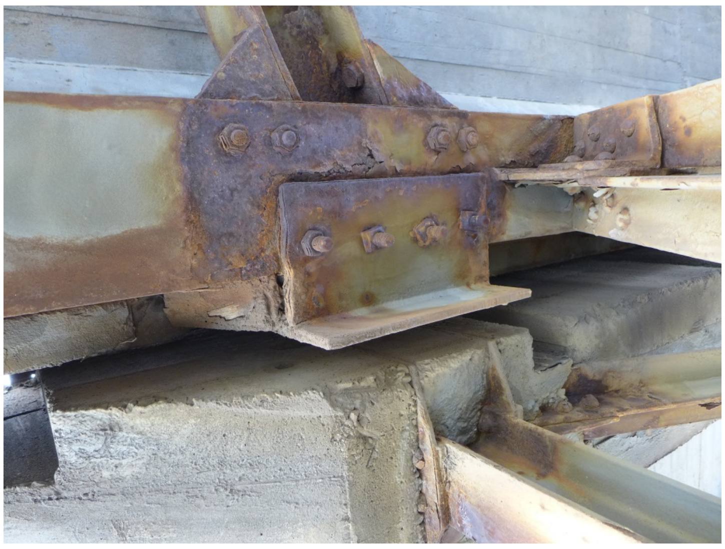


Figure 12: Illustration of corrosion on the steel construction of a bridge. This type of corrosion damages is often referred to as “red rust” because of the characteristic reddish colour. The image is downloaded from Brutus management system [21].

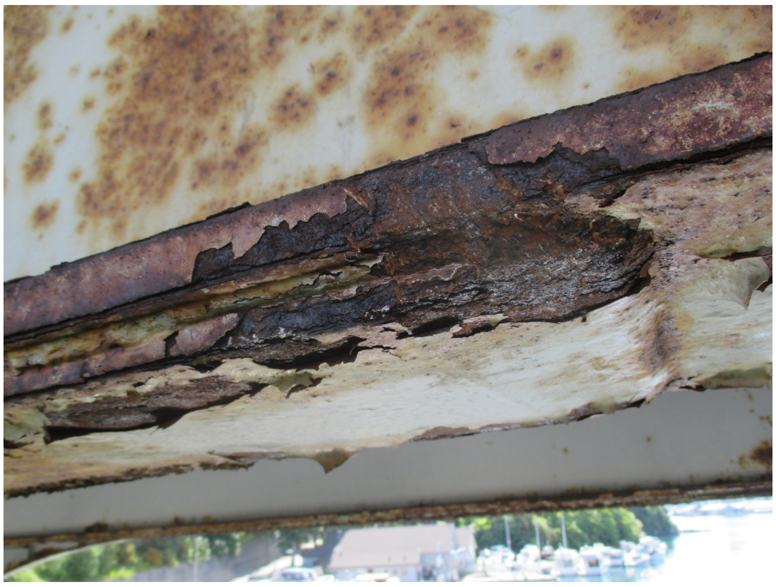


Figure 13: Illustration of corrosion on the steel construction of a bridge. This type of corrosion damages is often referred to as “red rust” because of the characteristic reddish colour. The image is downloaded from Brutus management system [21].

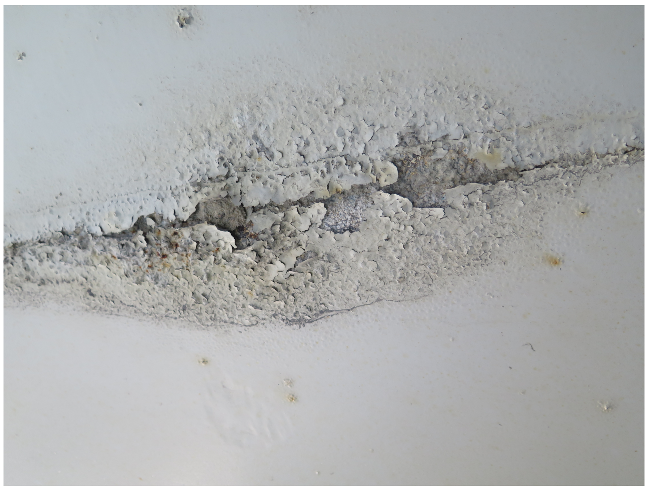


Figure 14: Illustration of corrosion on the steel construction of a bridge. This type of corrosion damages is often referred to as “white rust” because of the characteristic white-like colour. The image is downloaded from Brutus management system [21].

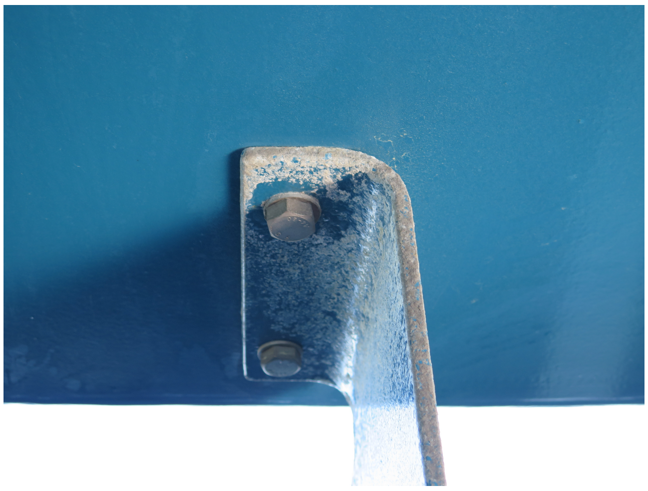


Figure 15: Illustration of corrosion on the steel construction of a bridge. This type of corrosion damages is often referred to as “white rust” because of the characteristic white-like colour. The image is downloaded from Brutus management system [21].

2.2 Corrosion protection and inspection of bridges

This section will introduce corrosion protection of steel bridges and will also explain the different types of bridge inspections, how inspections are implemented and evaluation of faults and damages.

2.2.1 Corrosion protection of steel bridges

Since the late 1960s, the Norwegian Public Roads Administration has used surface treatment systems on steel bridges called "duplex" systems, which consist of both metallic coating and paint layers. Earlier, a paint with lead oxide, also called *read lead*, was used [22]. Over the years, different compositions of coating and paint have been utilized, but thermal sprayed zinc (TSZ) is the metallic coating that has been used [23]. TSZ is zinc applied to a surface, commonly through the use of arc or flame spraying processes [24]. Figure 16 shows how the layers in a duplex surface treatment system typically are applied a steel construction.

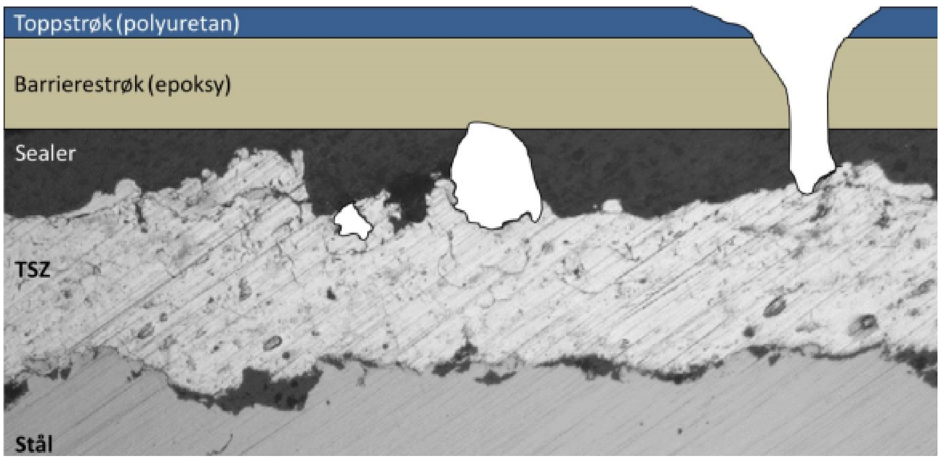


Figure 16: An illustration of typical coating and paint layers on a steel bridge construction [25]. At the bottom of the figure, the steel (in Norwegian:stål) is illustrated. Next, TSZ is applied, followed by a sealer. The two last paint layers illustrated here are epoxy and polyurethane. This figure also shows typical faults or damages on the protective surface layers. These are shown as a damage through the three first layers, to the top right, so that the TSZ is potentially exposed for corrosion, and as pores in the sealer, approximately in the middle of the illustration.

2.2.2 Types of inspections and implementation

The Norwegian Public Roads Administration has their own management system for bridges called *Brutus* [21] that is used in planning of inspections. Brutus contains, among other things, information about maintenance and inspection plans, load capacities, security management and images from inspections for overview of bridge condition. [26] refers to four types of routine inspections explained in table 1. Everything in this section is taken directly from [6].

Table 1: The four types of routine inspections on bridges [26].

Type of inspection	**Description**
Simple inspection	The purpose of a simple inspection is to check if any serious damages have occurred that in short term may affect the load capacity of the bridge, road safety, future maintenance and inspection as well as bridge esthetics.
Main inspection	A main inspection consists of a condition monitoring of the bridge construction above water. This is done to verify that the bridge meets necessary function requirements.
Main inspection of cables	The purpose of this inspection is to check that load cables, rods and fasteners fulfill specified requirements.
Main underwater inspection	Consists of a condition monitoring of the bridge underwater construction.

A simple inspection is carried out by simple visual inspection of the bridge construction above water. A simple visual inspection means that there is no use of access equipment like lifts, so the construction is inspected at a certain distance. Measurements and material testing is usually not required, but in case of great wear and tear some measurements may be necessary [26]. This type of inspection is carried annually except from the year a main inspection is performed [27].

Main inspections are carried out every third year for ferry bridges and moving bridges, and every fifth year for all other types of bridge constructions [27]. In a main inspection the entire bridge construction above water, except from cables, is visually inspected. The visual inspection must be close visual, which means that operators must be able to touch the construction that is inspected. If expected damages are detected with complete certainty from a further distance, a close visual inspection is not necessary.

A main inspection of cables is carried out by close visual inspection as in a main inspection. The same applies for a main underwater inspection where the diver has to be able to touch the underwater bridge construction that is inspected.

Damages and causes of damages that are discovered during inspections, like the four explained in table 1, can be further inspected through what is called a *special inspection*. A special inspection can also be carried out to achieve a basis for describing expensive and complicated measures [26].

2.2.3 Evaluation of faults and damages

When evaluating faults or damages on the bridge construction it has to be decided what type of fault one is dealing with, and considerations on how serious the faults are, as well as on main causes of the faults, has to be made. The Norwegian Public Roads Administration uses the terms *severity* and *consequence of damage* in the evaluation of faults [26]. Table 2 and 3 explain the two terms in more detail [6].

Table 2: Severity related to damage [26].

Code/numbering	**Explanation**	**Measures necessary**
1	Small damage	No
2	Medium damage	Yes, within 4-10 years
3	Large damage	Yes, within 1-3 years
4	Critical damage	Yes, latest within 6 months

Table 3: Consequence of damage [26].

Code/numbering	**Explanation**
B	Damage that affects the load capacity
T	Damage that affects traffic and road safety
V	Damage that may increase maintenance costs
M	Damage that may affect bridge esthetics and surroundings

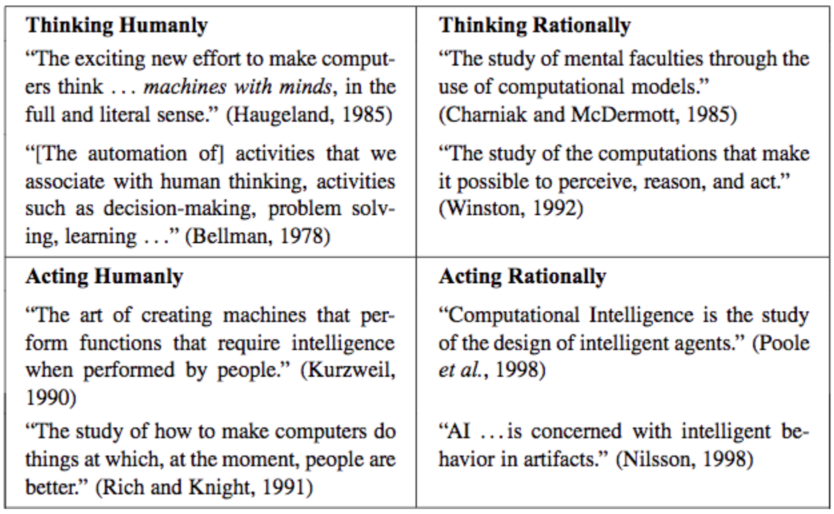
2.3 Machine learning

This section will first give an introduction to artificial intelligence, definitions and some general applications. Next, principles of machine learning are explained, including how to identify elements in a learning problem and what design choices have to be made. The different categories of machine learning and the main differences between them are introduced. In this thesis, deep learning approaches are used for classification of images, and the principles of artificial neural networks and deep learning are therefore described in this section. The descriptions include relevant functions and algorithms, as well as theory on how artificial neural networks are trained, validated and tested. Convolutional neural networks are introduced, which are the type of neural networks used for image classification in this thesis.

2.3.1 Introduction to artificial intelligence

There are several definitions of the term *artificial intelligence* (AI). One definition describes AI as “the study of how to make computers do things at which, at the moment, people are better” [28]. For example, in fast execution of many computational tasks computers outperform humans, but in the ability to enter an unfamiliar room and, within less than a second, being able to recognize surroundings and plan actions, humans clearly outperform computers today [28]. AI can also be defined as “the capability of a machine to imitate intelligent human behaviour” [29] [6]. In [30], eight definitions of AI are introduced and organized into four different categories. This is illustrated in figure 17. The different approaches for the four categories are further explained in table 4.

The foundation of AI is build on knowledge and methods from many different disciplines, such as mathematics, economics, neuroscience, computer science, psychology and cybernetics. Applications of AI today are for example in robotics, driverless vehicles and autonomous planning and scheduling [30]. A different application of AI is in learning, or more specific, in *machine learning*. Machine learning is a specialization within AI where a computer is trained to find patterns in data, and in this way learns instead of being programmed [31]. Section 2.3.2 will explain the principles of machine learning.



Thinking Humanly	**Thinking Rationally**
"The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense." (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Acting Humanly	**Acting Rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998) "AI . . . is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

Figure 17: Eight definitions of artificial intelligence, organized into four categories; thinking humanly, acting humanly, thinking rationally and acting rationally [30].

Table 4: Four different approaches based on definitions of artificial intelligence [30].

Approach	**Explanation**
Thinking humanly: *The cognitive modeling approach*	It is necessary to first determine how humans think in order to say that some program thinks like a human. This can be done through three methods; **introspection**, **psychological experiments** and **brain imaging**.
Acting humanly: *The Turing Test approach*	The Turing Test was designed to give a well functioning definition of intelligence. The test was proposed by **Alan Turing** [32], where a human interrogator evaluates, after some written questions, if the response comes from a human or a computer. If the interrogator can not tell, then the computer passes the test.
Thinking rationally: *The "laws of thought" approach*	This approach is related to the field known as **logic**. In building intelligent systems, the **logistic** tradition in AI is of importance, but it has its challenges.
Acting rationally: *The rational agent approach*	The definition of an agent here is just "something that acts". Computer agents are expected to have capabilities like adapting to changes, pursuing own goals and operating autonomously. An agent that acts in ways to achieve the best, or best expected, outcome, is called a **rational agent**.

2.3.2 Principles of machine learning

In this section, the principles of machine learning will be introduced, including relevant terms and methods, how to define a learning problem, and different categories of machine learning.

In machine learning, one seeks to create computer programs that improve with experience automatically. Concepts and results from fields like artificial intelligence, statistics, cognitive science and control theory are examples of elements that machine learning is based on. The definition of a computer program's learning from [33] says: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". To define a learning problem, E, T and P must be identified, being the source of experience, the class of tasks and the measure of performance, respectively. Consider the following example of a learning problem, with inspiration from [33]: *A computer program learning to play chess*. E, T and P can then, for example, be identified as in the list below.

- **Task, T**: Learn how to play chess.
- **Performance measure, P**: Amount of games won/lost against an opponent.
- **Training experience, E**: Play practice games of chess against itself.

An example of a learning problem more relevant to this thesis would be: *A computer program learning to detect corrosion from images*. E, T and P can then be identified as follows:

- **Task, T**: Learn to identify corrosion from images.
- **Performance measure, P**: Percentage correctly classified images from total amount of evaluated images.
- **Training experience, E**: Exposing the program to various types of images with and without corrosion.

When designing a machine learning system there are several design choices that have to be made. For example, it must be decided what exactly should be learned, what training experience the system should learn from, how the learning problem should be represented and what algorithm is most suited. The choice of what is called a *target function* is considered a key design choice. The target function can be explained as a function representing the problem of improving the performance, P, at the task, T. Using the example given earlier of a computer program learning to detect corrosion from images, a target function could be defined as $F(i)$, where i is the current image evaluated. Next, target function values could be $F(i) = 1$ if the image is correctly classified, and $F(i) = -1$ if the image is wrongly classified. Similarly, for the chess example, a target function can be defined as $F(b)$, where

b is a board state. The target function value can be $F(b) = 100$ if the board state results in victory, $F(b) = -100$ in the case of loss and $F(b) = 0$ if the result is a draw [33].

In general, it can be difficult for an algorithm to learn the exact target function, so an approximation is often used. Thus, the process of learning the target function is also called *function approximation*. For the chess example, the approximation of the target function can be denoted $\tilde{F}(b)$. One possibility is to represent the approximation as a linear function, a combination of *features* and *weights*. An example of a target function approximation, here $\tilde{F}(b)$ from the chess example, expressed as a linear combination of features is shown in (5). The features are denoted $x_1, x_2, ...x_n$, and weights are denoted $w_0, w_1, ...w_n$ [33].

$$\tilde{F}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n \tag{5}$$

The features represent certain type of information about the learning problem. The weights are numerical coefficients that the learning algorithm chooses, or learns to choose to determine what value the different board states have [33]. In the case of the chess learning problem, some relevant features are listed below:

- The number of white chess pieces: x_1
- The number of black chess pieces: x_2
- The number of pawns in black or white: x_3
- The number of knights in black or white: x_4
- and so on...

The next steps for a complete design of a learning system are first to choose a *function approximation algorithm*, which includes elements such as estimating training values and adjusting the weights, and second to create a final design of program modules to represent components in the learning system. Typical modules are the *performance system*, the *critic*, the *generalizer* and the *experiment generator*. The module that solves the performance task, using one or more target functions, is called the performance system. The output from the performance system is given as input to the critic, which produces training examples of the target function. The training examples are then used as inputs to the generalizer, the module that produces a hypothesis, or estimates the target function. This estimate is sent to the experiment generator, which creates a new problem to be an input to the performance system [33].

It is common to define three categories of machine learning; supervised, unsupervised and reinforcement learning. These three categories are illustrated in figure 18. A fourth category, *semi-supervised* learning, is sometimes mentioned in machine learning literature. The focus in this thesis will be on *supervised* learning.

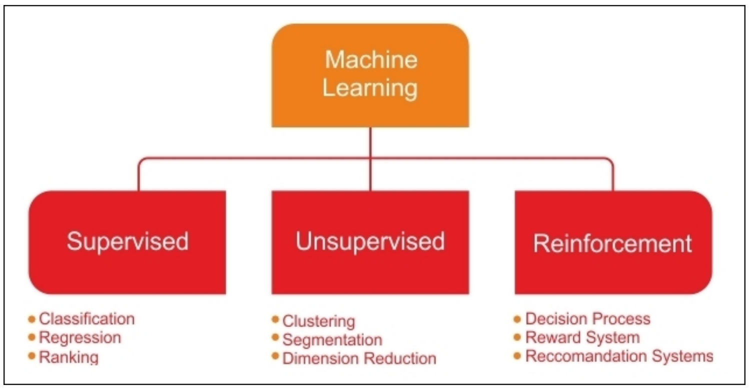
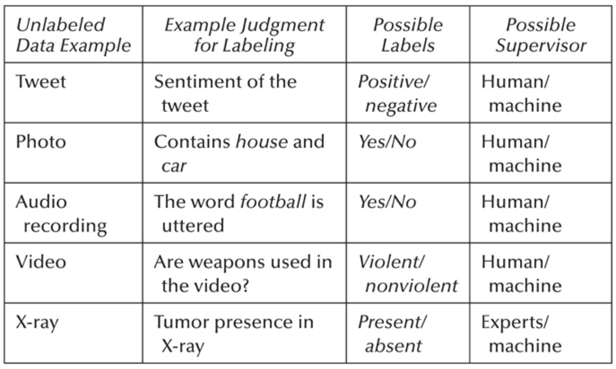


Figure 18: Three categories of machine learning; supervised, unsupervised and reinforcement learning [34].

In *supervised learning* the machine finds an unknown function from examples given as input in order to learn how to predict the output [31]. The training data is *labeled*, and consists of a input vector, **x**, and an output vector, **y**, with labels. Labels in **y** can be described as explanations of the inputs in **x**, thus the combination of the vectors **x** and **y** form training examples for the machine. The *supervisor* is a human or a machine that provides the labels [35]. Figure 19 shows some unlabeled examples and suggestions for labels and supervisors that can be used. A different example could be *Photo:Existence of corrosion*, with labels *corrosion/not corrosion* or *yes/no*.



Unlabeled Data Example	*Example Judgment for Labeling*	*Possible Labels*	*Possible Supervisor*
Tweet	Sentiment of the tweet	*Positive/ negative*	Human/ machine
Photo	Contains *house* and *car*	*Yes/No*	Human/ machine
Audio recording	The word *football* is uttered	*Yes/No*	Human/ machine
Video	Are weapons used in the video?	*Violent/ nonviolent*	Human/ machine
X-ray	Tumor presence in X-ray	*Present/ absent*	Experts/ machine

Figure 19: Unlabeled examples and suggestions for labels and supervisors [35].

Examples of common approaches and algorithms for supervised machine learning are decision trees, rule-based algorithms, the k-nearest neighbour algorithm, support vector machine (SVM) and neural networks. Decision tree is used in classification of data, and it is a statistical model where results are represented in a chart like a tree structure. Rule-based algorithms use so-called “if-then” rules to classify data, which means that *if* some condition is fulfilled *then* a decision is made. The k-nearest neighbour algorithm is a method used in pattern recognition where the input is the k nearest training examples in a feature space. It is common to assign weights to the examples in such a way that the closest “neighbours” contribute most to the decision making. Support vector machines are learning models where the training examples are represented as points in space and mapped for explicit separation between different categories [35]. Neural networks are explained in detail in the next section.

Supervised learning uses labeled training data, but in *unsupervised learning* the machine works with unlabeled data. Instead of training a machine on labeled data to be able to evaluate the inputs it is given, the key in unsupervised learning is to find some hidden structure in the provided input data [35]. Reinforcement learning is a method where the machine or program has interaction with an environment that gives feedback as rewards or penalties, depending on the performance of the model. In this way, the desired behaviour of the program is *reinforced*, without directly telling it what to do [31].

2.3.3 Artificial neural networks and deep learning

This section contains the following topics on artificial neural networks (ANNs) and deep learning:

1. Introduction to the term deep learning and artificial neural networks
2. Principles of neural networks
3. Gradient descent and stochastic gradient descent
4. The backpropagation algorithm
5. Training, validation and common challenges related to ANN
6. Convolutional neural networks
7. Transfer learning

1. Introduction to the term deep learning and artificial neural networks

Deep learning is a class of techniques in machine learning with an increasing amount of applications in for example speech recognition and object detection in images. Learning methods called *representation learning* allow a machine to automatically find the representations needed to detect or classify the input it is given. Deep learning methods can be described as representation learning methods with several representation levels. These levels are obtained by composing non-linear modules, where each module transforms the representation of its input at one level into a representation at a higher level, more abstract than the previous [36].

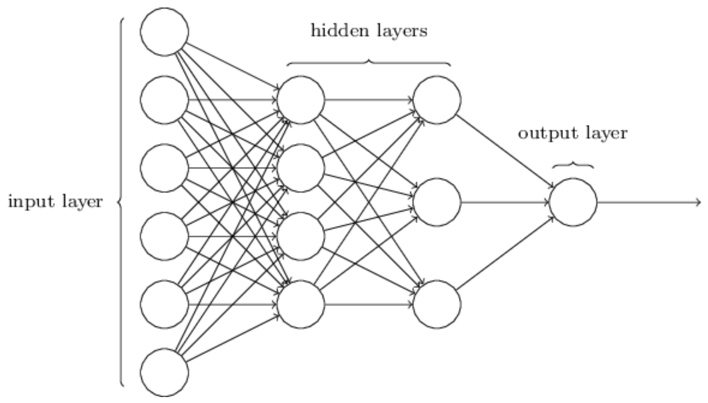


Figure 20: Illustration of a multi-layer artificial neural network [37]. The first layer is the input layer, followed by two hidden layers, and finally the output layer.

The learning process of deep learning consists of training what is called *artificial neural networks* (ANNs). ANNs consist of *neurons*, processing units, that are connected through *synapses*. Since ANNs are inspired by the human brain, expressions are based on physiological terms [38]. Neurons and synapses in ANNs are illustrated in figure 20, where the circles in the different layers are neurons, and the links (arrows) between them are synapses [6].

When an artificial neural network has more than one hidden layer, as shown in figure 20, the term *deep learning* is used. Each neuron has a certain capacity for processing information, and they influence each other through the synapses. The learning process for ANNs consists of deciding how much one neuron should affect the other, and this is referred to as deciding the synaptic weightings. Data, called a *training set*, is provided the network in the input layer, and the synaptic weightings are adjusted until the network is able to separate the given data in a desired way [38] [6]. The learning process for ANNs will be explained in more detail later in this section.

2. Principles of neural networks

To understand the working principle of neural networks, a type of ANN system based on units called *perceptrons* will first be explained.

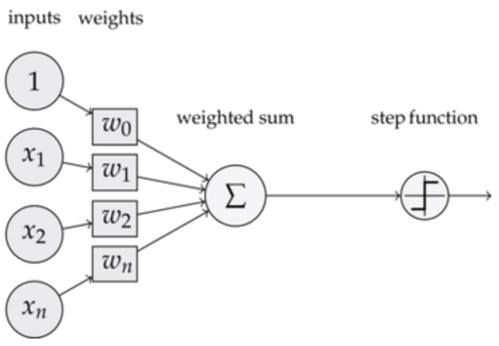


Figure 21: Illustration of a perceptron with inputs, x, and weightings, w. A weighted sum of the inputs is generated, and the output from the step function depends on the set threshold [39].

Figure 21 illustrates a perceptron which takes a vector of binary inputs, x, and calculates a linear combination of the inputs using the weightings, w. The output of the step function is typically equal to 1 if the weighted sum exceeds a certain threshold, otherwise it is equal to -1 (or 0) [33]. The mathematical expression for the output, y, is shown in (6). For the output to be 1, the weighted sum of inputs from x_1 to x_n must be larger than the quantity $-w_0$, also called a *bias* [33] [40].

$$y(x_1, ..., x_n) = \begin{cases} 1, & \text{if } w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n > 0. \\ -1, & \text{otherwise.} \end{cases} \tag{6}$$

A challenge with perceptrons is that a successful weight vector may not be found if the training data is not linearly separable. It is preferable that a small change in the weights results in a small, corresponding change in the output [40]. This is illustrated in figure 22. For perceptrons, a minor change in some of the weights, or the bias, may suddenly result in the output changing from 1 to -1, and this may again impact the rest of the network in undesirable ways. A different, newer type of neuron called a *sigmoid* neuron that overcomes the challenge related to perceptrons is therefore very common [40].

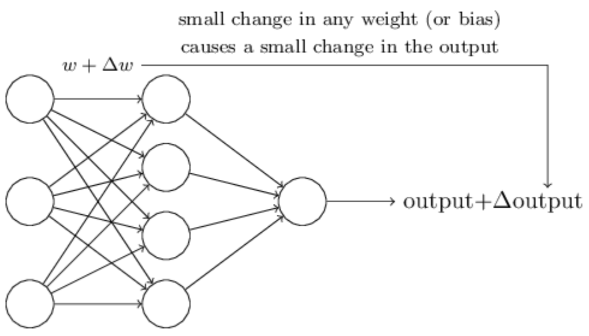

small change in any weight (or bias)
causes a small change in the output
$w + \Delta w$
output+Δoutput

Figure 22: The illustration shows how a small change in the weights, Δw, changes the output with $\Delta output$ [40].

The inputs of sigmoid neurons does not have to be exactly 0 or 1, like for perceptrons, but can also be values in between. Sigmoid neurons also have a weight associated with each input value, as well as a bias, b. However, the output is not only 1 and -1, or 1 and 0, but a *sigmoid function*, defined in (7) [40]. The sigmoid function is also called the *logistic function* [33]. A plot of the sigmoid function is shown in figure 23.

$$\sigma(z) = \frac{1}{1+e^{-z}}, \quad z = w_1x_1 + w_2x_2 + ... + w_nx_n + b \tag{7}$$

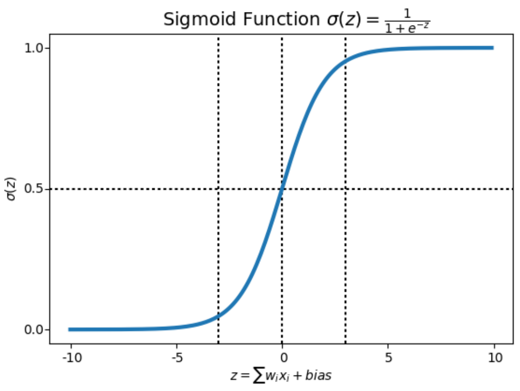


Figure 23: A plot of the sigmoid function. It is observed that the value of the sigmoid function goes toward zero for small values of z, and that it tends to 1 for larger values of z [41].

The *smoothness* of the sigmoid function makes it produce a small change in output, $\Delta output$, when small adjustments to the weights and bias are done. As mentioned earlier, this is preferable in the training process of neural networks, and the case was illustrated in figure 22. An approximation of $\Delta output$ is given by (8), as a linear function of Δw_j, change in weights, and Δb, change in bias [40].

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b \tag{8}$$

Functions like the step function and the sigmoid function are called *activation functions*. The main difference in using a different activation function is that the values of the partial derivatives in (8) will change [40]. Examples of some other activation functions are the *hyperbolic tangent* function, (9) and the *Rectified Linear Unit* (ReLU), (10). The hyperbolic tangent function and the ReLU are illustrated in figure 24 and 25, respectively.

$$f(x) = tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \tag{9}$$

$$f(x) = max(0, x) = \begin{cases} 0 & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases} \tag{10}$$

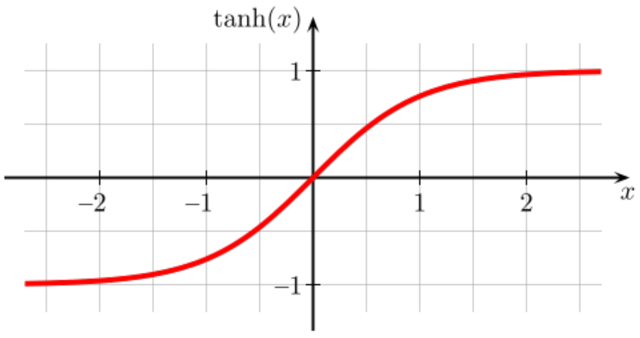


Figure 24: A plot of the hyperbolic tangent function, tanh(x) [42].

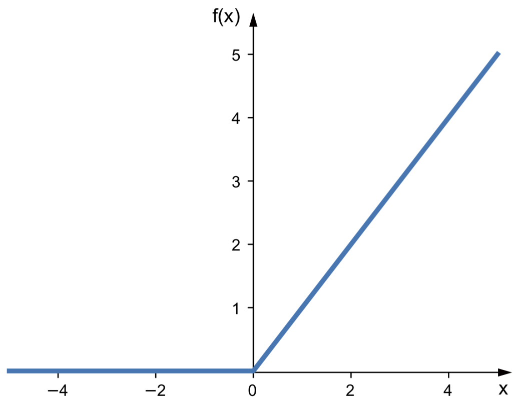


Figure 25: A plot of the Rectified Linear Unit [43]. It is observed that the value of the ReLU is equal to zero for all negative values of x, and otherwise equal to x.

3. Gradient descent and stochastic gradient descent

Consider a neural network for classification of images with an input vector, $\mathbf{x}$, and a desired output vector $\mathbf{y}(\mathbf{x})$. To find weights and biases for approximation of the desired output $\mathbf{y}(\mathbf{x})$ for the training inputs in $\mathbf{x}$, a *cost function* is defined. A *quadratic* cost function can be defined as in (11), where n is the total amount of training inputs and $\mathbf{a}$ is the actual output from the neural network, given $\mathbf{x}$, $\mathbf{w}$ and $\mathbf{b}$. The cost function in (11) is also known as the *mean square error* (MSE) [40]. In the field of optimization, a cost function is often referred to as an *objective function*.

$$C(\mathbf{w}, \mathbf{b}) = \frac{1}{2n} \sum_{x} \|\mathbf{y}(\mathbf{x}) - \mathbf{a}\|^2 \tag{11}$$

The *delta rule* is a training rule that converges to the best approximation, given training examples that are not linearly separable. To find weights and biases that fit the training examples best, the delta rule uses what is called a *gradient descent* algorithm to search a hypothesis space for possible weight and bias vectors [33]. The aim of using a gradient descent algorithm is to *minimize* the training error, (11). Thus, $C(\mathbf{w}, \mathbf{b}) \approx 0$, is the preferable result from using a gradient descent algorithm. For explanation of this algorithm in general, imagine a theoretical cost function of weights, $J(\mathbf{w})$, similar to (11), where each element in **w** could be a function of n variables, $w = w_0, w_1, ..., w_n$. The objective is to minimize $J(\mathbf{w})$, to find its *global minimum* [40]. To illustrate what a global minimum is, lets assume one of the elements in **w** has two variables, w_0 and w_1. The error surface created by the two variables and the corresponding values of the cost function could take on a form similar to the one in figure 26.

