Master's thesis

2022

Amalie Berge Holm, Kasper Kallseter

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Amalie Berge Holm
Kasper Kallseter

Elimination of Reflections in Laser Scanning Using Transfer Learning from Simulation-Based Data to Real-World Data

June 2022

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Elimination of Reflections in Laser Scanning Using Transfer Learning from Simulation-Based Data to Real-World Data

Amalie Berge Holm
Kasper Kallseter

Engineering and ICT

Submission date:	June 2022
Supervisor:	Olav Egeland
Co-supervisor:	-

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

Preface

The work presented was done as part of a Master's thesis in Mechanical and Industrial Engineering at the Norwegian University of Science and Technology (NTNU). The thesis was developed in the spring of 2022. The thesis is about computer vision and machine learning, and while some knowledge in these subjects is helpful, the necessary theoretical basis is covered in the Preliminaries. The thesis aims to see how transfer learning can be used on machine learning models to help reduce reflections that arise when a laser is focused directly on a bright surface. It can be challenging to acquire a large enough dataset to properly train a neural network. Therefore the convolutional neural network is initially trained on a large simulated dataset, before the transition to a smaller real-world dataset is studied in a transfer-learning approach.

Acknowledgements

We want to express our gratitude and thank our supervisor, professor Olav Egeland at NTNU, for providing an exciting task, good support, and constructive feedback. We would also like to thank Ph.D. candidate Ola Alstad at NTNU for sharing his insights, good discussions and for providing us with materials and a simulated dataset.

Summary

Robotic welding automation necessitates precise and correct information about the welding environment. Laser scanning is one method of gathering this data, but the laser might cause reflections on shiny materials, resulting in inaccurate data. This thesis explored the use of transfer learning with U-net. The initial training was executed on an extensive simulated dataset before transfer learning on a significantly smaller real-world dataset. The real-world dataset contained reflections on both steel and aluminium. Overall the transfer learning was successful, even with only a handful of real-world images. The models demonstrated that the learning rate and optimization algorithms are interconnected, particularly because non-adaptive and adaptive optimizers employ the learning rate in different ways. There were smaller differences between the results on the two materials than expected. The thesis discuss the reasons behind this, the main being the extensive simulated dataset, the thickness of the laser beam and differences between the real-world and simulated images. The success of the transfer learning models opens the possibility for industry implantation, though research of potential users and their needs must be conducted in order to customize the models. However, the promising results are exciting for the potential of this type of transfer learning in the industry.

Sammendrag

Automatisering av robotsveising krever presis og korrekt informasjon. Laserskanning er en metode for å samle relevant data om sveiseområde, men laseren kan forårsake refleksjoner på skinnende materialer, som resulterer i unøyaktig data. Denne oppgaven utforsket bruken av overføringslæring med U-net. Den første opplæringen ble utført på et omfattende simulert datasett før læringen ble overført på et betydelig mindre datasett med reelle bilder. Det reelle datasettet inneholdt refleksjoner på både stål og aluminium. Totalt sett var overføringslæringen vellykket, selv med bare en håndfull reelle bilder. Modellene demonstrerte en sammenheng mellom læringsraten og optimaliseringsalgoritmene, spesielt siden ikke-adaptive og adaptive optimaliseringsalgoritmer bruker læringsraten på forskjellige måter. Det var mindre forskjeller mellom resultatene på de to materialene enn forventet. Oppgaven diskuterer årsakene bak dette, og legger hovedvekt på det omfattende simulerte datasettet, tykkelsen på laserstrålen og forskjeller mellom reelle bilder og simulerte bilder. Resultatene viser muligheten for implementasjon i industrien, selv om undersøkelser av potensielle brukere og deres behov må utføres for å tilpasse modellene. De lovende resultatene er imidlertid spennende for potensialet til denne typen overføringslæring i industrien.

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

The motivation for this master thesis, the problems addressed, the goals the thesis sought out to reach, as well as relevant literature, will be presented in this chapter.

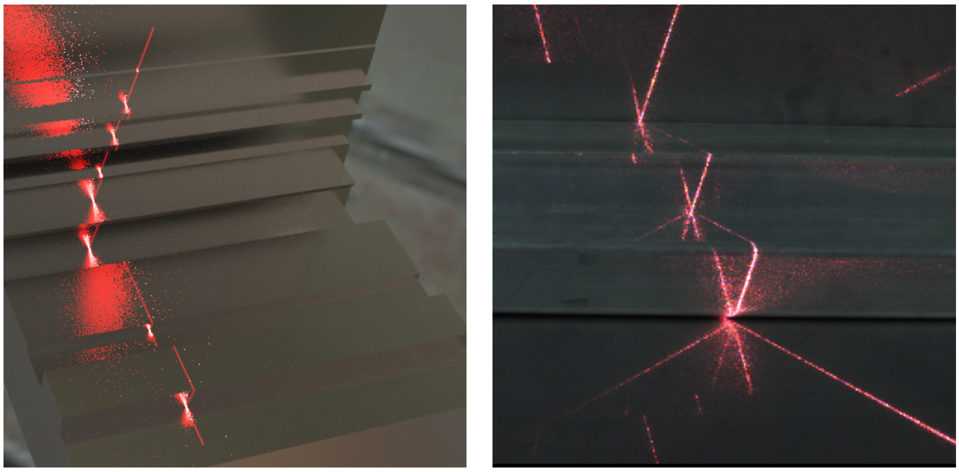
1.1. Motivation

In the engineering and manufacturing sector, as in most other sectors, convolutional neural networks are gaining acceptance and popularity as an efficiency and automation tool [15]. These networks can automate processes that would otherwise be manual and time-consuming, such as welding. Industrial robotic welding is one of the most widely used fields of robotics [53]. The ability to gather correct information concerning the welding process is required of a robot to generate results on par with a highly competent welder, according to Chen's paper on intelligentized welding manufacturing [10].

Data established through laser scanning can be used in robotic welding trajectory planning, seam tracking feedback management, quality control monitoring after a completed welding operation, and much more [26]. The ability to remove laser line reflections will improve the robot's ability to obtain this information, potentially expanding the range of activities that can be performed. The rapid advancements in machine learning and the rising use of artificial learning technology in engineering and manufacturing industries, like welding automation, allow for exciting opportunities. The specialization project by Kallseter and Holm [20] conducted in preparation for this thesis achieved promising results. The project presented machine learning models that minimized laser reflections on the reflective surfaces of simulated images, leading one step closer to a completely automated welding process. The next natural step would be to use transfer learning from the simulated-based to a real-world environment.

1.2. Problem Description

When a laser is aimed directly at a shining surface, it reflects, creating the illusion of several laser lines; examples are shown in Figure 1.1. Because the laser is designed to mark where a welding junction should be, the reflections generate confusion. Eliminating this ambiguity is time-consuming. The specialization project [20] managed to produce machine learning models that removed the reflections with adequate accuracy on simulated images. This master thesis will examine the task of transferring this machine learning knowledge from a simulated, as seen in Figure 1.1(a), to a real-world environment, as seen in Figure 1.1(b).



(a) Simulated Laser Reflection. (b) Real-World Laser Reflection.

Figure 1.1.: Examples of a laser line projected on reflective simulated and real-world surfaces.

1.3. Related Work

The use of various machine learning networks for image processing is a well-explored field, while the removal of unwanted reflections in images of laser scanning is less explored. This thesis use images generated by Ola Alstad during his thesis *Convolutional Neural Networks for Filtering Reflections in Laser Scanner Systems* [4] as well as real-world images generated during this thesis. One contribution listed by Alstad is a study executed by Sebastian Grans and Lars Tingelstad in 2021, in which Blender was used to simulate a laser scanner for use in neural network training [18]. They observed that the simulated images made by Blender were promising for transferring knowledge to the real-world domain.

An example of a study using machine learning methods to identify noise artifacts in images to be able to remove them was done in Washington in 2017 [3]. They trained a convolutional neural network to locate and classify source and reflection artifacts. The results showed that a network trained with only simulation data could distinguish experimental, real-world, information and display it in an artificial-free image. The approach highlights the potential for the elimination of reflection in images. In addition, they performed experiments to determine the feasibility of transfer learning and training with simulated data to identify and remove artifacts in real-world data. Though the percentage of misclassification increased, they concluded that networks trained with only simulated data could be transferred to experimental data and still maintain a high performance.

Another recent study from 2021 designed a U-Net that was trained on axial slices of cone-beam computed tomography (CBCT) [46]. CBCT is a solution providing accurate three-dimensional imaging of hard tissue structures with less exposure than regular CT, making it an increasingly accepted alternative to CT for dentists, among others [29]. CBCT is a fast and versatile solution, but has its drawbacks. A study published in The International Journal of Medical Physics Research and Practice sought to prevent these drawbacks by using a U-Net combined with transfer learning [46]. The network's weights were trained on synthetic CBCT scans generated from a public data set and the deepest layers of the network were trained again but then with real-world clinical data to fine-tune the weights. The study showed that U-Net was flexible enough to adapt to disturbances in the images. Their transfer learning successfully reduced prior knowledge in the network training, making the geometry possible to use on new data sets.

1.4. Goal

The main goal of this thesis was to successfully remove laser reflections on real-world images by transferring the knowledge obtained when initially training on simulated images. Additional goals were to develop the software further, include new accuracy metrics, connect to the Idun cluster to access more processing power and increase the size of the simulated dataset. To achieve the most successful transfer learning the results of models with different combinations of hyperparameters were compared. Furthermore, the thesis wanted to study the effects of the various reflections real-world surfaces would produce and the effects of the number of real-world images included in the transfer learning.

Chapter 2.

Preliminaries

The material presented in the preliminaries was also partly included in the preliminary study reported in the specialization project [20].

2.1. Geometry in Multidimensional Space

Computer vision is built on mathematical concepts. To give a better understanding of the mathematical foundation this section presents the geometry in multidimensional space. This first section is based on the relationship of homogeneous projective geometry in 2D and 3D as well as Plücker coordinates, from Potterman and Wallner [42] and Semple and Kneebone [48] as explained by Olav Egeland in his paper on robot vision [14]. The geometry will consist of points and lines in 2D and with the addition of planes in 3D.

2.1.1. Geometry in 2D

Points

A Euclidean vector $\boldsymbol{p}$ can be used to describe a point in the two-dimensional, 2D, Euclidean plane $\boldsymbol{R}^2$

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.1}$$

In this vector, x and y are the coordinates of the point. When operating with vision algorithms it is useful to represent the Euclidean geometry in the projective space $\boldsymbol{P}^2$. Then the point is described in terms of the homogeneous vector $\boldsymbol{x}$ given by

$$\boldsymbol{x} = \lambda \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.2}$$

In the equation above $\boldsymbol{x}$ represents the same point $\boldsymbol{p} \in \boldsymbol{R}^2$ for all $\lambda \neq 0$, where λ is a non-zero real number.

Lines

In the Euclidean plane a line $\boldsymbol{l}$ is given by a set of points (x, y) satisfying the equation

$$ax + by + c = 0 \tag{2.3}$$

The equation above (2.3) is a more general description than the following equation

$$y = Ax + B \tag{2.4}$$

The latter description has the flaw that lines parallel to the y-axis are not defined.

In homogeneous coordinates lines are described by the homogeneous vector

$$\boldsymbol{l} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{2.5}$$

2.1.2. Geometry in 3D

Points

Similarly to the 2D space, a point in the three-dimensional, 3D, Euclidean space $\boldsymbol{R}^3$ can be described by a coordinate vector $\boldsymbol{p}$ given by

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.6}$$

This point can also be represented by the homogeneous vector $\boldsymbol{x}$ given by

$$\boldsymbol{x} = \lambda \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.7}$$

The vector $\boldsymbol{x}$ will represent the same point $\boldsymbol{p} \in \boldsymbol{R}^3$ for all $\lambda \neq 0$.

Lines

Plücker coordinates can be used to describe lines in 3D in terms of a six-parameter representation. The geometric interpretation of Plücker coordinates consists of two vectors, the direction vector $\boldsymbol{a}$ and the moment $\boldsymbol{m}$. Considering two Euclidean points $(\boldsymbol{p}, p_4)$ and $(\boldsymbol{q}, q_4)$, the Plücker line is represented by

$$(\boldsymbol{l}, \boldsymbol{l}') = (p_4\boldsymbol{q} - q_4\boldsymbol{p}, \boldsymbol{p} \times \boldsymbol{q}) = (\boldsymbol{a}, \boldsymbol{m}) \tag{2.8}$$

Planes

A plane $\boldsymbol{\pi}$ is represented by the set of Euclidean points (x, y, z) and complete the equation

$$ax + by + cz + d = 0 \tag{2.9}$$

The plane can be described by the homogeneous vector

$$\boldsymbol{\pi} = \lambda \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \tag{2.10}$$

The geometric interpretation is that the plane has a normal vector $\boldsymbol{n} = [a, b, c]^{\mathrm{T}}$, and the distance from the origin to the plane in the direction of $\boldsymbol{n}$ is given by $-d/|\boldsymbol{n}|$. Together this implicates that a plane $\boldsymbol{\pi}$ can be constructed by a normal vector $\boldsymbol{n}$ and a point $\boldsymbol{p}$ on the plane $\boldsymbol{\pi}$

$$\boldsymbol{\pi} = \begin{bmatrix} \boldsymbol{n} \\ -\boldsymbol{n} \cdot \boldsymbol{p} \end{bmatrix} \tag{2.11}$$

A Point as the Intersection of a Line and a Plane

In Egeland's paper on robot vision [14] it is shown that a plane is found from a line and a point in the plane giving the intersection point between a dual line $(\boldsymbol{l}^*, \boldsymbol{l}'^*)$ and a plane $(\boldsymbol{u}, u_4)$ where $\boldsymbol{l} = \boldsymbol{l}'^*$ and $\boldsymbol{l}' = \boldsymbol{l}^*$

$$(\boldsymbol{x}, x_4) = (-u_4\boldsymbol{l} + \boldsymbol{u} \times \boldsymbol{l}', \boldsymbol{u} \cdot \boldsymbol{l}) \tag{2.12}$$

2.2. Computer Vision

This section will present some fundamentals in computer vision. It builds on the geometry already presented, and is based on the textbooks of Hart and Zisserman [19], and Ma, Soatto, Košeká and Sastry [30], and explained in further depth in Egeland's paper [14].

2.2.1. Camera Model

A camera model is the mapping from a point in the 3D Euclidean space scene to the 2D image plane. The pinhole model is the most widely used camera model in computer vision, it captures the geometric transformations that are a part of the image formations when using a camera.

In the pinhole camera model the light rays pass through a single point, called the optical center, before reaching the camera. Figure 2.1 illustrate how the mathematics behind the model is simplified by using a virtual image plane in front of the camera. Normalized image coordinates $\boldsymbol{s}$ are often introduced, they are vectors where the z value is equal to

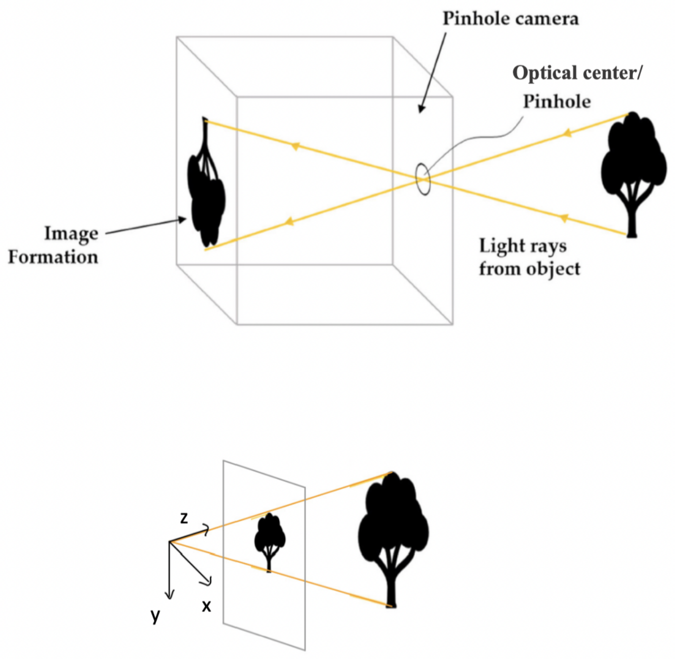


Figure 2.1.: (Above) Pinhole camera. (Below) pinhole camera model. *Illustration [20].*

one, making them on homogeneous form. To map the normalized image coordinates to pixel coordinates $\boldsymbol{p}$ the camera parameter matrix $\boldsymbol{K}$ is used

$$\boldsymbol{K} = \begin{bmatrix} \frac{f}{w} & 0 & u_0 \\ 0 & \frac{f}{h} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

By introducing the homogeneous form of the pixel $\tilde{\boldsymbol{p}}$, the mapping is written as

$$\tilde{\boldsymbol{p}} = \boldsymbol{K}\tilde{\boldsymbol{s}} \tag{2.14}$$

It is also possible to map the other way around, from pixel coordinates to normalized image coordinates, by using the inverse of the camera parameter matrix

$$\tilde{\boldsymbol{s}} = \boldsymbol{K}^{-1}\tilde{\boldsymbol{p}} \tag{2.15}$$

The inverse camera parameter matrix is given by $\boldsymbol{K}^{-1}$.

2.2.2. Epipolar Geometry

The image points of two cameras represented by the camera matrix, observing a scene from two different positions, have a geometrical relationship described by epipolar geometry. Figure 2.2 shows two cameras where the vector $\boldsymbol{r}_1$ runs from camera 1 through the normalized image plane to the point $\boldsymbol{p}$ in the scene. The intersection between $\boldsymbol{r}_1$ and the normalized image plane is known as $\boldsymbol{s}_1$, when observed from the view of camera 1. The length of $\boldsymbol{r}_1$ is unknown. This is the same for the corresponding relationships in camera 2. This arrangement is called a stereo arrangement.

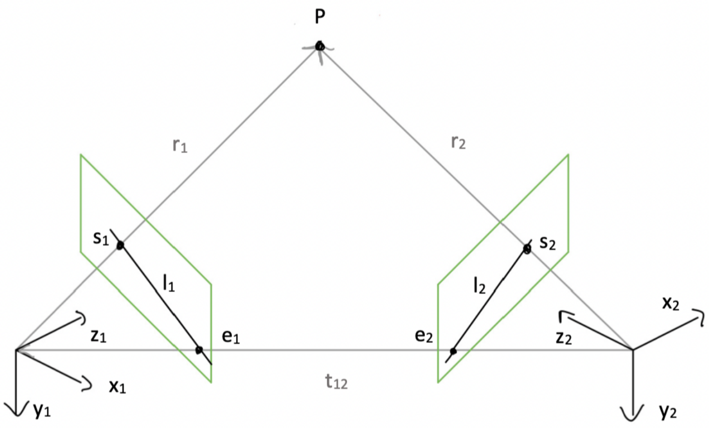


Figure 2.2.: Camera 1 and 2 in a stereo arrangement. $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ are the normalized image coordinates of point $\boldsymbol{p}$ in their respective images. $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are the epipoles, where $\boldsymbol{e}_1$ in image 1 is the image of the origin of frame 2, and the other way around. The figure also shows the epipolar lines $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$. *Illustration [20].*

The epipolar constraint results from the fact that the vectors $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ and the vector between the two cameras, $\boldsymbol{t}_{21}$, are all in the same plane. A consequence of this is that the triple scalar product of the vectors is equal to zero

$$\boldsymbol{r}_2 \cdot (\boldsymbol{t}_{21} \times \boldsymbol{r}_1) = 0 \tag{2.16}$$

The triple scalar product can be presented on coordinate form, here in the context of frame 2

$$\left(\boldsymbol{r}_2^2\right)^{\mathrm{T}} \left(\boldsymbol{t}_{21}^2\right)^{\times} \boldsymbol{R}_1^2 \boldsymbol{r}_1^1 = 0 \tag{2.17}$$

The essential matrix $\boldsymbol{E}$ is used to express the epipolar constraint between the vectors $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$

$$\boldsymbol{E} = \left(\boldsymbol{t}_{21}^2\right)^{\times} \boldsymbol{R}_1^2 \tag{2.18}$$

Then inserting the expression for the normalized image coordinates, $\boldsymbol{r}_1^1 = \lambda_1 \boldsymbol{s}_1$ and $\boldsymbol{r}_2^2 = \lambda_2 \boldsymbol{s}_2$, as well as the essential matrix gives the equation of the epipolar constraint

$$\lambda_2 \boldsymbol{s}_2^{\mathrm{T}} \boldsymbol{E} \lambda_1 \boldsymbol{s}_1 = 0 \tag{2.19}$$

Since the essential matrix is independent of scaling the constants, λ_1 and λ_2, can be made a part of it, simplifying the constraint to

$$\boldsymbol{s}_2^{\mathrm{T}} \boldsymbol{E} \boldsymbol{s}_1 = 0 \tag{2.20}$$

The epipolar lines are defined by using the essential matrix and the normalized image coordinates [14]

$$\boldsymbol{l}_1 = \boldsymbol{E}^{\mathrm{T}} \boldsymbol{s}_2 \tag{2.21}$$

$$\boldsymbol{l}_2 = \boldsymbol{E} \boldsymbol{s}_1 \tag{2.22}$$

The epipolar constraint can be used for pixel coordinates as well, by using the equation of $\boldsymbol{s}_1 = \boldsymbol{K}_1^{-1} \boldsymbol{p}_1$ and $\boldsymbol{s}_2 = \boldsymbol{K}_2^{-1} \boldsymbol{p}_2$. Then the constraint between the pixel coordinates is given by

$$\boldsymbol{p}_2^{\mathrm{T}} \boldsymbol{F} \boldsymbol{p}_1 = 0 \tag{2.23}$$

Where $\boldsymbol{F}$ is the fundamental matrix and can be derived from the following equation

$$\boldsymbol{F} = \boldsymbol{K}_2^{-T} \boldsymbol{E} \boldsymbol{K}_1^{-1} \tag{2.24}$$

The epipolar lines in pixel coordinates can be found in the same way as in normalized image coordinates

$$\boldsymbol{l}_1 = \boldsymbol{F}^{\mathrm{T}} \boldsymbol{p}_2 \tag{2.25}$$

$$\boldsymbol{l}_2 = \boldsymbol{F} \boldsymbol{p}_1 \tag{2.26}$$

2.2.3. Homography

A homography is defined in the 3D space as an invertible transformation from a point $\boldsymbol{x}$ to $\boldsymbol{x}'$

$$\boldsymbol{x}' = \boldsymbol{H} \boldsymbol{x} \tag{2.27}$$

The inverse transformation is given by

$$\boldsymbol{x} = \boldsymbol{H}^{-1} \boldsymbol{x}' \tag{2.28}$$

Generally, a homography consists of nine elements which are only equivalent under scaling, this means that there are eight independent elements. There is always a scaling factor, μ, that will give the homography on its normalized form with the bottom right element equal to one.

$$\mu \boldsymbol{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \tag{2.29}$$

Planar Homography

Planar homography builds on the stereo arrangement presented in the last section. Introducing a new point $\boldsymbol{x}$, which is given in each frame as $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. From [14] and [30] the geometric relationship between the frames of each camera is given as

$$c = \boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{t} \tag{2.30}$$

Considering that all points lie in the same plane, this plane has a normal vector, $\boldsymbol{n}$. The distance to the optical center from camera 1 is given as d. The following equation calculates the distance from the point in frame one, $\boldsymbol{x}_1$

$$d = \boldsymbol{n} \cdot \boldsymbol{x}_1 = \boldsymbol{n}^{\mathrm{T}}\boldsymbol{x}_1 \Longleftrightarrow \frac{1}{d}\boldsymbol{n}^{\mathrm{T}}\boldsymbol{x}_1 = 1 \tag{2.31}$$

Then inserting the equation describing the relationship between the frames (2.30) into the calculation of the distance

$$\boldsymbol{x}_2 = \boldsymbol{R}\boldsymbol{x}_1 + \boldsymbol{t}\frac{1}{d}\boldsymbol{n}^{\mathrm{T}}\boldsymbol{x}_1 = \left(\boldsymbol{R} + \boldsymbol{t}\frac{1}{d}\boldsymbol{n}^{\mathrm{T}}\right)\boldsymbol{x}_1 \tag{2.32}$$

This gives the following homography

$$\boldsymbol{H} = \boldsymbol{R} + \boldsymbol{t}\frac{1}{d}\boldsymbol{n}^{\mathrm{T}} \tag{2.33}$$

The normalized image coordinates of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ can be denoted as

$$\boldsymbol{x}_1 = \lambda_1\boldsymbol{x}_1 \tag{2.34}$$

$$\boldsymbol{x}_2 = \lambda_2\boldsymbol{x}_2 \tag{2.35}$$

This in term gives that

$$\boldsymbol{x}_2 = \boldsymbol{H}\boldsymbol{x}_1 \Longleftrightarrow \boldsymbol{x}_2 = \boldsymbol{H}'\boldsymbol{x}_1 \tag{2.36}$$

The fact that homographies are equivalent under scaling results in that $\boldsymbol{H}$ and $\boldsymbol{H}'$ are equivalent homography matrices. The scaling is expressed as

$$\boldsymbol{H}' = \frac{\lambda_2}{\lambda_1}\boldsymbol{H} \tag{2.37}$$

The mapping between the normalized image coordinates and the pixel coordinates can be written by using the camera matrices for the two cameras, resulting in

$$\boldsymbol{p}_1 = \boldsymbol{K}_1\boldsymbol{x}_1 \tag{2.38}$$

$$\boldsymbol{p}_2 = \boldsymbol{K}_2\boldsymbol{x}_2 \tag{2.39}$$

By substituting this into equation (2.36) the mapping can be written as

$$\boldsymbol{p}_2 = \boldsymbol{K}_2\boldsymbol{H}\boldsymbol{K}_1^{-1}\boldsymbol{p}_2 = \boldsymbol{H}'\boldsymbol{p}_1 \tag{2.40}$$

The end result is the following homographic mapping between pixel coordinates for two cameras viewing points in a plane, where the cameras have a known geometric relationship

$$\bar{\boldsymbol{H}} = \boldsymbol{K}_2\boldsymbol{H}\boldsymbol{K}_1^{-1} = \boldsymbol{K}_2\left(\boldsymbol{R} + \frac{1}{d}\boldsymbol{t}\boldsymbol{n}^{\mathrm{T}}\right)\boldsymbol{K}_1^{-1} \tag{2.41}$$

2.3. Laser Triangulation

The principle of triangulation concerns the projection of a light pattern. A laser beam is sent towards an object and captured by a camera. The distance from the object to the system can be calculated by trigonometry, given the prior distance between the scanning system and the camera [16].