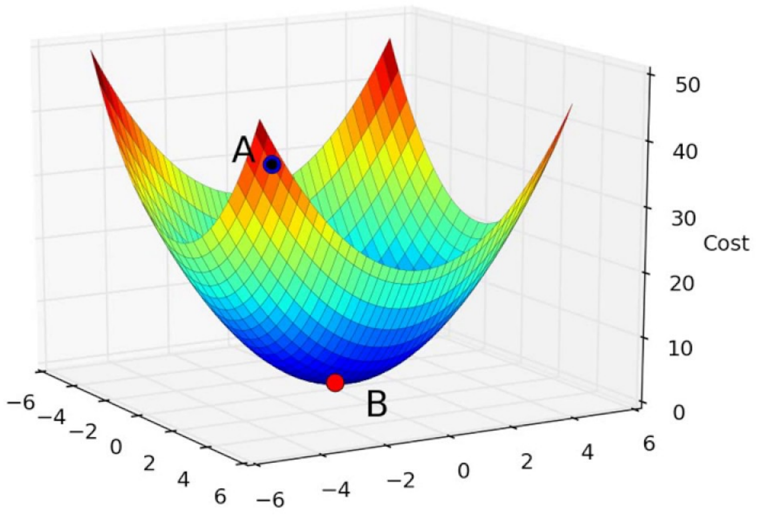


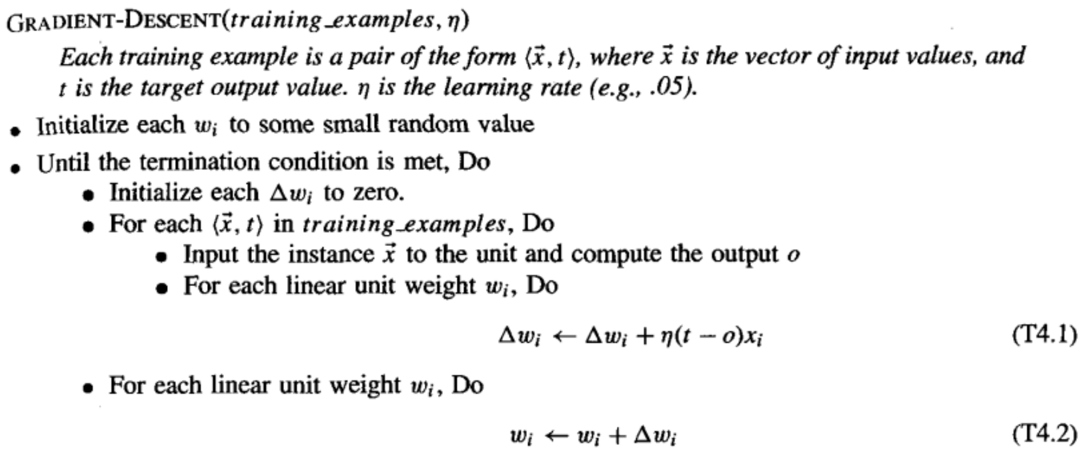
Figure 26: A simple illustration of what is desirable to achieve with the gradient descent algorithm [44]. Assuming w_0 and w_1 are the variables on the two horizontal axes in the figure, while $J(w_0, w_1)$, the cost, is on the vertical axis. Then, point A can be a point corresponding to the initial weight values of w_0 and w_1, while point B illustrates the global minimum of the cost function.

In order to find the global minimum, the direction on the error surface in which to go must be found, often referred to as the *steepest descent direction*. Through computing the *gradient* of the cost function, $\nabla J(\mathbf{w})$, the derivative of the cost function with respect to the components of $\mathbf{w}$, this direction can be found [33]. For line search methods, the steepest descent direction is considered the most obvious choice because it is the direction where $J(\mathbf{w})$ has the quickest decrease [45]. The gradient of $J(\mathbf{w})$ is given in (12), and $-\nabla J(\mathbf{w})$ specifies the steepest descent direction. A training rule for gradient descent is shown in (13), where η is the *learning rate* or the *step size*, a positive constant, in the search for the global minimum [33]. The gradient descent algorithm is illustrated in figure 27.

$$\nabla J(\mathbf{w}) = \left[\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \ldots, \frac{\partial J}{\partial w_n}\right] \tag{12}$$

$$\mathbf{w} \longleftarrow \mathbf{w} + \Delta\mathbf{w} \tag{13a}$$

$$\Delta\mathbf{w} = -\eta \nabla J(\mathbf{w}) \tag{13b}$$



```
GRADIENT-DESCENT($training\_examples, \eta$)
    Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and
    $t$ is the target output value. $\eta$ is the learning rate (e.g., .05).
• Initialize each $w_i$ to some small random value
• Until the termination condition is met, Do
    • Initialize each $\Delta w_i$ to zero.
    • For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
        • Input the instance $\vec{x}$ to the unit and compute the output $o$
        • For each linear unit weight $w_i$, Do
                    $\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$                (T4.1)
    • For each linear unit weight $w_i$, Do
                    $w_i \leftarrow w_i + \Delta w_i$                (T4.2)
```

Figure 27: The gradient descent algorithm for training of a linear unit [33].

Stochastic gradient descent (SGD) is a variation of gradient descent. The main difference between these two approaches is that in SGD the error for each training example is calculated and weights are updated *incrementally*, while gradient descent sums over all training examples and then calculates the weight updates. SGD is also called incremental gradient descent [33]. The SGD algorithm chooses a *mini-batch*, a sample of randomly chosen training examples, and approximates the gradient descent search by estimating (12) as the average gradient for all the training examples in the chosen mini-batch [40]. Summing over all training examples and then computing the gradient is in general computationally expensive, and SGD is therefore often a preferred approach. Concurrently, since gradient descent utilizes the real gradient, a larger step size can be used compared with the stochastic gradient descent approach where the gradient is approximated [33]. Stochastic gradient descent with momentum (SGDM) is an extension of SGD. A momentum is added as a factor to an extra term in the training rule in (13), deciding how much the previous step should contribute to the next. The momentum is added to reduce oscillations, which can occur using SGD [46].

4. The backpropagation algorithm

A common algorithm used for training multi-layer networks, such as the one illustrated in figure 20, is the *backpropagation* algorithm. The backpropagation algorithm makes these types of networks able to express a large variation of nonlinear decision surfaces. The algorithm uses SDG to minimize a quadratic cost function, and thus learn the optimal weights for a neural network [33]. The complete details and mathematics of backpropagation will not be explained here, but the general concept and some relevant equations are introduced.

The cost function in (11) is slightly rewritten in (14) to serve as an example for explanation of backpropagation. The number of training examples are denoted n, **x** are inputs and **y** the desired outputs, L denotes the number of network layers and $\mathbf{a}^L(\mathbf{x})$ are the actual outputs or *activations*, given **x** [40].

$$C(\mathbf{w}, \mathbf{b}) = \frac{1}{2n} \sum_x \|\mathbf{y}(\mathbf{x}) - \mathbf{a}^L(\mathbf{x})\|^2 \tag{14}$$

The aim of using backpropagation is to determine $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$, for all weights and biases. Some assumptions about (14) have to be made, specifically that it can be written as $\frac{1}{n}\sum_x C_x$, an average of cost functions for all individual inputs, x, and that it also can be formulated as $costC = C(a^L)$, a function of the output activations only (illustrated in figure 28) [40].

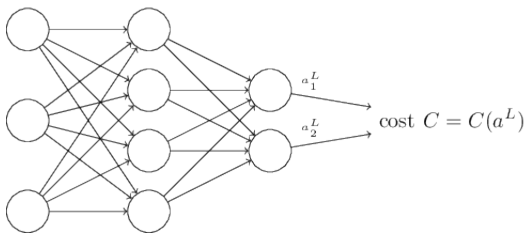


Figure 28: Illustration of two output activations , a_1^L and a_2^L, and a cost function as a function of these [40].

As illustrated in figure 28, activations in the output layer become inputs to a cost function, which tells how close to the desired output the activations are. There are four fundamental equations of backpropagation that together explain how the error and the gradient of the cost function is calculated and propagated backwards, starting from the final output layer. These four equations will not be explained in detail, but are shown in the algorithm for backpropagation below (step 3-5) [40]. A simple illustration of error propagating backwards in a neural network is given in figure 29.

1. **Input**, x: Set the corresponding activation, a^l, for the input layer.
2. **Feedforward**: For each $l = 2, 3, ..., L$, calculate $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.
3. **Output error**, δ^L: Compute $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Backpropagate the error**: For each $l = L-1, L-2, ..., 2$ calculate $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. **Output**: $\frac{\partial C}{\partial w^l_{jk}} = a_k^{l-1}\delta_j^l$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$, provides the gradient of the cost function.

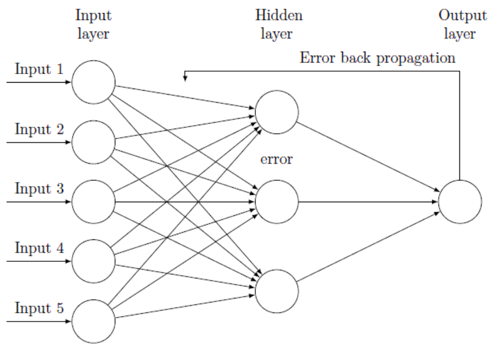


Figure 29: Illustration of the backpropagation principle in an ANN with one hidden layer [47]. The principle is the same for networks with multiple hidden layers, where errors then propagate back through several layers.

5. Training, validation and common challenges related to ANN

The underlying process for training an ANN is shown through the introduction of gradient descent, SGD and the backpropagation algorithm. Provided labelled training examples, the objective is to train a neural network so that it is capable of separating, or classifying, the inputs it is given in a desirable manner. There are some challenges related to this that will be introduced here, as well as existing techniques to address these challenges. Common methods for testing and evaluating performance of a neural network are also explained.

A common challenge related to supervised machine learning, thus also deep learning, is called *overfitting*. Overfitting occurs when weights and biases are tuned so that the neural network learns details, or peculiarities, about the training examples which makes it unable to generalize the given input. This is because the method of backpropagation can become overly complex after sufficiently many iterations of weight tuning, making it fit noise and unrepresentative properties in the training data [33].

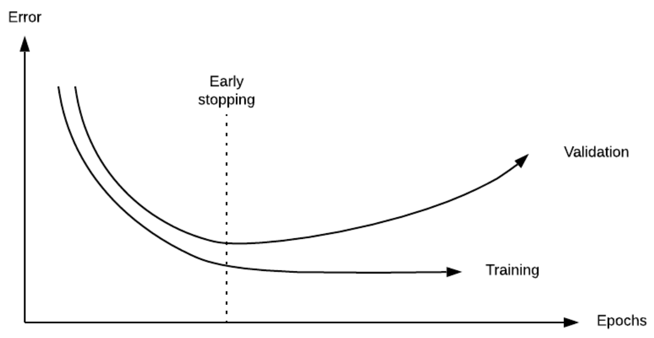


Figure 30: Illustration of curves for training and validation error over a certain number of epochs. The dashed line illustrates a point where the validation error increases even if the training error still decreases. This is the point where the validation error is at its lowest, and where early stopping is recommended to prevent overfitting [48]. The figure is made using Lucidchart [49].

In general, the original training data is split into three sets; a *training set*, *validation set* and a *test set*. The validation set is used to evaluate performance during training, while the test set is a final measure of performance after training. A common approach for detecting overfitting is utilization of the validation set and a method called *early stopping*, which says to stop training when the validation error increases, since this indicates overfitting of the training examples. Early stopping is a *regularization* technique for improving the ability of generalization in ANNs. It is common to use the validation error as an estimate of the generalization error [48]. Figure 30 illustrates how both the training and validation error is decreasing, but suddenly the validation error increases after a certain number of iterations, or *epochs*. A complete run through of the entire training set by a training algorithm is often referred to as an epoch.

Other examples of regularization techniques are *data augmentation*, *L1* and *L2* regularization, and *dropout*. Overfitting is a challenge that is most severe for small training sets, because generalization can be difficult when few examples are available [33]. Data augmentation is a technique for increasing the size of the training data through transformations and deformations like rotations, translations, mirroring and scaling of training examples. The labels on training examples are not changed after data augmentation [50]. L1 and L2 regularization are common techniques for reducing overfitting. In both techniques, an extra term, a *penalty* or regularization term, is added to the cost function. The difference between these two techniques is the regularization term, shown in (15) an (16), where cost functions for the two techniques are defined. λ is a strictly positive parameter known as the *regularization parameter* [40] [51].

$$C = \frac{1}{2n}\sum_x \|y - a^L\|^2 + \frac{\lambda}{2n}\sum_w |w| \tag{15}$$

$$C = \frac{1}{2n}\sum_x \|y - a^L\|^2 + \frac{\lambda}{2n}\sum_w w^2 \tag{16}$$

Dropout regularization is also a technique for reducing or preventing overfitting, where the method is to generate a probability, p, as number between 0 and 1 for each neuron. A neuron is “dropped out” from the neural network if the probability associated with the neuron is less than p. This also means that all connections from the neuron are blocked [52]. Figure 31 illustrates how flow from neurons in a layer is blocked with probability p when dropout is activated.

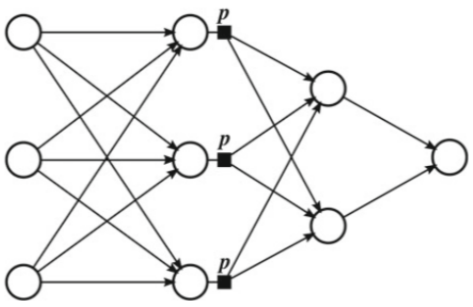


Figure 31: Illustration of the dropout technique, where information both forward and backward through neurons in a layer is blocked with probability, p, when dropout is activated [52].

Other challenges related to backpropagation and training of neural networks are vanishing and exploding gradients. In networks with many layers, gradients may be become very small, and even vanish, because of a long signal flow [48]. The opposite challenge is exploding gradients, when gradients become very large in early layers of a network. The challenges with vanishing and exploding gradients can be seen as a result of a more fundamental problem; unstable gradients. Vanishing gradients can in general be prevented by using an activation function that has a sufficient range [40]. One example of an activation function that can be used to prevent vanishing gradients is the ReLU. Exploding gradients can be prevented by reducing the learning rate, or by the use of *batch normalization* [51]. For convolutional neural networks, batch normalization is often placed between what is callled the fully connected layer and the associated activation function [52].

6. Convolutional neural networks

This part of the section regarding ANNs and deep learning will introduce convolutional neural networks, including a brief description of the concept and explanation of common layers and architectures of these types of neural networks. The next paragraph is taken directly from [6].

There is a class of neural networks called *convolutional neural networks* (CNN or ConvNets) that provides popular tools for image analysis. CNNs are highly suitable for analyzing inputs like images, text, and different continuous signals. This class of neural networks is inspired by the biological structure of a visual cortex, consisting of cells that are activated by subregions of a visual field called receptive fields. Unlike other types of neural networks, neurons in a convolutional layer connect to a subregion of a layer before the layer. Subregions may overlap, resulting in neurons producing spatially-correlated results. Figure 32 illustrates several different layers producing output suggestions based on the input image [53] [6].

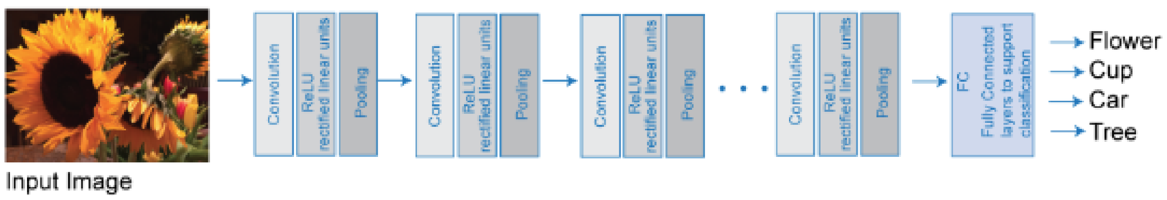

Input Image
Convolution
ReLU rectified linear units
Pooling
Convolution
ReLU rectified linear units
Pooling
Convolution
ReLU rectified linear units
Pooling
Convolution
ReLU rectified linear units
Pooling
FC
Fully Connected layers to support classification
Flower
Cup
Car
Tree

Figure 32: Illustration of typical modules in a CNN, each consisting of three layers; convolution, activation function (here:ReLU) and pooling. CNNs also have layers called fully-connected (FC) layers, the final layer in this illustration [53].

In general, a CNN consists of multiple layers. Some of the main types of layers are explained below [54]:

- **Input layer**: For an image input layer, this is where the type of network is specified, and choices regarding image size and data augmentation are made.
- **Convolution**: In a 2D convolutional layer, sliding filters are applied to the input to extract relevant information that is passed further into the network. This layer can consist of several components like: filters and stride, dilated convolution, feature maps, zero padding and learning parameters. A filter can be explained as a set of weights performing an operation on a region of the input. The input is convolved by the filter, using a matrix commonly called a *kernel*. Stride is a name for the step size of the filter, how it moves on the input. Figure 33 illustrates how a theoretical 3×3 filter is applied some input and creates a new output.

- **ReLU**: It is common to use the ReLU as an activation function. The ReLU was introduced earlier in this section under *Principles of neural networks* and illustrated in figure 25. The activation function performs a threshold operation to all elements of the input.
- **Pooling**: In a pooling layer, down-sampling is performed in order to reduce the number of connections and parameters, to simplify learning for the layers that follow. Examples of pooling techniques are *max-pooling* and *average-pooling*. Max-pooling divides the input into rectangular regions, and calculates the maximum of all these regions, while average-pooling calculates the average value of all regions after dividing the input.
- **Fully connected**: In a fully connected layer, each neuron is connected to the neurons in the preceding layer. Features from the preceding layer are combined by the fully connected layer to identify larger patterns. In image classification, this layer combines features to be able to perform the classification.
- **Output layers**: Examples of output layers are softmax and classification layers, or regression layers. The softmax activation function is defined in (17), where a_j^L is the activation of the jth neuron, z_j^L are the weighted inputs, and the denominator is the sum of all output neurons. The sum of all output activations from the softmax function is equal to 1 [40].

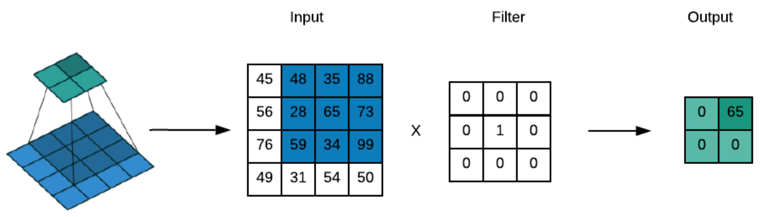


Figure 33: Illustration of a filtering example, where a 4×4 input is applied a 3×3 filter, and a 2×2 output is made. The theoretical filter, or kernel, shown here multiplies all numbers in the 3×3 region of the input with zero, except from the number in the middle which is multiplied by one. All multiplications are summed, and since 65 is the only element not multiplied with zero, this becomes one of the elements in the new output. The first part of this figure to the left is taken from [54], while the rest showing the filter operation is made using Lucidchart [49].

$$a_j^L = \frac{\exp z_j^L}{\sum_k \exp z_k^L}, \quad \sum_j a_j^L = 1 \tag{17}$$

7. Transfer learning

The idea behind transfer learning is to reuse a previously trained neural network, and train it to be able to classify a new set of inputs. Transfer learning is often referred to as *fine-tuning* an existing network, and this is in general faster than having to train a network from the beginning with arbitrarily initialisation of weights. For image classification, an important advantage with transfer learning is that fewer training images are required, because features are transferred from the existing network to solve the new classification problem [55]. Figure 34 illustrates a typical workflow for transfer learning.

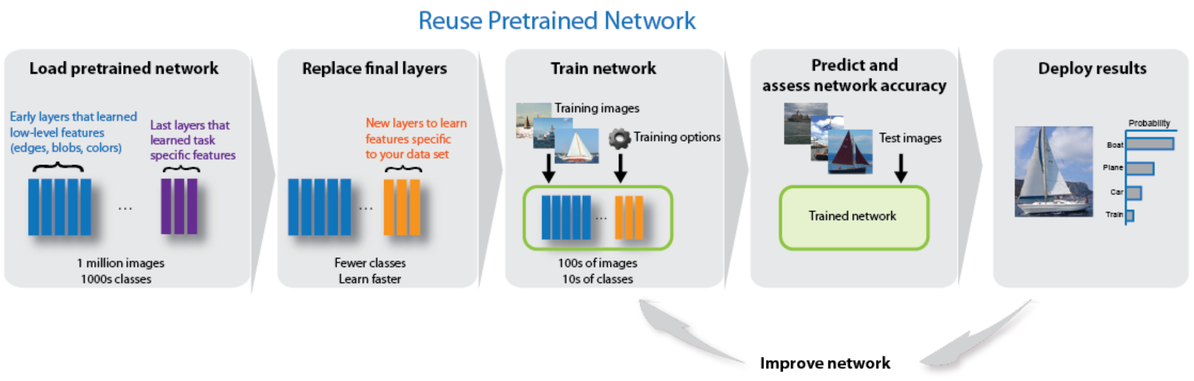


Figure 34: Illustration of a typical workflow for transfer learning. A pre-trained network, for example trained on 1 million images and 1000 classes, is first modified to fit the number of classes in the new classification problem. This involves replacing final layers, and maybe it is necessary to add or remove certain layers. Next, the network can be trained on much less images than the original network was (for example 100s of images and 10s of classes), and tested and fine-tuned until the desired accuracy is achieved [55].

Chapter 3

Previous work and literature study

This chapter will introduce existing neural networks, software and previous work relevant for image classification using deep learning approaches. Regarding networks, emphasis will be placed on pre-trained CNNs for image classification, and some of the most well known existing network architectures are presented here. Relevant software and frameworks for deep learning are also introduced. Previous work on automatic corrosion detection and evaluation using deep learning is presented, including types of neural networks used, software and results from image classifications.

3.1 Existing neural networks and relevant software

In section 2.3.3, neural networks and deep learning was introduced. There are many types of neural networks, and both their size and design can vary. Furthermore, there exists several combinations of software and network, depending on factors like the type and difficulty of the learning problem. This section will introduce existing convolutional neural networks and relevant software used in deep learning, to gain an overview of the possibilities in image classification. A selection of existing neural networks are introduced and explained, before relevant software is presented.

AlexNet

In 2010, a deep convolutional neural network, later called *AlexNet*, was trained to classify 1.2 million images into 1000 categories in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [56]. Approximately 50 000 images were used for validation, and 150 000 for testing. This neural network has 60 million parameters and 650 000 neurons. A version of this model also won the ILSVRC in 2012 [57]. AlexNet consists of five convolutional layers, max-pooling layers and three fully-connected layers. The input images to the network has to be RGB images and have a size of 256×256 pixels. Images of a different dimension can be resized before used as input. From inside the 256×256 images, random crops of size 227×227 are generated and used in the first layer of the network [58]. The architecture of AlexNet is illustrated in figure 35. Regarding the approach for network training, SGD was used with an initial learning rate of 0.01 which was reduced three times during training. The learning rate was adjusted manually and divided by 10 when validation error did not improve with current learning rate. The mini-batches contained 128 training examples each. Network training was set to approximately 90 epochs, or cycles, and this took between five to six days to complete using two graphics processing units (GPUs) of the type NVIDIA GTX 580 3GB [57].

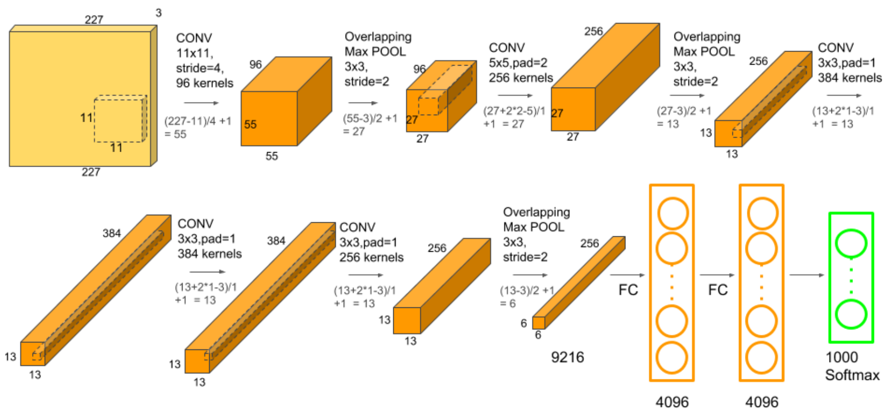

227
3
11
11
227
CONV 11x11, stride=4, 96 kernels
(227-11)/4 +1 = 55
96
55
55
Overlapping Max POOL 3x3, stride=2
(55-3)/2 +1 = 27
96
27
27
CONV 5x5,pad=2 256 kernels
(27+2*2-5)/1 +1 = 27
256
27
27
Overlapping Max POOL 3x3, stride=2
(27-3)/2 +1 = 13
256
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 256 kernels
(13+2*1-3)/1 +1 = 13
256
13
13
Overlapping Max POOL 3x3, stride=2
(13-3)/2 +1 = 6
256
6
6
9216
FC
4096
FC
4096
1000 Softmax

Figure 35: The architecture of AlexNet, consisting of five convolutional layers, max-pooling layers and three fully-connected layers [58]. Both the first and the second convolutional layer are followed by a ReLU and a normalization layer before max pooling. The third, fourth and fifth convolutional layer are followed by a ReLU only before max pooling. The first fully-connected layer is followed by a ReLU and a dropout layer. After the last fully-connected layer there is a softmax activation function that produces a distribution over the 1000 labels [57].

ZF Net

The architecture of ZF Net is very similar to AlexNet. However, some modifications to layer 3, 4 and 5 were done through changing from sparse to dense connections. This is related to the choice of GPU. An illustration of the network architecture is given in figure 36. SDG was used, with a mini-batch size of 128 images and initial learning rate 0.01. The network was trained for 70 epochs which took 12 days using a single GTX580 GPU [59].

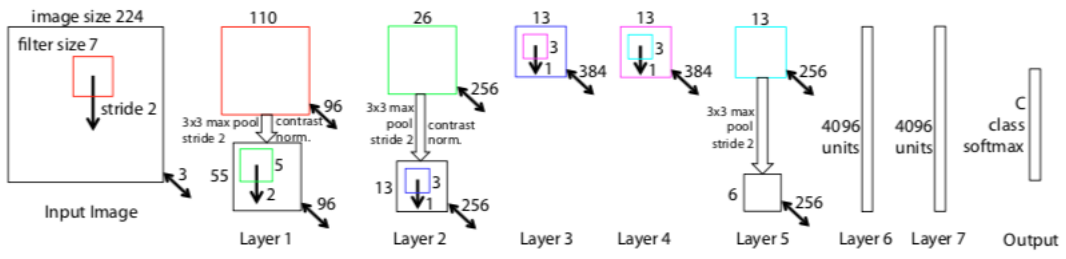


Figure 36: Illustration of the ZF Net architecture [59].

Inception (GoogLeNet)

In ILSVRC 2014, a convolutional neural network called *GoogLeNet*, based on the *Inception* architecture, became the new state of the art in detection and classification of images. This network was, like AlexNet, trained on 1.2 million images, but the validation and test sets contained 50 000 and 100 000 images, respectively. GoogLeNet has 22 layers and 12 times fewer parameters than AlexNet. Inputs to the network are RGB images with size 224×224 pixels. The network architecture consists of "Inception modules" stacked on top of each other. These modules contain several convolutions and max pooling, and are illustrated in figure 37 [60].

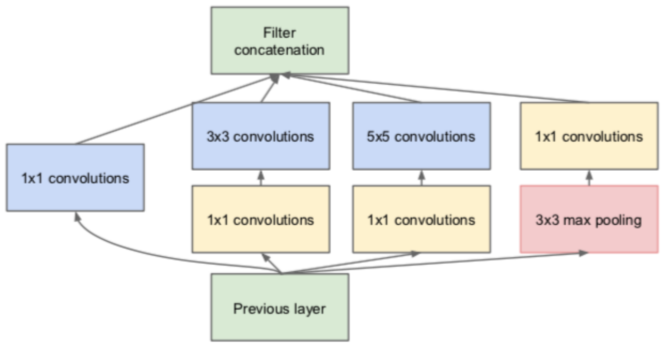


Figure 37: Illustration of an Inception architecture module containing convolutions and max pooling [60]. The filtered output from one module becomes the input to the next.

VGG

A different submission for the ILSVRC 2014 was two convolutional networks with an architecture of 16 and 19 layers, presented by a team called *VGG*. These two network architectures are sometimes referred to as VGG-16 and VGG-19. The inputs to both networks are RGB images with size 224×224 pixels. The distribution of images in training, validation and test sets was the same as for GoogLeNet explained previously. Regarding the training procedure, this was very similar to the one for AlexNet. However, the size of the mini-batches was 256, and training was stopped after approximately 74 epochs [61]. The VGG network architecture is illustrated in figure 38.

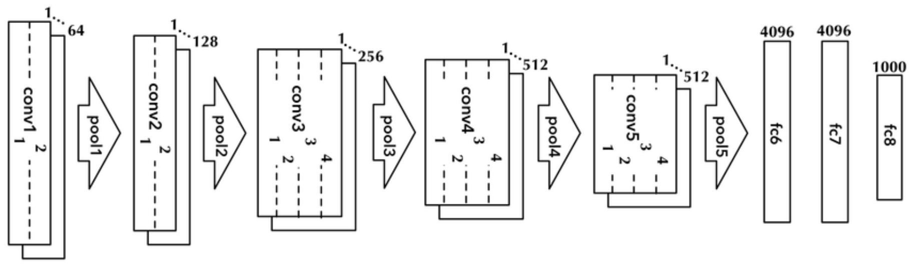


Figure 38: Illustration of the VGG network architecture. Two architectures, VGG-16 and VGG-19, was mentioned earlier, and the difference is the number of layers. The architecture illustrated here has 16 convolutional layers and 3 fully-connected layers, and is therefore denoted the VGG-19 network [62].

ResNet

A deep *residual* learning framework, hence the name *ResNet*, was designed to address the problem of degradation in deep network architectures. Degradation refers to the problem of accuracy saturation as the network depth increases. Adding more layers to an architecture can therefore lead to higher training error. The building block of residual learning is illustrated in figure 39.

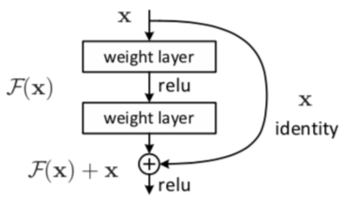

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 39: A “building block” of residual learning [63]

Considering an input, x. The desired underlying mapping can be denoted $H(x)$, while the nonlinear layers fit a mapping denoted $F(x) = H(x) - x$. Thus, the original mapping can be written as $H(x) = F(x) + x$. This is achieved through the use of a feed forward neural network with so called “shortcut connections” that skip one or more layers, as illustrated in figure 39 where the input, x, skips the weight layers and ReLU, and is added to the filtered input. This type of shortcut connection performs an identity mapping. One thought related to residual learning is that it is easier to optimize the residual mapping, $F(x)$, and drive this to zero, rather than to optimize the original mapping through fitting it by nonlinear layers. [63] presents several residual network architectures, from 18 to 152 layers. Network architectures without shortcut connections are often referred to as *plain* networks [63].

A brief comparison of networks

The networks presented in this section differs in size and architecture, and have also performed differently on tests. This part of the section aims to give a brief comparison of the networks that have been introduced. The comparisons here will be made from test results in image classification from ILSVRC.

Figure 40 compares the test error and number of layers for some of the network architectures that have competed in ILSVRC throughout the years. In ILSVRC 2012, AlexNet had a test error of 16.4 % when averaging the results from five similar CNNs, and a test error of 15.3% when averaging the predictions of seven similar CNNs [57]. Ensembles of ResNet architectures competed in ILSVRC 2015, and won the image classification task with a test error of 3.57 %. For comparison, GoogLeNet and VGG had a test error of 6.66 % and 7.32% respectively in ILSVRC 2014 [63]. Other networks that have shown great results in image classification the last years are *Trimps-Soushen*, winner of ILSVRC 2016 with a test error of 2.99% [64], and *WMW*, winner of ILSVRC 2017 with a test error of 2,25% [65].

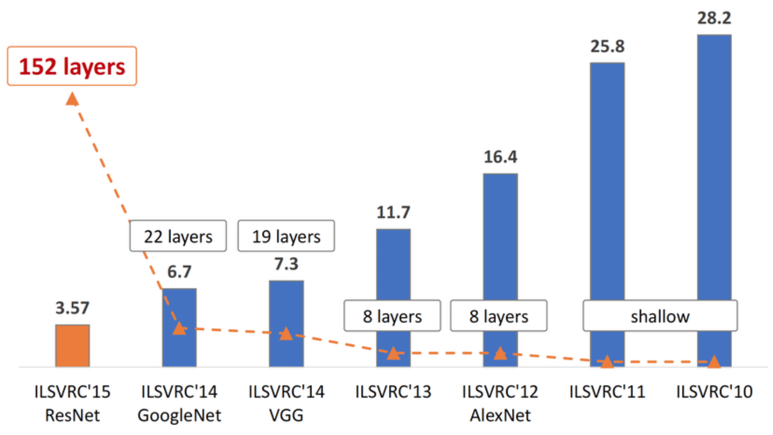


Figure 40: Illustration of how the number of layers and test error in image classification for networks competing in ILSVRC have changed throughout the years [66]. The value on the top of the bars show the test error in percent.

Relevant software and frameworks

There exists a large variety of software and frameworks for implementation of neural networks. In this part of the section, examples of commercial software and frameworks relevant for deep learning are presented.

Python [67] is a programming language with applications in fields like web-and Internet development, scientific and numeric computing, and software development. Tensorflow [68] is an open source machine learning library that is compatible with Python, and Keras [69] is an application programming interface (API) often used in combination with TensorFlow. PyTorch [70] is an open source deep learning platform also compatible with Python.

MATLAB [71] is a programming platform that provides solutions within, for example, data analytics, signal processing, robotics, computer vision and deep learning. A framework for design and implementation of neural networks is provided by the Deep Learning Toolbox [72], and pre-trained networks like AlexNet, ResNet, VGG-19 and GoogLeNet are accessible for performance of transfer learning. Originally Python based frameworks like Keras+TensorFlow and PyTorch are also accesble in MATLAB, either directly or through ONNX [73], an open format for representation of deep learning models. Caffe [74], another example of an existing deep learning framework, can be used with both MATLAB and Python.

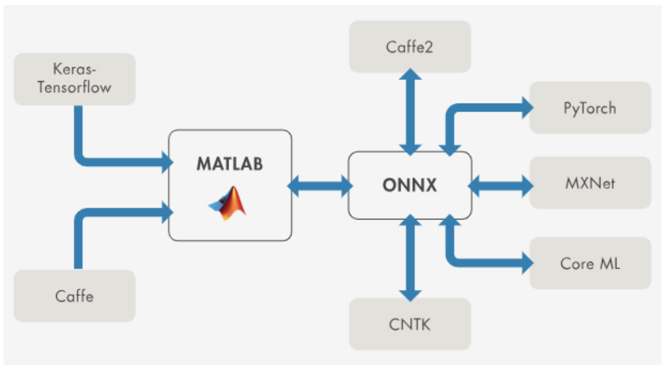


Figure 41: Interoperability in MATLAB using ONNX, which makes it possible to access frameworks like Caffe2, PyTorch, MXNet, Core ML and CNTK. The figure also shows that frameworks like Caffe and Keras-TensorFLow are directly available in MATLAB [75].

3.2 Previous work on classification of corrosion damages

This section will introduce previous work on automatic detection and evaluation of corrosion damages, including software and networks used, methods and modifications, and results from image classifications. First, previous work on corrosion *detection* is introduced, where the goal is to detect if corrosion exists on an image, or not. Next, previous work on *evaluation* of corrosion damages is presented. Evaluations can for example be done through labelling images after the severity on the corrosion damage, or after forms of corrosion damages, as introduced in section 2.1.2.

Corrosion *detection* using deep learning

In [7] and [8], different approaches for corrosion detection from images using deep learning are tested. Some of the methods and results presented in these two papers will be introduced in this section. Most of the summary from [8] presented here is taken directly from [6].

[7] compares standard computer vision techniques using OpenCV [76] with a deep learning approach for detection of corrosion from images taken during bridge inspections. The OpenCV approach, concerning counting of red pixels/components in images, will not be further explained here. For the deep learning approach, Python and the Caffe framework was used. A network called "$bvlc\_reference\_caffenet$", based on AlexNet, was tuned for corrosion detection. A dataset of 3500 images, 2200 without corrosion and 1300 containing corrosion, was used for training and validation of the network (80% for training and 20% for validation). Regarding the training process, the learning rate was set to 5×10^{-5} and the number of iterations was 100 000. Using Ubuntu 14.04 with GPU CUDA support, a i7 Skylake CPU and a Nvidia GTX 980 Ti GPU, the network was trained for approximately two days. The test set included 100 images, with 37 of these containing corrosion. Examples of images in the test set are shown in figure 42. Results from tests performed in [7] are given in table 5. The positive class is *corrosion*, while the negative class is *not corrosion*. Thus, a false positive is an image that does not contain corrosion being classified as *corrosion*, while a false negative is an image that contains corrosion being classified as *not corrosion*.

Table 5: Results from OpenCV approach and network testing in [7].

	OpenCV	**Deep learning**	**Deep learning Probability>0.8**
False positives	27/63	14/63	5/51
Partial accuracy for “not corrosion”	57 %	78 %	90 %
False negatives	4/37	8/37	7/34
Partial accuracy for “corrosion”	89 %	78 %	79.4 %
Total accuracy	69 %	78 %	88 %

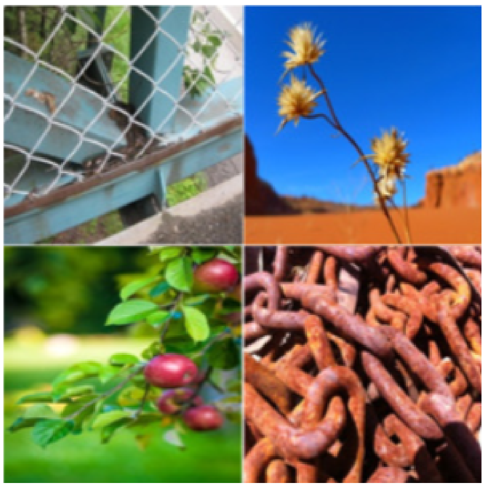


Figure 42: Examples of images in the test set used in [7].

Table 5 presents a total accuracy of 78 % for the deep learning approach. Out of 100 images, 15 images in the test set gave a confidence level on classification of less than 80 %. When these images were disregarded, the network had an accuracy of 88 %. The network classified all images in figure 42 correctly, even though images of for example apples were not included in the training set. For future work, the paper suggests refining the created deep learning model for corrosion detection, and to train it with even more images for possibly achieving a higher accuracy than obtained with the existing model.

In [8], deep learning approaches based on CNNs for corrosion detection are evaluated. The study presented in this paper aims to quantify performance of corrosion classification and find optimal inputs for a CNN to create robust systems for automatic corrosion detection. Figure 43 shows a basic structure of a CNN with output predictions "non-corroded" and "corrosion", and this is the type of structure used in the study to classify image regions as corroded or not.

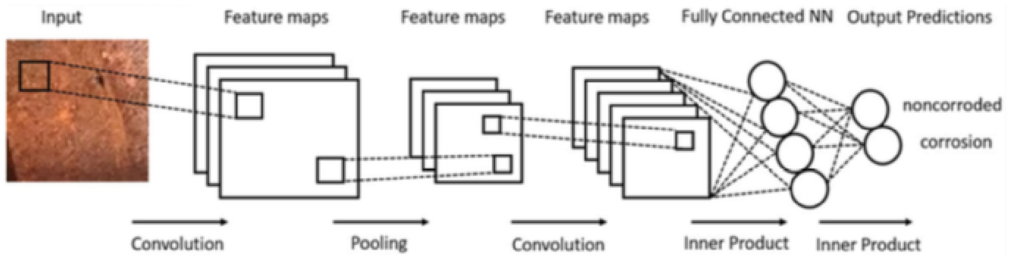


Figure 43: Illustration of a basic CNN architecture for corrosion detection on images [8].

A total of 926 original images were cropped in order to generate even more images. Through cropping regions of corrosion and non-corrosion, 33039 images of corrosion and 34148 of non-corrosion were created. The distribution of images in a training, validation and test set was 70%, 10%, and 20%, respectively. Two types of existing networks, VGG16 and ZF Net, were used. The paper also proposes two network architectures named Corrosion5 and Corrosion7. Different colour spaces such as RGB, YCbCr, CbCr and grayscale were tested to find the optimal colour space for corrosion detection. A sliding window approach was used to classify different regions of an image. This is illustrated in figure 44.

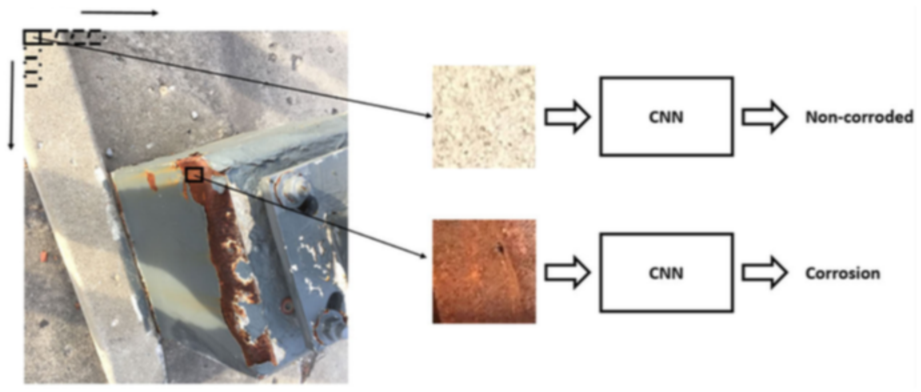
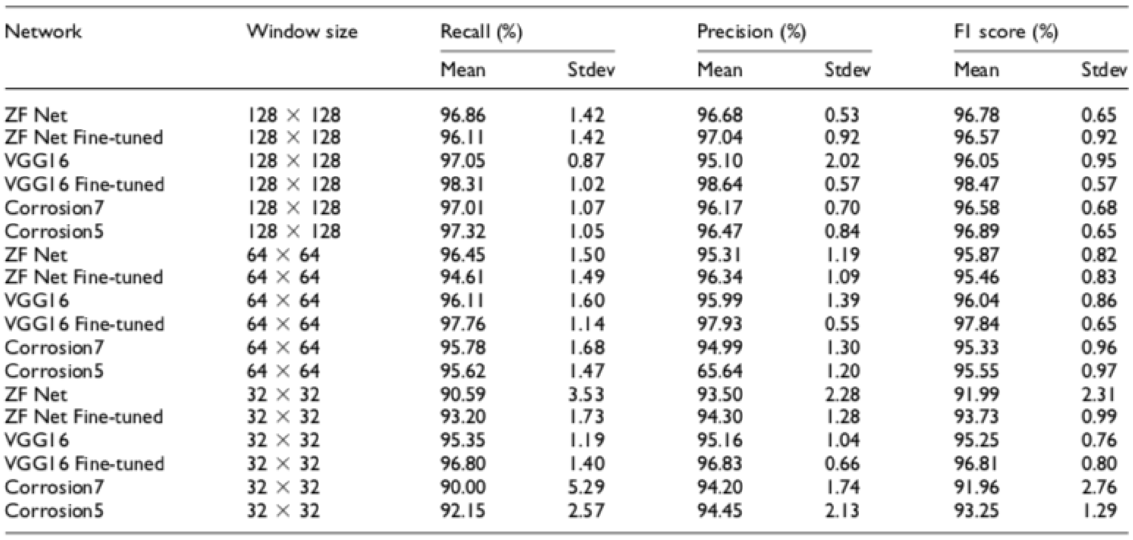


Figure 44: Illustration of region classification using a sliding window approach [8].

The following training options for the CNNs were chosen:

- Number of epochs: 22 (8800 iterations)
- Mini-batch size: 256
- Learning rate: 1×10^{-3}

Through testing the two networks with different colour spaces and sliding window sizes it was found that the best input parameters were the RGB and YCbCr colour spaces, using a sliding window size of 128×128. After fine-tuning, the VGG16 network turned out to be the most robust architecture. However, with no fine-tuning, the two networks performed equally. Results after testing of the different CNN architectures on RGB images are shown in figure 45. For future work, the paper suggests using the results found to look at the possibilities for classification of different types of corrosion using CNNs [8] [6].



Network	Window size	Recall (%)		Precision (%)		FI score (%)	
		Mean	Stdev	Mean	Stdev	Mean	Stdev
ZF Net	128 × 128	96.86	1.42	96.68	0.53	96.78	0.65
ZF Net Fine-tuned	128 × 128	96.11	1.42	97.04	0.92	96.57	0.92
VGG16	128 × 128	97.05	0.87	95.10	2.02	96.05	0.95
VGG16 Fine-tuned	128 × 128	98.31	1.02	98.64	0.57	98.47	0.57
Corrosion7	128 × 128	97.01	1.07	96.17	0.70	96.58	0.68
Corrosion5	128 × 128	97.32	1.05	96.47	0.84	96.89	0.65
ZF Net	64 × 64	96.45	1.50	95.31	1.19	95.87	0.82
ZF Net Fine-tuned	64 × 64	94.61	1.49	96.34	1.09	95.46	0.83
VGG16	64 × 64	96.11	1.60	95.99	1.39	96.04	0.86
VGG16 Fine-tuned	64 × 64	97.76	1.14	97.93	0.55	97.84	0.65
Corrosion7	64 × 64	95.78	1.68	94.99	1.30	95.33	0.96
Corrosion5	64 × 64	95.62	1.47	65.64	1.20	95.55	0.97
ZF Net	32 × 32	90.59	3.53	93.50	2.28	91.99	2.31
ZF Net Fine-tuned	32 × 32	93.20	1.73	94.30	1.28	93.73	0.99
VGG16	32 × 32	95.35	1.19	95.16	1.04	95.25	0.76
VGG16 Fine-tuned	32 × 32	96.80	1.40	96.83	0.66	96.81	0.80
Corrosion7	32 × 32	90.00	5.29	94.20	1.74	91.96	2.76
Corrosion5	32 × 32	92.15	2.57	94.45	2.13	93.25	1.29

Figure 45: Performance for different CNN architectures from testing on RGB images [8].

Corrosion *evaluation* using deep learning

In [77], tests on classification of grounding grid corrosion damages from images using CNNs were performed. For implementation of the CNNs, the authors used MATLAB and the Deep Learning Toolbox. Four different categories of corrosion damages were chosen as labels for the images; very mild, mild, moderate and severe. Through the use of Photoshop [78] image processing software, 2500 images of each category were pre-processed and given the size of 32 $\times$ 32 pixels. The data set of images was divided into a training set and a test set, containing 80% and 20% of the images respectively. A convolutional neural network architecture called LeNet-5 [79] was modified for use in this paper. Two different classifiers were used; Softmax and SVM. Results from training and tests of original and improved CNNs performed in [77] are shown in table 6. The training options were set to the following, using the SGD algorithm:

- Number of epochs: 50
- Mini-batch size: 20
- Learning rate: 0.1
- Momentum: 0.95

Table 6: Results from network training and testing in [77].

Algorithm	**Pre-processing sample bank**	
	Training set recognition rate	Test set recognition rate
CNN-Softmax	83.14 %	79.26 %
CNN-SVM	85.21 %	81.55 %
Improved CNN-Softmax	87.58 %	84.09 %
Improved CNN-SVM	91.25 %	89.37 %

The improved CNNs differ from the original LeNet-5 architecture through use of the ReLU activation function, and the number of convolutional layers were changed from two to three. More details regarding the improved network architecture is found in [77]. The results in table 6 show that the improved CNNs have a higher accuracy both for training and testing, and that using a SVM classifier gave the highest overall accuracy.

Chapter 4

Methods

In this chapter, the methods for data collection and image categorization based on classes of corrosion and coating damages are explained. The setup of training and tests of convolutional neural networks is presented. MATLAB is the chosen software to be used in this thesis. The approach for implementation of networks in MATLAB is given, including image pre-processing, labeling of images, modifications on network layers and choice of training options such as the training algorithm, learning rate, mini-batch size and number of epochs.

4.1 Data collection

This section will explain the methods of data collection. A hard drive containing approximately 240 000 images from previous bridge inspections was received from The Norwegian Public Roads Administration, and this serves as the main source of images used in this thesis. Some images were downloaded directly from Brutus, a management system for bridges in Norway introduced in section 2.2.2. The data set from the hard drive contains images such as; corrosion damages on steel construction and rebar, deformation damages, missing construction elements, bridge overview images and vegetation around bridges. All the 240 000 images were manually evaluated, and applicable images were categorized in folders. This section will explain in detail how data collection and categorization was carried out.

For the very first network training and test, approximately 400 images were downloaded directly from Brutus and categorized into two classes; corrosion and not corrosion. The corrosion class contained images of corrosion both on steel construction and rebar. These images were collected before the hard drive was received, and served as a first set of images for setup and test of a script in MATLAB. The results from training and tests of the first network are shown in section 5.1.

Damages on bridge constructions like deformations, cracks, paint flaking, and corrosion on rebar and steel all vary in scope and shape within their respective categories. In the category of corrosion damages, the variation is quite large, and it is therefore necessary to choose reasonable classes of corrosion damages to be used as labels in network training. Table 7 shows different degrees of damages related to corrosion and coating damages on bridge constructions.

Table 7: Degrees of damages related to corrosion as classified by The Norwegian Public Roads Administration [80].

Degree of damage	**Explanation**
Degree 1	Discoloration of the surface
Degree 2	Pores/ crazing/ flaking
Degree 3	Decomposition to zinc or aluminium primer
Degree 4	Exposure of metal surface

With inspiration from table 7, it was chosen to categorize images from previous bridge inspections into five classes; (1) paint flaking, (2) white corrosion, (3) red corrosion on steel construction, (4) red corrosion on rebar and (5) not corrosion, as shown in table 8. The main focus in this thesis is classification of corrosion damages, but it is also of great importance to be able to detect a potential start of a corrosion damage. If a coating damage is detected early, necessary measures related to maintenance can be implemented in order to avoid a corrosion damage. Therefore, paint flaking is added as a class to be identified by neural networks. The white corrosion class is also useful because corrosion directly on the steel construction of a bridge can be avoided if damage on the TSZ coating is detected and repaired. The different coating layers used on the steel construction of a bridge were previously illustrated in figure 16, section 2.2.1. However, because very few images of white corrosion were found in the available data set, this class will not be used in network training. Classification of damages like cracks or deformations are not investigated in this thesis, but will be mentioned as possible future work later in the report. Table 9 shows the amount of images collected for each of the five classes. Figures 46, 47, 48, and 49 illustrate image examples within the categories 1, 3, 4 and 5.

Table 8: Classes of corrosion and surface damages manually categorized from the 240 000 images received from the Norwegian Public Roads Administration. All classes except from class 2-white corrosion will be used in network training and testing.

Number	**Class/label**	**Description**
1	Paint flaking	Flaking or peeling paint on steel. Some images contain small areas of beginning corrosion damages due to paint flaking.
2	White corrosion	Corrosion on zinc coating.
3	Red corrosion, steel construction	Corrosion on exposed steel on construction elements such as beams, wires and railings.
4	Red corrosion, rebar	Corrosion on exposed rebar in concrete of varying degrees.
5	Not corrosion	Everything that does not contain elements of the other classes above. Examples are; intact/undamaged concrete, cracks in concrete, undamaged steel, leaves, flowers, graffiti, trees, roads and cars. (for example: a red car, autumn leaves, and steel or woodwork painted red).

Table 9: Amount of images collected of each class and in total. The number in parentheses denotes the amount of images including augmented images.

Number	Class/label	Amount of images
1	Paint flaking	248 (2050)
2	White corrosion	31
3	Red corrosion, steel construction	2557
4	Red corrosion, rebar	2380
5	Not corrosion	2307
Total		7523 (9325)

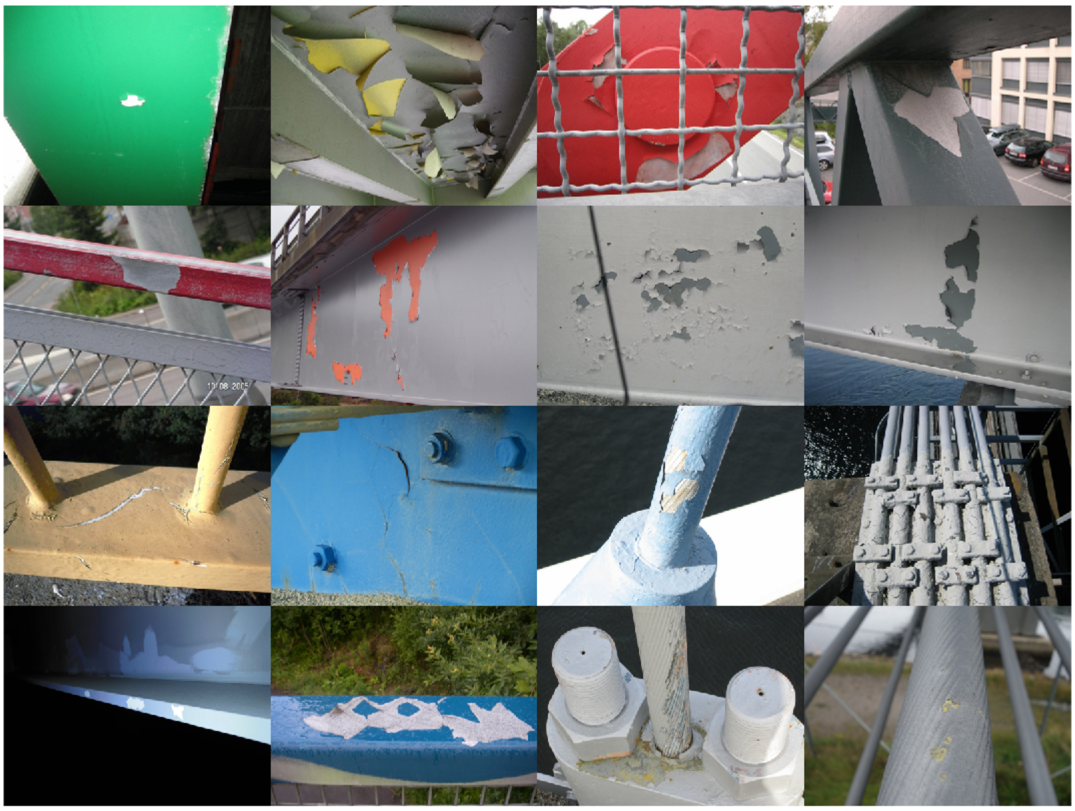


Figure 46: Examples of images collected in the class *paint flaking*.

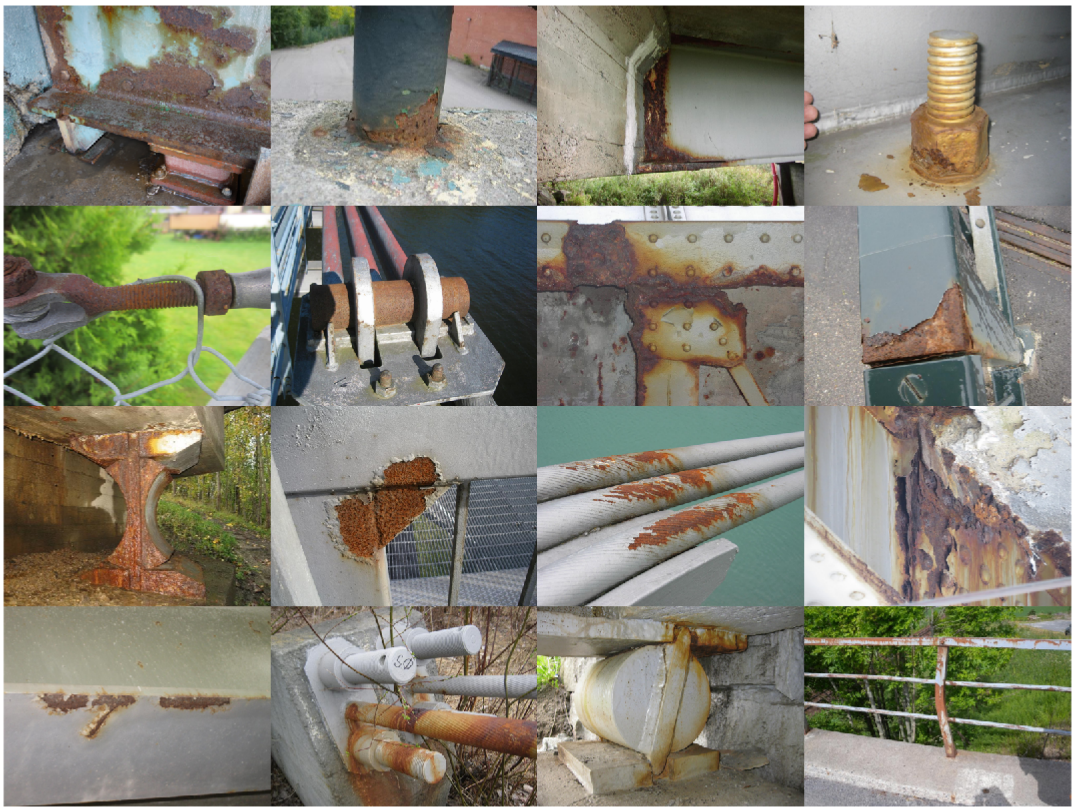


Figure 47: Examples of images collected in the class *red corrosion, steel construction*.

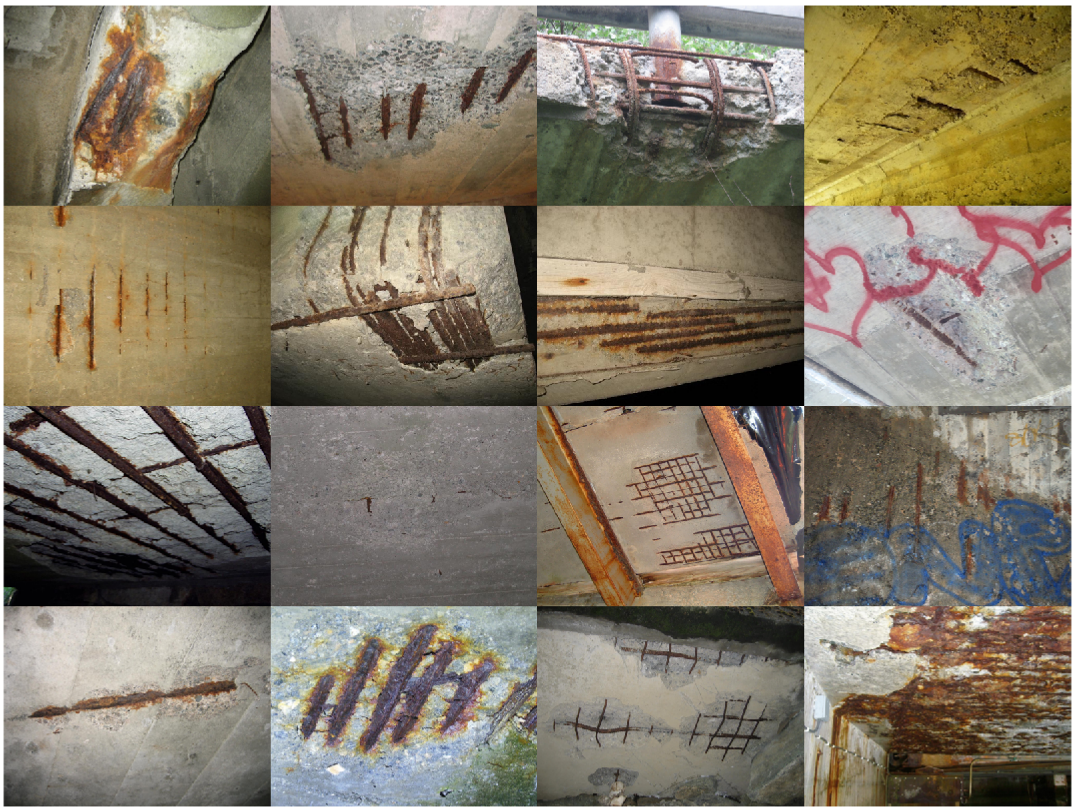


Figure 48: Examples of images collected in the class *red corrosion, rebar*.

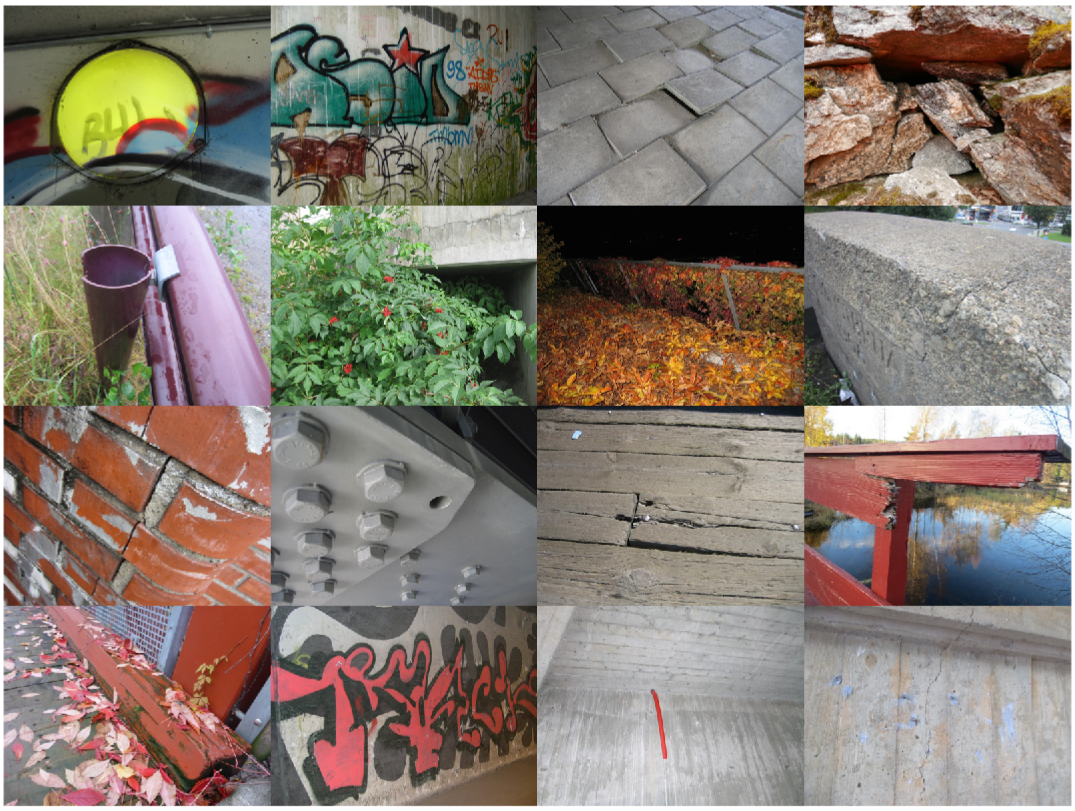


Figure 49: Examples of images collected in the class *not corrosion*.

4.2 Setup of training and tests

This section will explain the setup of training and tests for different convolutional neural networks. Relevant networks are AlexNet, GoogLeNet, ResNet-50 and VGG-16. The trained CNNs are named networks 1-9. The motivation for training and testing of network 1 was to become familiar with the setup of code in MATLAB, as well as to get an initial idea of how accurate a CNN could classify images of corrosion and not corrosion. Networks 2-5 were trained with three classes of images; *red corrosion on steel construction*, *red corrosion on rebar* and *not corrosion*, while networks 6-9 were trained with four classes, including the class *paint flaking*, in addition to the three classes used in networks 2-5. Available hardware and training options used for each CNN is presented in this section. The setup of tests is also explained, including relevant methods for testing of CNNs, and which metrics are used to evaluate classification performance for the different classes of images.

4.2.1 Training and validation

Table 10 shows the setup of CNN training, including relevant network architectures and the classes that are used for training of networks 1-9 in this thesis. The existing CNN architectures AlexNet, GoogLeNet, ResNet and VGG were introduced in section 3.1, and are some of the most well known neural networks for image classification. Each of these four CNNs have previously provided state of the art results within image classification, and were therefore considered highly relevant for comparison in this master's thesis. ResNet-50 and VGG-16 are the chosen versions of ResNet and VGG.

Table 10: Setup of network training through transfer learning using existing pre-trained networks. The class numbers with description are given in table 8.

Name	**Pre-trained network**	**Classes**
Network 1	AlexNet	Corrosion Not corrosion
Network 2	AlexNet	3,4,5
Network 3	GoogLeNet	3,4,5
Network 4	ResNet-50	3,4,5
Network 5	VGG-16	3,4,5
Network 6	AlexNet	1,3,4,5
Network 7	GoogLeNet	1,3,4,5
Network 8	ResNet-50	1,3,4,5
Network 9	VGG-16	1,3,4,5

Table 11: Available hardware for training of convolutional neural networks.

Name	**Pre-trained network**	**Hardware**
Network 1	AlexNet	Intel Core i7-8700 CPU 3.20 GHz
Network 2	AlexNet	Intel Core i7-8700 CPU 3.20 GHz
Network 3	GoogLeNet	Intel Core i7-8700 CPU 3.20 GHz
Network 4	ResNet-50	Intel Core i7-8700 CPU 3.20 GHz
Network 5	VGG-16	Intel Core i7-8700 CPU 3.20 GHz
Network 6	AlexNet	NVIDIA Titan X (Pascal) Single GPU
Network 7	GoogLeNet	NVIDIA Titan X (Pascal) Single GPU
Network 8	ResNet-50	Intel Core i7-6850K CPU 3.60 GHz
Network 9	VGG-16	Intel Core i7-6850K CPU 3.60 GHz

Table 12: Training options for network training, including training algorithm and parameters such as learning rate, momentum, mini-batch size and the number of epochs.

Network	**Training options**				
	Learning algorithm (MATLAB notation)	Initial learning rate	Epochs	Mini-batch size	Momentum
Network 1	SGD with momentum ('sgdm')	3×10^{-4}	6	10	0.90 (default)
Network 2-training 1	SGD with momentum ('sgdm')	1×10^{-1}	10	20	0.95
Network 2-training 2	SGD with momentum ('sgdm')	1×10^{-5}	10	20	0.95
Networks 3-9	SGD with momentum ('sgdm')	1×10^{-5}	10	20	0.95

The collected images, introduced in section 4.1, were divided in such a way that 80% were used as a training set and 10% as a validation set for the CNNs. The remaining 10% of all images were assigned to a test set. The partitioning of images into different sets using MATLAB is further explained in section 4.3.2. Table 11 shows the available hardware for training of each of the networks 1-9, and relevant training options with values used in MATLAB are given in table 12.

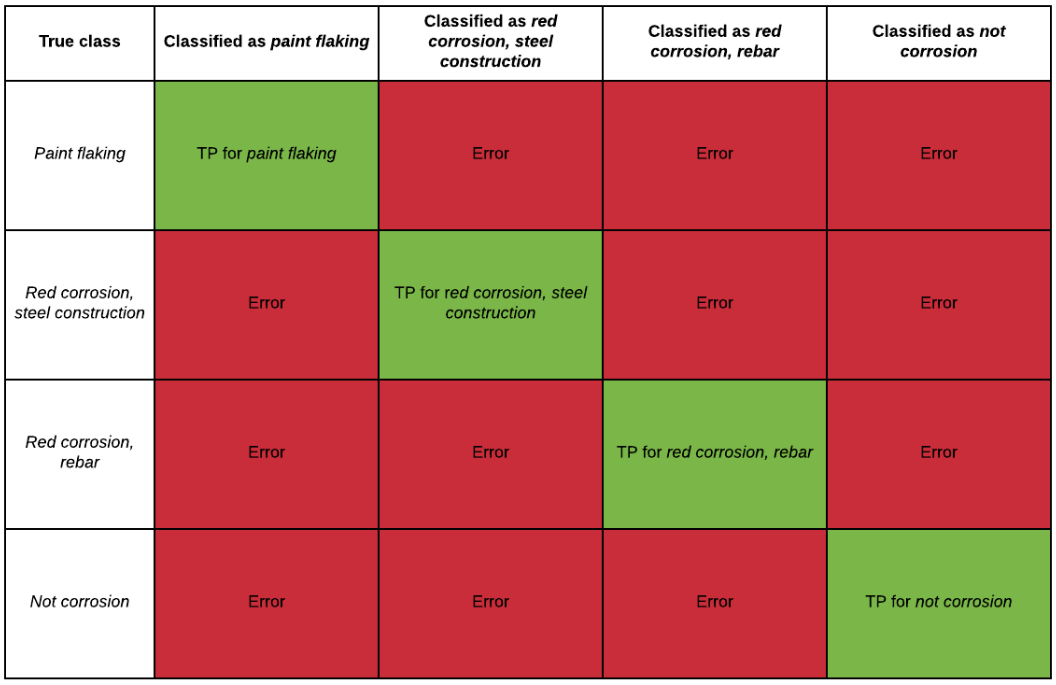
4.2.2 Testing

In order to test a trained CNN, relevant terms and metrics for evaluation of multi-class classification problems must first be defined. For a binary classification problem, such as in [7] where the objective is to classify images as *corrosion* or *not corrosion*, one of the classes can be defined as *positive*, while the other is defined as *negative*. Results from classifications are often evaluated in what is called a *confusion matrix*. A confusion matrix for a binary classification problem is shown in figure 50.

True class	**Classified as positive**	**Classified as negative**
Positive	True positive (TP)	False negative (FN)
Negative	False positive (FP)	True negative (TN)

Figure 50: Illustration of a confusion matrix for a binary classification problem. Defining a true positive, false positive, true negative and false negative. The true classifications, being TP and TN, are on the diagonal of the 2×2 confusion matrix. The illustration is made using Lucidchart [49], with inspiration from [81].

In this thesis, a maximum number of four classes is to be classified, which makes it a multi-class classification problem. Figure 51 illustrates a confusion matrix containing the four relevant classes from table 8. In the multi-class case, the definition of TP, TN, FP and FN requires some more explanation than in the binary case illustrated in figure 50. A TP is defined very similar in both cases, specifically as a class being correctly classified, or *predicted*, as the true class. For example, an image of *paint flaking* being classified as *paint flaking* is a TP. The TNs for a class in the multi-class case are defined as the sum of predictions in all columns and rows of the confusion matrix, except from the ones included in rows and columns related to that specific class. The total amount of FPs for a class is the sum of all predictions in the column related to that specific class, except from the TPs. The total amount of FNs for a class is the sum of all predictions in its corresponding row, except from the TPs. Figure 52 illustrates a confusion matrix generated after classification of images performed in this thesis. This confusion matrix serves as a basis for a calculation example to show how TPs, TNs, FPs and FNs, as well as performance metrics called *recall*, *precision*, *accuracy* and *F1 score*, are calculated.



True class	Classified as *paint flaking*	Classified as *red corrosion, steel construction*	Classified as *red corrosion, rebar*	Classified as *not corrosion*
Paint flaking	TP for *paint flaking*	Error	Error	Error
Red corrosion, steel construction	Error	TP for *red corrosion, steel construction*	Error	Error
Red corrosion, rebar	Error	Error	TP for *red corrosion, rebar*	Error
Not corrosion	Error	Error	Error	TP for *not corrosion*

Figure 51: Illustration of a confusion matrix for classification of the four classes *paint flaking*, *red corrosion, steel construction*, *red corrosion, rebar* and *not corrosion*. The correct classifications, TPs for all classes, are observed on the green diagonal of the 4×4 confusion matrix. The illustration is made using Lucidchart [49].

An explanation of the performance metrics that will be used to evaluate CNNs after testing is given below:

Recall: says what proportion of actual positives was correctly identified.

Precision: says what proportion of positive identifications was correct.

Accuracy: the fraction of predictions the network classified correctly.

F1 score: a metric that combines recall and precision as shown in (21).