2.3.1. Mapping from 2D to 3D

The goal of a laser scanning setup with a camera and a laser is to get an accurate 3D point cloud of the object. As shown in Figure 2.3 the laser scanning setup has a constant geometric relationship between the camera and laser while the object has a relative motion in comparison to the two other parts, as the laser line is swept along the object surface.

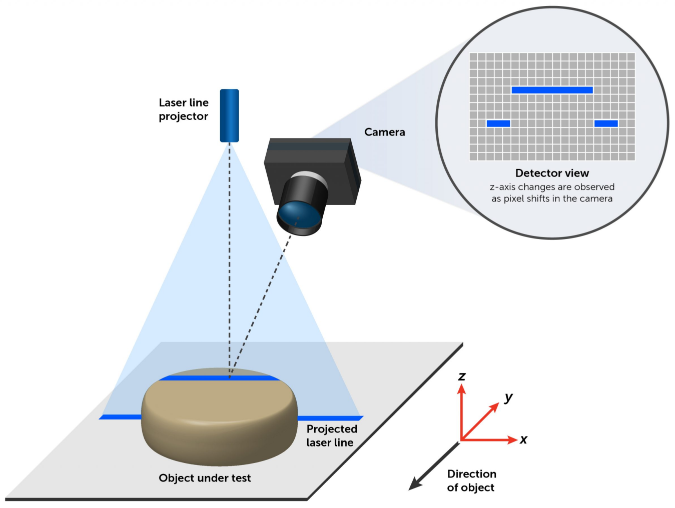


Figure 2.3.: A laser scanning setup, where the object scanned is moving along the y axis relative to the camera and the laser. The laser line is projected onto the object and the detector view that is formed is an image of a 2D pixel array, $\boldsymbol{p}$, for each scan. *Illustration [21]*

To map the surface scanned from the 2D image into 3D, points the geometrical principles described earlier in the preliminaries will be relevant. The plane made when projecting the laser will be denoted as $\tilde{\boldsymbol{u}}$. Then the normalized pixel coordinate of $\boldsymbol{s}$ is found by using equation (2.15). The line in the camera frame going through the optical center of the camera and the normalized image coordinate, $\boldsymbol{s}$, is given by the following equation

$$\boldsymbol{\ell} = (\boldsymbol{l}, \boldsymbol{l}') = (\boldsymbol{s}, \boldsymbol{0}) \tag{2.42}$$

The direction vector of the line is $\boldsymbol{s}$, while the moment is 0. This is because the distance from the line to the optical center is 0. From Egeland's paper [14] the equation from the calculation of the intersection between the line and the plane is the point $\boldsymbol{x}$

$$\boldsymbol{x} = -\frac{u_4}{\boldsymbol{u} \cdot \boldsymbol{s}} \tag{2.43}$$

2.3.2. Sub-Pixel Accuracy

To analyze the accuracy it is necessary to study the laser line presented in equation (2.42) up close. To be able to extract an accurate 2D coordinate for the measurement of the laser plane, the accuracy of the sub-pixel must be determined. A method to calculate the sub-pixel accuracy is to calculate the weighted center of mass for each row, i, of pixels in the images, like the images in Figure 2.4. Figure 2.5 illustrate a plot of the calculated weighted center of mass for a similar image [33]. The laser line center coordinate x_{ic} is calculated with the following equation

$$x_{ic} = \frac{\sum_{j=s}^{e} j \tilde{I}_{il}\left(j\right)^2}{\sum_{j=s}^{e} \tilde{I}_{il}\left(j\right)^2} \tag{2.44}$$

The normalized pixel intensity is given by $\tilde{I}_{il}$, where j is the row index, starting on s and ending on e for each row. The laser intensity profile $I\left(x\right)$ of each row in the image is based on each unit, and is normalized using the following equation

$$\tilde{I} = \frac{I\left(x\right) - \min\left(I\left(x\right)\right)}{\max\left(I\left(x\right)\right) - \min\left(I\left(x\right)\right)} \tag{2.45}$$

2.4. Convolutional Neural Networks

Convolutional neural networks, CNNs, are a subset of machine learning used to optimize successive filters when given a dataset. This section explains relevant concepts and is based on excerpts from the book by Goodfellow, Bengio and Courville [17], mainly from the chapter *Convolutional Networks*. Though CNNs can complete a wide range of tasks, this section will present them in the context of 2D visual imagery, with 2D images as input.

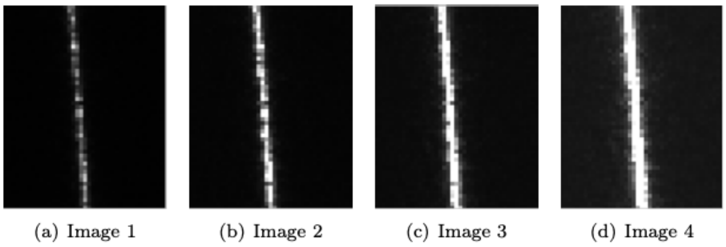


Figure 2.4.: A zoomed in view of the laser line, where the rows of pixels are visible. *Illustration [33]*

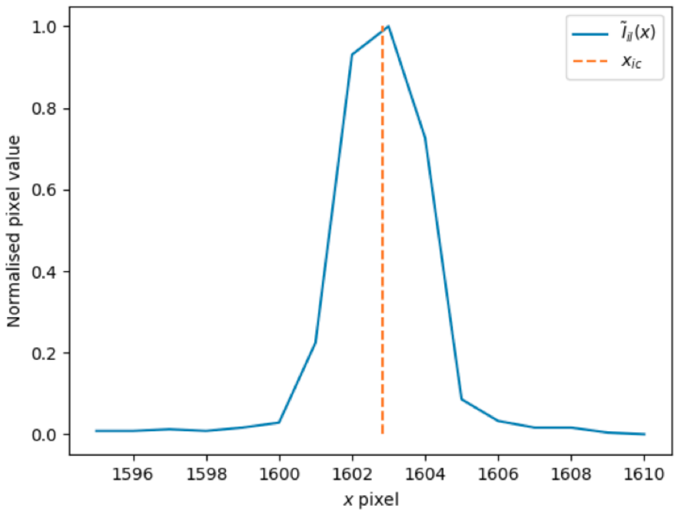


Figure 2.5.: Weighted center of mass, where the laser line center coordinate x_{ic} is estimated using normalized pixel intensity $\tilde{I}_{il}$. *Illustration [33]*

2.4.1. Convolution

A tensor is a multidimensional array and often the input in a CNN. For example, this tensor can be an image, given by height, width, and a number of channels. Looking at an example with a 2D image and a kernel with indices i and j the convolution operation would be

$$F\left(i,j\right)=\left(K*I\right)\left(i,j\right) \tag{2.46}$$

The output is given by F, referred to as a feature map. I is the input and K is the kernel. The term convolution is widely used for this equation, even though the correct mathematical term would be cross-correlation since mathematical convolution would have the inDices i and j the other way around.

Due to the practical applications of filtering an image, looking for features such as lines, a kernel can also be referred to as a filter. Each output, $f_{i,j}$ can be calculated by

$$F(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{2.47}$$

Illustrating the calculation of one output, $f_{1,1}$

$$f_{1,1} = t_{1,1}k_{1,1} + t_{1,2}k_{1,2} + t_{2,1}k_{2,1} + t_{2,2}k_{2,2} \tag{2.48}$$

This operations is illustrated in Figure 2.6.

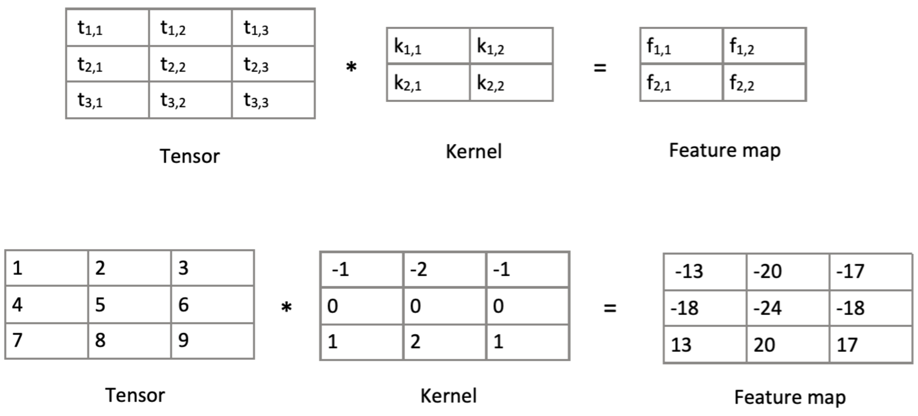


Figure 2.6.: (Above) Symbolic 2D convolution example using a 2×2 Kernel. (Below) 2D convolution example using a 3×3 kernel. *Illustration [20].*

Convolutions can also be used to calculate 3D inputs, this is illustrated in Figure 2.7. The depth dimension is given as d, and must be the same on both the tensor and the kernel. This will result in a collection of 3D feature maps, also illustrated in Figure 2.7. Each 3D kernel is independently convolved with the input tensor and the output of each is stacked in the output. In Figure 2.7 the four kernels are used to make four feature maps. The equation for the 3D convolution with multiple filters is

$$F(i,j,d) = \sum_l \sum_m \sum_n I_{l,j+m-1,d+n-1} K_{d,l,m,n} \tag{2.49}$$

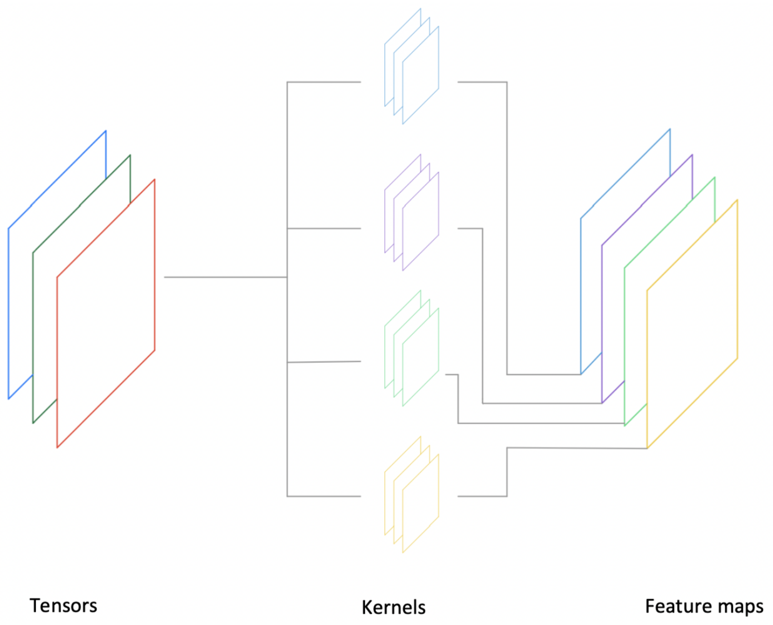
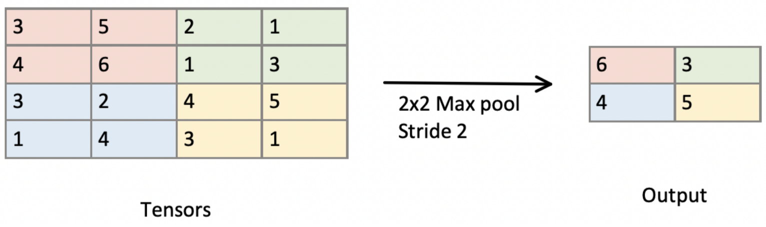


Figure 2.7.: A convolution with multiple filters, more precise with four kernels making four feature maps. *Illustration [20].*

2.4.2. Pooling and Downsampling

Pooling is the operation where the output becomes a summary statistic of the nearby inputs. There are often pooling layers in CNNs, the most common of them is max pooling. Max pooling outputs the maximum value within the neighborhood of the input. The pooling layers are an effective way to downsample feature maps since they summarize the presence of features in patches of the previous feature maps.

Max pooling is a good way to downsample since it keeps the highest activations, which are the activations that can be interpreted as the most important. Figure 2.8 shows a tensor where there is a two-by-two max pooling kernel with stride two in progress. The stride is the horizontal or vertical steps the kernel is moving between each calculation of a value from the input tensor.



3	5	2	1
4	6	1	3
3	2	4	5
1	4	3	1

6	3
4	5

Figure 2.8.: 2×2 max pooling example with a kernel with stride two. *Illustration [20].*

It is possible to use a filter with a stride bigger than one, as an alternative to pooling when downsampling. When a filter with a stride bigger than one is used, the weights of the filter are turned optimally. Giving the interpretation that the filter learns the optimal way to downsample an image. Where the pooling operations have no learnable parameters, a filter's weights will be tuned during optimization. In consequence, a pooling layer is more computationally efficient, without a significant drop in performance.