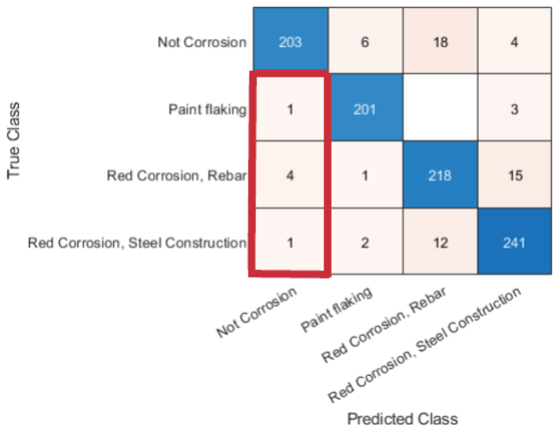
Equations for recall, precision, accuracy and F1 score, (18), (19), (20) and (21), are taken from [81], where i denotes the number of classes that are evaluated.

$$\text{Recall} = \frac{\sum_{i=1}^{I} \frac{TP_i}{TP_i+FN_i}}{I} \tag{18}$$

$$\text{Precision} = \frac{\sum_{i=1}^{I} \frac{TP_i}{TP_i+FP_i}}{I} \tag{19}$$

$$\text{Accuracy} = \frac{\sum_{i=1}^{I} \frac{TP_i+TN_i}{TP_i+FN_i+FP_i+TN_i}}{I} \tag{20}$$

$$\text{F1} = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{21}$$



True Class \ Predicted Class	Not Corrosion	Paint flaking	Red Corrosion, Rebar	Red Corrosion, Steel Construction
Not Corrosion	203	6	18	4
Paint flaking	1	201		3
Red Corrosion, Rebar	4	1	218	15
Red Corrosion, Steel Construction	1	2	12	241

Figure 52: Confusion matrix for use in calculation example. This is an actual confusion matrix for test of network 6 on 930 images. Seen from a bridge inspection perspective, the matrix elements inside the red frame are considered the most critical of wrong classifications. This is because these numbers represent images of damages on a bridge construction that are not detected, since they are classified as *not corrosion*. The elements inside the red frame are called the false positives (FPs) of *not corrosion*.

Calculation example using results from confusion matrix in figure 52

This calculation example shows how the different performance metrics are calculated from results of CNN testing in this thesis. The confusion matrix in figure 52 serves as the basis for this example. The example will go through calculation of TPs, TNs, FPs and FNs, as well as recall, precision, accuracy and F1 score for the class *Not corrosion* in figure 52. It will also be explained how average values for recall, precision, accuracy and F1 score are calculated.

True positives (TPs)

The first element on the blue diagonal of the confusion matrix shows the number of images that are predicted as *Not corrosion*, and that also belong to the true class *Not corrosion*. Thus, TP $= \underline{\underline{203}}$.

True negatives (TNs)

The sum of elements on all rows and columns, excluding the ones belonging to the class *Not corrosion*, shows the number of true negatives (TNs). TN $= 201 + 0 + 3 + 1 + 218 + 15 + 2 + 12 + 241 = \underline{\underline{693}}$.

False positives (FPs)

The number of false positives for the class *Not corrosion* is the sum of all elements on the column related to this class, except from the TPs. Thus, the number of false positives is: FP $= 1+4+1 = \underline{\underline{6}}$. All the false positives are marked with a red frame in figure 52 because these are wrong classifications that are considered most critical for bridge inspections, since FPs for *Not corrosion* means that there are damages on the bridge that are not detected as a damage. In this example, there would be 6 damages that are not detected, thus 6 FPs.

False negatives (FNs)

The number of false negatives for the class *Not corrosion* is the sum of all elements on the row related to this class, except from the TPs. Thus, the number of false negatives is: FN $= 6 + 18 + 4 = \underline{\underline{28}}$.

The sum of all TPs, TNs, FPs and FNs should equal to the total number of test images, which is 930: Total amount of test images $= TP+TN+FP+FN = 203+693+6+28 = \underline{\underline{930}} \rightarrow OK$.

Recall

Using (18) and the results found above: Recall $= \frac{\sum_{i=1}^{I} \frac{TP_i}{TP_i+FN_i}}{I} = \frac{\frac{203}{203+28}}{1} \approx \underline{\underline{0.8788}}$.

Precision

Using (19) and the results found above: Precision $= \frac{\sum_{i=1}^{I} \frac{TP_i}{TP_i+FP_i}}{I} = \frac{\frac{203}{203+6}}{1} \approx \underline{\underline{0.9713}}$.

Accuracy

Using (20) and the results found above: Accuracy $= \frac{\sum_{i=1}^{I} \frac{TP_i+TN_i}{TP_i+FN_i+FP_i+TN_i}}{I} = \frac{\frac{203+693}{203+28+6+693}}{1} \approx \underline{\underline{0.9634}}$.

F1 score

Using (21) and the values of recall and precision found above: F1 $= \frac{2 \times Recall \times Precision}{Recall + Precision} = \frac{2 \times 0.8788 \times 0.9713}{0.8788+0.9713} \approx \underline{\underline{0.9227}}$.

Average values

Average values, considering all classes in a network, are calculated the following way for all CNNs in this thesis:

Average recall $= \frac{\sum_{i=1}^{I} \frac{TP_i}{TP_i+FN_i}}{I} = \frac{\text{Sum of recall for all classes}}{\text{Number of classes}}$

Average precision $= \frac{\sum_{i=1}^{I} \frac{TP_i}{TP_i+FP_i}}{I} = \frac{\text{Sum of precision for all classes}}{\text{Number of classes}}$

Average accuracy $= \frac{\sum_{i=1}^{I} \frac{TP_i+TN_i}{TP_i+FN_i+FP_i+TN_i}}{I} = \frac{\text{Sum of accuracy for all classes}}{\text{Number of classes}}$

Average F1 score $= \frac{2 \times \text{Average recall} \times \text{Average precision}}{\text{Average recall} + \text{Average precision}}$

4.3 Implementation of classification system in MATLAB

This section will explain how convolutional neural networks for classification of corrosion and coating damages were implemented in MATLAB using the Deep learning Toolbox. The implementation consists of loading networks, labeling images, image pre-processing and network training and testing. Some lines of code are added in this chapter to support explanations related to implementation of CNNs, and the MATLAB code in its entirety is given in the appendix in section C.

4.3.1 Loading CNNs

A CNN is loaded in MATLAB using the command `net = "network"`, for example `net = alexnet` or `net = googlenet`. In order to run this command, the Deep learning Toolbox must be installed, and it is also necessary to download specific packages for each network. In addition to the Deep Learning Toolbox, the following toolboxes for the specific networks were installed:

- Deep Learning Toolbox Model for AlexNet Network [82]
- Deep Learning Toolbox Model for GoogLeNet Network [83]
- Deep Learning Toolbox Model for ResNet-50 Network [84]
- Deep Learning Toolbox Model for VGG-16 Network [85]

4.3.2 Image labeling and data sets

Images recieved from The Norwegian Public Roads Administration were categorized into folders, as explained in section 4.1. From the folder names the images were labeled using `imageDatastore()` [86]. The code for image labeling is given in listing 4.1. In this code example the folder *ThreeLabels* is used as input, and the names of its subfolders are set as the label source for all images.

Listing 4.1: MATLAB code for labeling of images from folder names made during categorization of images.

```
%Loading images to a image datastore, label images from
   foldernames:
images=imageDatastore('ThreeLabels','IncludeSubfolders',true
   ,'LabelSource','foldernames');
```

The images were divided into training, validation and test sets as shown in listing 4.2. The different sets were denoted *imageTrain*, *imageValidation* and *imageTest*, and the command `splitEachLabel()` [87] was used to split the total amount of images into proportions. As listing 4.2 shows, 80% of all images were assigned to the training set, and 10% to the validation set. The remaining 10% of the images were automatically put aside to be used as a test set.

Listing 4.2: MATLAB code for dividing the images into training, validation and test sets.

```
%The data is divided into three sets;training(80%),
   validation(10%) and
%testing (10%):
[imageTrain,imageValidation,imageTest]=splitEachLabel(images
   ,0.8,0.1);
```

4.3.3 Image pre-processing

Cropping and rotation of images

The amount of images for each class of damages to be used in this thesis was given in table 9. It was decided not to continue with class 2-white corrosion, because very few images were found of this class during data collection. Class 1, paint flaking, also has few images compared with the remaining classes. Using only the original 248 images in class 1 together with the classes 3, 4 and 5 would give an unbalanced data set. Therefore, data augmentation, in the form of image cropping and rotations, was performed to increase the number of images in class 1. To perform image cropping, the MATLAB command `imCrop()` [88] was used, which opens a new window with the image and an area can be chosen to crop. Figure 53 illustrates an example of how an image was cropped and rotated. Images were rotated using `imrotate()` [89], and the two inputs to this command were the relevant image and the desired angle of rotation.

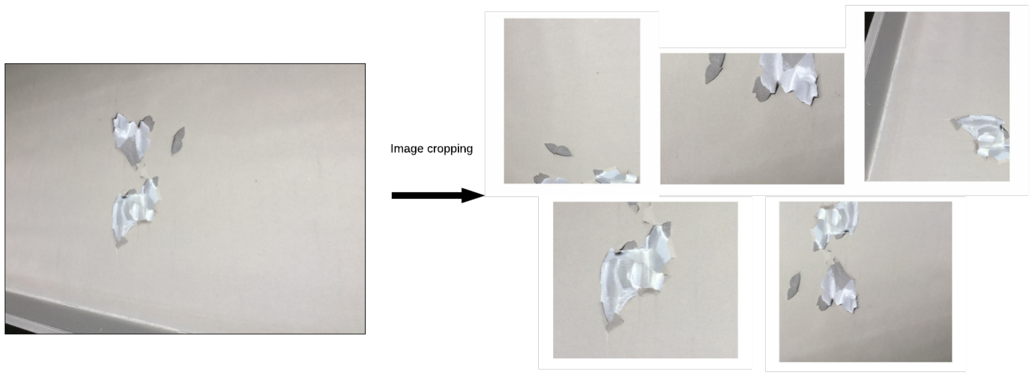


Figure 53: Example of how image cropping and rotation was performed to create more images in class 1-paint flaking.

Image resizing and colour pre-processing

The size of all images had to be scaled down to fit the input layer of the networks, for example to 227×227 pixels for AlexNet, and to 224×224 pixels for GoogLeNet and ResNet-50. Both image resizing and colour pre-processing was performed using the command `augmentedImageDatastore()` [90]. The operation `'ColorPreprocessing'` was used to ensure that all input images had the same amount of colour channels, here three channels. Although all collected images were RGB images, some of the images had a type of grayscale representation. Thus, it became necessary to use this operation with the input `'gray2rgb'`. The MATLAB code for changing image input size and for colour pre-processing is shown in listing 4.3. The augmented training set in listing 4.3 is denoted *AugTrain*. The exact same operation was performed on the validation and training set, and the augmented sets were denoted *AugValidation* and *AugTest*, respectively.

Listing 4.3: MATLAB code for changing image input size to fit the first layer of a network and for colour processing of images.

```matlab
%Define image input size for the network:
inputSize = net.Layers(1).InputSize;

%Change input image size and perform colour pre-processing:
AugTrain = augmentedImageDatastore(inputSize(1:2),imageTrain
   ,'ColorPreprocessing','gray2rgb');
```

4.3.4 Modification of layers in existing CNNs

The pre-trained CNNs AlexNet, GoogLeNet, ResNet-50 and VGG-16 used for transfer learning are all originally trained to classify 1000 classes of images. In this thesis, the maximum number of classes to be classified is four. Therefore, it was necessary to modify some of the last layers in the pre-trained CNNs to make them fit the new classification problem. More specifically, the three last layers of the existing networks were modified as explained by the MATLAB code in listing 4.4. New layers were specified in `newLayers`, including a fully-connected layer where the damage categories specified in `classes` were used as input to define the new classification task.

Listing 4.4: MATLAB code for modifying layers in CNNs.

```matlab
%Modifying network layers:
layersExtract = net.Layers(1:end-3);
classes = numel(categories(imageTrain.Labels));
newlayers = [
    layersExtract
    fullyConnectedLayer(classes,'WeightLearnRateFactor',20,'
       BiasLearnRateFactor',20)
    softmaxLayer
    classificationLayer];
```

The code in listing 4.4 was sufficient for modifying the layers in the AlexNet network, but for all the other CNNs, an additional function was required due to these networks being DAG (Directed Acyclic Graph) networks. The function called `findLayersToReplace.m` was downloaded from [91], and is given in the appendix in section C.

4.3.5 Training of CNNs

Figure 54 illustrates a typical workflow for transfer learning in MATLAB. Three main elements creates the basis for CNN training through transfer learning; *modified network*, *training data* and *training algorithm options*. Modification of a network was performed as described in section 4.3.4, and the set of images introduced in section 4.1 serves as both training and test data. The final element is the training algorithm options, which contain information on parameters such as what type of learning algorithm is used, the learning rate, mini-batch size and the number of epochs. Chosen training options for each CNN were shown in table 12, and specified in MATLAB using the command `trainingOptions()` [46] as shown in listing 4.5.

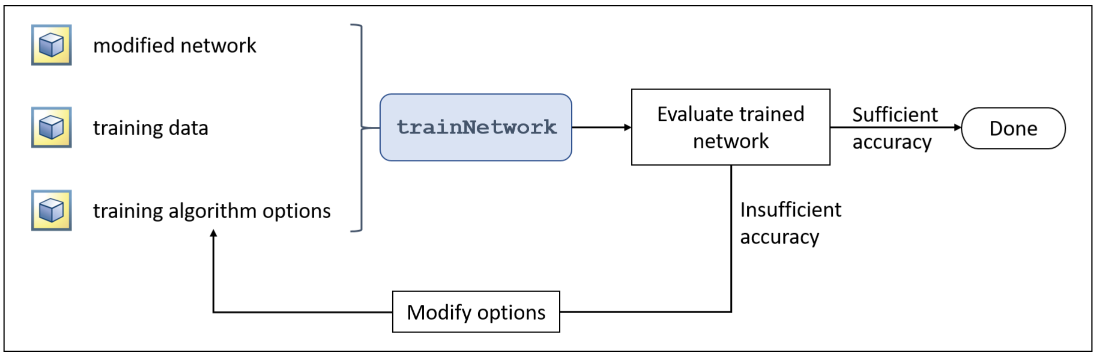


Figure 54: Transfer learning workflow in MATLAB as specified by MathWorks [92].

Listing 4.5: MATLAB code for deciding the training options for a CNN.

```
%Specifying training options:
optionsTrain = trainingOptions('sgdm','MiniBatchSize',20,'
   MaxEpochs',10,'Momentum',0.95,'InitialLearnRate',1e-5, '
   Shuffle','every-epoch','ValidationData',AugValidation,'
   ValidationFrequency',3, 'Verbose',false,'Plots','training
   -progress');
```

4.3.6 Testing of CNNs

The setup for testing of CNNs in this thesis was explained previously in section 4.2.2. This section will show how tests were performed in MATLAB using relvant commands, including how to make use of a defined test set to classify images using a trained CNN, and how to create confusion matrices for illustrating test results.

After training a CNN, the network was saved as a .mat-file in MATLAB. Listing 4.6 shows how a saved CNN is loaded in MATLAB as a .mat-file, and the command `classify()` [93] is used with the two inputs being the trained CNN and the augmented test set of images. Predictions and scores are extracted as `Pred` and `scores`. A confusion matrix is created using `confusionchart`, which is a MATLAB command that automatically generates an illustration of true and predicted classes, given labels from test images and the predictions found using `classify()`.

Listing 4.6: MATLAB code for testing of trained CNN, and for obtaining a confusion matrix.

```matlab
%Classify test images:
load ('Network.mat')
[Pred,scores] = classify(Network,AugTest);

%Creating a confusion matrix after classifying test images:
Test = imageTest.Labels;
confusionchart(Test,Pred)
```

Chapter 5

Results

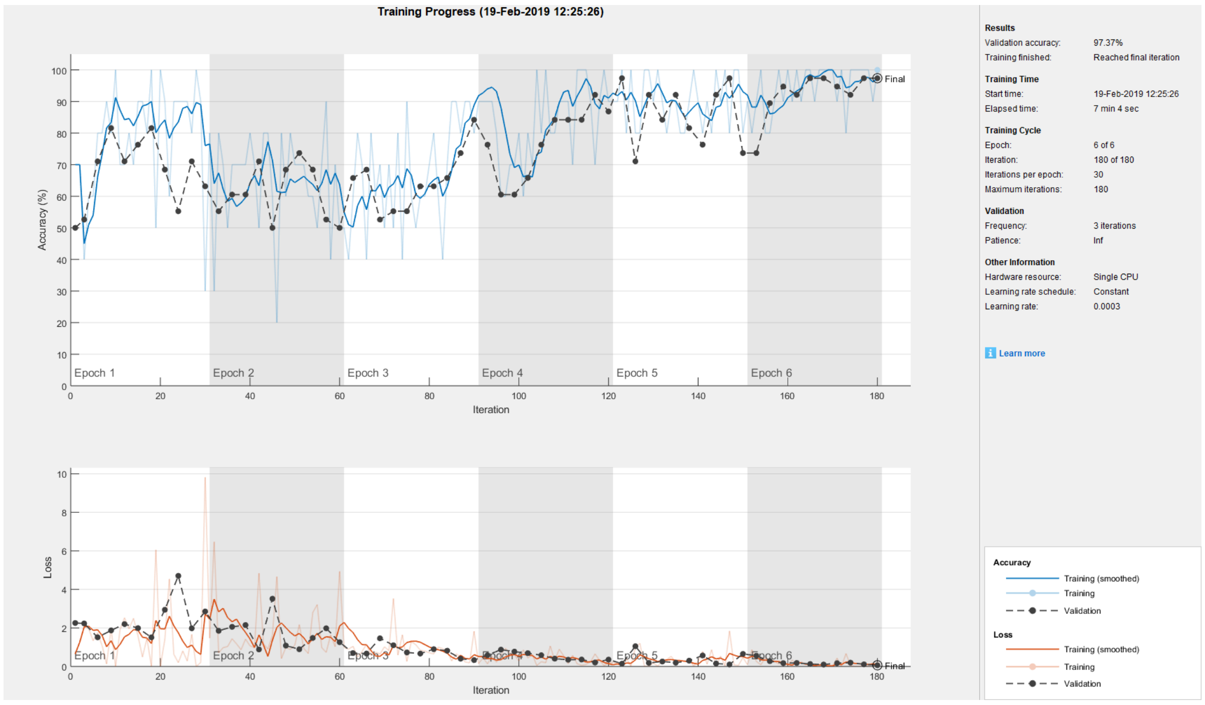
This chapter will show the obtained results after training and test of CNNs performed as explained in section 4.2. Training progress for each network is illustrated by accuracy and loss for both training and validation. Results from testing of the different CNNs are illustrated with confusion matrices, showing both true and predicted classes. The metrics recall, precision, accuracy and F1 score are used as performance measurements when evaluating the results. All calculations of these metrics were performed as shown in the detailed example in section 4.2.2, using equations (18), (19), (20) and (21). The CNNs are also evaluated on damage detection accuracy, meaning that the number of false positives for *not corrosion* are studied. A summary and comparison of results for the different CNNs are given in in the last section of this chapter, section 5.10. The scope of the discussion for each network, from section 5.1 to 5.9, will vary and is quite brief for some of the networks. The majority of discussion related to the results will be given in the summary and comparison of all CNNs in section 5.10. Relevant terms used to present and discuss the results in this chapter were explained in detail in section 4.2.2, and a brief recap of some of the terms is given below:

- **Recall**: The amount of actual positives for a class that are correctly classified.
- **Precision**: The amount of positive predictions that are correct.
- **F1 score**: A metric that combines results from recall and precision, shown in (21).
- **Accuracy**: The fraction of predictions that are correctly classified.

5.1 Network 1-AlexNet

5.1.1 Training, validation and test of network 1

The training progress for network 1, including accuracy and loss related to both training and validation, is shown in figure 55. A total of 376 images, 188 of each category (corrosion and not corrosion) were used for training, validation and testing of network 1. Out of 376 images, 80 % were used as a training set, and the remaining images were divided equally into a validation set and a test set.



Results

Validation accuracy:	97.37%
Training finished:	Reached final iteration

Training Time

Start time:	19-Feb-2019 12:25:26
Elapsed time:	7 min 4 sec

Training Cycle

Epoch:	6 of 6
Iteration:	180 of 180
Iterations per epoch:	30
Maximum iterations:	180

Validation

Frequency:	3 iterations
Patience:	Inf

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.0003

Figure 55: Training progression. 97,37% validation accuracy. Learning rate 3×10^{-4}. Total of 6 epochs. Elapsed time: 7 min and 4 sec on single CPU.

Network 1 was tested on two different test sets as explained below:

- The first test set contained a total of 38 images, 19 of each category *corrosion* and *not corrosion*. Figure 56 illustrates the confusion matrix for this test, showing a total test accuracy of 100%.
- The second test was performed using a test set containing 100 images of the corrosion class only. Figure 57 illustrates the confusion matrix for this test, showing a total test accuracy of 83.00%.

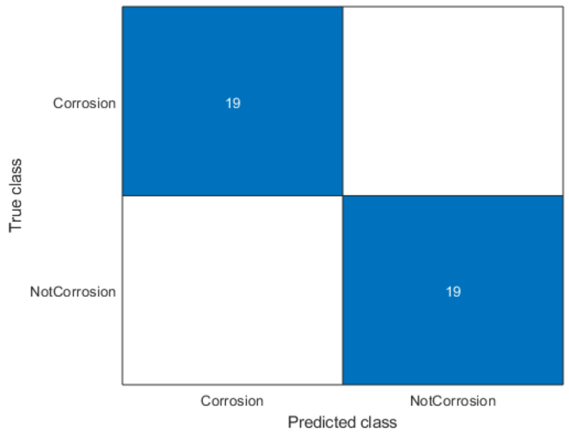


Figure 56: Confusion matrix for test set of 38 images (19 of each label). There are 19 TPs for *corrosion* and 19 TPs for *not corrosion*.

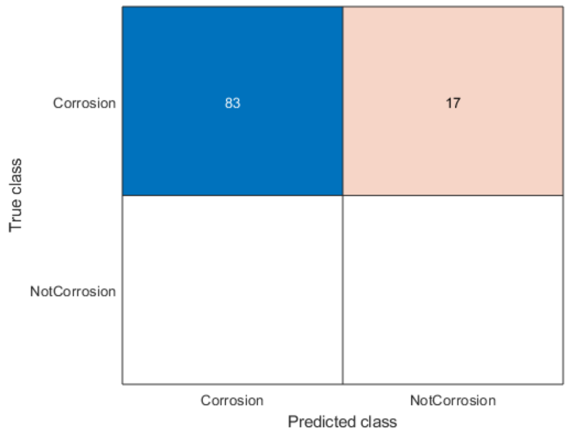


Figure 57: Confusion matrix for test set of 100 images containing the corrosion class only. There are 83 TPs for *corrosion* and 17 FPs for *not corrosion*.

5.1.2 Discussion of network 1

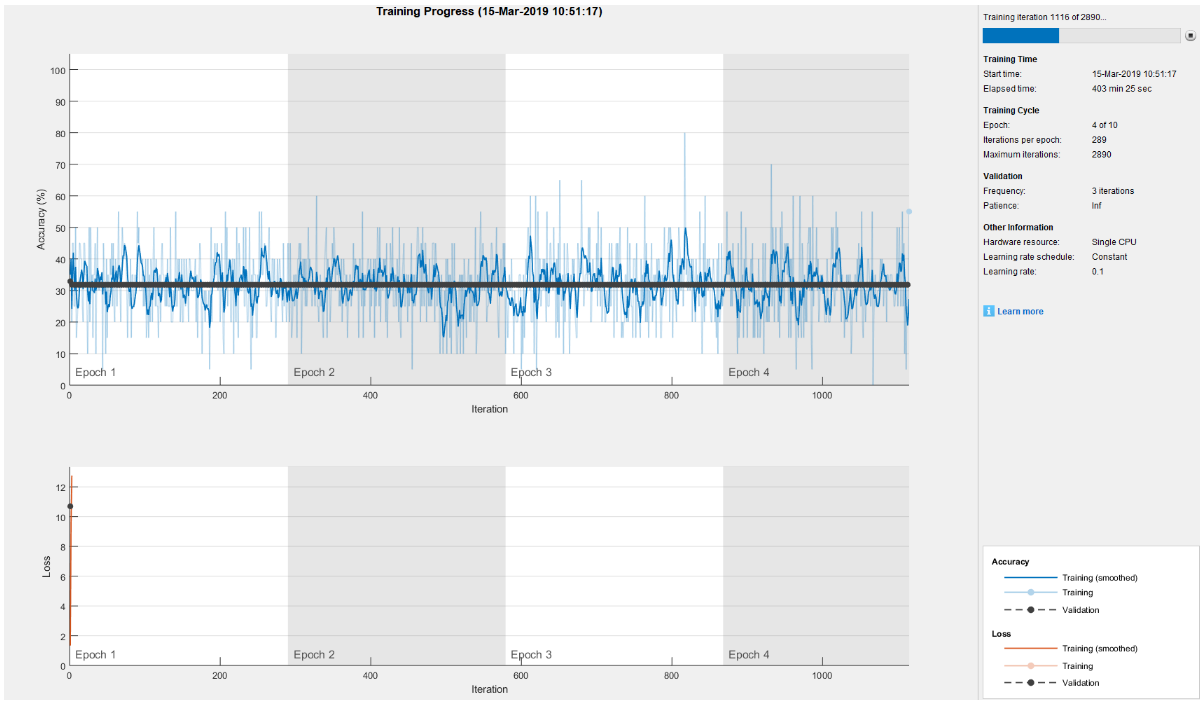
The results from training and testing of network 1 are not considered very relevant for conclusions to be made in this thesis, but a few interesting observations are worth discussing. This network training and testing was performed mainly to set up a MATLAB script and to become familiar with how network training is carried out. Since a relatively small amount of images was used and no data augmentation was performed, the chance is high that overfitting did occur. When comparing the results from the two tests, it is observed that the 38 test images in figure 56 were all correctly classified, while for the 100 images of only corrosion in figure 57 the results are different. Out of 100 images of corrosion, 17 were wrongly classified as not corrosion. The network clearly is not able to classify such a relatively large amount of test images with the same accuracy as obtained in the first test with 38 images. The validation accuracy during training was 97.37% as shown in figure 55, while the test accuracy for the 100 images was 83.00%. This result supports the theory of overfitting occurring in network 1.

From the presented results it was observed that having images of red corrosion on elements of steel construction and on rebar in the same class called *corrosion* seemed to be a challenge. This is because many of the 17 wrongly classified images in the second test contained corrosion on rebar. The reason is assumed to be the fact that differences in appearance between these two categories of corrosion damages can be quite large, resulting in images of corrosion on rebar being classified as *not corrosion* because these images often contain smaller areas of corrosion compared with those of elements of steel constructions. To avoid this potential challenge in further training of neural networks, it was chosen to divide these types of corrosion damages into two separate classes. One could of course argue that a neural network trained on more images than network 1 could be able to classify images of corrosion on rebar as *corrosion* with higher accuracy, while being defined as the same class as corrosion on steel construction. However, corrosion on rebar has a very characteristic appearance with horizontal and vertical "lines" as shown in figure 48, which makes it reasonable to define it as an own class of corrosion damages. In addition, if any of the neural networks trained in this thesis can be used in an actual system for automatic damage detection on bridge constructions, being able to separate between different types of corrosion damages is considered to be preferable in order to get the best possible overview of a construction and what measures of maintenance are required.

5.2 Network 2-AlexNet

5.2.1 Training, validation and test of network 2

The training progresses for network 2-training 1 and 2, including accuracy and loss related to both training and validation, are shown in figures 58 and 59, respectively. The confusion matrix after testing of network 2-training 2 is shown in figure 60, and table 13 shows the result for the different metrics used to evaluate CNN performance.



Training iteration 1116 of 2890...

Training Time

Start time:	15-Mar-2019 10:51:17
Elapsed time:	403 min 25 sec

Training Cycle

Epoch:	4 of 10
Iterations per epoch:	289
Maximum iterations:	2890

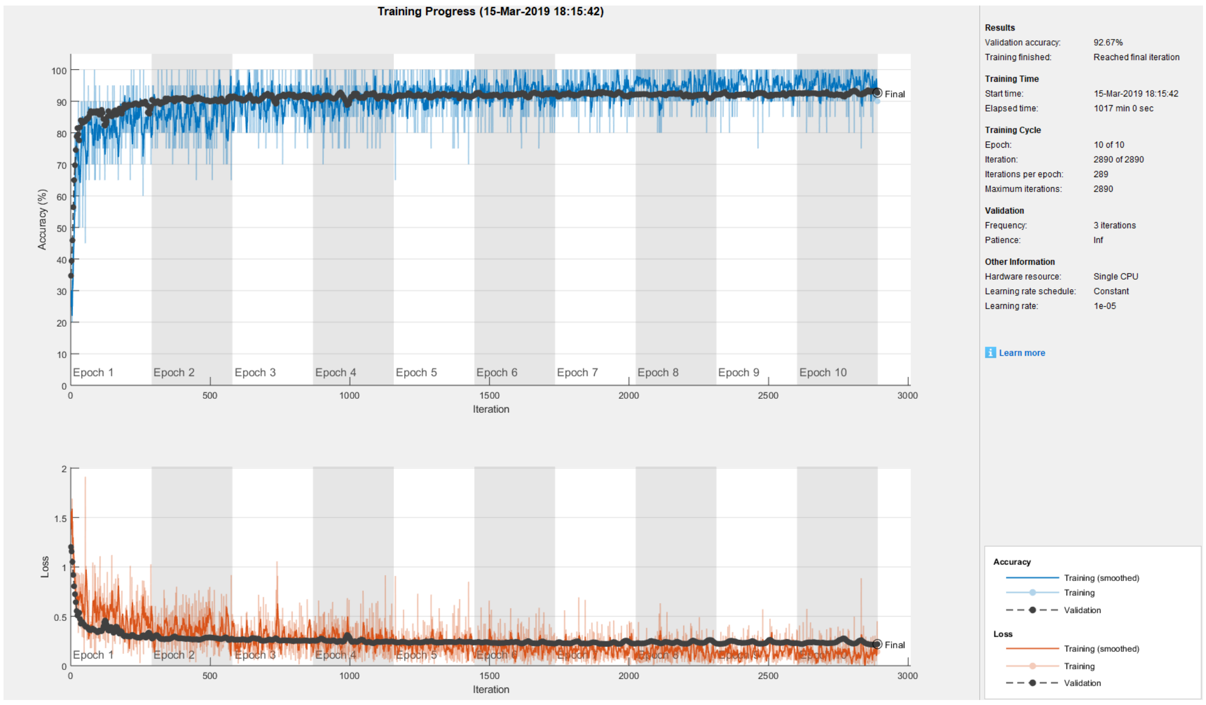
Validation

Frequency:	3 iterations
Patience:	Inf

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.1

Figure 58: Training progression for network 2-training 1. 32.80% validation accuracy. Learning rate 1×10^{-1} .Training stopped after 4 epochs. Elapsed time: 403 min and 25 sec ($\sim$ 7 hours) on single CPU.



Results

Validation accuracy:	92.67%
Training finished:	Reached final iteration

Training Time

Start time:	15-Mar-2019 18:15:42
Elapsed time:	1017 min 0 sec

Training Cycle

Epoch:	10 of 10
Iteration:	2890 of 2890
Iterations per epoch:	289
Maximum iterations:	2890

Validation

Frequency:	3 iterations
Patience:	Inf

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	1e-05

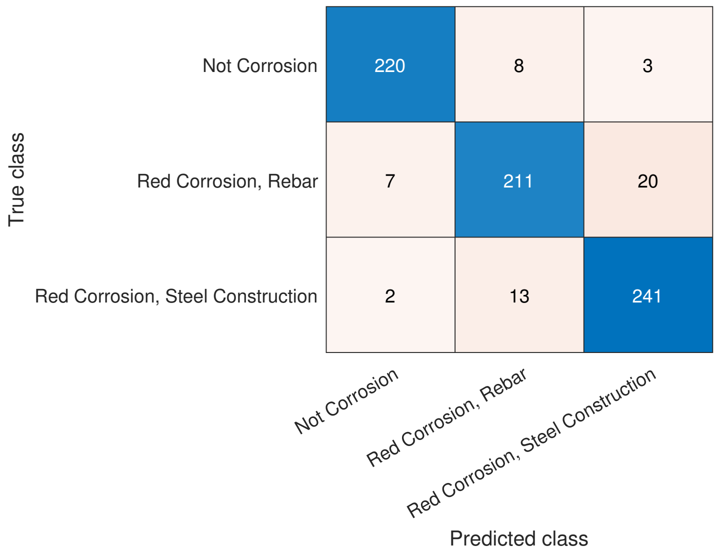
Accuracy

- Training (smoothed)
- Training
- Validation

Loss

- Training (smoothed)
- Training
- Validation

Figure 59: Training progression for network 2-training 2. 92,67% validation accuracy. Learning rate 1×10^{-5} . Total of 10 epochs. Elapsed time: 1017 min and 0 sec ($\sim$ 17 hours) on single CPU.



True class \ Predicted class	Not Corrosion	Red Corrosion, Rebar	Red Corrosion, Steel Construction
Not Corrosion	220	8	3
Red Corrosion, Rebar	7	211	20
Red Corrosion, Steel Construction	2	13	241

Figure 60: Confusion matrix for test of network 2-training 2 on 725 images.

Table 13: Results from testing of network 2 presented with the metrics recall, precision, accuracy and F1 score. The average values show the overall performance of the CNN for all classes.

Class/label	Recall	Precision	Accuracy	F1 score
Red corrosion, steel construction	0.9414	0.9129	0.9476	0.9269
Red corrosion, rebar	0.8866	0.9095	0.9338	0.8979
Not corrosion	0.9524	0.9607	0.9724	0.9565
Average, all classes	0.9268	0.9277	0.9513	0.9272

5.2.2 Discussion of network 2

The validation accuracy for training 1 remained constant during training as shown in figure 58. This is most likely due to the SGD algorithm reaching a local minimum, and interpreting this as a global one, because the learning rate was set too high. Training 1 was stopped after four epochs since there were no signs of the validation accuracy increasing with time. There was something wrong with illustration of the training loss curve for training 1 (orange curve in figure 58); it did not appear during training. It is not known exactly what caused this, but the plot command in MATLAB must have frozen in some way, causing the plot to stop in the very beginning of the training.

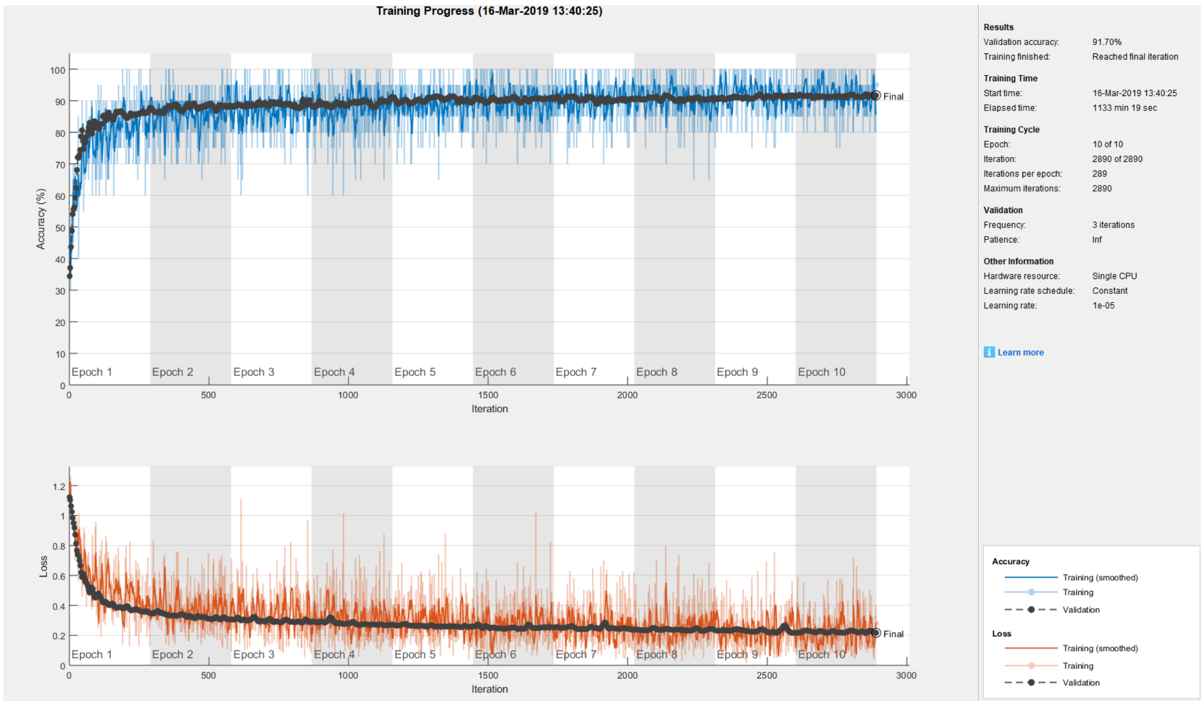
From the observations on validation accuracy made in training 1 (figure 58), a new network training was performed. The learning rate was first changed to 1×10^{-3}, but it was not a sufficiently small value because after only a few iterations the validation accuracy decreased and the training loss increased rapidly. This network training was therefore cancelled, and the learning rate was changed to 1×10^{-5}. With this learning rate the training progression improved significantly, as illustrated in figure 59.

Results from testing of network 2-training 2 are shown both in the confusion matrix in figure 60 and in table 13. From these results it is observed that the class *not corrosion* has the highest scores for all performance metrics, while the results for *red corrosion, rebar* show the overall lowest scores of all three predicted classes. As illustrated in the first column of the confusion matrix in figure 60, there are a total of 9 (7+2) FPs for *not corrosion*. This means that 9 corrosion damages were not detected by network 2.

5.3 Network 3-GoogLeNet

5.3.1 Training, validation and test of network 3

The training progress for network 3 is illustrated in figure 61. The confusion matrix after testing of network 3 is shown in figure 62, and table 14 shows the result for the different metrics used to evaluate CNN performance.



Results

Validation accuracy:	91.70%
Training finished:	Reached final iteration

Training Time

Start time:	16-Mar-2019 13:40:25
Elapsed time:	1133 min 19 sec

Training Cycle

Epoch:	10 of 10
Iteration:	2890 of 2890
Iterations per epoch:	289
Maximum iterations:	2890

Validation

Frequency:	3 iterations
Patience:	Inf

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	1e-05

Figure 61: Training progression for network 3. 91,70% validation accuracy. Learning rate 1×10^{-5} . Total of 10 epochs. Elapsed time: 1133 min and 19 sec ($\sim$ 19 hours) on single CPU.

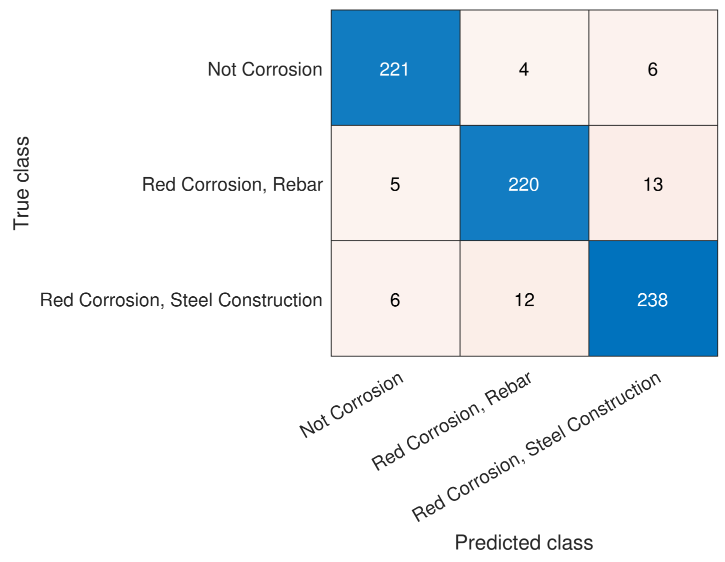


Figure 62: Confusion matrix for test of network 3 on 725 images.

5.3.2 Discussion of network 3

The validation accuracy is approximately 1 percent lower for network 3 compared with network 2. However, the average test scores for all metrics shown in table 14 are higher than those for network 2 shown in table 13. From table 14 it is observed that the class *not corrosion* has the highest performance scores of all classes in network 3. The confusion matrix for network 3, illustrated in figure 62, show that there are a total of 11 (5+6) FPs for *not corrosion*. This is given in the first column of the confusion matrix. These FPs represent 5 cases of red corrosion on rebar and 6 cases of red corrosion on steel construction that are not detected as a damage by network 3.

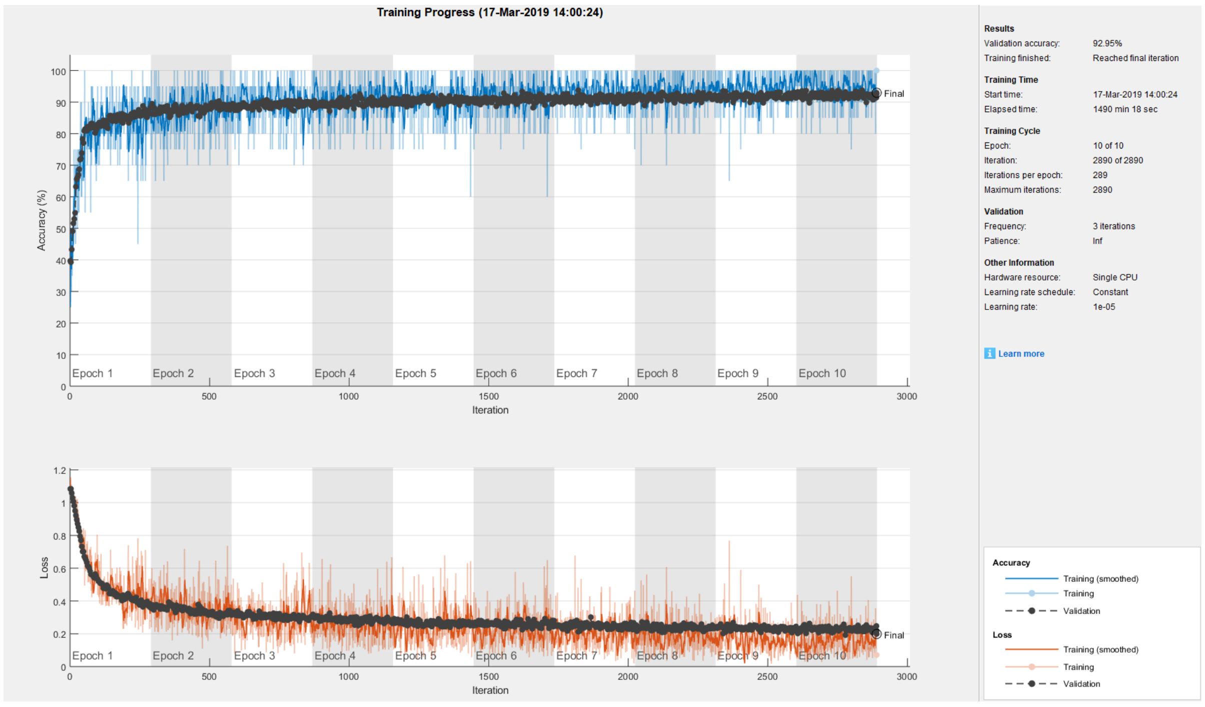
Table 14: Results from testing of network 3 presented with the metrics recall, precision, accuracy and F1 score. The average values show the overall performance of the CNN for all classes.

Class/label	**Recall**	**Precision**	**Accuracy**	**F1 score**
Red corrosion, steel construction	0.9297	0.9261	0.9490	0.9279
Red corrosion, rebar	0.9244	0.9322	0.9531	0.9283
Not corrosion	0.9567	0.9526	0.9710	0.9546
Average, all classes	0.9369	0.9370	0.9577	0.9369

5.4 Network 4-ResNet-50

5.4.1 Training, validation and test of network 4

The training progress for network 4 is illustrated in figure 63. The confusion matrix after testing of network 4 is shown in figure 64, and table 15 shows the result for the different metrics used to evaluate CNN performance.



Results	
Validation accuracy:	92.95%
Training finished:	Reached final iteration
Training Time	
Start time:	17-Mar-2019 14:00:24
Elapsed time:	1490 min 18 sec
Training Cycle	
Epoch:	10 of 10
Iteration:	2890 of 2890
Iterations per epoch:	289
Maximum iterations:	2890
Validation	
Frequency:	3 iterations
Patience:	Inf
Other Information	
Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	1e-05

Figure 63: Training progression for network 4. 92,95% validation accuracy. Learning rate 1×10^{-5} . Total of 10 epochs. Elapsed time: 1490 min and 18 sec ($\sim$ 25 hours) on single CPU.

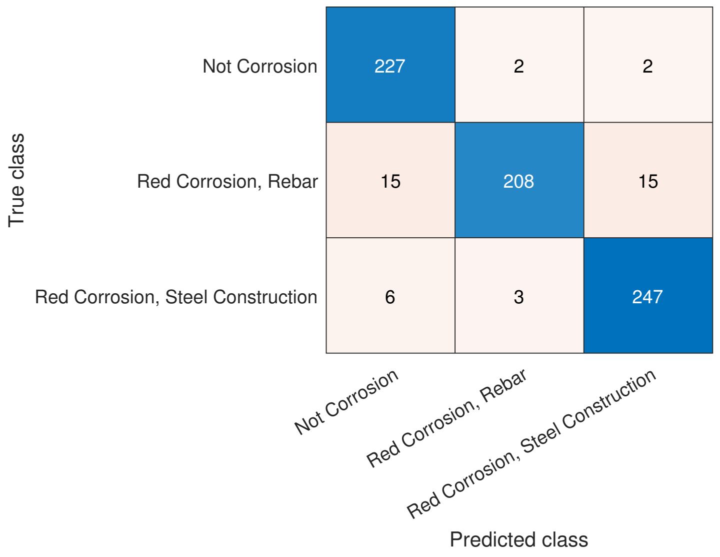


Figure 64: Confusion matrix for test of network 4 on 725 images.

5.4.2 Discussion of network 4

The recall related to classifying *not corrosion* is higher compared with both network 2 and network 3. Only four out of 231 images in this class were wrongly classified as either red corrosion on rebar or steel construction, meaning there are four FNs for *not corrosion*. Table 15 shows that the average performance scores for all metrics are higher for network 4, compared with both network 2 and 3. As illustrated in the first column of the confusion matrix in figure 64, there are a total of 21 (15+6) FPs for *not corrosion*. This means that 21 corrosion damages were not detected by network 4, making it the CNN with most FPs for *not corrosion* of all networks introduced until now.

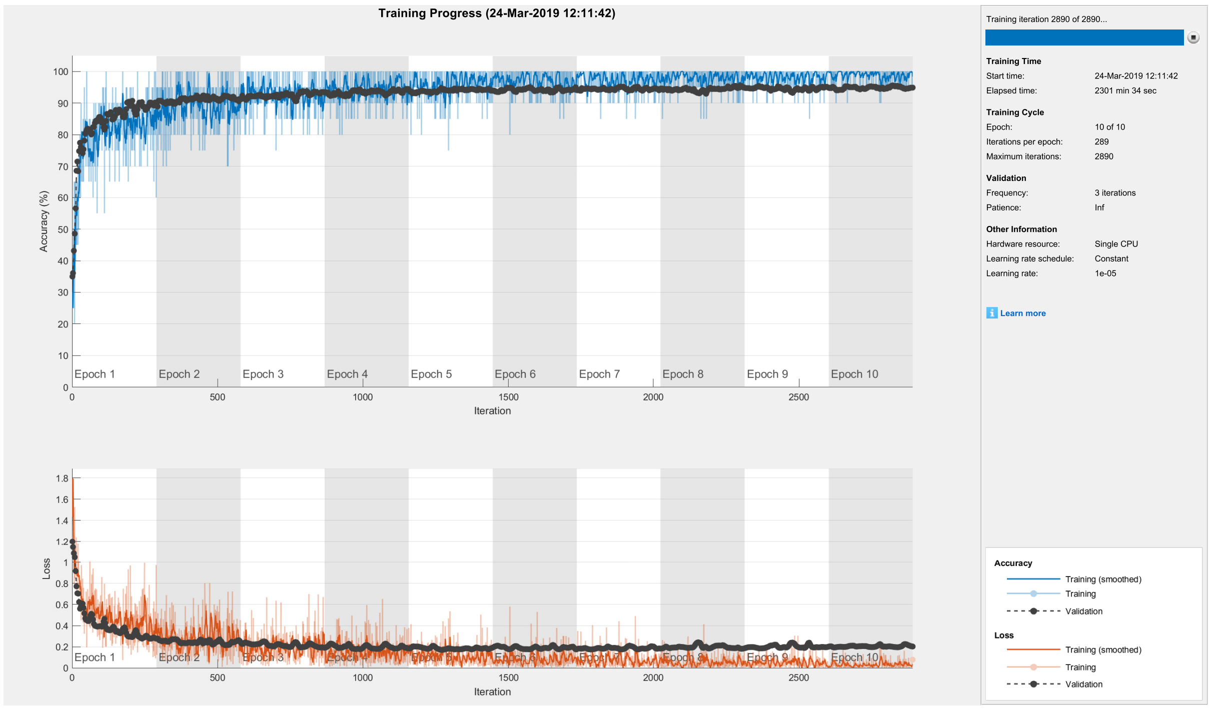
Table 15: Results from testing of network 4 presented with the metrics recall, precision, accuracy and F1 score. The average values show the overall performance of the CNN for all classes.

Class/label	**Recall**	**Precision**	**Accuracy**	**F1 score**
Red corrosion, steel construction	0.9648	0.9356	0.9641	0.9500
Red corrosion, rebar	0.8739	0.9765	0.9517	0.9224
Not corrosion	0.9827	0.9153	0.9655	0.9478
Average, all classes	0.9405	0.9425	0.9605	0.9415

5.5 Network 5-VGG-16

5.5.1 Training, validation and test of network 5

The training progress for network 5 is illustrated in figure 65. The confusion matrix after testing of network 5 is shown in figure 66, and table 16 shows the result for the different metrics used to evaluate CNN performance.



Training iteration 2890 of 2890...

Training Time
Start time: 24-Mar-2019 12:11:42
Elapsed time: 2301 min 34 sec

Training Cycle
Epoch: 10 of 10
Iterations per epoch: 289
Maximum iterations: 2890

Validation
Frequency: 3 iterations
Patience: Inf

Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 1e-05

Figure 65: Training progression for network 5. 94,88% validation accuracy. Learning rate 1×10^{-5} . Total of 10 epochs. Elapsed time: 2301 min and 34 sec ($\sim$ 38 hours) on single CPU.

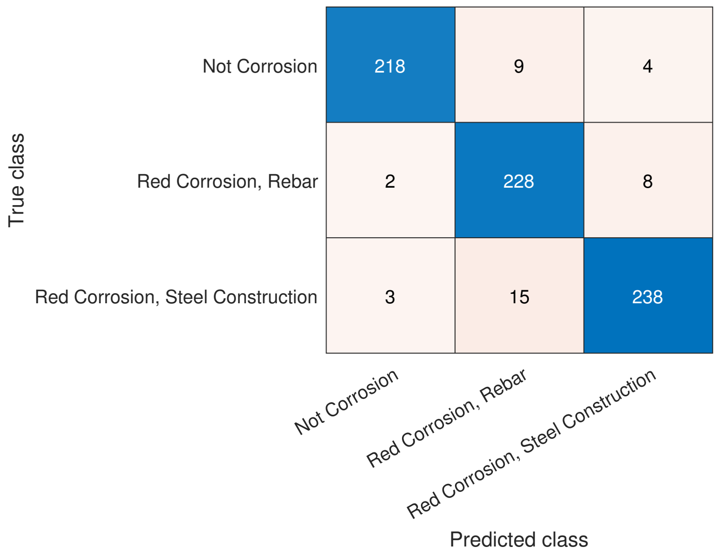


Figure 66: Confusion matrix for test of network 5 on 725 images.

Table 16: Results from testing of network 5 presented with the metrics recall, precision, accuracy and F1 score. The average values show the overall performance of the CNN for all classes.

Class/label	Recall	Precision	Accuracy	F1 score
Red corrosion, steel construction	0.9297	0.9520	0.9586	0.9407
Red corrosion, rebar	0.9580	0.9048	0.9531	0.9306
Not corrosion	0.9437	0.9776	0.9752	0.9604
Average, all classes	0.9438	0.9448	0.9623	0.9443

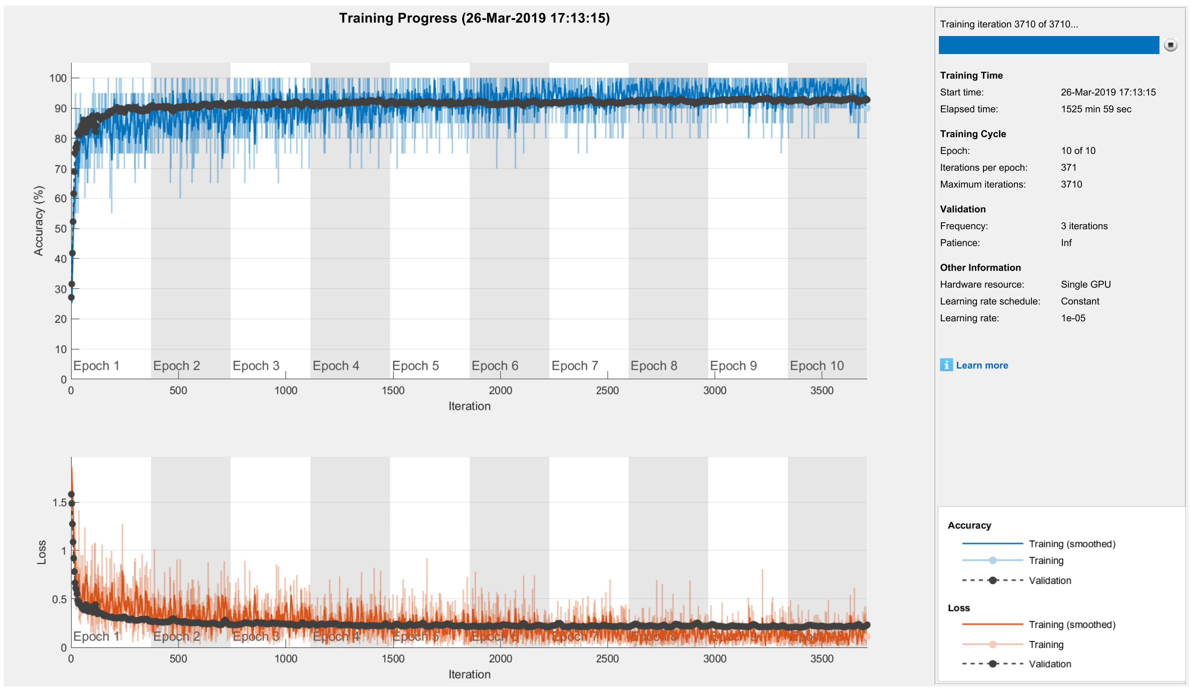
5.5.2 Discussion of network 5

Network 5 has the highest validation accuracy obtained so far, being 94.88 %, as shown in figure 65. With a training duration of approximately 38 hours on a single CPU, network 5 was also the CNN that took the longest time to train of networks 2-5. As illustrated in the first column of the confusion matrix in figure 66, there are a total of 5 (2+3) FPs for *not corrosion*. This means that 5 corrosion damages were not detected by network 5, making it the CNN with fewest FPs for *not corrosion* of networks 2-5. The average values for metrics in the last row of table 16 also show that all values are higher compared with network 2-4.

5.6 Network 6-AlexNet

5.6.1 Training, validation and test of network 6

The training progress for network 6 is illustrated in figure 67. The confusion matrix after testing of network 6 is shown in figure 68, and table 17 shows the result for the different metrics used to evaluate CNN performance.



Training iteration 3710 of 3710...

Training Time
Start time: 26-Mar-2019 17:13:15
Elapsed time: 1525 min 59 sec

Training Cycle
Epoch: 10 of 10
Iterations per epoch: 371
Maximum iterations: 3710

Validation
Frequency: 3 iterations
Patience: Inf

Other Information
Hardware resource: Single GPU
Learning rate schedule: Constant
Learning rate: 1e-05

Accuracy
Training (smoothed)
Training
Validation

Loss
Training (smoothed)
Training
Validation

Figure 67: Training progression for network 6. 92,78 % validation accuracy. Learning rate 1×10^{-5} . Total of 10 epochs. Elapsed time: 1525 min and 59 sec ($\sim$ 25 hours) on single GPU.

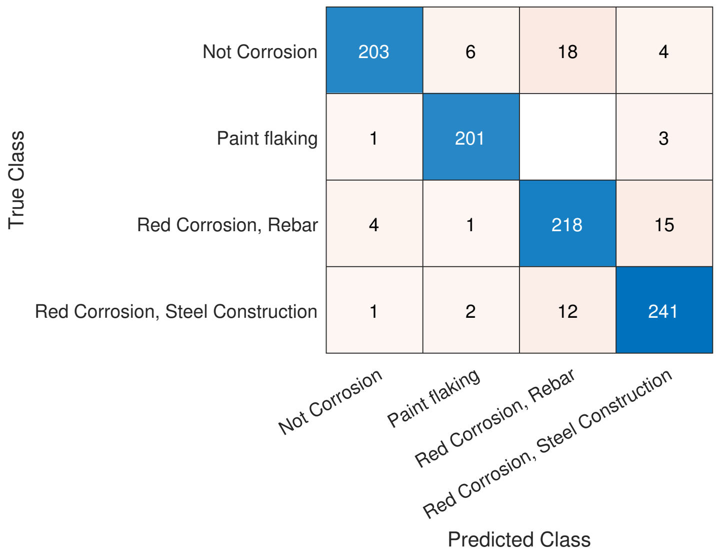


Figure 68: Confusion matrix for test of network 6 on 930 images. The blank cell in the confusion matrix means that there are zero wrongly classified images for the specific class. In this case, no images of paint flaking were wrongly classified as *red corrosion, rebar*.

Table 17: Results from testing of network 6 presented with the metrics recall, precision, accuracy and F1 score. The average values show the overall performance of the CNN for all classes.

Class/label	**Recall**	**Precision**	**Accuracy**	**F1 score**
Paint flaking	0.9805	0.9571	0.9860	0.9687
Red corrosion, steel construction	0.9414	0.9163	0.9602	0.9287
Red corrosion, rebar	0.9160	0.8790	0.9462	0.8971
Not corrosion	0.8788	0.9713	0.9634	0.9227
Average, all classes	0.9292	0.9310	0.9640	0.9301

5.6.2 Discussion of network 6

First, a learning rate of 5×10^{-5} was set in the training options for the network, in order to see if a slightly higher learning rate than the one used for networks 2-5 could be beneficial. It was observed a much more unstable progress in accuracy and loss for both the training and validation set, so the learning rate was set back to 1×10^{-5}, giving a smoother progress and a stable increase towards higher accuracy, as well as stable decrease towards lower loss. This is illustrated in figure 67.

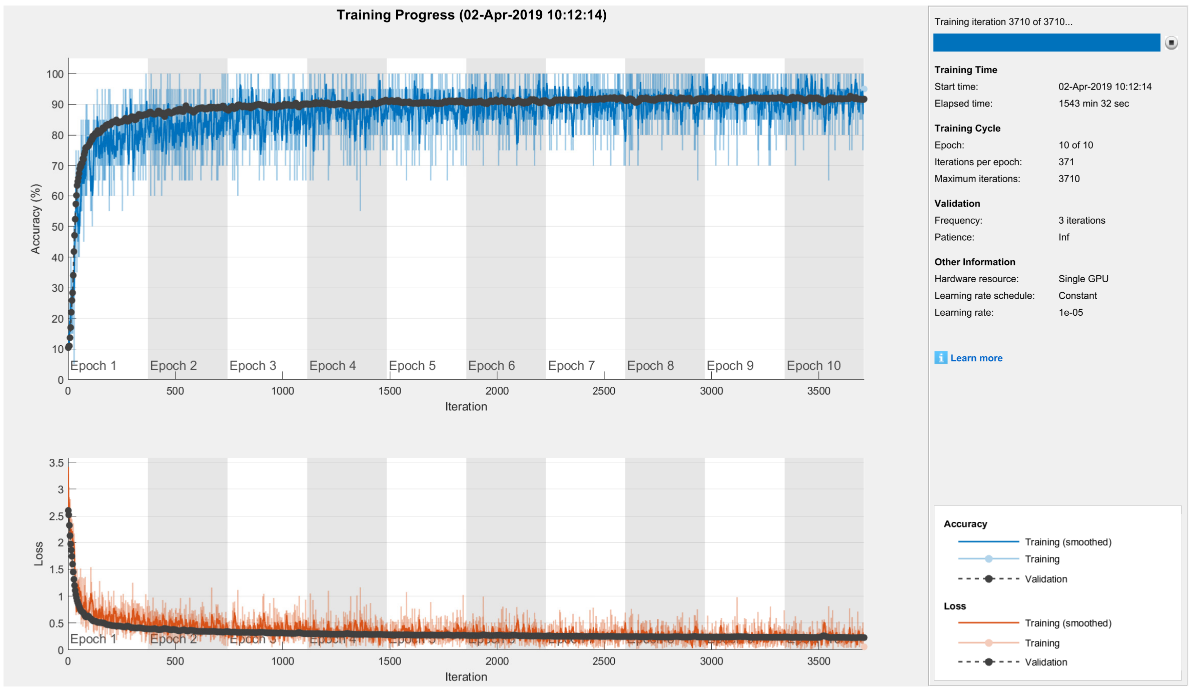
Network 6 was the first CNN in this thesis to be trained with four classes, as the class *paint flaking* was not included in networks 2-5. As shown in the first row of table 17, the results for classification of paint flaking are promising. For example, out of 205 test images in this class only four images of paint flaking were wrongly classified as either *not corrosion* or *red corrosion, steel construction*, which can be observed in the confusion matrix illustrated in figure 68. The high scores on the metrics for classification of paint flaking in table 17 are worth a few comments. Since the majority of the images in this class are generated through image cropping and rotation, some images are relatively similar to each other, however not identical. The crops were generated in such a way that areas of focus from the original images were changed for each new crop. It might be that the crops still became too similar for the CNN or that there were too few images, leading to overfitting. However, there are no indications of overfitting in the training progress illustrated in figure 67, which would have given a sudden increase in the validation loss. In addition, the validation accuracy does not exceed the average test accuracy, being 92.78 % and 96.40 % respectively. The test accuracy for *paint flaking* was 98.60 %, as shown in table 17. If the validation accuracy was higher than the average test accuracy, this would also have indicated overfitting. Paint flaking has a very characteristic appearance, and is quite unlike the images in the other classes, so this is expected to be the main reason for the high performance scores. Wrongly classified images of paint flaking are further discussed in the summary and comparison of networks in section 5.10.

As illustrated in the confusion matrix for network 6 in figure 68, there are a total of 6 (1+4+1) FPs for *not corrosion*. This means that 6 images containing damages were not detected by network 6.

5.7 Network 7-GoogLeNet

5.7.1 Training, validation and test of network 7

The training progress for network 7 is illustrated in figure 69. The confusion matrix after testing of network 7 is shown in figure 70, and table 18 shows the result for the different metrics used to evaluate CNN performance.



Training iteration 3710 of 3710...

Training Time

Start time:	02-Apr-2019 10:12:14
Elapsed time:	1543 min 32 sec

Training Cycle

Epoch:	10 of 10
Iterations per epoch:	371
Maximum iterations:	3710

Validation

Frequency:	3 iterations
Patience:	Inf

Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	1e-05

Learn more

Accuracy
- Training (smoothed)
- Training
- Validation

Loss
- Training (smoothed)
- Training
- Validation

Figure 69: Training progression for network 7. 91,70 % validation accuracy. Learning rate 1×10^{-5} . Total of 10 epochs. Elapsed time: 1543 min and 32 sec ($\sim$ 26 hours) on single GPU.

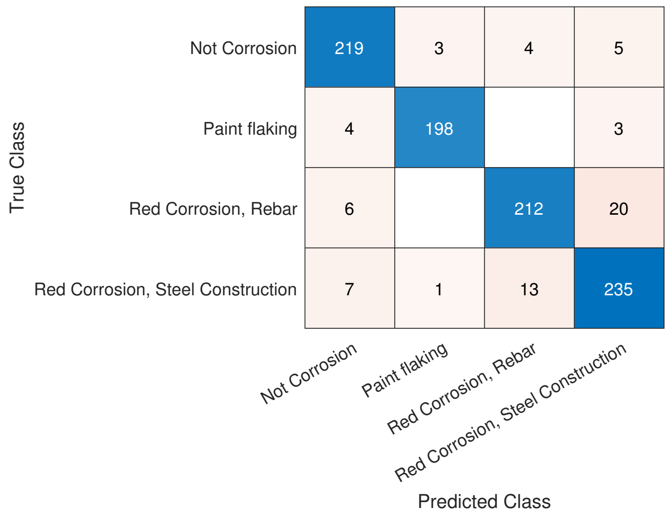


Figure 70: Confusion matrix for test of network 7 on 930 images. The blank cells in the confusion matrix means that there are zero wrongly classified images for the specific classes. In this case, no images of paint flaking were wrongly classified as *red corrosion, rebar*, and no images of red corrosion on rebar were wrongly classified as *paint flaking*.

5.7.2 Discussion of network 7

It is observed that the average performance scores are higher for network 7 than for network 6 when comparing table 17 and 18. However, the number of FPs for *not corrosion* illustrated in the confusion matrix in figure 70, show that network 7 miss out on more damages. In fact, a total of 17 (4+6+7) images of damages are not detected by network 7, compared to only 6 for network 6.

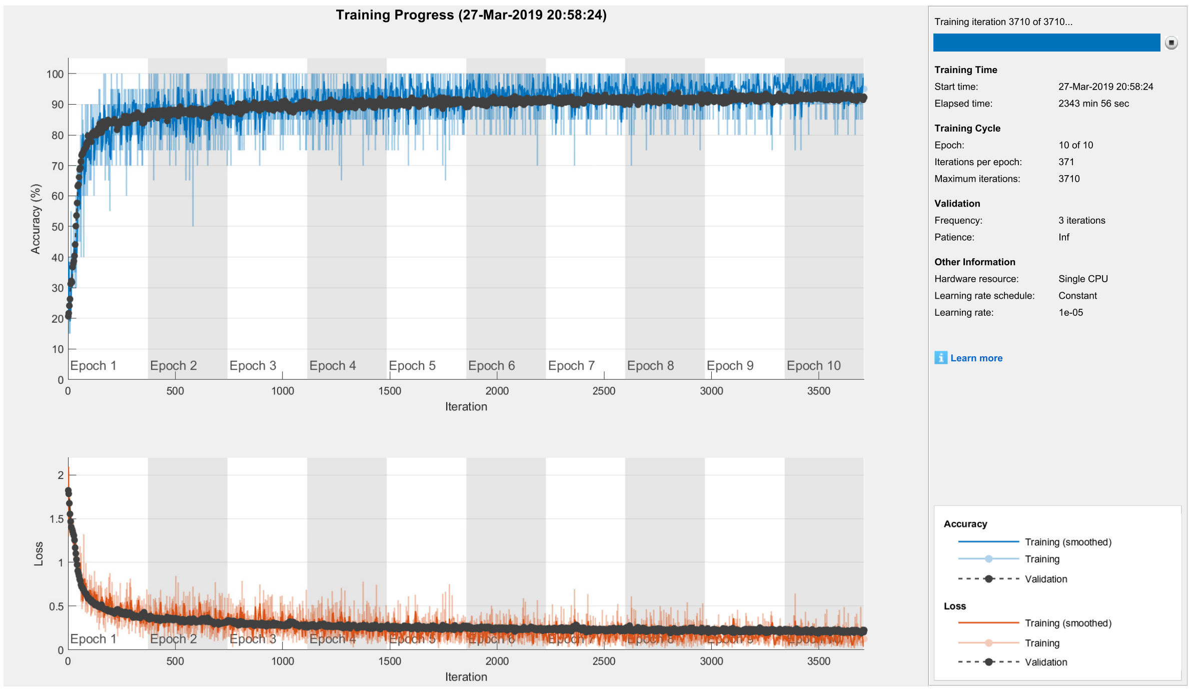
Table 18: Results from testing of network 7 presented with the metrics recall, precision, accuracy and F1 score. The average values show the overall performance of the CNN for all classes.

Class/label	**Recall**	**Precision**	**Accuracy**	**F1 score**
Paint flaking	0.9659	0.9802	0.9882	0.9730
Red corrosion, steel construction	0.9180	0.8935	0.9473	0.9056
Red corrosion, rebar	0.8908	0.9258	0.9538	0.9079
Not corrosion	0.9481	0.9280	0.9688	0.9379
Average, all classes	0.9307	0.9319	0.9645	0.9313

5.8 Network 8-ResNet-50

5.8.1 Training, validation and test of network 8

The training progress for network 8 is illustrated in figure 71. The confusion matrix after testing of network 8 is shown in figure 72, and table 19 shows the result for the different metrics used to evaluate CNN performance.



Training iteration 3710 of 3710...

Training Time

Start time:	27-Mar-2019 20:58:24
Elapsed time:	2343 min 56 sec

Training Cycle

Epoch:	10 of 10
Iterations per epoch:	371
Maximum iterations:	3710

Validation

Frequency:	3 iterations
Patience:	Inf

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	1e-05

Figure 71: Training progression for network 8. 94,18 % validation accuracy. Learning rate 1×10^{-5} . Total of 10 epochs. Elapsed time: 2343 min and 56 sec ($\sim$ 39 hours) on single CPU.