2.4.3. Padding

Up to this point, the convolutions included as examples have been what is called *valid*. In other words, they have not used any padding in the input. The convolutions using padding are called *same* convolutions. These are padded so that the spatial dimensions of the input and output are the same. When the input is padded, the spatial resolution is artificially increased by adding numeric values to the boundaries of the tensor. A common way of padding is *zero padding* where there are added 0s around the tensor, which is illustrated in Figure 2.9.

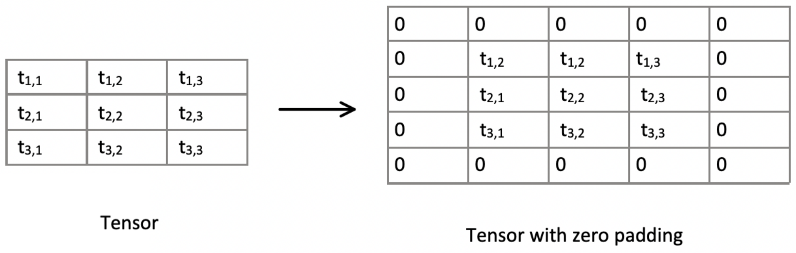


Figure 2.9.: The same tensor with and without zero padding. *Illustration [20].*

The dimensions of a subsequent layer $l+1$ of a tensor is given as a function of the spacial dimensions $n_l \times n_l$ of the previous layer l as

$$n_{l+1} = \frac{n_l + 2p - k}{s} + 1 \tag{2.50}$$

The padding is represented by p, the kernel size by k and the stride by s. For a same convolution the spatial dimensions will be the same, $n_{l+1} = n_l$, and the stride will be $s = 1$. Solving (2.50) for the padding, p, results in a padding equal to

$$p = \frac{k-1}{2} \tag{2.51}$$

2.4.4. Non-Linear Activation

A network of convolutions consists of several convolution operations being applied step by step to the input. These steps apply only linear operations to the input, making the output linearly dependent on the input, and the whole network could, in theory, be reduced to a single convolution. For the network to be able to learn non-linear relationships between input and output, a non-linear activation function f is applied to the output in the following manner

$$\bar{F}\left(i,j\right) = f\left(F\left(i,j\right)\right) = f\left(K * I\right)\left(i,j\right) \tag{2.52}$$

The non-linear function f is called an activation function and it is often computationally efficient to calculate the derivative from it. Two common activation functions are the *rectified linear unit, ReLU*, and the *Sigmoid*, which are both illustrated in Figure 2.10.

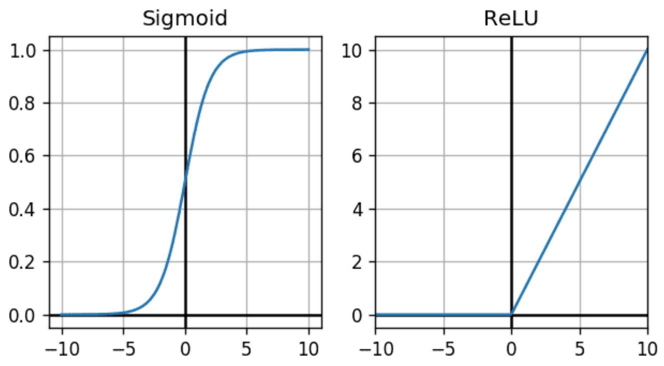


Figure 2.10.: The Sigmoid and ReLU activation functions.

2.4.5. Architecture of CNN

The architecture of CNNs typically follow the same pattern shown in Figure 2.11. A pattern of layers consisting of a convolution followed by a non-linear activation function and then an optional pooling layer.

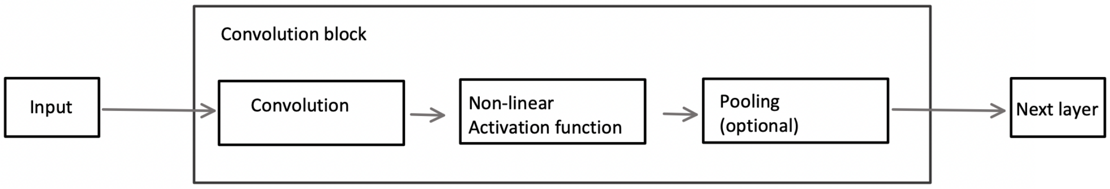


Figure 2.11.: Typical architecture of layers in a CNN network. *Illustration [20].*

2.4.6. Receptive Field

Convolutions are locally connected in a network, which means that each part of the output is a function of a certain input region. The certain input region that each part of the output relies on is called the receptive field. Considering a three-layer network with a kernel of size 3×3, as seen in Figure 2.12. Each pixel in the last feature map is a function

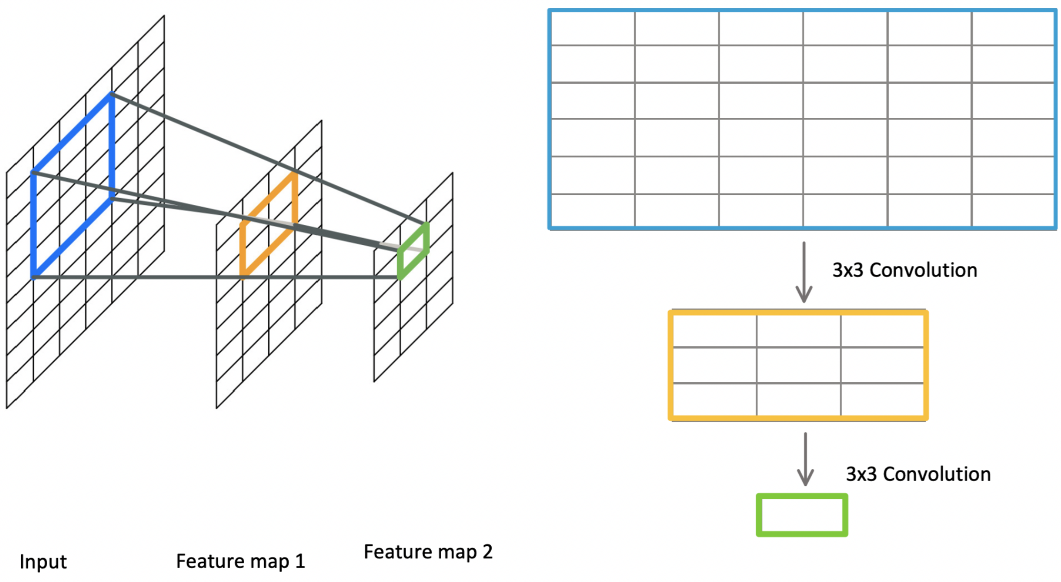


Figure 2.12.: (Left) Receptive field for tree layers with a 3 × 3 kernel. (Right) Convolutions increase receptive field. *Illustration [20].*

of a larger region of the input tensor. The receptive field in the previous network can be written as

$$r_{l-1} = s_l r_l + (k_l - s_l) \tag{2.53}$$

In the equation above k_l is the kernel size and s_l is the stride of layer l. The equation can be solved recursively for a whole single-path network [6], this would result in the following equation

$$r_0 = \sum_{l=1}^{L} \left((k_l - 1) \prod_{i=1}^{l-1} s_i \right) + 1 \tag{2.54}$$

A single-path network with the same kernel size and stride for all layers has tree parameters that can vary to increase the receptive field, and those are the kernel size, the stride and the number of layers. The most effective way to increase the receptive field is to change the stride since it is a multiplicative term in the equation, while the kernel size is an additive term.

2.4.7. Computing the Output Shape of Convolution Layers

The input image of a convolutional layer has the shape $\boldsymbol{H}_1 \times \boldsymbol{W}_1 \times \boldsymbol{C}_1$, where $\boldsymbol{H}$ corresponds to the height, $\boldsymbol{W}$ to the width and $\boldsymbol{C}$ to the channel. The output will be $\boldsymbol{H}_2 \times \boldsymbol{W}_2 \times \boldsymbol{C}_2$, where $\boldsymbol{H}_2 \times \boldsymbol{W}_2$ depends on the size of the receptive field $\boldsymbol{F}$ of the convolution filter and the stride $\boldsymbol{S}$ at which they are applied, in addition to the amount of zero padding $\boldsymbol{P}$ applied to the input. The following equations can calculate the height and width of the output

$$H_2 = \frac{H_1 - F_H + 2P_H}{S_H} + 1 \tag{2.55}$$

$$W_2 = \frac{W_1 - F_W + 2P_W}{S_W} + 1 \tag{2.56}$$

2.4.8. Optimization

When a network is set up, as explained in the previous sections, the filters only contain random weights and are not able to solve meaningful tasks. The results the network achieves depend on the dataset it is given. When working in the context of 2D visual imagery, the dataset has a large number of images, usually in the range from 500 to 100000. Each image in the dataset must contain an associated ground truth, but it can vary how this ground truth is defined in different tasks.

Machine Learning methods are usually implemented in a framework, the two most popular being *TensorFlow* [2] and *PyTorch* [40]. The main feature of these frameworks is the automatic calculation of gradients for a model, which could contain millions of tune-able parameters. To do this efficiently, the model uses the relatively simple method of the chain rule for derivatives. When considering a small computation graph, as Figure 2.13, containing a series of functions, f, g and h, applied to the input a to produce the output d. This leads to $b = f(a)$, $c = g(b)$ and $d = h(c)$. In this context, the intrigue is to calculate the gradient of each parameter with respect to the output d. If the interest is to find the partial derivative of d for each of the variables, for instance a, the partial derivative would be the following string of derivatives

$$\frac{\partial d}{\partial a} = \frac{\partial d}{\partial c}\frac{\partial c}{\partial b}\frac{\partial b}{\partial a} = f'(c)\, g'(b)\, h'(a) \tag{2.57}$$

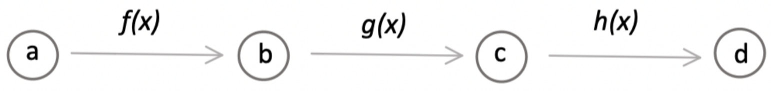


Figure 2.13.: A small computational graph. *Illustration [20].*

2.4.9. Loss Function

The parameters, weights and biases, of a network are usually given by $\boldsymbol{\theta}$. The optimization problem in deep learning is tuning the parameters $\boldsymbol{\theta}$ of a network, which corresponds to reducing the loss function $L(\boldsymbol{\theta})$. The loss function summarizes the error between the prediction and the ground truth of each guess the neural network performs. One of the most straightforward loss functions is to average the least square error between the predictions and the ground truths. When the network is solving problems where it has to predict a distinct class, the loss function is often based on probabilities, being the cross-entropy loss. To calculate the cross-entropy loss, the network's outputs have to be converted into probabilities through a function called the softmax function. Letting the output of the network predicting a class, k, be given by z_k, then the output of the softmax function $\hat{y}$ is interpreted as the probability of belonging to the class k. If there are K classes, then the softmax function for a prediction of class k would look like

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{k'}^{K} e^{z_{k'}}} \tag{2.58}$$

When the output of the softmax function $\hat{y}_k$ is calculated and the ground truth is denoted as y_k, the cross-entropy loss function C_n can be calculated by

$$C_n = -\sum_{k=1}^{K} y_k \log(\hat{y}_k) \tag{2.59}$$

The cross-entropy loss function can be weighted for specific classes, this can be useful if the training set is unbalanced. A training set is unbalanced if it has a large deviation in the number of examples for the different classes. When the cross-entropy loss function is weighted, a specific weight for each class is introduced w_k and it is multiplied with the loss for the specific class

$$C_n = -\sum_{k=1}^{K} w_k y_k \log(\hat{y}_k) \tag{2.60}$$

The total cross-entropy loss C for multiple examples n is the average of the individual losses C_n

$$C = \frac{1}{N}\sum_{n=1}^{N} C_n \tag{2.61}$$

2.4.10. Optimization Algorithms

The loss function must be minimized to minimize overall errors since it summarizes a defined error. Gradient descent and related methods are the optimization methods that have proved to be the most efficient at this task for neural networks. These methods iteratively update the parameters to minimize the error. When gradient descent is used for updating weights, the partial derivative $\frac{\partial}{\partial \theta_1}$ concerning each weight, for instance θ_1, and that weight's loss function, $J(\theta_1)$, has to be determined. These calculations are efficient because of the chain rule and partial derivative calculations, as explained above. The intent is to adjust the weights to minimize the loss function. These adjustments are made by modifying the next weight θ_1 based on the previous θ_1 with the following rule for gradient descent

$$\theta := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1) \tag{2.62}$$

The learning rate is represented by α. Many different optimization algorithms build on the gradient descent method.

2.4.11. Semantic Segmentation

Semantic segmentation is the process where each pixel in an input image is predicted to belong to a class [25]. Figure 2.14 shows an example of a semantic image segmentation. This example shows a classification problem where the network has to identify the different components in a street view, separating the cars from the road, sidewalk, buildings, and so on. In this example, there are eight classes, as the unlabeled is also a class.