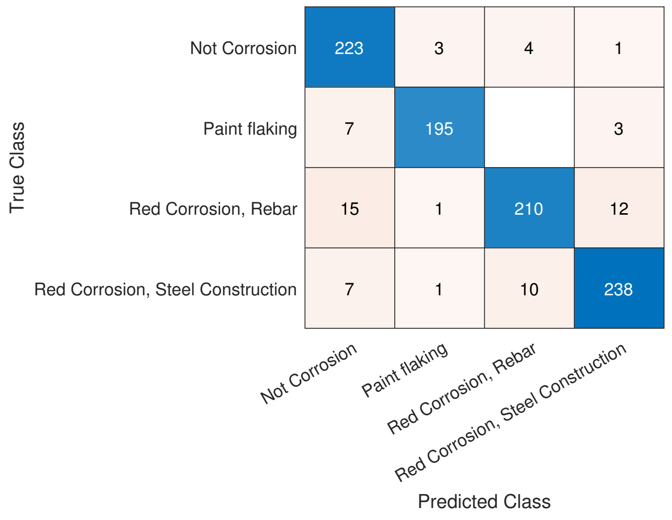


Figure 72: Confusion matrix for test of network 8 on 930 images. The blank cell in the confusion matrix means that there are zero wrongly classified images for the specific class. In this case, no images of paint flaking were wrongly classified as *red corrosion, rebar*.

5.8.2 Discussion of network 8

Comparison of table 18 and 19 show that the average performance scores are higher for network 8 than for network 7. Thus, network 8 has the highest average performance scores of all trained CNNs so far. Concurrently, the confusion matrix in figure 72 shows that network 8 has a total of 29 (7+15+7) FPs for *not corrosion*. Network 8 has the highest average performance scores so far, but out of the CNNs trained with four classes, it is also the network which misses out on most damages.

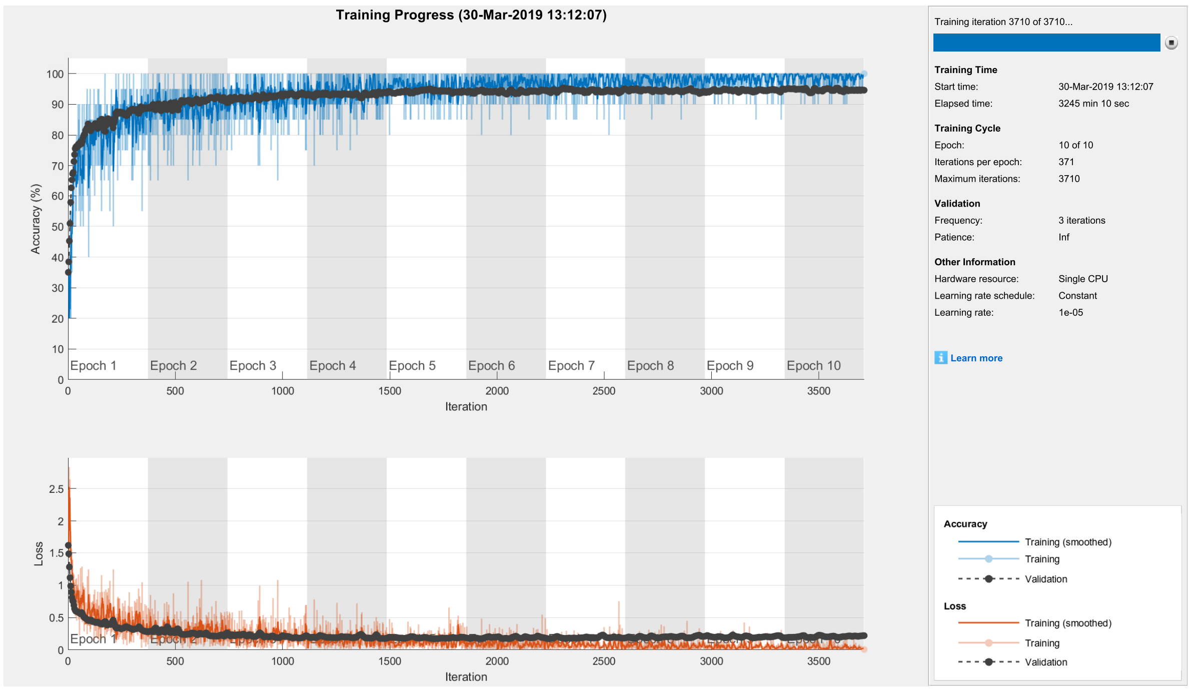
Table 19: Results from testing of network 8 presented with the metrics recall, precision, accuracy and F1 score. The average values show the overall performance of the CNN for all classes.

Class/label	**Recall**	**Precision**	**Accuracy**	**F1 score**
Paint flaking	0.9512	0.9750	0.9839	0.9630
Red corrosion, steel construction	0.9297	0.9370	0.9634	0.9333
Red corrosion, rebar	0.8824	0.9375	0.9548	0.9091
Not corrosion	0.9654	0.8849	0.9602	0.9234
Average, all classes	0.9322	0.9336	0.9656	0.9329

5.9 Network 9-VGG-16

5.9.1 Training, validation and test of network 9

The training progress for network 9 is illustrated in figure 73. The confusion matrix after testing of network 9 is shown in figure 74, and table 20 shows the result for the different metrics used to evaluate CNN performance.



Training iteration 3710 of 3710...

Training Time

Start time:	30-Mar-2019 13:12:07
Elapsed time:	3245 min 10 sec

Training Cycle

Epoch:	10 of 10
Iterations per epoch:	371
Maximum iterations:	3710

Validation

Frequency:	3 iterations
Patience:	Inf

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	1e-05

Accuracy

Training (smoothed)
Training
Validation

Loss

Training (smoothed)
Training
Validation

Figure 73: Training progression for network 9. 94,61 % validation accuracy. Learning rate 1×10^{-5} . Total of 10 epochs. Elapsed time: 3245 min and 10 sec ($\sim$ 54 hours) on single CPU.

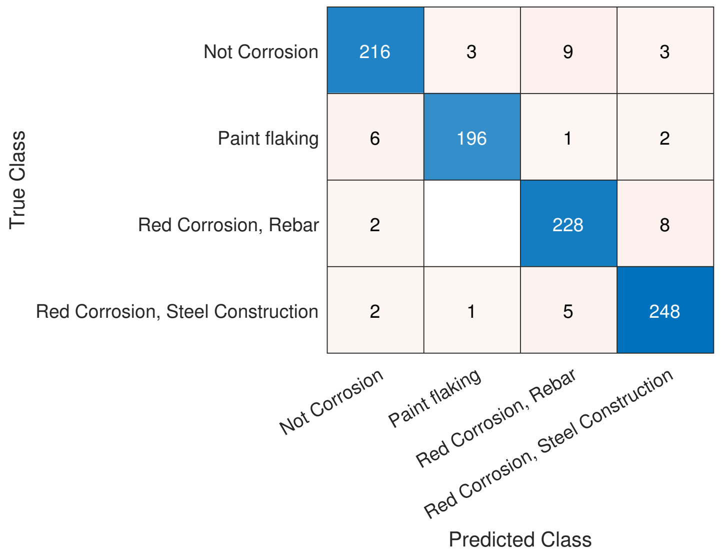


Figure 74: Confusion matrix for test of network 9 on 930 images. The blank cell in the confusion matrix means that there are zero wrongly classified images for the specific class. In this case, no images of red corrosion on rebar were wrongly classified as *paint flaking*.

5.9.2 Discussion of network 9

Table 20 shows the performance metrics after testing of network 9. When comparing with table 19 for network 8, it is observed that all average performance scores are higher for network 9. Thus, network 9 using VGG-16 is the CNN which obtains the highest average performance scores out of all networks trained in this thesis. The confusion matrix for network 9 in figure 74 illustrates that there are a total of 10 (6+2+2) FPs for *not corrosion*.

Table 20: Results from testing of network 9 presented with the metrics recall, precision, accuracy and F1 score. The average values show the overall performance of the CNN for all classes.

Class/label	**Recall**	**Precision**	**Accuracy**	**F1 score**
Paint flaking	0.9561	0.9800	0.9860	0.9679
Red corrosion, steel construction	0.9688	0.9502	0.9774	0.9594
Red corrosion, rebar	0.9580	0.9383	0.9731	0.9480
Not corrosion	0.9351	0.9558	0.9731	0.9453
Average, all classes	0.9545	0.9561	0.9774	0.9553

5.10 Summary and comparison of results

In this section, a summary of the results from CNN training and testing presented in this chapter is given. The test results for the different networks are compared using graphs and tables in order to give a more clear overview of strength and weaknesses for each CNN. This section includes an extensive discussion of the obtained results, and relevant elements for further work are suggested based on the results and discussion of these. Further work is summarized in chapter 7, together with the conclusion of this thesis. Networks 2-5 and networks 6-9 are compared in order to observe how well the CNNs perform depending on the number of classes, here three or four classes. Examples of images that were wrongly classified by several or all of the networks 2-9 are shown and discussed in this section.

A comparison of average values for recall, precision, accuracy and F1 scores for networks 2-9 is given in table 21. In an actual analysis system for damages on bridge constructions, it is important that corrosion damages and paint flaking is detected so that necessary measures of maintenance can be performed. Therefore, a comparison of how well networks 2-9 recognize a damage is given in table 22.

Table 21: Comparison of results from the different networks given as a comparison of average values for recall, precision, accuracy and F1 score for networks 2-9.

Network	Average recall	Average precision	Average accuracy	Average F1 score
Network 2	0.9268	0.9277	0.9513	0.9272
Network 3	0.9369	0.9370	0.9577	0.9369
Network 4	0.9405	0.9425	0.9605	0.9415
Network 5	0.9438	0.9448	0.9623	0.9443
Network 6	0.9292	0.9310	0.9640	0.9301
Network 7	0.9307	0.9319	0.9645	0.9313
Network 8	0.9322	0.9336	0.9656	0.9329
Network 9	0.9545	0.9561	0.9774	0.9553

The example on the next page, using the confusion matrix illustrated in figure 75, will show in detail what is meant by damage detection accuracy, and how it is calculated for all networks 2-9.

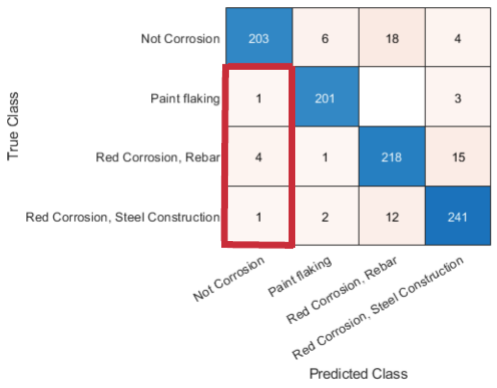


Figure 75: Illustration of a confusion matrix for use in calculation example. This is an actual confusion matrix for test of network 6 on 930 images. The elements inside the red frame are the false positives (FPs) of *not corrosion*.

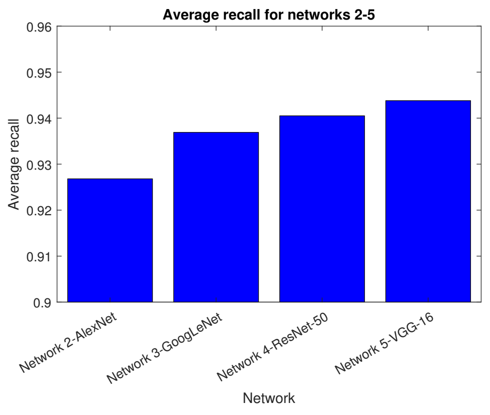
Example: The total number of images containing a damage is the sum of all test images in the classes *paint flaking*, *red corrosion, steel construction* and *red corrosion, rebar*; $205 + 256 + 238 = \underline{699}$. There are 6 FPs for *not corrosion*, as illustrated in the red frame in figure 75. The damage detection accuracy is calculated as: $\frac{699-6}{699} * 100 \approx \underline{99.14\%}$.

Table 22: Comparison of how well networks 2-9 detect images that contain a damage (corrosion and paint flaking). The proportion of damages not detected are the number of FPs for *not corrosion* divided on the number of test images containing damages. Damage detection accuracy is the percentage of all test images containing a damage that actually were detected.

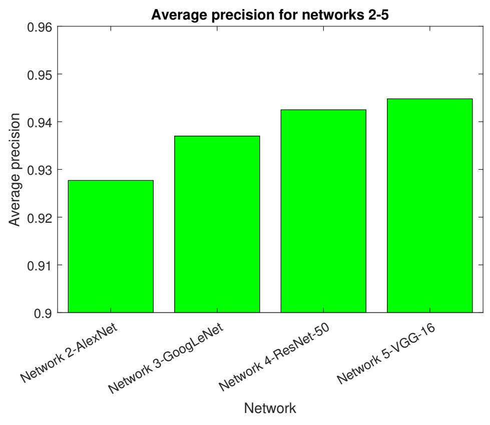
Network	**Proportion of damages not detected**	**Damage detection accuracy [%]**
Network 2	9/494	98.18
Network 3	11/494	97.77
Network 4	21/494	95.75
Network 5	5/494	98.99
Network 6	6/699	99.14
Network 7	17/699	97.57
Network 8	29/699	95.85
Network 9	10/699	98.57

5.10.1 Networks 2-5

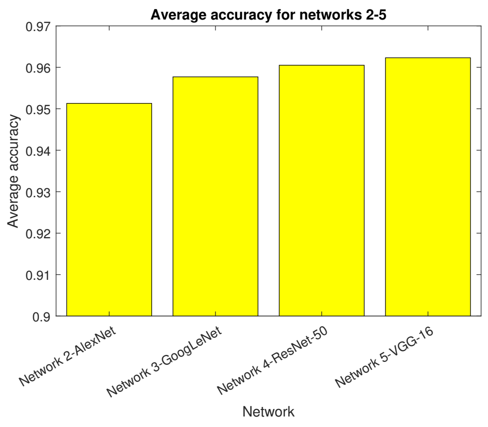
A comparison of networks 2-5 is given in figure 76. Networks 2-5 were trained and tested with the three classes *red corrosion, steel construction*, *red corrosion, rebar* and *not corrosion*. Figure 76 illustrates average values from results of image classification, using the metrics recall, precision, accuracy and F1 score. It is observed that VGG-16 has the overall highest performance scores, while AlexNet has the lowest scores of networks 2-5 after testing. A comparison of damage detection accuracy for networks 2-5 is shown in figure 77, from results in table 22.



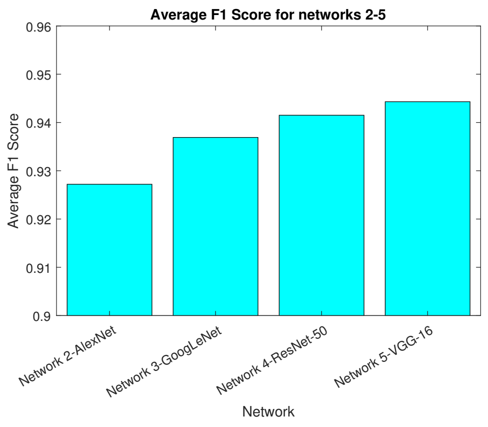
(a) Average recall



(b) Average precision



(c) Average accuracy



(d) Average F1 score

Figure 76: Comparison of average recall, precision, accuracy and F1 score for networks 2-5.

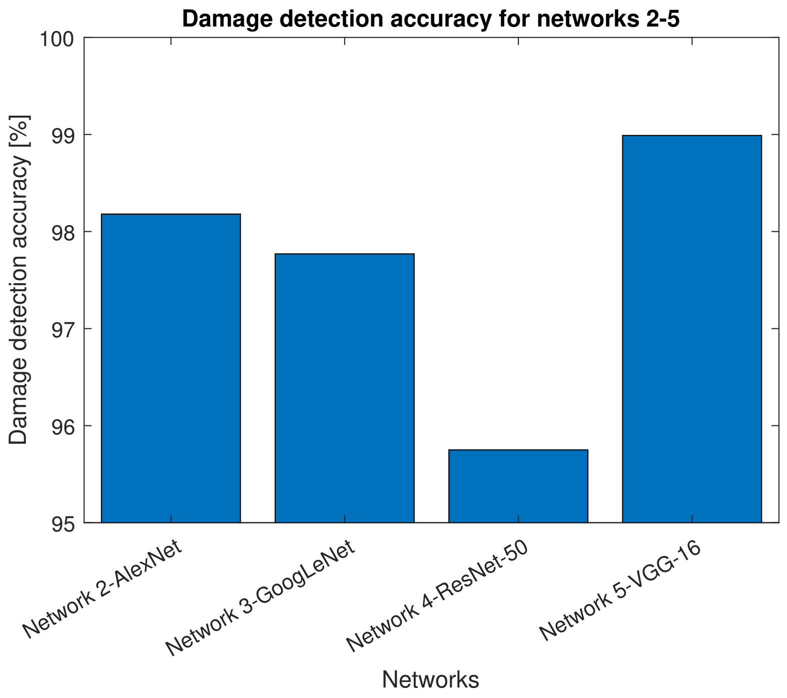
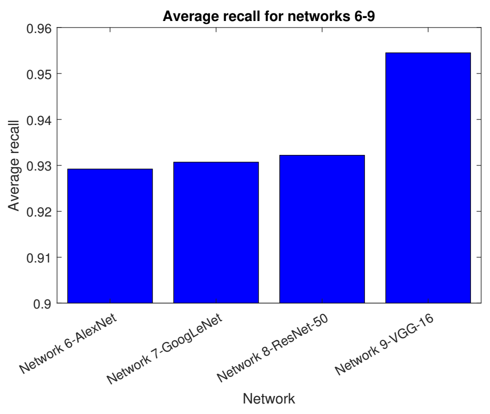


Figure 77: A comparison of damage detection accuracy for networks 2-5.

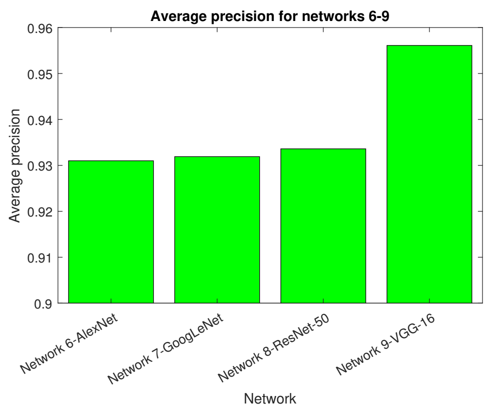
Figure 77 illustrates that network 5 has the highest damage detection accuracy of all networks 2-5. This means that network 5 has fewer FPs for *not corrosion* compared with network 2, 3, and 4, thus detects images containing a damage better. A FP for *not corrosion* means that an image containing a damage is wrongly classified as *not corrosion*, which results in an actual damage not being detected. Therefore, a high damage detection accuracy for a CNN is important so that damages on a bridge construction are detected and can be repaired. The damage detection accuracy does not say how well the network separate between different types of damages, only how well any type of damage is detected. However, as illustrated in figure 76, network 5 using the VGG-16 architecture also has the highest overall performance scores from testing.

5.10.2 Networks 6-9

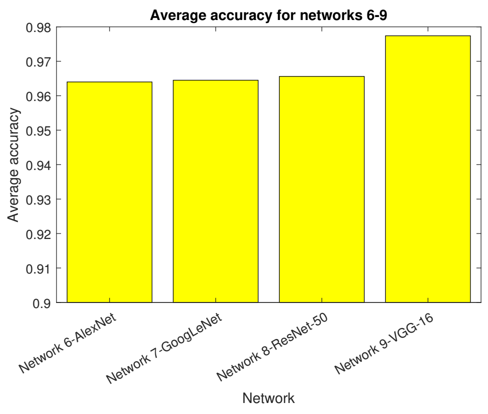
A comparison of networks 6-9 is given in figure 78. Networks 6-9 were trained and tested with the four classes *red corrosion, steel construction*, *red corrosion, rebar*, *paint flaking* and *not corrosion*. Figure 78 illustrates average values from results of image classification, using the metrics recall, precision, accuracy and F1 score. As for networks 2-5, VGG-16 has the overall best performance scores of networks 6-9. A comparison of damage detection accuracy for networks 6-9 is shown in figure 79, from results in table 22.



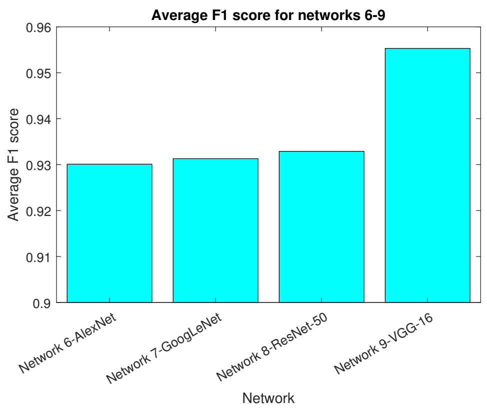
(a) Average recall



(b) Average precision



(c) Average accuracy



(d) Average F1 score

Figure 78: Comparison of average recall, precision, accuracy and F1 score for networks 6-9.

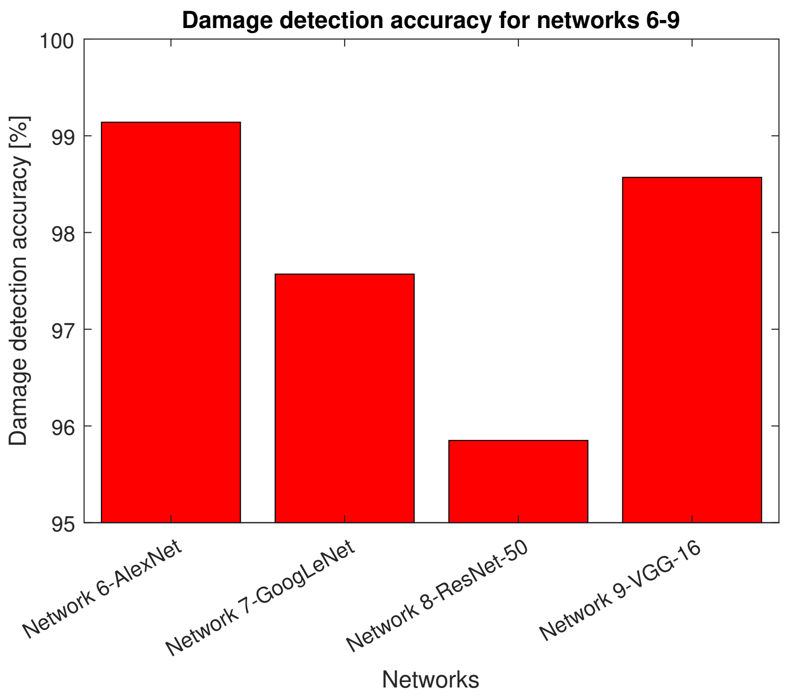
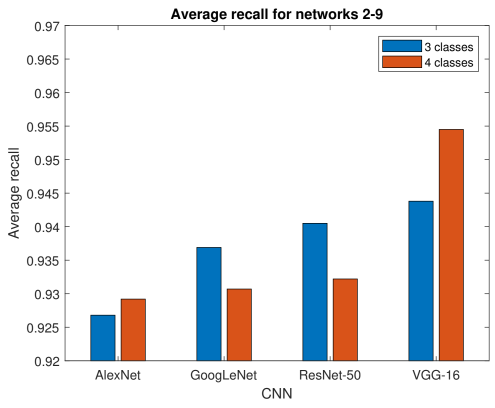


Figure 79: A comparison of damage detection accuracy for networks 6-9.

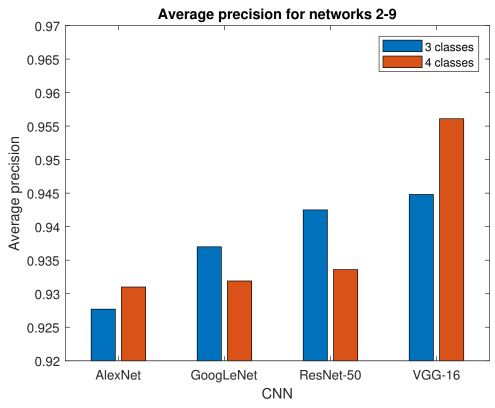
Figure 79 illustrates that network 6 has the highest damage detection accuracy of all networks 2-5. This means that network 6 has fewer FPs for *not corrosion* compared with network 7, 8, and 9, thus detects images containing a damage better. As explained previously, a FP for *not corrosion* means that an image containing a damage is wrongly classified as *not corrosion*, which results in an actual damage not being detected. Therefore, a high damage detection accuracy for a CNN is important so that damages on a bridge construction are detected and can be repaired. The damage detection accuracy does not say how well the network separate between different types of damages, only how well any type of damage is detected. For networks 6-9, network 6 has highest damage detection accuracy, but the lowest average performance scores, as illustrated in figure 78. Network 9 has the highest average performance scores of networks 6-9, and the second highest damage detection accuracy. Thus, the overall best results from testing was obtained with network 9 and the VGG-16 architecture.

5.10.3 Networks 2-9

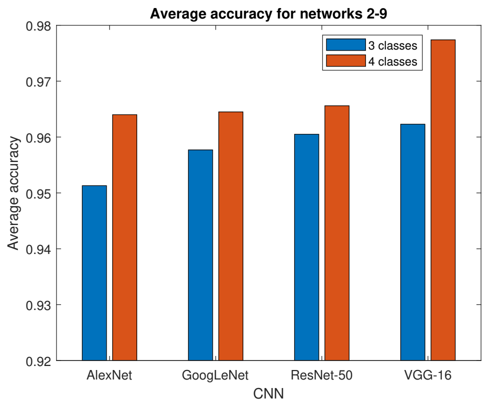
A comparison of CNN performance for all networks 2-9 is given in figure 80 by average values for the metrics recall, precision, accuracy and F1 score. This comparison illustrates how the average test results change depending on the number of classes for each of the CNNs AlexNet, GoogLeNet, ResNet-50 and VGG-16. A comparison of damage detection accuracy for networks 2-9 is shown in figure 81, from results in table 22.



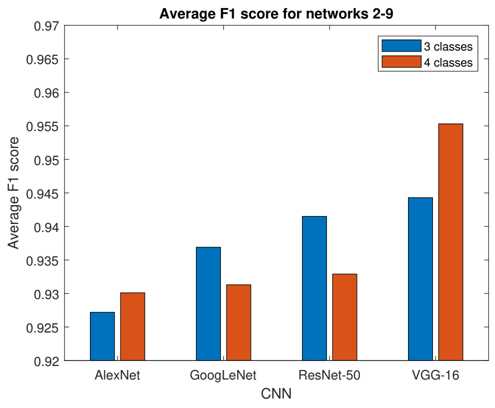
(a) Average recall



(b) Average precision



(c) Average accuracy



(d) Average F1 score

Figure 80: Comparison of average recall, precision, accuracy and F1 score for networks 2-9. The illustrations give an overview of CNN performance for 3 versus 4 classes.

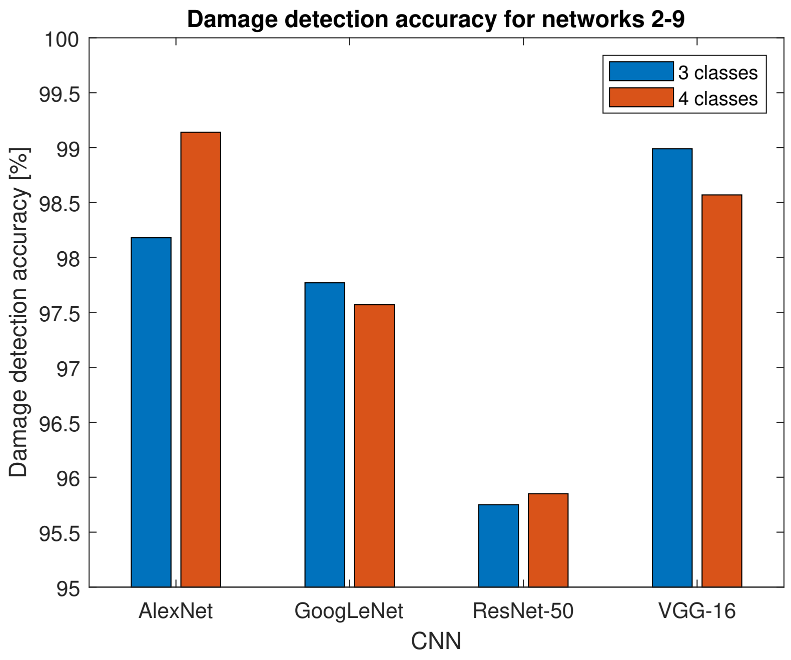
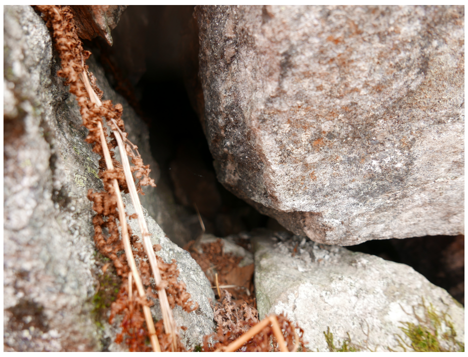
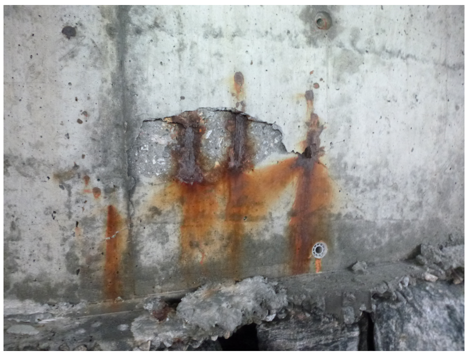


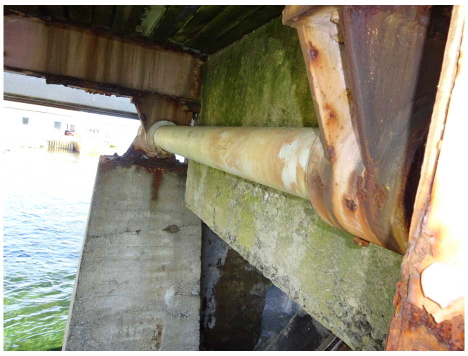
Figure 81: A comparison of damage detection accuracy for all networks 2-9.

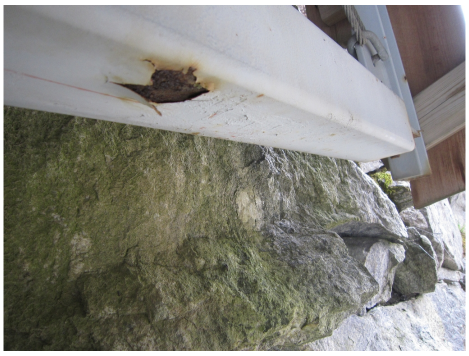
It is of importance to investigate how the CNNs perform depending on the number of classes used in training. From figures 80a, 80b and 80d it is observed that the CNNs trained using GoogLeNet and ResNet-50 has lower recall, precision and F1 score when trained with four classes. The accuracy for both GoogLeNet and ResNet-50 is however higher than in the case with three classes, as shown in figure 80c. For the CNNs AlexNet and VGG-16, all performance metrics are higher when training and testing with four classes.

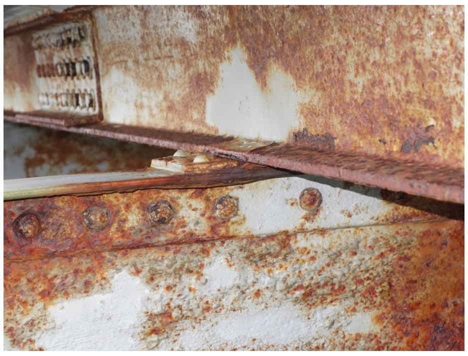
Figure 81 illustrates a comparison of damage detection accuracy for all networks 2-9, and shows how the accuracy changes depending on the number of classes used in training. As for the other performance metrics illustrated in figure 80, it is difficult to say if an additional class, thus more images, in general results in higher damage detection accuracy since the networks trained using AlexNet and ResNet-50 obtains higher accuracy with four classes, while GoogLeNet and VGG-16 performs better with three classes.


(a) *Not corrosion* classified as *Red corrosion, steel construction* by all networks 2-9.


(b) *Red corrosion, rebar* classified as *Red corrosion, steel construction* by networks 2-9.


(c) *Red corrosion, steel construction* classified as *Red corrosion, rebar* by all networks 2-9.


(d) *Red corrosion, steel construction* classified as *Red corrosion, rebar* by networks 2,3,5,6,7 and 9.


(e) *Red corrosion, steel construction* classified as *Not corrosion* by all networks, except network 6 (paint flaking).

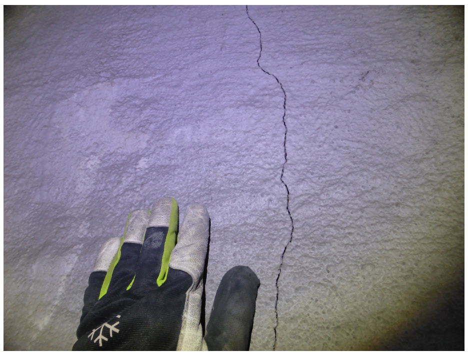

(f) *Not corrosion* classified as *Red corrosion, rebar* by networks 3,4,5, 8 and 9.

Figure 82: Examples of images that were wrongly classified by several, or all, of the networks 2-9.

It is observed that the images in figures 82a, 82b and 82c were wrongly classified by all networks 2-9. The image in figure 82a is labeled *not corrosion*, and contains rocks and plants with a brown colour. This image was chosen during data collection because it was expected to become a challenge for a CNN. The brown plants on the lighter background of rocks have both the colour and shape similar to a corrosion damage, and it is therefore not too surprising that all CNNs classify this image as *red corrosion, steel construction*. One can argue that images of plants are not relevant during bridge inspections, an thus probably wont occur in an actual analysis system. However, a wrong classification such as this should not be ignored. Therefore, collection of a even larger variety of images in the class *not corrosion* to be used in CNN training is considered a relevant element for further work.

The image in figure 82b is labeled *red corrosion, rebar*, but all networks classify this image as *red corrosion, steel construction*. One reason for this is expected to be that the pieces of corroded rebar are very close to each other, and a solution of corrosion products and water has begun to run down on the concrete, increasing the area that looks corroded. Thus, the corroded rebar looks quite like a larger area of corrosion on steel construction. As in all the other classes of images, *red corrosion, rebar* contains a variety of images with corroded rebar. A second reason for the wrong classification might be that there were not enough images of different types of corroded rebar. Thus, collecting even more images to be used in CNN training is expected to be a relevant measure for increasing the level of generalization in this class. In general, collection of more images in all classes is considered to be one of the best steps in future work for increasing performance of CNNs.

Figure 82c illustrates an image of red corrosion on steel construction that was wrongly classified as *red corrosion, rebar* by all networks 2-9. The image contains several elements of vertical and horizontal “lines” that could be interpreted as rebar. Figure 83 shows nine different activations from the third last layer in VGG-16 called fully-connected layer 8. It is observed that the original image is filtered in such a way that most edges become very visible. The light edges result in longer lines which looks quite similar to the characteristic lines created by corrosion on rebar in concrete.

The image in figure 82f was wrongly classified as *red corrosion, rebar*, but differs from the images in figures 82c and 82d, that were also classified as red corrosion on rebar, because it does not contain corrosion at all. Figure 84 illustrates nine activations from fully-connected layer 8 in VGG-16, and these images give quite a reasonable explanation for why the image of a glove and a longer crack in concrete is classified as *red corrosion, rebar*. Both the crack and the glove in figure 82f is filtered by the CNN in such a way that the characteristics of corrosion on rebar in concrete is observed. An argument for this not being a challenge in an actual inspection system for bridge constructions, is that a drone will not take images from the same view as in figure 82f. Thus, a hand or a glove is not expected to be observed on images of bridge constructions. However, images of cracks are definitely expected on images from inspections, and can be a challenge since the activations in figure 84 clearly makes it look like corrosion on rebar. In this thesis, damages like cracks in concrete or steel have not been assigned to an own class, but show up in the class *not corrosion*. One way to deal with the challenge of cracks being wrongly classified as *red corrosion, rebar* is to collect images of cracks and create an own class to be used in training of a CNN. A technique like creating regions of interest (ROI) in the images is then expected to be preferable, so that cracks in an image are found and specified instead of the entire image being labeled as, for example, *crack in concrete*. In general, utilizing ROIs could be an approach for improving the results in this thesis, since the specific areas of an image containing corrosion damages or paint flaking then are marked and labeled, instead of the entire image.