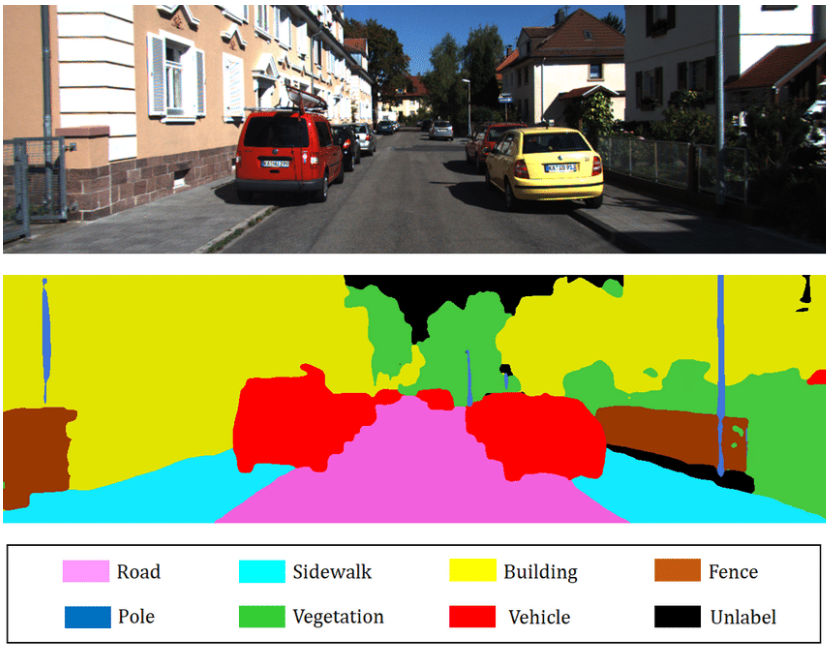


Figure 2.14.: An example of semantic segmentation of a street view picture, where the input image is above and the segmented output is presented below. *Illustration [24].*

2.4.12. Dice Coefficient

The Sørensen-Dice coefficient [63], also known as the Dice score, is a common metric for determining the overlap between the ground truth and the predicted segmentation. The Sørensen-Dice coefficient, DSC, calculates the spatial overlap for two sets A and B

$$DSC = \frac{2 \times |A \bigcap B|}{|A| + |B|} \tag{2.63}$$

The equation times the area of overlap by two and divides it by the total number of pixels in both images, it is visually presented in Figure 2.15.

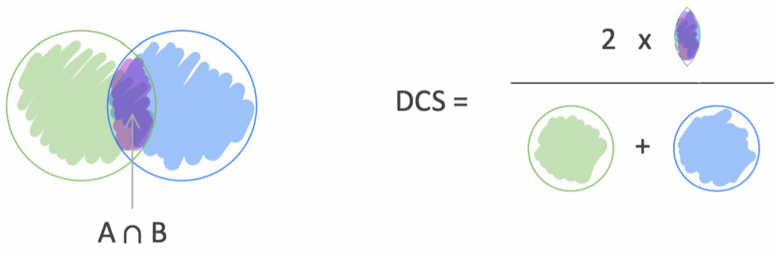


Figure 2.15.: (Left) Showing the spatial overlap that is $A \bigcap B$. (Right) Visual representation of the Dice score equation (2.63). *Illustration [20].*

2.4.13. U-Net

U-net is a fully convolutional network. It was initially developed for biomedical image segmentation [45], but is now widely used in many types of segmentation tasks. The general idea behind the U-net is to have a wide receptive field for each spatial location in the output, while simultaneously maintaining high-resolution information from the input image.

To explain the motivation for using U-net, take the receptive field from Figure 2.12, then consider stacking the convolution blocks after each other, without the pooling. Compared to the input, the width and height of the receptive field of the output increase by two for each convolution block. From the equation calculating the receptive field (2.54), it is shown that if the input image has a size of, for example 1024 there must be a large number of convolution blocks to produce a considerable receptive field. If the receptive field is enlarged by pooling layers or convolutions with stride $s > 1$, the equation (2.50) resolution of the output decrease, which could lead to a loss of finer-grained spatial information. U-net is able to solve this problem by having two different paths, one that decreases the resolution for a large receptive field and another that increases the resolution and concentrates finer-grained spacial information.

U-Net Architecture

Figure 2.16 shows the original U-net architecture. The network is formed as a U, thereof its name. The left side of the U is the contracting path and the right side is the expansive path. To be able to pass on the finer-grained spatial information, skip-connections are made from the contracting to the expansive path, the gray horizontal arrows. The contracting path consists of the following block being repeated:

- 3×3 convolution
- ReLU
- 3×3 convolution
- ReLU
- 3×3 maxpool

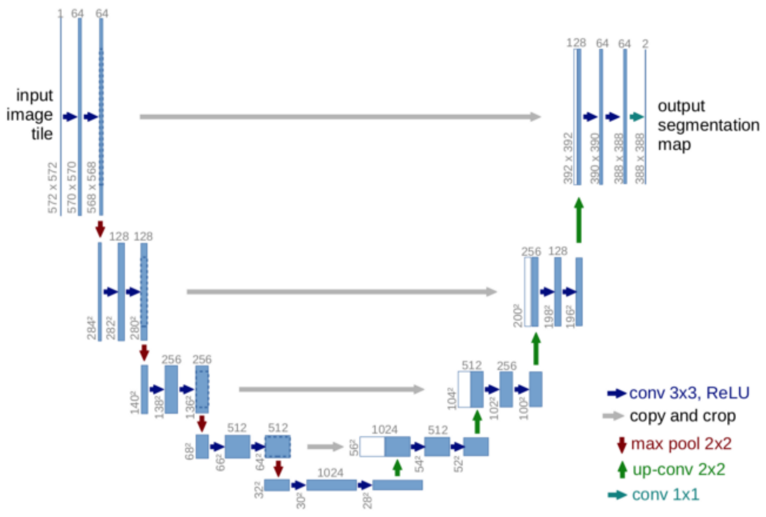


Figure 2.16.: Original U-net architecture. *Illustration [45].*

This is a typical CNN architecture. The expansive path is constructed similarly, but instead of downsampling with max pool, it runs a convolutional up-sampling, halving the number of feature channels. Each block in the expansive path starts with concentrating the correspondingly cropped feature maps into the up-sampled feature maps of the expansive path. Several variants of the U-net have been made, due to the impressive performance in segmentation tasks. The several variants can differ in their depth of the contracting path, in addition to more advanced inner workings. The similarity between the different variants is the idea behind a contracting and expansive path, with information flow between them.

2.5. Physically Based Rendering

Rendering is the process of generating an image given the description of a 3D scene. It is used extensively in computer games, movies and simulation [31]. The different computational complexity versus realism demands has resulted in different rendering techniques. Physically based rendering is the rendering technique that focuses most on realism, in other words an attempt to simulate reality.

2.5.1. Models for Reflection

The theory of rendering an image is to choose the color and intensity of each pixel in the image. The scene's object, material, and light sources decide intensity and color. When creating a realistic image from a scene, the most crucial factor is the accurate calculation of light and how it interacts with the materials and surfaces in the scene. There are three basic reflection models: specular, diffuse and spread, all illustrated in Figure 2.17.

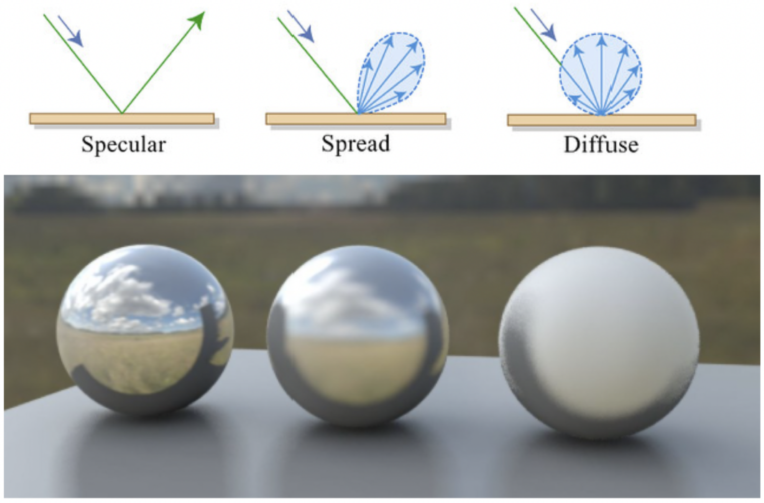


Figure 2.17.: The three basic reflections. *Illustration [20].*

Specular

To model surfaces as smooth metal mirrors, specular reflection is used. There are two basic principles in specular reflection: the law of reflection and the Fresnel equation. The law of reflection states that the angle of incident is the same as the angle of reflection, so that the incident direction, surface normal and direction of reflection are co-planar. The Fresnel equation describes the fraction of light reflected, and through this the fraction that is absorbed [49].

Spread

Spread, glossy or imperfect specular, reflections are the reflections that in large scale appear blurry because of the irregularities on a surface, even though the surface can have a perfect specular reflection for a single light ray.

Diffuse

A diffuse, or Lambertain, surface is a surface that reflects light equally in all directions regardless of the incident angle. In theory, a perfectly diffuse surface would mathematically conserve energy, but this surface does not exist in reality. In reality, diffuse surfaces reflect light unequally, but can reflect in all directions and represent the majority of surfaces.

Combined Reflection Models

In reality, most surfaces are a mixture of specular, diffuse and spread reflections. In this context, the most relevant combination is the combination of a strong spread or specular reflection combined with a weak diffuse reflection. The combinations of diffuse/specular and diffuse/spread are illustrated in Figure 2.18. Both combined reflections have the diffuse lobe that comes from the diffuse reflection.

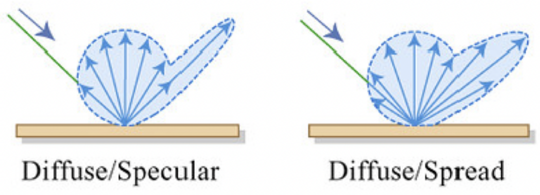


Figure 2.18.: Combined reflections. *Illustration [20].*

2.5.2. Ray Tracing

Ray tracing is the technique of tracing the path of the light in photorealistic rendering. The ray tracing algorithm follows a path of a ray of light through the scene as it interacts with the objects in it. The algorithm's goal is to make a realistic 2D image, and the only tool to make this happen is the realistic simulation of the light in the scene. This makes room for some simplifications, for example that the light that is certain not to hit the camera can be discarded.

Forward Ray Tracing

Forward ray tracing calculates how the light from a source moves around the scene and possibly hits the camera. The method simulates how light behaves in nature, however it is very computational inefficient. The light rays of interest are the ones that hit the camera, while most rays calculated with forwarding ray tracing do not.

Backward Ray Tracing

Backward ray tracing is the reverse of forward ray tracing. Calculating the paths from the camera, then how it interacts with objects in the form of reflections, before eventually hitting a light source. This method is more computationally efficient than forward ray tracing, since it only calculates the rays that actually hit the camera. Since all optical systems are reversible, the backward ray tracing method can theoretically reach the same result as forward ray tracing.

Hybrid Ray Tracing

Several render engines only use backward ray tracing because of its computational efficiency. However, it comes short in terms of caustics. Caustics are the light that goes through a specular reflection, then a diffuse reflection before hitting the camera. The light reflected for a diffuse surface can have any incident angle, since the position of this light is known, it is easy to trace it back to the source. On the other hand, if the light comes from a concentrated specular reflection, this traceback is much harder, as the location of the reflection is unknown. Hybrid ray tracing is the solution to the caustics problem. It uses forward ray tracing to track one of the specular reflections from the source before they hit the diffuse surface, making the reflections easily accounted for since they now are known in advance. Then the hybrid ray tracing combines this method with the computational efficiency of backward ray tracing.

2.6. Transfer Learning

Torrey and Shavlik define transfer learning as the improvement of learning a new task through the transfer of knowledge from a related task that has already been learned, in their work on the subject [57]. An intuitive example could be to transfer the knowledge of riding a bicycle to driving a motorcycle. The creation of algorithms that facilitate transfer learning is a topic of high interest in the machine learning field, as most algorithms are built to address a particular task.

It is possible to categorize transfer learning problems as transductive, inductive or unsupervised transfer problems, when viewing them from a label-setting view [62]. Situations where the label information only appears in the source domain are categorized as transductive, while the situations where the label information of the target domain is available are categorized as inductive. Situations where there label information is unknown for both source and target domain are categorized as unsupervised.

2.6.1. Applications of Transfer Learning

The ability to transfer knowledge successfully would make the task of performing classification in a domain possible, even though the training data is not sufficient by utilizing training data from another domain of interest [39]. Figure 2.19 illustrates a possible example of this, where the bottom network utilizes transfer learning on a network trained on cars. If the knowledge transfer is successful, it could improve learning performance extensively by surpassing a large amount of expensive data labeling.

Transfer learning could be used when convolutional neural networks transition from training on simulated data to real-world data [39]. One advantage of transfer learning is that the dataset containing the real-world data can be relatively small compared to the simulated dataset since many of the features are learned through the training on the simulated data.

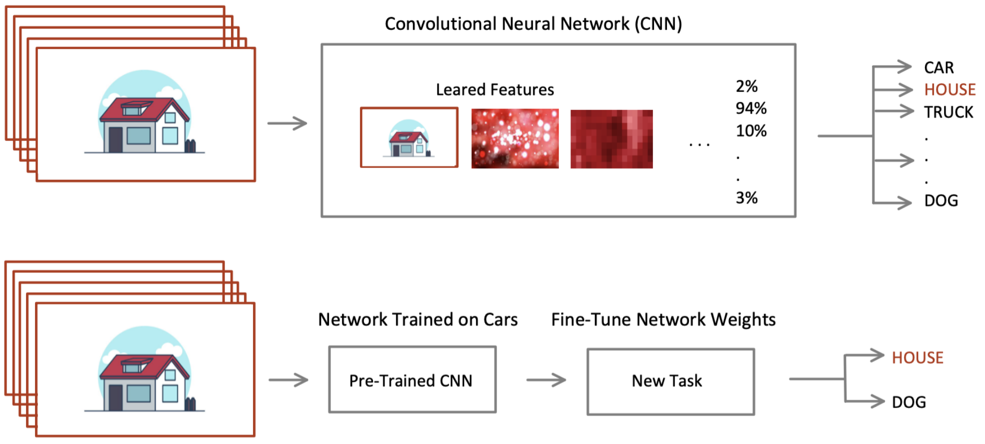


Figure 2.19.: Two different convolutional networks. The top one is trained directly on the dataset for houses, while the bottom network use transfer learning to take a network already trained on cars to classify houses in stead.

Chapter 3.

Machine Learning Networks

After the machine learning networks have been fully implemented, the learning process can begin, also known as model training. The multiple hyperparameters in the networks such as batch size, number of epochs, optimization algorithms, and loss functions can be varied to train different models. Different accuracy metrics can be used to assess the output of trained models. The different variables used to tune models are presented in this section.

The term *network* will be used in this thesis to refer to the neural network that was employed. A single uniquely trained model with specified parameters and weights is referred to as a *model*. Each model is created with a convolutional neural network. The U-Net architecture, presented in the preliminaries 2.4.13, is widely used for image segmentation in the medical field due to its ability for precise segmentation, outstanding operational speed and need for fewer training images [51]. This thesis strives to achieve the same features, making U-Net a natural choice of network.

3.1. Loss Functions

A deep neural network learns to map given inputs to given outputs through the training data. The perfect weights for a neural network cannot be calculated because there are too many unknown factors. As a result, the learning problem is approached as an optimization problem, with an algorithm employed to discover the optimum feasible set of weights for the model to make decisions. A loss function is a mathematical function that computes a single numerical value that the algorithm tries to minimize. Ian Goodfellow stated the following [17],

> "The function we want to minimize or maximize is called the objective function or criterion. When we are minimizing it, we may also call it the cost function, loss function, or error function"

The purpose of training a model is to reduce both the training and the validation loss. The loss can for instance be calculated as the difference between the predictions generated by the model when fed an input and the corresponding ground truth of the input [50]. Optimally both the training and validation loss should be approximately the same value or within a reasonable amount, since this would mean that the model is learning generalized features about the data. If the validation loss deviates upward from the training loss, it indicates that the model improves on the training set, but deteriorates on the validation set, the model may be overfitting. That is, instead of generalizing, the model memorizes the desired output for the training set.

3.1.1. Cross-Entropy Loss

To ensure that a model produces adequate results, it is necessary first to define the search aim and then select an error function that is appropriate for the task at hand. When working with semantic image segmentation, pixel-wise cross-entropy loss is one of the most often used loss functions, according to a publication by Stevens, Antiga and Viehmann [50]. Cross-entropy loss examines each pixel in the image and compares the depth-wise pixel vector, also known as the class predictions, to the ground truth vector. Because cross-entropy loss assigns equal learning to each pixel in the image by assigning class predictions, one by one, and then averaging over all pixels, it effectively assigns equal learning to each pixel in the image.

An imbalanced class distribution can be a significant disadvantage because the most extensive class can rapidly dominate the training. The dataset utilized in this thesis contains images where the main part is the background, and just a small percentage is the actual weld line. To minimize the impact of the imbalanced images, it is possible to assign a weight to each output channel. Positive masks, which are used to define the weld lines, will become more influential than negative masks, used to create backgrounds. Be aware that when positive masks are favored to improve sensitivity, it is possible that a more significant proportion of negative masks will be mispredicted, resulting in a higher number of false positives, and by that a lower accuracy, which is discussed further in Section 3.6. In this thesis, the background, the negative masks, was given a weight of 0.3 while the weld line, the positive masks, was given a weight of 0.7, based on the results reached in the specialization project done in advance of this thesis [20].

3.1.2. Dice Loss

Dice loss is another popular loss function for semantic image segmentation. This loss function is based on the Dice score, which was explained in Section 2.4.12. Dice loss is assumed to be better suited to handle imbalanced class distributions than cross-entropy loss. This is because the Dice loss function is formulated as 1 - *Dice score*, and the Dice score is based on the ratio of properly predicted pixels to the sum of all predicted pixels and the actual pixels. Equation (3.1), which corresponds to equation (2.63) in the preliminaries, is a way of conceptualizing the Dice score. True positive (TP) are positive pixels correctly predicted to be positives, false positives (FP) are negative pixels wrongly predicted to be positives and false negatives (FN) are positive pixels wrongly predicted to be negatives. The formula for Dice score can be presented as:

$$DSC = \frac{2 \times TP}{(TP + FP) + (TP + FN)} \tag{3.1}$$

3.2. Learning Rate

The hyperparameter learning rate determines how much the weights in the model change in response to the error of the predictions compared to the ground truth. Gradient descent can be used to adjust the weight, with the newly adjusted weight equal to the current weight minus the learning rate, multiplied by the gradient and the cost function at the given point. The gradient descent for weights is significantly dependent on the learning rate α, as seen in equation (3.2), equal to equation (2.62) in Section 2.4.10 in the preliminaries.

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1) \tag{3.2}$$

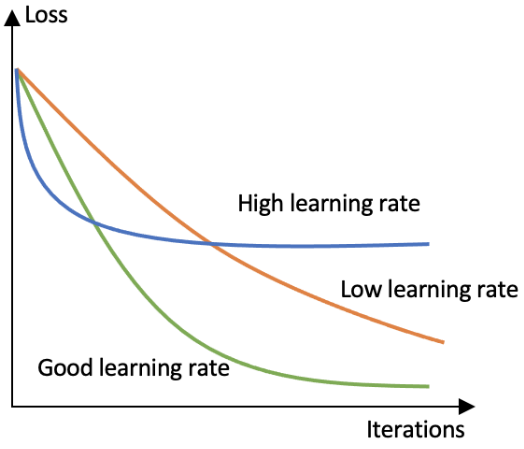


Figure 3.1.: Different learning rates and how their loss develops over the course of iterations. *Illustration [20].*

Equation (3.2) can be considered the step size for each iteration as the loss function approaches its minimum. By using a low learning rate, the accuracy converges slower, meaning that more iterations and time is needed for training, as can be seen in Figure 3.1. Furthermore, a low learning rate may cause the model to become trapped at a local, rather than global, minimum of the loss function. As a result, the training process may become stuck with less-than-ideal weights and biases. Choosing a learning rate that is too high may result in unstable training. When the step size is too big, it can overshoot the minimum, forcing a step back and overreach once more, resulting in the failure to converge or even begin to diverge. Figure 3.2 illustrates this. Adaptive learning rate approaches, which are further detailed in Section 3.5, are an upgrade to gradient descent algorithms.

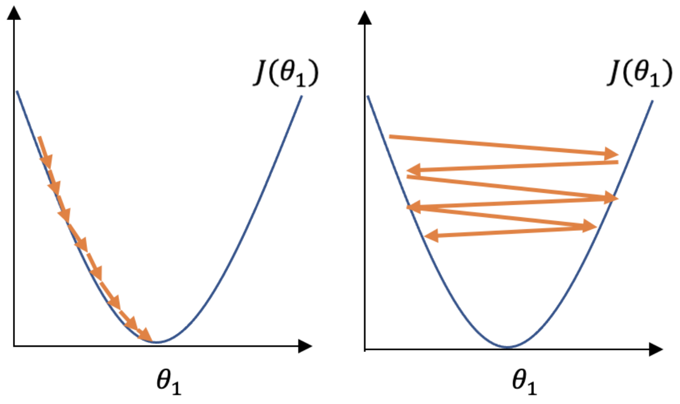


Figure 3.2.: (Left) Illustrating a too low learning rate, resulting in a very slow gradient descent. (Right) Illustrating a too high learning rate, resulting in overshooting. *Illustration [20].*

3.3. Batch Size

The hyperparameter batch size specifies how many photos should be processed before updating the model parameters. It is crucial to consider how frequently the weights should be changed when training a model. This will impact the training time of the model as well as the final result. Mini-batch learning was employed for this thesis. The predictions were compared to the ground truth after each batch, and an error was determined. The model's parameters were adjusted as a result of this inaccuracy.

The size of a batch can range from a single image to the complete training set. As a result, a small batch size will result in numerous batches and updates per epoch, but a batch the size of the entire training set will only result in one batch per epoch. Smaller batch sizes have been shown to converge faster to a good, but not always the optimal, solution empirically [27]. This is because the model does not need to complete the entire training set before learning, but can begin to learn and make minor improvements much sooner. A bigger batch size may be more efficient in terms of gradient calculation by handling more of the training set at once. However, because smaller batches begin learning sooner, they may converge faster, requiring fewer epochs. Another disadvantage of utilizing a larger batch size is that it may converge to a local optimum, resulting in poor generalization. A bigger batch size is exposed to less unpredictability, or "noise", making it more challenging to escape steep-sided local optimums. However, a too small batch size can also get stuck in a local optimum, since it computes an approximation of the true gradient and not the true gradient itself.

3.4. Epochs

An entire run-through of the dataset through the algorithm is referred to as an epoch. Further discussed in Section 4.3, the number of epochs in a model is the number of cycles the dataset is cycled through during the training phase. The model parameters of the dataset are modified at each epoch [47]. The number of times the underlying model parameters, such as weights, are updated grows as the number of epochs increases.

The number of epochs is tightly connected to the number of images in the dataset and their diversity. Traditionally, the number is high to allow the learning procedure to run until the model's error has been reduced to a reasonable level. Typical examples are 10, 100 and 500, as well as even higher numbers of epochs. Other models, such as those that use an adaptive optimization method, terminate after the loss has plateaued after a certain number of epochs. Though the number of epochs varies, it is generally understood that if a model has a relatively low number of epochs, the training process will take a short time, but may not be very accurate. In contrast, a model with a traditionally large number will take longer and could produce more accurate results as long as the model does not experience errors, such as overfitting.

3.5. Optimization Algorithms

Adaptive and non-adaptive methods for optimizing neural networks are based on the gradient descent idea for reducing the loss function. Section 2.4.10 about this topic in the preliminaries and the section on loss functions 3.1 describe how the model is optimized as the loss function is minimized. Adaptive approaches are becoming more popular for various reasons, one of which is the quick learning time.

Stochastic gradient descent, **SGD** [60], is a popular gradient descent method since it only employs a portion instead of the entire training set as in standard gradient descent, making it significantly faster. To update the weights, SGD uses the equation (3.2) presented in Section 3.2 about learning rate.

Adaptive gradient methods are different from non-adaptive gradient methods in that they include a local distance measure that is calculated using the whole sequence of weights determined up to the current iteration k, $w_1, \ldots, w_k$ [60]. Adaptive methods try to match the algorithm to the geometry of the data, whereas gradient descent uses the geometry of the parameter space itself. The *PyTorch* framework includes several different adaptation algorithms; the three utilized in this thesis are Adagrad, RMSProp, and Adam.

Adagrad [13] adjusts the learning rate based on the parameters, resulting in smaller updates. This makes it ideal for handling sparse data. Adagrad greatly enhances the resilience of SGD, making it suitable for training large-scale neural networks. However, the rapidly dwindling learning rates are one of Adagrad's drawbacks.

RMSProp [55] was created in the course of addressing Adagrad's declining learning rates. It divides the learning rate by average squared gradients that decay exponentially.

Adaptive moment estimate, **Adam** [28], computes adaptive learning rates for each parameter. In addition to retaining an exponentially decaying average of past gradients, it retains an exponentially decaying average of past squared gradients, similar to RMSProp. It has been shown to minimize training time and give resilience in hyperparameter selection.

For training a neural network, adaptive optimization methods have become increasingly prominent. When tackling simple overparameterized problems in [60], Wilson, Roelofs, Stern, Srebro, and Recht describe how these popular approaches often discover drastically different solutions than non-adaptive methods, such as stochastic gradient descent. SGD is compared to adaptive optimization methods such as Adagrad, RMSProp, and Adam in this research.

The binary least-square classification problem that is studied in [60] is used to solve the minimization problem

$$\min_{w} R_s[w] := \frac{1}{2}||X_w - y||_2^2 \tag{3.3}$$

Here w is the best linear classifier, X is a matrix of features and y is a vector of labels.

Solving this classification problem will show both which solutions the algorithms identify and how well they work on unseen data. They discovered that non-adaptive approaches such as SGD converge to the minimal Euclidean norm solution with the most considerable margin of all the following equation's solutions

$$Xw = Y \tag{3.4}$$

On the other hand, adaptive approaches converge on the notion that there is a solution where $Xw = Y$ is proportional to

$$\operatorname{sign}(X^{\mathrm{T}}y) \tag{3.5}$$

When the solution described above exists, all adaptive gradient algorithms will converge. The solution is easier to understand than the one reached using non-adaptive approaches.