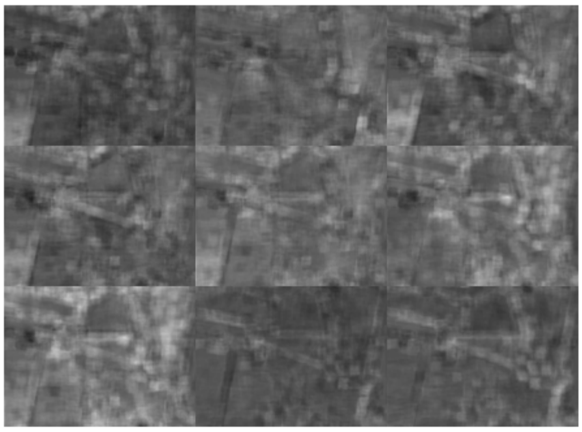


Figure 83: Examples of nine activations from the original image in figure 82c after fully-connected layer 8 in VGG16. This is the third last layer in the CNN. The characteristic vertical lines often seen for red corrosion on rebar results in this image of red corrosion on steel construction being classified as *red corrosion, rebar*.

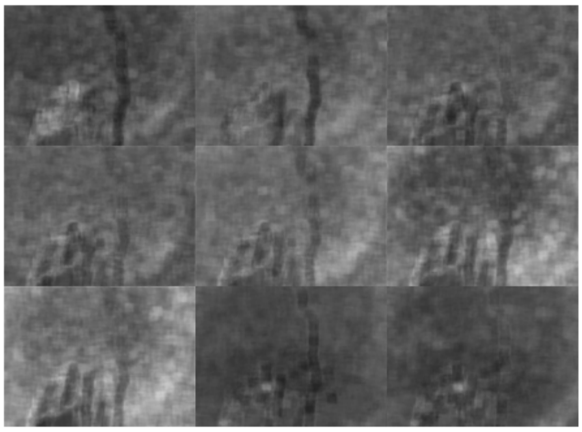


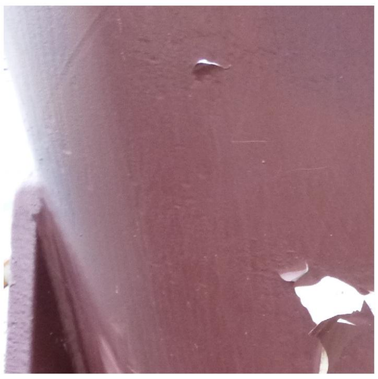
Figure 84: Examples of nine activations from the original image in figure 82f after fully-connected layer 8 in VGG16. This is the third last layer in the CNN. The characteristic vertical lines often seen for red corrosion on rebar results in this image of a glove and a long crack in concrete being classified as *red corrosion, rebar*.

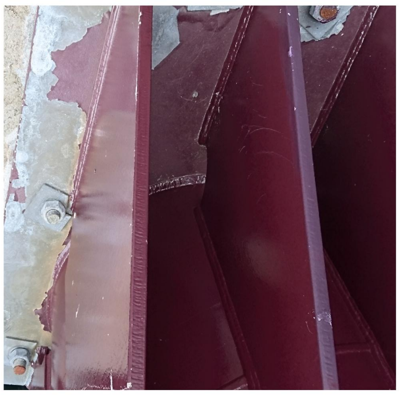
Table 23: Comparison of scores from the Softmax activation function for all classes related to classification of the image of red corrosion on steel construction in figure 82e. Networks 2-5 were not trained with images of paint flaking. Therefore, "-" is placed in the columns for these networks.

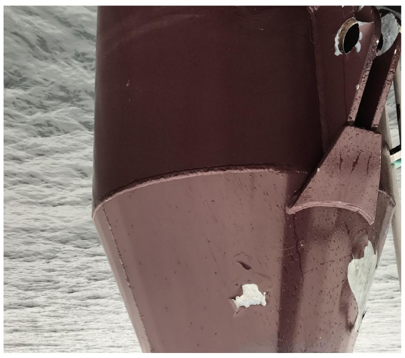
Network	**Not corrosion**	**Red corrosion steel construction**	**Red corrosion rebar**	**Paint flaking**
Network 2	0.70239055	0.22553548	0.072073989	-
Network 3	0.74095970	0.040013120	0.21902709	-
Network 4	0.97803819	0.014278459	0.0076833963	-
Network 5	0.99992251	4.7976435×10^{-5}	2.9519224×10^{-5}	-
Network 6	0.088003285	0.13495609	0.015899600	0.76114100
Network 7	0.47708878	0.11149754	0.31681144	0.094602264
Network 8	0.91197115	0.0065946043	0.016795645	0.064638525
Network 9	0.99893183	1.6765011×10^{-5}	1.4550410×10^{-5}	0.0010368616

The image in figure 82e is perhaps the most surprising of all the wrongly classified images. For the human eye, it is clear that the steel construction has corrosion damages, but all networks, except for network 6, classifies the image as *not corrosion*. Network 6 classifies the image as *paint flaking*. Table 23 shows scores from the Softmax activation function for each class related to the image in figure 82e. The Softmax activation function returns a score between 0 and 1 for each class, which is to be interpreted as a probability for the image belonging to a specific class. The sum of all scores belonging to an image is equal to 1. The Softmax activation function was introduced previously in section 2.3.3.

As the scores in table 23 show, the image of *red corrosion, steel construction* in figure 82e is classified as *not corrosion* with over 70% certainty by most networks. Networks 5 and 9, both VGG-16, have over 99% certainty in the classification of the image as *not corrosion* even though the image does contain large areas of corrosion. It is of importance to point out that even though this is somewhat of a critical wrong classification, the trained CNNs in this thesis have high performance scores in general, so this is not a common fault for the networks. However, wrong classifications such as this should not appear in an actual analysis system for bridge constructions. Therefore, the classification requires further analysis; The reason for the image in figure 82e being classified as *not corrosion* is expected to have connections to the appearance of the corrosion damages. As introduced previously in section 2.1.2, there are different types of corrosion damages, and appearances can vary a lot. The corrosion damages on the beams in figure 82e are quite spread across the surfaces, and there are areas in between that seem undamaged and therefore have a lighter colour than the corroded areas. As mentioned in some of the other cases of wrong classifications, a solution to this challenge is expected to be training of CNNs with even more images than in this thesis, and with an even larger variety in appearance.


(a) *Paint flaking* classified as *Red corrosion, steel construction* by all networks 6-9.


(b) *Paint flaking* classified as *Not corrosion* by networks 7, 8 and 9. Correctly classified as *Paint flaking* by network 6.


(c) *Paint flaking* classified as *Not corrosion* by networks 7, 8 and 9. The image was correctly classified as *Paint flaking* by network 6.

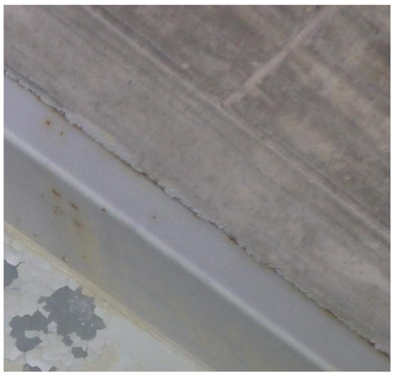

(d) *Paint flaking* classified as *Red corrosion, steel construction* by networks 6, 7 and 8, and as *Red corrosion, rebar* by network 9.

Figure 85: Examples of images containing paint flaking that were wrongly classified by several, or all, of the networks 6-9.

Figure 85 shows four examples of images containing paint flaking that were wrongly classified by several, or all, of the networks 6-9. Figure 85a was wrongly classified as *red corrosion, steel construction* by all networks 6-9. This image contains a large area of a reddish colour, being the paint on an element of steel construction. It is likely that this colour, which is similar to colours seen on corrosion, results in the networks classifying the image as *red corrosion, steel construction*. However, the image in figure 85b, which has a very similar paint colour to the image in figure 85a, is wrongly classified as *not corrosion* by network 7, 8 and 9. This observation weakens the theory of the reddish colour being the main reason for the wrong classification. The image in figure 85c is also classified as *not corrosion* by networks 7-9, and correctly classified as *paint flaking* by network 6.

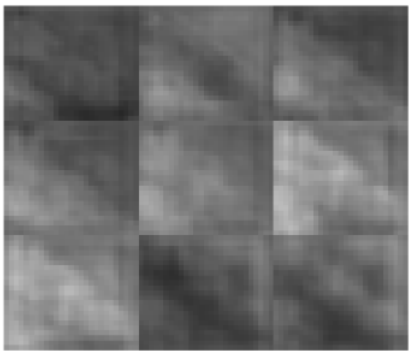


Figure 86: Examples of nine activations from the original image in figure 85d after fully-connected layer 8 in VGG16. This is the third last layer in the CNN. The characteristic vertical lines often seen for red corrosion on rebar are being classified as *red corrosion, rebar*.

The image in figure 85d was wrongly classified as *red corrosion, steel construction* by networks 6-8, and as *red corrosion, rebar* by network 9. If the image in figure 85d is studied a bit closer, one can see that it actually contains small areas of initial corrosion damages on the left from the image centre. The image was assigned to the class *paint flaking* because paint flaking is the most prominent damage in the image. The area of corrosion was considered so small that it was not expected to become a challenge. However, the classification result indicates that having two different damages in the same image can be a challenge for a CNN. Network 9 classifies the image in figure 85d as *red corrosion, rebar*, which may be explained by the transition between steel and concrete that creates a long line on the image diagonal, similar to a crack or rebar. To study the reason for this wrong classification, an image of different activations from fully-connected layer 8 in VGG-16 is given in figure 86. It is difficult to tell from these activations why networks 6-8 classify the image in figure 85d as *red corrosion, steel construction*. However, the diagonal lines on all activations in figure 86 give a reasonable explanation for the wrong classification *red corrosion, rebar* by network 9.

As stated previously in the discussion of results in this section, utilization of ROIs is considered relevant in further work for improvement of test results, through labeling of the specific areas of an image containing paint flaking, and not the entire image.

Chapter 6

Discussion

This chapter will discuss the main elements of the master's thesis such as data collection and categorization, the choice of software and CNNs, choices regarding setup of CNN training and testing, results and further work. The majority of discussion related to obtained results was given in section 5.10, so this section includes a more general discussion of training, testing and results. Relevant elements for further work were also given in section 5.10, and are suggested here as well. Further work is summarized in chapter 7, together with the conclusion of this master's thesis.

Data collection and choices regarding images

The images received from the Norwegian Public Roads Administration were manually evaluated and categorized, which means that the choice of images for CNN training and testing was somehow subjective. However, it was put great effort in ensuring a variety of images in all classes to reflect the many types of damage appearances that exist within each category. In addition, images with very poor image quality or no relevant information were manually filtered and put aside during data collection. Even though a critical evaluation and choice of images was performed, it was also considered important to choose some images that were difficult to classify even for a human. In that way, the images used to train and test the CNNs became both realistic and challenging, similar to damage classification during real inspections of bridge constructions.

Regarding image quality, the received data set contained images from bridge inspections taken with several different camera types, depending on the year of inspection and by whom it was performed. Thus, the training, validation and test sets used for the CNNs contain images with varying resolution. As mentioned, images with very poor quality were not chosen. In addition, light conditions on images differ to a certain degree, meaning that some images are taken in daylight while others are taken at evening or night. The majority of images used in this thesis were taken in daylight. The images used in this thesis also differ in terms of how close to a bridge construction they were taken. All images in the received data set were taken manually by bridge inspectors, not by drones. In a potential application for inspection, using a drone with a camera to provide images to be classified by a CNN, it is considered preferable to set requirements related to how close to the bridge construction an image should be taken. If an image is taken from a relatively far distance, relevant information regarding a damage may not be captured. For example, corrosion might not be detected if a corrosion damage makes up a too small amount of the total area in an image. On the other hand, an image taken too close to the construction may not either be preferable.

The types of images in the category *not corrosion* vary a lot, from images of concrete and steel with and without cracks to images of multi coloured graffiti, leaves and trees. The reason that so many different categories of images were assigned to a single class was to "challenge" the CNNs. During the manual categorization of images, autumn leaves and reddish graffiti were expected to be difficult for a CNN to classify because it sometimes can remind of corrosion in appearance. It was considered important to not only choose "obvious" images of *not corrosion*, since this could have questioned the CNNs' ability of generalizing *not corrosion* and separating it from an actual corrosion and coating damage. As for example the images in figure 82a and 82f show, a brown plant and a glove combined with a crack in concrete were apparently not easy to classify as *not corrosion* for the CNNs in these two specific cases. However, as mentioned earlier, one can question how likely it is that images of plants will occur in an actual analysis system for damages on bridge constructions, but at the same time this is not a classification fault that should be ignored. Collecting more images of *not corrosion* containing elements very similar to corrosion, and utilization of ROIs or segmentation, are considered relevant measures to avoid classification faults like those illustrated in figures 82a and 82f. It would have been preferable to create an own class for cracks in concrete, since this also is a surface damage. Unfortunately, not enough images of this category were found to realize an additional class. However, through the use of data augmentation techniques on the images of cracks collected during this thesis, one could perhaps be able to generate enough images for creating such a class.

Regarding the class *white corrosion* that was not included in this thesis due to few collected images; this class is considered important for an inspection analysis system because it reveals damages on zinc coating, damages that have evolved from for example paint flaking, but not yet reached a point of red corrosion on steel. In further work, the collection of more such images should therefore be investigated in order to create an own class in CNN training for this damage category. The Norwegian Public Roads Administration should be consulted for perhaps finding even more relevant classes, since they are the potential end user of a damage analysis system such as the one evaluated in this master's thesis using CNNs.

Choice of software and CNNs

MATLAB was the utilized software for deep learning in this thesis, and one reason for this choice of software is that MATLAB provides very practical and understandable user interfaces on software tools for tasks like image pre-processing, and neural network training and testing. Also, because MATLAB is the software in which the student has most programming experience, this became the reasonable choice of programming language. Python is perhaps the most popular programming language for deep learning, but there were not experienced any limitations with the use of MATLAB in this thesis.

The CNNs that were compared in this thesis were chosen because each of these network architectures are some of the most well known CNNs, and all have contributed to the state of the art in image classification throughout the years. It would definitely been interesting to study even more CNNs in this thesis, but at the same time the chosen four are some of the best CNN architectures throughout the years. In addition, if more tuning is performed, these could obtain even better performance scores than shown from results in this thesis. Also, more training images could help. Regarding future work, it is considered relevant to both test other CNN architectures than the four used here, and to create a network architecture from scratch, using results obtained here to modify layers in a preferable manner.

Training and testing

The training options were chosen to be equal for all networks. This means that the same learning algorithm was used, here SGDM, and options such as learning rate, mini-batch size and number of epochs were the same for all networks 2-9. This was done in order to ensure a best possible basis for comparison of the CNNs. Because the same training options were used for all networks, the optimal performance of each network is not expected to have been achieved. It means that CNNs like AlexNet and GoogLeNet, which have lowest performance scores in this thesis, not necessarily have the lowest performance scores in general. However, finding the optimal training options for each CNN was not the aim of this thesis. This thesis presents a comparison of different CNN architectures in order to say something about how well CNNs can classify corrosion and coating damages on bridge constructions, and the presented results are promising. Testing of other training algorithms and parameters could be relevant for further work, if one is interested to find the optimal training options for a CNN.

The partitioning of training, validation and test sets were chosen as 80%, 10% and 10% of all categorized images, respectively. In MATLAB, the images were then automatically divided into each set by assigning the first 80 % of the images in all folders to the training set, and then dividing the remaining 20% equally into the two sets for validation and testing. It could be that a different partitioning of training, validation and test set would have been preferable, and given different results than those obtained in this thesis. However, more training images are in general expected to be of greater significance for the results than the partitioning of images itself.

The available hardware for training of CNNs in this thesis was shown in table 11, and specifies exactly what hardware was used for training of each network 1-9. The reason most networks were trained on a single CPU is that this was the only available hardware at the time. Network 6 and 7 were trained using a computer with GPU, while the rest of the networks were trained on a single CPU.

Results

The obtained results in this thesis show that CNNs have a great potential in analyzing corrosion and coating damages on bridge constructions. For evaluation of classification results from the CNNs in this thesis, the performance metrics recall, precision, accuracy and F1 score were chosen. These are very common performance metrics for evaluation of both binary and multi-class classification problems in machine learning. The obtained test results for all networks were compared and discussed in section 5.10. Network 9 (VGG-16) obtained the highest average performance scores, while network 6 (AlexNet) obtained the highest damage detection accuracy of all CNNs trained and tested in this thesis. Even if a CNN has higher average scores on metrics, a different CNN could do better on classification of a certain class. For example, network 7 (GoogLeNet) has higher performance scores on classification of *paint flaking* compared with network 8 (ResNet-50), even though network 8 has higher average performance scores for all metrics.

Each of the networks 2-9 were trained once on the categorized data sets, thus the performance scores for each network show results based on a single CNN training. In general, it would be preferable to perform more than one training per network in order to calculate an average and standard deviation, thus being able to say with higher confidence that the results for each network are representative for the CNNs AlexNet, GoogLeNet, ResNet-50 and VGG-16. However, since training CNNs is quite time consuming and the time for implementation of the master's thesis is limited, the chosen setup of training and testing was considered to give a sufficient comparison of CNNs to reach the aim of the thesis.

Chapter 7

Conclusion and further work

This master's thesis has presented a comparison of performance for the convolutional neural networks AlexNet, GoogLeNet, ResNet-50 and VGG-16 on image classification of corrosion and coating damages from bridge constructions. A total of 9300 images were collected through manual categorization and augmentation techniques, and divided into four classes; *paint flaking*, *red corrosion on rebar*, *red corrosion on steel construction* and *not corrosion*, which were used to train and test the convolutional neural networks. The aim of the thesis was to compare performance for convolutional neural networks in order to evaluate the potential in using deep learning for automatic analysis of damages on bridge constructions found during inspections. From the presented results, it is concluded that convolutional neural networks are well suited for use in an automatic analysis system for corrosion and coating damages. VGG-16 was the network architecture with highest performance scores on recall, precision, accuracy and F1 score, being 95.45%, 95.61%, 97.74% and 95.53%, respectively. Training and testing of AlexNet with four classes gave the highest score on general damage detection with an accuracy of 99.14%. This thesis show that convolutional neural networks can be used in bridge inspection to both detect and analyze corrosion and coating damages. In combination with a drone and localization system using a 3D model of a bridge, an automatic analysis system based on a convolutional neural network can provide information automatically regarding where maintenance and repairs are needed. Regarding further work, the following elements are considered relevant based on discussion of results: (1) Collection of more images in the four classes used in this thesis, and evaluation of the need for additional classes such as *white corrosion* and *cracks in concrete*. (2) Utilization of regions of interest for labeling and/or segmentation of images to improve the results presented in this thesis. More images may not be necessary if the use of such techniques proves to be successful. (3) Testing of other training algorithms and parameters, and optionally modification of layers in CNN architectures.

Bibliography

[1] SafeControl AS. Fakta om korrosjon og kostander. https://safecontrolgruppen.no/frosio-ns-476/. (Accessed: 08.01.2019).

[2] Overflateportalen. Kort om korrosjon. https://www.overflateportalen.no/kort-om-korrosjon/. (Accessed: 08.01.2019).

[3] NACE Intenational. International measures of prevention, application and economics of corrosion technology (impact). http://impact.nace.org/economic-impact.aspx. (Accessed: 08.01.2019).

[4] Statistisk Sentralbyrå (Statistics Norway). Årlig nasjonalregnskap, 1970-, tabell 9. https://www.ssb.no/nasjonalregnskap-og-konjunkturer/tabeller/nr-tabeller. (Accessed: 29.01.19).

[5] Statens Vegvesen. Bruer. https://www.vegvesen.no/fag/teknologi/bruer. (Accessed: 07.02.2019).

[6] E. Holm. Automatic inspection of bridge constructions. *TTK4551 Specialization project, NTNU*, 2018.

[7] L. Petricca, T. Moss, G. Figueroa, and S. Broen. Corrosion detection using a.i : A comparison of standard computer vision techniques and deep learning model. *Computer Science and Information Technology*, 6, 2016.

[8] D.J. Atha and M.R. Jahanshahi. Evaluation of deep learning approaches based on convolutional neural networks for corrosion detection. *SAGE journals*, 17(5), 2018.

[9] ICMV. Icmv 2019 official website. http://icmv.org/. (Accessed: 16.05.19).

[10] E. Bardal. *Korrosjon og korrosjonsvern*. Tapir Akademisk Forlag, Trondheim, 1994.

[11] W.D. Callister and D.G. Rethwisch. *Materials Science and Engineering*. John Wiley and Sons, 2015.

[12] Unknown. Corrosion electrochemistry. https://corrosion-doctors.org/

Electrochemistry-of-Corrosion/Introduction.htm. (Accessed: 22.01.19).

[13] Nada Al-Eidan. Types of corrosion. https://www.researchgate.net/post/What_are_the_most_dangerous_types_of_corrosion_and_does_the_dangerous_vary_depending_on_the_type_of_metal. (Accessed: 22.01.19).

[14] NACE International. Uniform corrosion. https://www.nace.org/corrosion-central/corrosion-101/uniform-corrosion/. (Accessed: 22.01.19).

[15] Gibson Stainless and Speciality Inc. Corrosion types and prevention. https://www.gibsonstainless.com/types-of-corrosion.html. (Accessed: 23.01.19).

[16] Steel Fabrication Services. Pitting corrosion and how to treat it. https://steelfabservices.com.au/pitting-corrosion-how-to-treat-it/. (Accessed: 23.01.19).

[17] Burt Victoria. *Corrosion in the Petrochemical Industry*. ASM International, 2015.

[18] NACE International. Galvanic corrosion. https://www.nace.org/corrosion-central/corrosion-101/galvanic-corrosion/. (Accessed: 28.01.19).

[19] M.G Fontana. *Corrosion Engineering*. McGraw-Hill Book Company, 3rd edition, 1986.

[20] STRUCTURE Magazine. Fastener corrosion. https://www.structuremag.org/?p=9609. (Accessed: 28.01.19).

[21] Statens Vegvesen. Bruforvaltningssystemet brutus. https://www.vegvesen.no/fag/teknologi/bruer/bruforvaltning. (Accessed: 07.02.2019).

[22] Statens Vegvesen. Overflatebehandling av stål. https://www.vegvesen.no/fag/fokusomrader/forskning+og+utvikling/Avsluttede+FoU-program/Varige+konstruksjoner/Prosjekter/Fremtidens+bruer/overflate-stal/fb-9-overflatebehandling-av-st%C3%A5l. (Accessed: 12.02.19).

[23] Statens Vegvesen. Varige konstruksjoner, tilstandsutvikling bruer. https://www.vegvesen.no/_attachment/433872/binary/729734?fast_title=Overflatebehandling+av+st%C3%A5lbruer.pdf. (Accessed: 12.02.19).

[24] S. Kuroda. Thermal spray coatings for corrosion protection in atmospheric and aqueous environments. *ASM Handbook*, 13B, 2005.

[25] O.Ø. Knudsen. Korrosjonsbeskyttelse for stålbruer. *Statens Vegvesen, Etatsprogrammet Varige konstruksjoner 2012-2015*, 2015.

[26] Statens Vegvesen. *Inspeksjonshåndbok for bruer V441*. Vegdirektoratet, 2000.

[27] Statens Vegvesen. *Bruforvaltning riksveg:Forvaltning av bærende konstruksjoner på riksveg*. Vegdirektoratet, 2018.

[28] W. Ertel. *Introduction to Artificial Intelligence*. Springer International Publishing, Cham, 2017.

[29] A. Gupta. *Introduction to deep learning: Part 1*. American Institute of Chemical Engineers, 2018.

[30] S.J. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach, Third Edition*. Pearson, Boston, 2016.

[31] A. Tidemann and A.C. Elster. Maskinlæring. https://snl.no/maskinl\T1\aering. (Accessed: 16.01.19).

[32] Ola Nordal. Alan turing. https://snl.no/Alan_Turing. (Accessed: 23.01.19).

[33] T. Mitchell. *Machine learning*. McGraw Hill, 1997.

[34] Md.Rezaul Karim and Giancarlo Zaccone. *Deep Learning with TensorFlow*. Packt Publishing, second edition, 2018.

[35] Mohssen Mohammed, Muhammad Badruddin Khan, and Eihab Bashier Mohammed Bashier. *Machine learning*. Taylor and Francis Group, 2017.

[36] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521, 2015.

[37] Unknown. How neural networks work. https://chatbotslife.com/how-neural-networks-work-ff4c7ad371f7. (Accessed: 16.01.19).

[38] A. Tidemann. Dyp læring. https://snl.no/dyp_l\T1\aering. (Accessed: 16.01.19).

[39] Knoldus. Introduction to perceptron: neural network. https://blog.knoldus.com/introduction-to-perceptron-neural-network/. (Accessed: 24.01.19).

[40] M.A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[41] Nahua Kang. Multi-layer neural network with sigmoid function. https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for- (Accessed: 24.01.19).

[42] Wikimedia Commons. Hyperbolic tangent. https://commons.wikimedia.org/wiki/File:Hyperbolic_Tangent.svg. (Accessed: 30.01.19).

[43] Sebastian Raschka. Machine learning faq. https://sebastianraschka.com/faq/docs/relu-derivative.html. (Accessed: 30.01.19).

[44] Mwamba Capital. Simple gradient descent. https://www.youtube.com/watch?v=riplXsNf_zs. (Accessed: 31.01.19).

[45] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, second edition, 2006.

[46] MathWorks. Options for training deep learning neural network. https://se.mathworks.com/help/deeplearning/ref/trainingoptions.html. (Accessed: 09.05.19).

[47] M. Mahmoud, A. Bukhary, A. G Ramadan, and M. Al-Zahrani. Prediction of breach peak outflow and failure time using artificial neural network approach. In *Proceedings of the 2nd World Congress on Civil, Structural, and Environmental Engineering (CSEE'17)*, 2017.

[48] G.B. Orr and K.R Muller. *Neural Networks: Tricks of the Trade*. Springer, 1998.

[49] Lucidchart. Official website. https://www.lucidchart.com. (Accessed: 15.02.19).

[50] J. Ahmad, K. Muhammad, and S.W. Baik. Data augmentation-assisted deep learning of hand-drawn partially colored sketches for visual search. *PLoS ONE*, 12, 2017.

[51] Google. Machine learning crash course. https://developers.google.com/machine-learning/crash-course/. (Accessed: 15.02.19).

[52] H.H. Aghdam. *Guide to convolutional neural networks : a practical application to traffic-sign detection and classification*. Springer, 2017.

[53] MathWorks. Learn about convolutional neural networks. https://se.mathworks.com/help/deeplearning/ug/introduction-to-convolutional-neural-networks.html. (Accessed: 18.02.19).

[54] MathWorks. Specify layers of convolutional neural network. https://se.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html. (Accessed: 18.02.19).

[55] MathWorks. Get started with transfer learning. https://se.mathworks.com/help/deeplearning/examples/get-started-with-transfer-learning.html. (Accessed: 18.02.19).

[56] ImageNet. Official website. http://www.image-net.org/. (Accessed: 21.02.19).

[57] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60, 2017.

[58] Sunita Nayak. Understanding alexnet. https://www.learnopencv.com/understanding-alexnet/. (Accessed: 30.01.19).

[59] M.D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

[60] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-, 2015.

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

[62] K. He, Y. Wang, and J.E. Hopcroft. A powerful generative model using random weights for the deep image representation. *CoRR*, abs/1606.04801, 2016.

[63] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[64] ImageNet. Large scale visual recognition challenge 2016 (ilsvrc2016). http://image-net.org/challenges/LSVRC/2016/results. (Accessed: 25.02.19).

[65] ImageNet. Large scale visual recognition challenge 2017 (ilsvrc2017). http://image-net.org/challenges/LSVRC/2017/results. (Accessed: 25.02.19).

[66] K. Nguyen, C. Fookes, A. Ross, and S. Sridharan. Iris recognition with off-the-shelf cnn features: A deep learning perspective. *IEEE Access*, 6, 2018.

[67] Python. Official website. https://www.python.org/. (Accessed: 30.01.19).

[68] TensorFlow. An open source machine learning framework for everyone. https://www.tensorflow.org/. (Accessed: 30.01.19).

[69] Keras. Keras: The python deep learning library. https://keras.io/. (Accessed: 25.02.19).

[70] PyTorch. From research to production. https://pytorch.org/. (Accessed: 30.01.19).

[71] MATLAB. Official website. https://se.mathworks.com/products/matlab.html. (Accessed: 30.01.19).

[72] MATLAB. Deep learning toolbox. https://se.mathworks.com/products/deep-learning.html. (Accessed: 30.01.19).

[73] ONNX. Open neural network exchange format. https://onnx.ai/. (Accessed: 25.02.19).

[74] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[75] MathWorks. Deep learning. https://se.mathworks.com/solutions/deep-learning.html. (Accessed: 25.02.19).

[76] OpenCV Team. Opencv website. https://opencv.org/. (Accessed: 15.01.19).

[77] J. Du, L. Yan, H. Wang, and Q. Huang. Research on grounding grid corrosion classification method based on convolutional neural network. *MATEC Web of Conferences*, 160, 2018.

[78] Adobe. Adobe photoshop cc. https://www.adobe.com/products/photoshop.html. (Accessed: 11.03.19).

[79] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86, 1998.

[80] Statens Vegvesen. Na rundskriv 2018/4 endringer til håndbok v441 inspeksjonshåndbok for bruer. https://www.vegvesen.no/_attachment/2265596/binary/1251794?fast_title=NA-rundskriv+2018%2F4+-+Endringer+til+h%C3%A5ndbok+V441+Inspeksjonshandbok+for+bruer.pdf. (Accessed: 25.01.19).

[81] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45, 2009.

[82] MathWorks. Deep learning toolbox model for alexnet network. https://se.mathworks.com/matlabcentral/fileexchange/59133-deep-learning-toolbox-model-for-alexnet-network?s_tid=FX_rc1_behav. (Accessed: 18.03.19).

[83] MathWorks. Deep learning toolbox model for googlenet network. https://se.mathworks.com/matlabcentral/fileexchange/64456-deep-learning-toolbox-model-for-googlenet-network?s_tid=FX_rc3_behav. (Accessed: 18.03.19).

[84] MathWorks. Deep learning toolbox model for resnet-50 network. https://se.mathworks.com/matlabcentral/fileexchange/64626-deep-learning-toolbox-model-for-resnet-50-network?s_tid=FX_rc2_behav. (Accessed: 18.03.19).

[85] MathWorks. Deep learning toolbox model for vgg-16 network. https://se.mathworks.com/matlabcentral/fileexchange/61733-deep-learning-toolbox-model-for-vgg-16-network. (Accessed: 24.03.19).

[86] MathWorks. Datastore for image data. https://se.mathworks.com/help/matlab/ref/matlab.io.datastore.imagedatastore.html. (Accessed: 25.03.19).

[87] MathWorks. Split imagedatastore labels by proportions. https://se.mathworks.com/help/matlab/ref/matlab.io.datastore.imagedatastore.spliteachlabel.html. (Accessed: 13.05.19).

[88] MathWorks. Crop image. https://se.mathworks.com/help/images/ref/imcrop.html. (Accessed: 25.03.19).

[89] MathWorks. Rotate image. https://se.mathworks.com/help/images/ref/imrotate.html. (Accessed: 02.04.19).

[90] MathWorks. Transform batches to augment image data. https://se.mathworks.com/help/deeplearning/ref/augmentedimagedatastore.html. (Accessed: 25.03.19).

[91] MathWorks. Train deep learning network to classify new images. https://se.mathworks.com/help/deeplearning/examples/train-deep-learning-network-to-classify-new-images.html. (Accessed: 02.04.19).

[92] MathWorks. Deep learning onramp course- transfer learning workflow. https://matlabacademy.mathworks.com/R2018b/portal.html?

course=deeplearning#chapter=4&lesson=2&section=2. (Accessed: 14.01.2019).

[93] MathWorks. Classify data using a trained deep learning neural network. https://se.mathworks.com/help/deeplearning/ref/classify.html. (Accessed: 02.04.19).

Appendix

- **A**: Tasks and progress plan
- **B**: Scientific paper
- **C**: MATLAB code
  -AlexNet
  -GoogLeNet, ResNet-50 and VGG-16
  -findLayersToReplace.m (downloaded from [91])
- **D**: MATLAB course certificate

A: Tasks and progress plan

Tasks and progress plan for master's thesis, spring 2019
Student: Egil Holm

Tasks

- Read and write about previous work/methods on corrosion detection and classification from images using deep learning. Write theory on artificial intelligence, machine learning, deep learning and corrosion.

- Find a relevant dataset containing images of corrosion damages. Images from previous bridge inspections in Brutus are relevant. Label images using appropriate software tools.

- Choose software and networks for the classification tasks. Modify existing neural network, add/remove layers and adjust parameters/weights. Perform training and testing of neural networks.

- Present results in a suitable way using graphs and tables. Find and express measurements of performance for the trained neural networks.

- Discuss results from testing of the neural network, explain what the results show and how this can be used, and what could have been done differently. Suggest relevant elements for further work.

In addition, the student will follow relevant lectures in the three courses;

- TDT4171- Artificial Intelligence Methods
- TDT4173- Machine Learning and Case-Based Reasoning
- TDT4265- Computer Vision and Deep Learning

Progress plan

Week	Progress
2	**January and February**: Read and write about previous work and theory (corrosion and machine learning) Start working with images; gathering and labelling of images from Brutus and the Norwegian Public Roads Administration.
3	
4	
5	
6	
7	**End of February**: finish work with theory, previous work and image collection and labelling. Deliver 1. draft of report.
8	
9	**March**: Setup of code for neural networks, modify existing networks.
10	
11	
12	**End of March**: finish general setup of code and modification of CNNs
13	
14	**April**: Training and testing of neural networks. Gather results and calculate metrics. Compare results.
15	
16	
17	**End of April**: finish training and testing of networks. Deliver 2. draft of report.
18	**May:** Write discussion of results, conclusion and further work
19	
20	
21	
22	
23	**Submission of master thesis**: 03.06.2019

B: Scientific paper

Classification of Corrosion and Coating Damages on Bridge Constructions from Images using Convolutional Neural Networks

Egil Holm[1]
[1]*M.Sc student, Department of Engineering Cybernetics, NTNU*

The global cost related to corrosion damages was estimated to be 3.4 percent of the global GDP in 2013. To properly handle corrosion damages at an asset we need to understand the state of the assets - such as bridges. Unmanned vehicles can gather large amount of imagery from bridges and such data could improve asset management. However, it can be very labour intensive to go through all the data manually. To this end, in this paper, we present a comparison of performance for different convolutional neural networks for automatic classification of corrosion and coating damages on bridge constructions from images taken during previous inspections. Through manual categorization and data augmentation, a total of 9300 images were collected and divided into five classes; (1) paint flaking, (2) white corrosion, (3) corrosion on steel construction, (4) corrosion on rebar, and (5) not corrosion. All classes except from (2) white corrosion were used for training and testing of convolutional neural networks. Four different convolutional neural networks called AlexNet, GoogLeNet, ResNet-50 and VGG-16 were trained using transfer learning in MATLAB. We have evaluated test performance through the metrics recall, precision, accuracy and F1 score. Test performance was also evaluated on damage detection accuracy, meaning how well the networks detect images that contain a damage. The convolutional neural network trained with four classes using VGG-16 had the overall best performance results, with average recall, precision, accuracy and F1 score being 95.45%, 95.61%, 97.74% and 95.53%, respectively. AlexNet, trained and tested with four classes, had the highest score on damage detection with 99.14% accuracy. The obtained results are promising, and make it possible to conclude that convolutional neural networks have a great potential in bridge inspections for automatic analysis of corrosion and coating damages.