Adaptive approaches find solutions that generalize poorer than non-adaptive methods, according to the study. Adaptive algorithms tend to have fast initial progress on the training set, but their performance quickly plateaus on the validation set. The study suggests that the non-adaptive approaches are most efficient, even though the adaptive optimizer Adam remains one of the most used. They speculate that Adam's appeal stems from its compatibility with optimization-free iterative search techniques, but add that a properly-tuned SGD might also function well in these applications. During this thesis, all four optimizers were used to train models to compare their results in this particular image segmentation problem.

3.6. Accuracy Metrics

The term *accuracy*, when working with image processing, refers to a measure of consistency with accurate information in a spatial point with data [23].

3.6.1. Pixel Accuracy

Pixel accuracy, according to the article written by Ekin Tiu from the ML group at Standford University [56], may be the most intuitive measure for the accuracy of a model prediction. It calculates the percentage of correctly classified pixels in an image. The pixel accuracy is calculated by dividing the numbers of correctly classified pixels by the total number of classified pixels in terms of true or falsely predicted positive or negative pixels presented in Section 3.1.2:

$$PixelAccuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.6}$$

Class imbalance is a problem that might arise when calculating pixel accuracy. This happens when classes are largely imbalanced, in other words, if one or more classes dominate the image, while others only make up a small portion. As seen in Figure 3.3, if the classes only make up a fraction of the image, a high pixel accuracy does not always guarantee a precise segmentation. The predicted image is pure black in this example, shown in Figure 3.3(b), despite the high accuracy of 0.95, this is an excellent example of class imbalance.

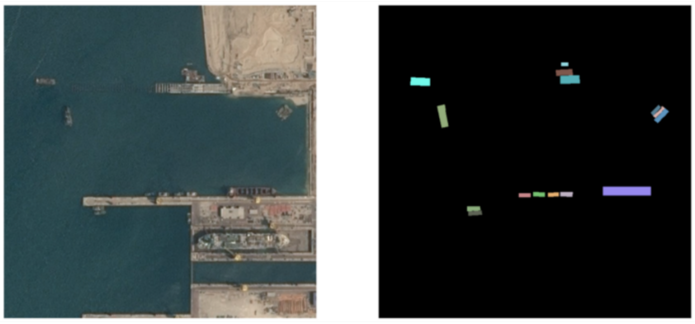
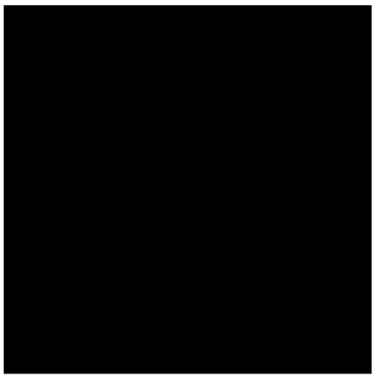


Image from Vlad Shmyhlo in article: Image Segmentation: Kaggle experience (Part 1 of 2) in TDS

(a) (Left): Input image. (Right): Ground truth.



(b) Model prediction.

Figure 3.3.: Example of class imbalance, where the predicted model is a completely black image. *Illustration [56].*

3.6.2. Sensitivity and Specificity

Because the weld line is such a minor element of the image, losing a few pixels or even the entire weld line could still result in a high pixel accuracy. This may give the wrong impression of the model's performance, as a partial weld line forecast will not be helpful. Sensitivity, also known as recall, is vital for the segmentation in this thesis since it indicates the number of correctly recognized weld line pixels. This can be seen as the true positive rate. Specificity is the ability to categorize background pixels correctly, the true negative rate. This metric is also essential in segmentation, but notice in this thesis that the background pixels make up such a large part of the images that many background pixels must be predicted wrong to get a noticeable change in the specificity. If focusing solely on accuracy, it is easy to disregard sensitivity and specificity, oblivious to the possibility of many false positives or negatives. To get a clear overview of how the different models perform, it is important to analyze and compare these metrics.

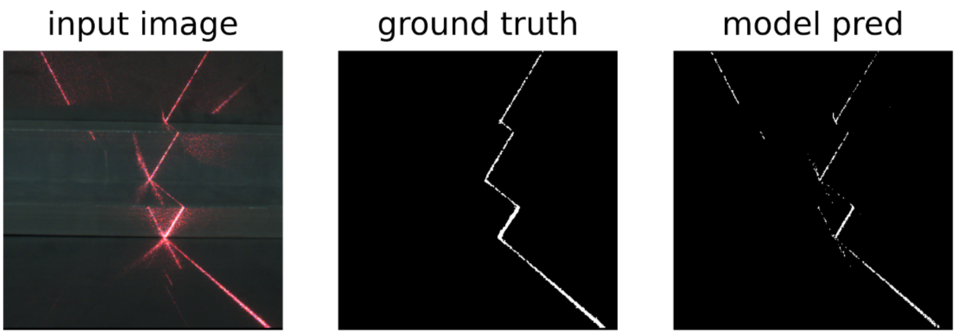
3.6.3. Dice Score

The Dice score or the Sørensen-Dice coefficient explained in Section 2.4.12 is a typical metric to determine the overlap between the ground truth and the predicted segmentation. The equation multiplies the area of overlap and divides it by the total number of pixels in the images, as illustrated in equation (2.63). The equation can also be described in terms of the four possible outcomes of classification

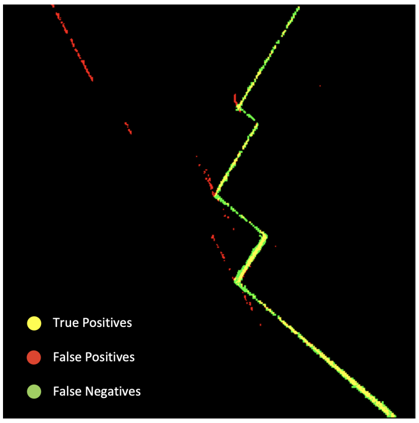
$$DiceScore = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{3.7}$$

Figure 3.4 illustrates the classification of pixels, either as a weld line line pixel or a background pixel. The number of pixels classified as each outcome is often gathered in a confusion matrix, illustrated in Figure 3.4(c), to calculate the accurate Dice score.

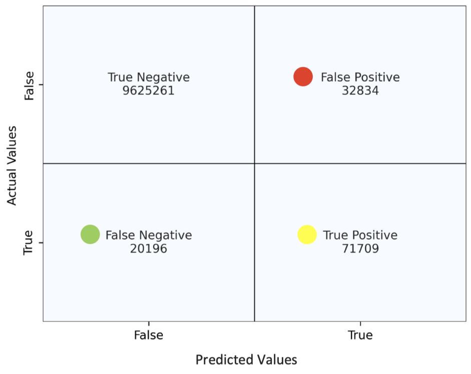
Dice score is a favored accuracy metric by many as it only values the true positives, TP, as correctly predicted pixels. By multiplying the number of the correctly predicted true positives by two and ignoring the correct true negatives, the Dice score counters the class imbalance problem that can occur in pixel accuracy. Though the issue of class imbalance is addressed, neither pixel accuracy nor Dice score considers the distance from the wrongly predicted scan line to the ground truth, which should be taken into account to evaluate the degree of the wrong prediction.



(a) Comparison of an input image, its ground truth and the model prediction.



(b) The model prediction with the different possible outcomes of classification in different colors.



(c) Confusion Matrix of the entire last validation epoch.

Figure 3.4.: Classification of pixels used to calculate accuracy.

Chapter 4.

Method

This chapter explains how the results were obtained. From the creation of datasets, handling large datasets and the machine learning techniques used to extract findings from the datasets, to the agile collaboration approaches.

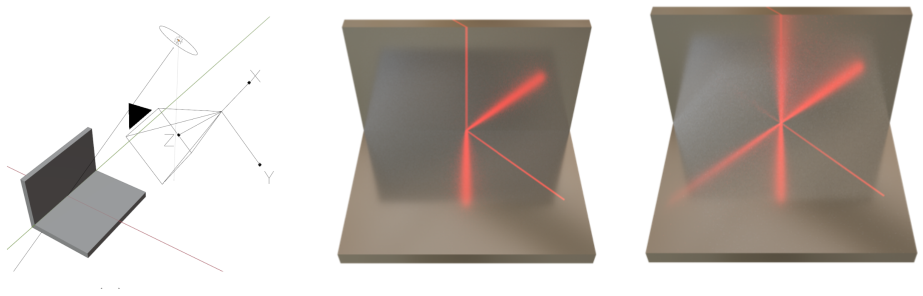
4.1. Dataset Generation

4.1.1. Simulated Images

The initial training of all models was carried out on a simulated dataset. Ola Alstad created the dataset, including the images and ground truths of fictitious welds, as a part of his master thesis and this process is explained in detail in his thesis [4]. Because the simulation of light interacting with both materials and objects was required, LuxCoreRender, a physically-based render engine, was used to produce the images. Blender, a 3D creation suite, was used with LuxCoreRender. Blender is an open-source 3D computer graphics tool commonly used for visual arts and modeling. It has a Python API and a graphical user interface, making it a suitable development platform.

To create a rendered image in Blender a scene must comprise objects, lighting, and at least one camera. The render engine receives the scene data from Blender. Tracing the light paths flowing into the chosen camera produces an image. Cycles is a fast physically-based renderer, and the default ray tracer engine of Blender, but it only allows backward ray tracing. Backward ray tracing struggles to calculate caustics, as mentioned in Section 2.5.2. LuxCoreRender, on the other hand, is compatible with hybrid ray tracing, which is why it was selected as the render engine. The difference between the engines is crucial for obtaining a true simulation, as seen in Figure 4.1 from Alstad's thesis [4]. A scene with a corner mesh, camera, and laser line is shown in 4.1(a). Cycles and LuxCoreRender are used to render this scene in 4.1(b) and 4.1(c), respectively. Cycle, as seen, cannot calculate the caustics in the scene, which accounts for a significant portion of the reflections, whereas LuxCoreRender produces more realistic reflections.

The pinhole camera, discussed in Section 2.2.1, is already implemented in Blender and LuxCoreRender. However, a wrapper class was created using the Python API for additional functionality and to simplify the initialization of the camera. To simulate the laser

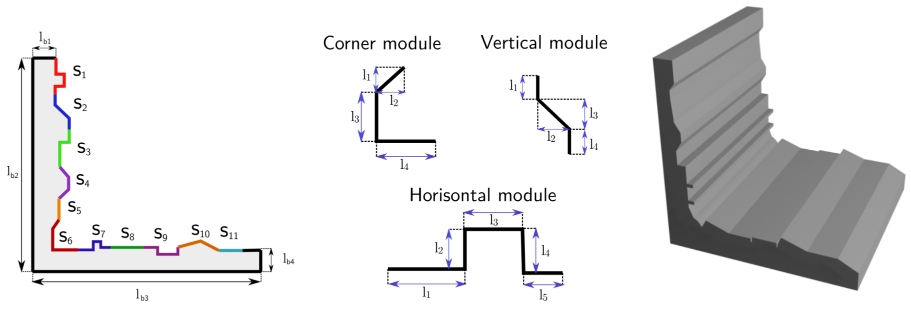


(a) Scene setup (b) Cycle (c) LuxCoreRender

Figure 4.1.: Cycle and LuxCoreRender comparison. *Illustration [4].*

line a precise line was projected onto the center of an images with otherwise black pixels. Using an image to project the laser line allows for controlling the line's color, width, and appearance. The parametric 3D CAD program CadQuery [59] was used to create the 3D weld corner models. Python was able to produce 3D models with unique pieces due to this. Alstad writes the following in his thesis [4]:

> The parts were generated by defining a set of 2D modules which were assembled to generate a corner cross- section as shown in 4.2(a). The cross-section consists of a base defined by the constant lengths $l_{b1}, l_{b2}, l_{b3}, l_{b4}$ and a set of 11 sections $s_1, s_2, \ldots, s_{11}$. At each section s_n, a random module is chosen. Each module m is defined by vertical, horizontal, diagonal or curved lines with randomized lengths. Modules were defined for the vertical sections, corner section and horizontal sections respectively. 9 vertical modules, 4 corner modules and 4 horizontal modules were made. Three example modules of each type is shown in 4.2(b), where the lengths $l_1, l_2, \ldots, l_n$ are randomized in a uniform interval. Once the cross-section was defined by the set of random modules, it was extruded a constant length to get a 3D part as shown in 4.2(c).



(a) (b) (c)

Figure 4.2.: Randomized model generation, *Illustration [4].*

By assigning PBR texture pictures to the parts, the materials for the parts were created. PBR is a two-dimensional picture creation technology that stores color and surface information. The PBR materials in this example were images that described color, roughness, normals, and metalness. A total of 40 PBR textures were downloaded from ambientCG, a public domain resource for physically based rendering [5], and utilized on the dataset.

4.1.2. Real-World Images

The real-world images were taken by a Teledyne DALSA Genie Nano-GigE camera [52] with a Computar MPZ Series Machine Vision Lens V1226-MPZ 1" 12 mm F2.6 [12]. According to Teledyne DALSA the scanning camera redefines low-cost performance with its industry-leading Complementary Meta Oxide Semiconductor (CMOS) sensors. The electrical semiconductor image sensors convert light into electrical signals [8] and are a proprietary technology for break through speed and robust build quality. The setup includes an industrial-grade laser of class 3B, which is considered to be a medium-powered laser with a range from 5 to 499 milliwatts [33]. A module head glass lens was attached to the laser to create the laser line. Figure 4.3 shows the entire setup of how the images were taken.