I. INTRODUCTION

In the western part of the world, costs related to corrosion damages are estimated to be 3-4 percent of a country's Gross Domestic Product (GDP) [1] [2], and the global cost was estimated to be 3.4 percent of the global GDP in 2013 [3]. Corrosion damages are common challenges in many industries and on elements of infrastructure like bridges, tunnels and vehicles. In Norway there are more than 17500 bridges to be inspected and maintained by The Norwegian Public Roads Administration [4]. Corrosion damages, cracks and faults in surface treatment are examples of elements of interest during inspection of bridges. Every year there are high economic costs related to inspection of bridge constructions, and there are also safety challenges related to implementation of certain types of manual inspection methods that require use of access equipment. Manual inspection methods also have challenges in terms of subjectivity when evaluating corrosion damages, which are eliminated when using automatic analysis methods instead. Therefore, it is important to investigate the potential in automatic inspection methods.

There are three main contributions in this paper:

1. The collected data set is the basis for the work introduced in this paper, but is also highly relevant for future research and work on image classification and development of an automatic analysis method for damages on bridge constructions. All the collected images are considered a basis for further work within training of convolutional neural networks to be used for automatic inspection purposes. Through manual categorization and image pre-processing, a total of 9300 images from previous bridge inspections were collected to be used in training and testing of convolutional neural networks.

2. A maximum of four classes were used in the training process for the convolutional neural networks in this paper. In previous work [5] on analysis of damages on bridge construction using deep learning, two classes; *corrosion* and *not corrosion* were used. In bridge inspection, it is of importance to get the best possible overview of a construction to decide relevant measures of maintenance and repair. This paper introduces convolutional neural networks trained on four classes; *paint flaking*, *red corrosion on steel construction*, *red corrosion on rebar* and *not corrosion*. Thus, more information regarding the type of damage is obtained compared with the case of using only two classes.

3. Training, testing and comparison of image classification performance of four different convolutional neural networks; AlexNet, GoogLeNet, ResNet-50 and VGG-16. The obtained results in this paper are relevant for creating an automatic analysis system for bridge constructions. In combination with a drone and a localization system based on, for example, Visual Simultaneous Localization and Mapping (VSLAM), an analysis system based on convolutional neural networks would give a complete overview of the state of a bridge construction.

II. PREVIOUS WORK ON CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks (CNNs) belong to a class of neural networks that is popular for classification tasks such as classifying different categories of images. CNNs are inspired by the structure of a visual cortex, which consists of cells that are activated by what is called receptive fields [6]. A more technical explanation of CNNs can be done through introducing the main layers of such neural networks. The main layers of a CNN for image classification purposes are listed and briefly explained below [7]. Figure 1 illustrates the general concept for classifying images from bridge constructions using a CNN.

- **Input layer**: The first layer of a CNN where choices regarding elements such as image input size, the type of neural network and data augmentation are made.
- **Convolution**: Considering a 2D convolutional layer, this is a type of layer where sliding filters are applied an image to extract relevant information.
- **ReLU**: The ReLU is a very common activation function for CNNs. In general, an activation function works as a threshold operation to inputs, deciding what information is passed further into the network.
- **Pooling**: Down-sampling is performed in what is called a pooling layer to reduce the number of connections and parameters, to simplify the learning process.
- **Fully connected**: Features from the previous layers are combined in this layer in order to recognize a larger pattern for performing classification.
- **Output layers**: Output layers can be what is called softmax or classification layers. The softmax function is very common, a function where the sum of all output activations are equal to one. The function can be interpreted as a way of assigning a probability to each class for deciding what class an image most likely belongs to.

One of the most well known CNN architectures is AlexNet [8], and this is by many considered the first real state of the art CNN for image classification. AlexNet has five convolutional layers, overlapping pooling layers and three fully-connected layers. Since AlexNet won the 2010 and 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC)[9], the number of existing CNN architectures have increased significantly. In ILSVRC, CNNs are trained on approximately 1.2 million images divided into 1000 classes, and compete in different classification tasks. ZF Net [10] is a CNN that is quite similar to AlexNet, but with modifications in certain layers. In 2014, a CNN called GoogLeNet, based on the Inception architecture [11], won the ILSVRC. Compared with AlexNet, GoogLeNet has a total of 22 layers and 12 times fewer parameters. A team called VGG also competed in ILSVRC 2014, and introduced both a 16 and a 19 layer CNN architecture named VGG-16 and VGG-19, respectively [12]. ResNet [13] is another example of a CNN architecture that has competed in ILSVRC and obtained state of the art results within image classification.

Corrosion detection from images of bridge constructions using deep learning has been investigated previously in [5]. The authors used Python and a framework called Caffe for the implementation of a CNN based on AlexNet. A total of 3500 images were used in CNN training, and 100 images for testing, divided into two classes; *corrosion* and *not corrosion*.

In [14], corrosion detection performance from images for different CNNs are evaluated. As in [5], two classes; *corrosion* and *not corrosion* were utilized. VGG-16 and ZFNet were the two CNN architectures mainly used, but the authors also proposed their own network architectures.

Classification of different degrees of corrosion damages on grounding grids using CNNs was performed in [15]. A total of 10000 images equally divided into four classes; *very mild*, *mild*, *moderate* and *severe*, were used. The authors modified a CNN architecture called LeNet-5 [16] for use in the paper, and the utilized software was MATLAB and the Deep Learning Toolbox.

III. METHODS

A. Data collection and categorization

A hard drive containing approximately 240 000 images was received from the Norwegian Public Roads Administration (Statens Vegvesen), and the images were manually evaluated and categorized into the following five classes; (1) paint flaking, (2) white corrosion, (3) corrosion on steel construction, (4) corrosion on rebar, and (5) not corrosion. However, because few images of white corrosion were found in the provided data set, it was chosen not to continue with this class. Thus, the remaining four classes served as labels in training of CNNs. Table I explains the different classes in more detail, and table II shows the amount of images for each class that were manually evaluated and categorized. For the paint flaking class, data augmentation was used to generate more images from the 248 originally collected images. Image cropping and rotations were the utilized augmentation techniques.

B. Software, hardware and implementation

In this section, relevant software tools and methods for implementation of CNNs are introduced and explained

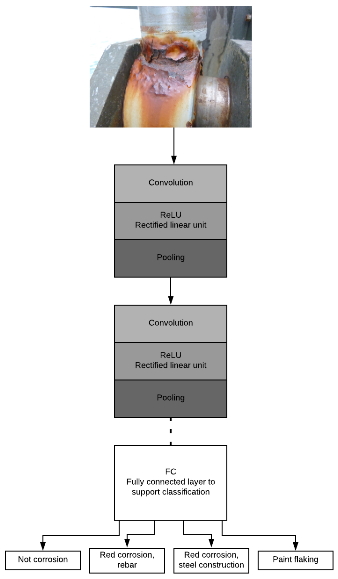


FIG. 1: Illustration of typical modules in a CNN, each consisting of three layers; convolution, activation function (here:ReLU) and pooling. CNNs also have layers called fully-connected (FC) layers, the final layer in this illustration. The input is an image of a corrosion damage from a bridge, and the CNN can classify the image as paint flaking, red corrosion on rebar, red corrosion on steel construction or not corrosion. The illustration is made using Lucidchart, with inspiration from [6].

in detail. Training and testing of CNNs was performed through transfer learning on existing neural networks in MATLAB using the Deep Learning Toolbox [17]. The available hardware for CNN training was Intel Core i7-8700 CPU 3.20 GHz, Intel Core i7-6850K CPU 3.60 GHz and NVIDIA Titan X (Pascal) Single GPU. All networks were trained using the following training algorithm and parameters in MATLAB:

- **Training algorithm**: Stochastic gradient descent with momentum (SGDM)
- **Learning rate**: 1×10^{-5}
- **Momentum**: 0.95
- **Epochs**: 10
- **Mini-batch size**: 20

TABLE I: Classes of corrosion and surface damages manually categorized from the 240 000 images received from the Norwegian Public Roads Administration. All classes except from class 2-white corrosion was used in network training and testing.

Number	**Class/label**	**Description**
1	Paint flaking	Flaking or peeling paint on steel. Some images contain small areas of beginning corrosion damages due to paint flaking.
2	White corrosion	Corrosion on zinc coating.
3	Red corrosion, steel construction	Corrosion on exposed steel on construction elements such as beams, wires and railings.
4	Red corrosion, rebar	Corrosion on exposed rebar in concrete of varying degrees.
5	Not corrosion	Everything that does not contain elements of the other classes above. Examples are; intact/undamaged concrete, cracks in concrete, undamaged steel, leaves, flowers, graffiti, trees, roads and cars.

TABLE II: Amount of images collected of each class and in total. The number in parentheses denotes the amount of images including augmented images.

Number	**Class/label**	**Amount of images**
1	Paint flaking	248 (2050)
2	White corrosion	31
3	Red corrosion, steel construction	2557
4	Red corrosion, rebar	2380
5	Not corrosion	2307
Total		7523 (9325)

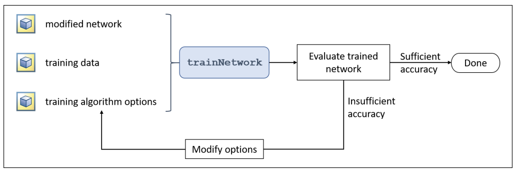


FIG. 2: Illustration of transfer learning workflow in MATLAB [18].

C. Setup of training and test of the neural networks

This section will explain how training and testing of CNNs are performed, including which existing CNNs are used, the training options and relevant hardware, how many classes that are to be classified and which metrics are relevant for evaluating the performance of the different CNNs.

Table III shows the setup of CNN training. Network 1 was trained mainly to become familiar with the setup of convolutional neural networks in MATLAB, and training was performed using approximately 400 images and two classes; *corrosion* and *not corrosion*. The results from training and testing of network 1 are not considered very relevant for conclusions to be made in this paper, but were evaluated in a process of deciding to divide the class *corrosion* into two classes; *red corrosion, rebar* and *red corrosion, steel construction*. Thus, networks 2-9 are the CNNs that will be evaluated and compared in this paper. Networks 2-5 are trained and tested on three classes, while four classes were used for networks 6-9.

TABLE III: Setup of network training through transfer learning using existing pre-trained networks. The class numbers with description are given in table I.

Name	**Pre-trained network**	**Classes**
Network 1	AlexNet	Corrosion Not corrosion
Network 2	AlexNet	3,4,5
Network 3	GoogLeNet	3,4,5
Network 4	ResNet-50	3,4,5
Network 5	VGG-16	3,4,5
Network 6	AlexNet	1,3,4,5
Network 7	GoogLeNet	1,3,4,5
Network 8	ResNet-50	1,3,4,5
Network 9	VGG-16	1,3,4,5

The metrics recall, precision, accuracy and F1 score were chosen as performance measurements for the CNNs. Equations (3.1), (3.2), (3.3) and (3.4) show how the metrics were calculated. All equations are taken from [19]. The notation is as follows; TP: true positive, FP: false positive, TN: true negative, FN: false negative and I: number of classes.

$$\text{Recall} = \frac{\sum_{i=1}^{I} \frac{TP_i}{TP_i + FN_i}}{I} \tag{3.1}$$

$$\text{Precision} = \frac{\sum_{i=1}^{I} \frac{TP_i}{TP_i + FP_i}}{I} \tag{3.2}$$

$$\text{Average accuracy} = \frac{\sum_{i=1}^{I} \frac{TP_i + TN_i}{TP_i + FN_i + FP_i + TN_i}}{I} \tag{3.3}$$

$$\text{F1} = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{3.4}$$

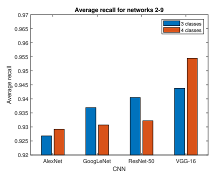
IV. RESULTS

This section will introduce the obtained results from testing of the four convolutional neural networks AlexNet, GoogLeNet, ResNet-50 and VGG-16.

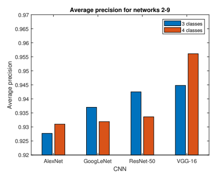
Table IV shows the results from testing of networks 2-9, and a comparison of performance metrics is illustrated in figure 3. A comparison of damage detection accuracy for networks 2-9 is shown in table V and in figure 4. Figure 3 show that network 9, trained on the VGG-16 architecture, has the highest average performance scores of all networks 2-9. Network 6, trained with four classes using AlexNet, has the highest damage detection accuracy, as illustrated in figure 4.

TABLE IV: Comparison of results from the different CNNs, denoted networks 2-9. The average metric values were calculated using equations (3.1), (3.2), (3.3) and (3.4).

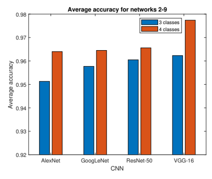
Network	**Average recall**	**Average precision**	**Average accuracy**	**Average F1 score**
Network 2	0.9268	0.9277	0.9513	0.9272
Network 3	0.9369	0.9370	0.9577	0.9369
Network 4	0.9405	0.9425	0.9605	0.9415
Network 5	0.9438	0.9448	0.9623	0.9443
Network 6	0.9292	0.9310	0.9640	0.9301
Network 7	0.9307	0.9319	0.9645	0.9313
Network 8	0.9322	0.9336	0.9656	0.9329
Network 9	0.9545	0.9561	0.9774	0.9553



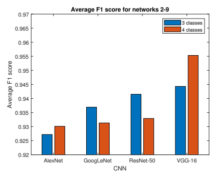
(a) Average recall for networks 2-9



(b) Average precision for networks 2-9



(c) Average accuracy for networks 2-9



(d) Average F1 score for networks 2-9

FIG. 3: Comparison of performance metrics for networks 2-9.

TABLE V: A comparison of how well networks 2-9 detect images that contain a damage (corrosion and paint flaking). The proportion of damages not detected are the number of FPs for *not corrosion* divided on the number of test images containing damages. Damage detection accuracy is the percentage of all test images containing a damage that actually were detected.

Network	**Proportion of damages not detected**	**Damage detection accuracy [%]**
Network 2	9/494	98.18
Network 3	11/494	97.77
Network 4	21/494	95.75
Network 5	5/494	98.99
Network 6	6/699	99.14
Network 7	17/699	97.57
Network 8	29/699	95.85
Network 9	10/699	98.57

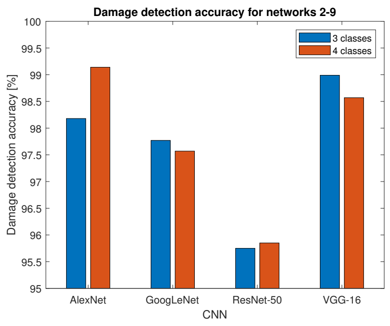


FIG. 4: A comparison of damage detection accuracy for networks 2-9. The values for each network is given in table V.

V. DISCUSSION

The obtained results show that there is a quite large variation in performance scores depending on both the CNN architecture and the number of classes, thus the number of images, used in training and testing. In this section, obtained results from testing of the CNNs will be compared and discussed.

From table IV and figure 3 it is observed that networks 5 and 9, both trained on the VGG-16 architecture, have the highest average performance scores of all networks 2-9. The second highest scores were obtained with network 4 and 8, trained on ResNet-50. Regarding the difference in performance depending on the number of classes, figure 3 illustrate that all networks trained on ResNet-50 and VGG-16 have higher average performance scores when trained with four classes (red bars) compared with three classes (blue bars). At the same time, networks trained on AlexNet and GoogLeNet have lower performance scores when the number of classes are increased from three to four. Table V shows how well the different networks detect a damage in the test set of images, and a comparison of this is illustrated in figure 4. Since only a single training per network was performed, it is difficult to say if the obtained results necessarily are representative for each of the CNN architectures. If each of the networks 2-9 had been trained and tested three or five times, an average total score and standard deviation could be calculated, and one would be able to say with higher confidence that the results were representative for AlexNet, GoogLeNet, ResNet-50 and VGG-16. However, training of CNNs is time consuming, and it was therefore considered more relevant to perform training and testing as shown in this paper to give a clear overview of the possibilities in corrosion and surface damage classification using CNNs. The presented results in table IV and figure 3 show that values for performance metrics are generally high for all networks 2-9, so the results are promising and show a great potential in the use of CNNs for classification of damages on bridge constructions during inspections.

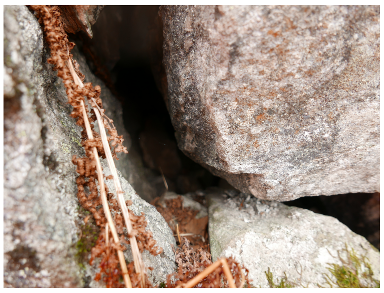


FIG. 5: Example of an image of *not corrosion* wrongly classified as *red corrosion, steel construction* by all networks 2-9.

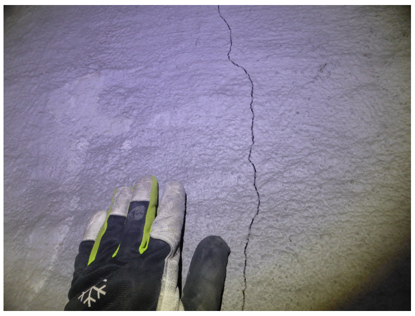


FIG. 6: Example of an image of *not corrosion* wrongly classified as *red corrosion, rebar* by networks 3, 4, 5, 8 and 9.

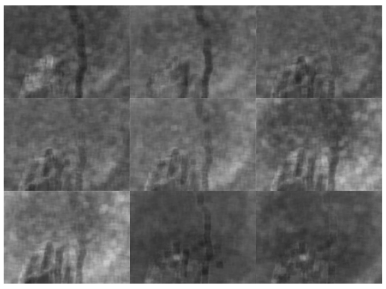


FIG. 7: Illustration of nine activations from fully-connected layer 8 in VGG-16 of the image in figure 6.

Even though the trained CNNs in this paper have overall high performance scores, there are a few of the wrong classifications that are worth discussing in order to evaluate how the performance can be improved. Figure 5 and 6 show examples of images that were wrongly classified by several, or all, of the networks 2-9.

The image of rocks and brown, or reddish, plants in figure 5 was classified as *red corrosion, steel construction* by all networks 2-9. This image, and others similar, were chosen during data collection because they were expected to become a challenge for the CNNs. The reason for this wrong classification is most likely connected to the fact that both the colour and shape of the plants are very similar to a corrosion damage, especially when the rocks in the background are so much brighter. One can argue that images like the one in figure 5 probably will not occur in an actual analysis system for bridge constructions, and thus not become a challenge. However, to increase the robustness in classification for the CNNs, collection of more images with larger variety in appearance to be used in CNN training is considered relevant in further work to avoid such wrong classifications.

Figure 6 illustrates an image of a glove and a long crack in concrete that was wrongly classified as *red corrosion, rebar* by several of the CNNs. To study this classification further, activations from fully-connected layer 8 in VGG-16 were extracted, and these are illustrated in figure 7. It is observed from these activations that the appearance of both the glove and the crack becomes very similar to red corrosion on rebar, with characteristic darker "lines" on a brighter background.

Regarding the choice of training algorithm and parameters, this was chosen with inspiration from previous work [14], and through a trial and error approach until parameters that gave sufficient training progress were found.

VI. CONCLUSION AND FURTHER WORK

This paper has presented a comparison of image classification performance for the convolutional neural networks AlexNet, GoogLeNet, ResNet-50 and VGG-16. The highest average performance scores were obtained with VGG-16, trained and tested with four classes. The values for the performance metrics recall, precision, accuracy and F1 score were 95.45%, 95.61%, 97.74% and 95.53%, respectively. A damage detection accuracy of 99.14% was achieved with AlexNet, trained on four different classes. From the obtained results in this paper it is concluded that convolutional neural networks have a great potential in classification of surface and corrosion damages on bridge constructions, and thus can be used to create an automatic analysis system to be used during bridge inspections.

The following elements are suggested for further work: (1) Collection of even more images containing damages to be used in training of CNNs. Evaluate the need for more classes such as *white corrosion* and *cracks in concrete.* (2) Utilize regions of interest (ROI) and/or segmentation techniques for labeling of specific areas of an image that contain damages instead of labeling the entire image as a damage. If such techniques are successful, collection of more images might not be necessary to obtain better results. (3) Investigate how other training parameters on CNNs affect the classification results, and if larger modifications on CNN architecture is preferable.

ACKNOWLEDGMENTS

Thanks to Sølvi Austnes from the Norwegian Public Roads Administration for providing images from previous bridge inspections. This paper is partially funded by the RINVE network (Co-funded by the BIA program by the Norwegian Research Council, 277234)

[1] SafeControl AS. Fakta om korrosjon og kostander. `https://safecontrolgruppen.no/frosio-ns-476/`. (Accessed: 08.01.2019).

[2] Overflateportalen. Kort om korrosjon. `https://www.overflateportalen.no/kort-om-korrosjon/`. (Accessed: 08.01.2019).

[3] NACE Intenational. International measures of prevention, application and economics of corrosion technology (impact). `http://impact.nace.org/economic-impact.aspx`. (Accessed: 08.01.2019).

[4] Statens Vegvesen. Bruer. `https://www.vegvesen.no/fag/teknologi/bruer`. (Accessed: 07.02.2019).

[5] L. Petricca, T. Moss, G. Figueroa, and S. Broen. Corrosion detection using a.i : A comparison of standard com-

puter vision techniques and deep learning model. *Computer Science and Information Technology*, 6, 2016.

[6] MathWorks. Learn about convolutional neural networks. `https://se.mathworks.com/help/deeplearning/ug/introduction-to-convolutional-neural-networks.html`. (Accessed: 18.02.19).

[7] MathWorks. Specify layers of convolutional neural network. `https://se.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html`. (Accessed: 18.02.19).

[8] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60, 2017.

[9] ImageNet. Official website. `http://www.image-net.org/`. (Accessed: 21.02.19).

[10] M.D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-, 2015.

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

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[14] D.J. Atha and M.R. Jahanshahi. Evaluation of deep learning approaches based on convolutional neural networks for corrosion detection. *SAGE journals*, 17(5), 2018.

[15] J. Du, L. Yan, H. Wang, and Q. Huang. Research on grounding grid corrosion classification method based on convolutional neural network. *MATEC Web of Conferences*, 160, 2018.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86, 1998.

[17] MATLAB. Deep learning toolbox. `https://se.mathworks.com/products/deep-learning.html`. (Accessed: 30.01.19).

[18] MathWorks. Deep learning onramp course- transfer learning workflow. `https://matlabacademy.mathworks.com/R2018b/portal.html?course=deeplearning#chapter=4&lesson=2&section=2`. (Accessed: 14.01.2019).

[19] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45, 2009.

C: MATLAB code

AlexNet

```matlab
%Loading Alexnet:
net=alexnet;
%analyzeNetwork(net)

%Loading images to a image datastore, label images from foldernames:
images=imageDatastore('ThreeLabels','IncludeSubfolders',true,'LabelSource','foldernames');

%The data is divided into three sets;training(80%), validation(10%) and
%testing (10%):
[imageTrain,imageValidation,imageTest]=splitEachLabel(images,0.8,0.1);

%Define the image input size for AlexNet:
inputSize = net.Layers(1).InputSize;

%Change image size of images to fit AlexNet:
AugTrain = augmentedImageDatastore(inputSize(1:2),imageTrain,'ColorPreprocessing','gray2rgb');

AugValidation = augmentedImageDatastore(inputSize(1:2),imageValidation,'ColorPreprocessing','gray2rgb');

AugTest = augmentedImageDatastore(inputSize(1:2),imageTest,'ColorPreprocessing','gray2rgb');


%Modifying network layers:
layersExtract = net.Layers(1:end-3);
classes = numel(categories(imageTrain.Labels));
newlayers = [
    layersExtract
    fullyConnectedLayer(classes,'WeightLearnRateFactor',20,'
       BiasLearnRateFactor',20)
    softmaxLayer
    classificationLayer];

%Specifying training options:
optionsTrain = trainingOptions('sgdm','MiniBatchSize',20,'
   MaxEpochs',10,'Momentum',0.95,'InitialLearnRate',1e-5, '
   Shuffle','every-epoch','ValidationData',AugValidation,'
   ValidationFrequency',3, 'Verbose',false,'Plots','training
   -progress','OutputFcn',@(info)savetrainingplot(info));

%Train the network:
%newNet = trainNetwork(AugTrain,newlayers,optionsTrain);

%NetworkSix=newNet;
%save NetworkSix

function stop=savetrainingplot(info)
stop=false;  %prevents this function from ending
   trainNetwork too early
if info.State=='done'   % to check if all iterations/epochs
   are completed

        figHandles = findall(0, 'Type', 'figure');
        saveas(figHandles,'TrainingProgressNetSix.eps','epsc
           ')  % save figure in desired file format
end
end


%Classify test images:
%load ('NetworkTwo.mat')
%[Pred,scores] = classify(NetworkTwo,AugTest);

%Calculate accuray after classifying test images, and
   creating a confusion matrix:
%Test = imageTest.Labels;
%accuracy = mean(Pred == Test)

```

```matlab
%confusionchart(Test,Pred)

%trained_net_vars = load('NetworkThree.mat');
%handles.neural_net = trained_net_vars.NetworkThree;
```

GoogLeNet, ResNet-50 and VGG-16

```matlab
net=googlenet; %net=vgg16, net=resnet50

%Loading images to a image datastore, and label images after
%foldernames:
images=imageDatastore('ThreeLabels','IncludeSubfolders',true
   ,'LabelSource','foldernames');

%The data is divided into three sets;training(80%),
   validation(10%) and
%testing (10%):
[imageTrain,imageValidation,imageTest]=splitEachLabel(images
   ,0.8,0.1);

%Define the image input size for GoogLeNet:
inputSize = net.Layers(1).InputSize;

%Change image size of images to fit GoogLeNet:
AugTrain = augmentedImageDatastore(inputSize(1:2),imageTrain
   ,'ColorPreprocessing','gray2rgb');

AugValidation = augmentedImageDatastore(inputSize(1:2),
   imageValidation,'ColorPreprocessing','gray2rgb');

AugTest = augmentedImageDatastore(inputSize(1:2),imageTest,'
   ColorPreprocessing','gray2rgb');


%Modifying network layers:
if isa(net,'SeriesNetwork')
  NewLayersReplace = layerGraph(net.Layers);
else
```

```
  NewLayersReplace = layerGraph(net);
end

[learnableLayer,classLayer] = findLayersToReplace(
   NewLayersReplace);

numClasses = numel(categories(imageTrain.Labels));

if isa(learnableLayer,'nnet.cnn.layer.FullyConnectedLayer')
    newLearnableLayer = fullyConnectedLayer(numClasses, ...
        'Name','new_fc', ...
        'WeightLearnRateFactor',20, ...
        'BiasLearnRateFactor',20);

elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer
   ')
    newLearnableLayer = convolution2dLayer(1,numClasses, ...
        'Name','new_conv', ...
        'WeightLearnRateFactor',20, ...
        'BiasLearnRateFactor',20);
end

NewLayersReplace = replaceLayer(NewLayersReplace,
   learnableLayer.Name,newLearnableLayer);

newClassLayer = classificationLayer('Name','new_classoutput'
   );
NewLayersReplace = replaceLayer(NewLayersReplace,classLayer.
   Name,newClassLayer);

%Specifying training options:
optionsTrain = trainingOptions('sgdm','MiniBatchSize',20,'
   Momentum',0.95, 'MaxEpochs',10,'InitialLearnRate',1e-5, '
   Shuffle','every-epoch','ValidationData',AugValidation,'
   ValidationFrequency',3, 'Verbose',false,'Plots','training
   -progress','OutputFcn',@(info)savetrainingplot(info));

%Train the network:
%newNet = trainNetwork(AugTrain,NewLayersReplace,
   optionsTrain);
```

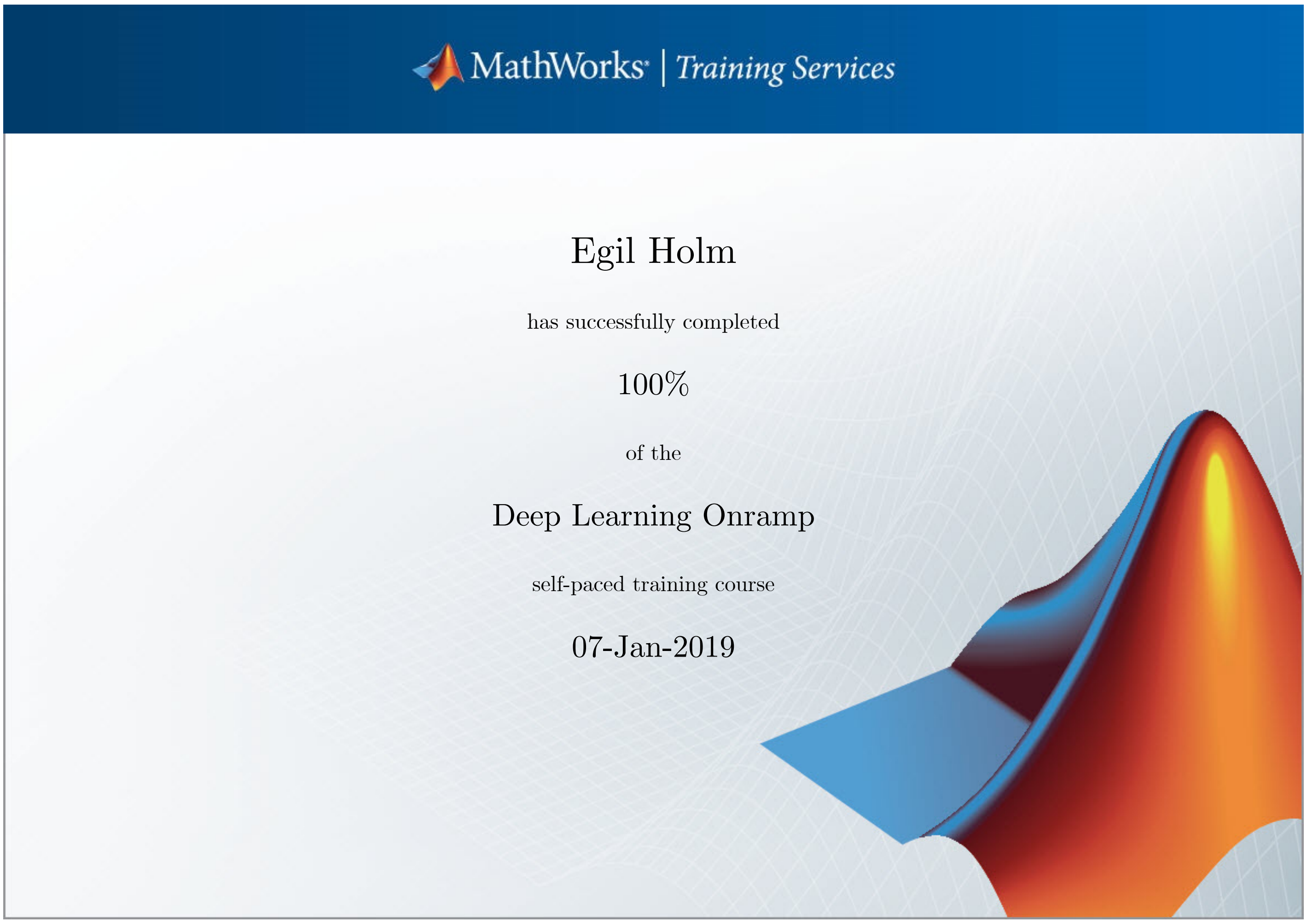
```

%NetworkSeven=newNet;
%save NetworkSeven

function stop=savetrainingplot(info)
stop=false;  %prevents this function from ending trainNetwork too early
if info.State=='done'   % to check if all iterations/epochs are completed

        figHandles = findall(0, 'Type', 'figure');
        saveas(figHandles,'TrainingProgressNetSeven.eps','epsc')  % save figure in desired file format
end
end


%load ('NetworkThree.mat')
%Classify test images:
%[Pred,scores] = classify(NetworkThree,AugTest);

%Calculate accuray after classifying test images, and creating a confusion matrix:
%Test = imageTest.Labels;
%accuracy = mean(Pred == Test)

%confusionchart(Test,Pred)

% trained_net_vars = load('NetworkThree.mat');
%handles.neural_net = trained_net_vars.NetworkThree;
%[Pred,scores] = classify(handles.neural_net,AugTest);
```

findLayersToReplace.m

```
% findLayersToReplace(lgraph) finds the single classification layer and the
% preceding learnable (fully connected or convolutional) layer of the layer
```

```matlab
% graph lgraph.
function [learnableLayer,classLayer] = findLayersToReplace(NewLayersReplace)

if ~isa(NewLayersReplace,'nnet.cnn.LayerGraph')
    error('Argument must be a LayerGraph object.')
end

% Get source, destination, and layer names.
src = string(NewLayersReplace.Connections.Source);
dst = string(NewLayersReplace.Connections.Destination);
layerNames = string({NewLayersReplace.Layers.Name}');

% Find the classification layer. The layer graph must have a single
% classification layer.
isClassificationLayer = arrayfun(@(l) ...
    (isa(l,'nnet.cnn.layer.ClassificationOutputLayer')|isa(l,'nnet.layer.ClassificationLayer')), ...
    NewLayersReplace.Layers);

if sum(isClassificationLayer) ~= 1
    error('Layer graph must have a single classification layer.')
end
classLayer = NewLayersReplace.Layers(isClassificationLayer);


% Traverse the layer graph in reverse starting from the classification
% layer. If the network branches, throw an error.
currentLayerIdx = find(isClassificationLayer);
while true

    if numel(currentLayerIdx) ~= 1
        error('Layer graph must have a single learnable layer preceding the classification layer.')
    end

    currentLayerType = class(NewLayersReplace.Layers(
       currentLayerIdx));
    isLearnableLayer = ismember(currentLayerType, ...
        ['nnet.cnn.layer.FullyConnectedLayer','nnet.cnn.
           layer.Convolution2DLayer']);

    if isLearnableLayer
        learnableLayer =  NewLayersReplace.Layers(
           currentLayerIdx);
        return
    end

    currentDstIdx = find(layerNames(currentLayerIdx) == dst)
       ;
    currentLayerIdx = find(src(currentDstIdx) == layerNames)
       ;

end

end
```

D: MATLAB course certificate



MathWorks | *Training Services*

Egil Holm

has successfully completed

100%

of the

Deep Learning Onramp

self-paced training course

07-Jan-2019

Egil Holm

Classification of Corrosion and Coating Damages on Bridge Constructions using Deep Learning