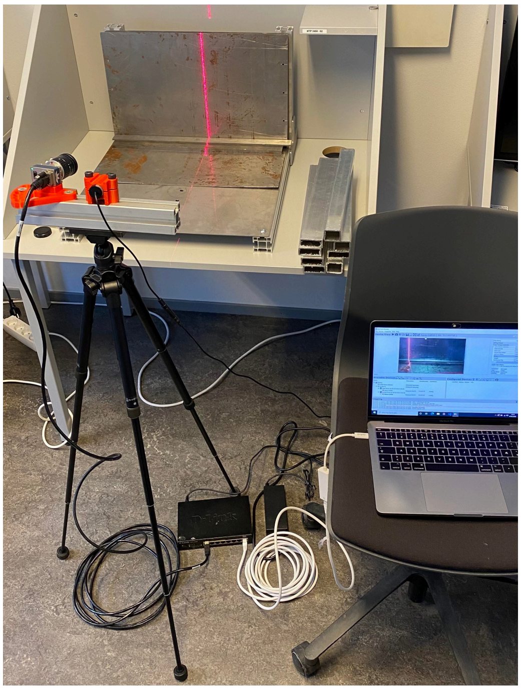
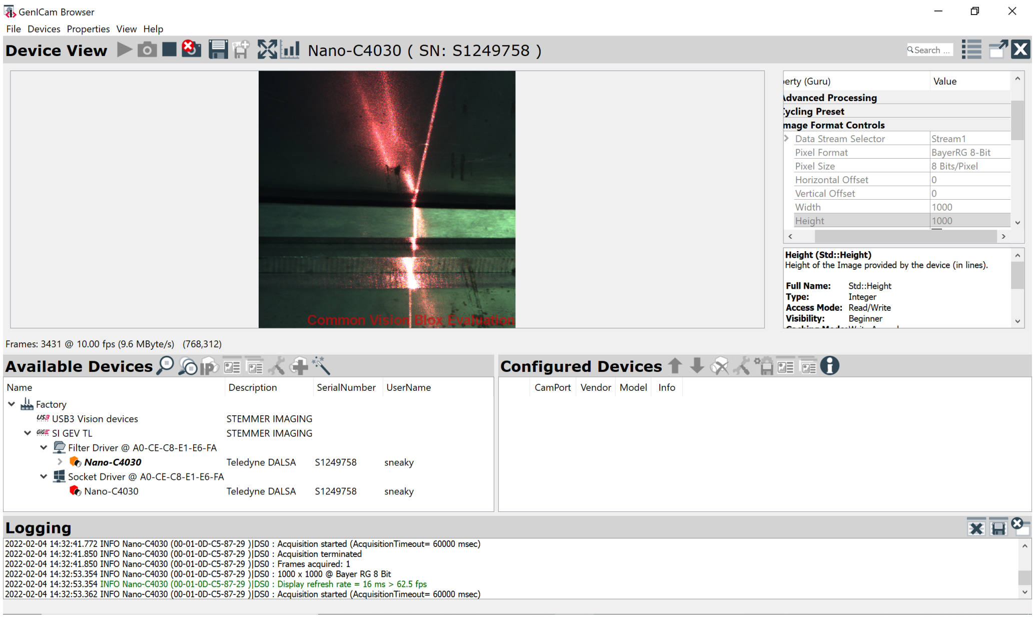


Figure 4.3.: The setup used to take images. The camera and laser mounted on a tripod to the left and the D-Link Desktop beneath it connected to both the camera and the computer with the GenICam Browser. The metal profiles and parts were placed on a desk in front of the setup.

To operate the camera Common Vision Blox (CVB) [11] was downloaded on a windows machine. CVB is a high-speed and high-end vision library, allowing users to develop fast and powerful applications. Using a D-Link DGS 1008P 8-port Gigabit PoE Unmanaged Desktop Switch provided the functionality to operate the camera through the CVB application called GenICam Browser. GenICam provided functionality to set the dimensions and save the images, Figure 4.4 shows the layout of the application. The images were uploaded to the GitHub repository containing the software, explained further in Section 4.5.1.


GenICam Browser
File Devices Properties View Help
Device View
Nano-C4030 (SN: S1249758)
Search ...
Common Vision Blox Evaluation
Available Devices
Configured Devices
Logging

erty (Guru)	Value
dvanced Processing	
ycling Preset	
mage Format Controls	
Data Stream Selector	Stream1
Pixel Format	BayerRG 8-Bit
Pixel Size	8 Bits/Pixel
Horizontal Offset	0
Vertical Offset	0
Width	1000
Height	1000

Height (Std::Height)
Height of the Image provided by the device (in lines).

Full Name:	Std::Height
Type:	Integer
Access Mode:	Read/Write
Visibility:	Beginner

Frames: 3431 @ 10.00 fps (9.6 MByte/s) (768,312)

Name	Description	SerialNumber	UserName
Factory			
USB3 Vision devices	STEMMER IMAGING		
SI GEV TL	STEMMER IMAGING		
Filter Driver @ A0-CE-C8-E1-E6-FA			
Nano-C4030	Teledyne DALSA	S1249758	sneaky
Socket Driver @ A0-CE-C8-E1-E6-FA			
Nano-C4030	Teledyne DALSA	S1249758	sneaky

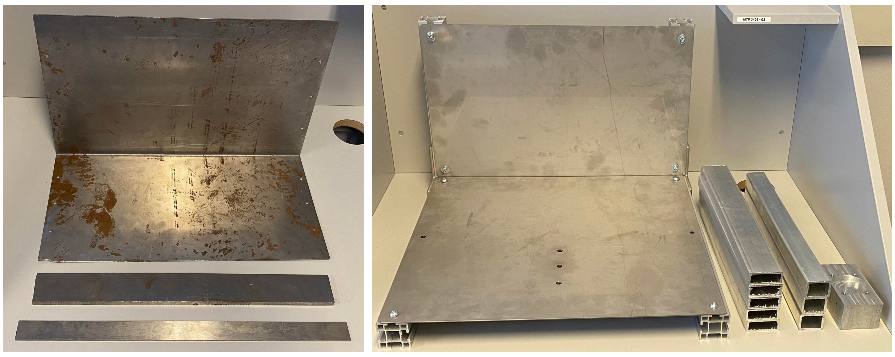
CamPort	Vendor	Model	Info

```
2022-02-04 14:32:41.772 INFO Nano-C4030 (00-01-0D-C5-87-29 )|DS0 : Acquisition started (AcquisitionTimeout= 60000 msec)
2022-02-04 14:32:41.850 INFO Nano-C4030 (00-01-0D-C5-87-29 )|DS0 : Acquisition terminated
2022-02-04 14:32:41.850 INFO Nano-C4030 (00-01-0D-C5-87-29 )|DS0 : Frames acquired: 1
2022-02-04 14:32:53.354 INFO Nano-C4030 (00-01-0D-C5-87-29 )|DS0 : 1000 x 1000 @ Bayer RG 8 Bit
2022-02-04 14:32:53.354 INFO Nano-C4030 (00-01-0D-C5-87-29 )|DS0 : Display refresh rate = 16 ms > 62.5 fps
2022-02-04 14:32:53.362 INFO Nano-C4030 (00-01-0D-C5-87-29 )|DS0 : Acquisition started (AcquisitionTimeout= 60000 msec)
```

Figure 4.4.: The layout of the GenICam Browser application.

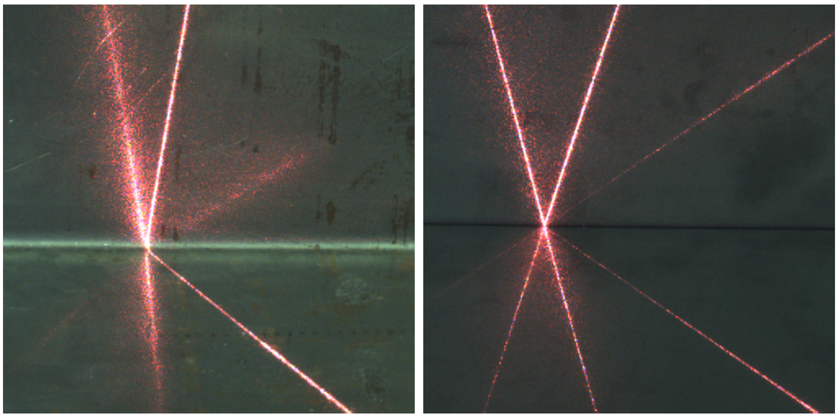
It was necessary to have a significant variation in the reflections to compare results adequately. A variation of metal profiles in two different materials, steel and aluminium, were used to create reflections of different types and intensities. Figure 4.5 displays the different parts used to generate unique images. A profile was used as the background and the smaller parts were placed in varying formations to illustrate different welds. The tripod with laser and camera was pointed onto the formations from different angles to ensure that every image was unique.

The two different metals were chosen because their surfaces result in various reflections. Figure 4.6 illustrate these differences. Steel is a rougher material with a more matte finish, and the profile and parts used in this thesis also include a degree of rust, resulting in even more diffuse reflections as shown in Figure 4.6(a). Aluminium is shinier and has a smoother finish, leading to generally more reflections, in particular, more specular reflections, illustrated in Figure 4.6(b).



(a) Steel parts. (b) Aluminium parts.

Figure 4.5.: The metal parts that were used when generating unique images. The smaller parts were placed in different formations on top of the larger profiles to simulate different types of welds.

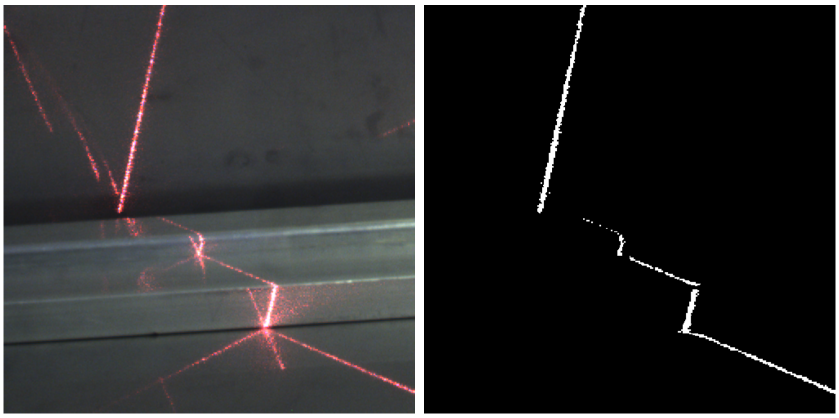


(a) Steel. (b) Aluminium.

Figure 4.6.: Images taken of laser reflection on steel and aluminum materials.

Generating Ground Truth Images

The ground truth images corresponding to the real-world images were generated manually by the use of GIMP, an image manipulation program [32]. GIMP allowed copying of each image, layering a threshold and transforming the image to black and white, automatically removing the majority of the laser reflections with low intensity before finally removing all the remaining reflections manually. This resulted in a black and white ground truth to be used in the transfer learning of the networks. An example of a real-world image and its corresponding ground is illustrated in Figure 4.7.



(a) Real-world image. (b) Generated ground truth.

Figure 4.7.: A real-world image taken of reflections on aluminium and its corresponding ground truth generated in GIMP.

After the ground truths had been generated, both the original images and their corresponding ground truths had to be cropped into a format that would suit the algorithm. The original images had a resolution of 500 × 504 pixels, and the 4 bottom pixels were therefore cropped from the image, resulting in an image of 500 × 500 pixels. This was done so the real-world images would have the exact resolution as the simulated dataset.

4.2. Dataset Handling

The simulated dataset received from Alstad was extensive, and the addition of real-world images would increase its size even further. It would require a large amount of processing power and memory to be able to train models with this dataset, more than that of a standard personal computer with an ordinary graphics card.

For convenience, the algorithm were developed on personal computers during the implementation stage with only a few images in the training and validation sets to build a functioning network. A significant increase in processing power was necessary to train the models properly with a large enough training set. This demonstrated a fundamental understanding of neural networks, the necessity for an extensive database to adequately train a model.

In preparation for this thesis, a specialization project was conducted. The project utilized physical computers at a computer lab at NTNU called Cybele. The computers used NVIDIA GeForce GTX 1080 Ti graphics card, which was the fastest desktop consumer graphics card when it was released in 2017. While these computers provided a lot of processing power, one of the goals for this thesis was to increase the processing power even further. This goal was reached by connecting to the Idun cluster.

4.2.1. The Idun Cluster

Personal computers often only have a central processing unit, CPU, while networks of more extensive size, training on many images, require a graphic processing unit, GPU. GPUs break complex problems into many separate tasks to work them out at once, which is why they are ideal for processing graphics, and in this case, image segmentation. CPUs race through a series of tasks requiring a large amount of interaction and restricting them to only performing a few operations at a time, while the GPU can perform thousands at the same time, making a GPU significantly faster and more efficient than a CPU [7].

The Idun cluster is a project created at NTNU to provide rapid testing and prototyping of high-performance computing software [36]. The cluster has access to several GPUs that can run in parallel. The nodes used in this thesis are of type Dell PE730, and they have two processors, 48 cores and ten GPUs. The GPUs are of the type NVIDIA A100 and have 40GB memory. Having ten GPUs of this strength is an enormous improvement compared to only one NVIDIA GeForce GTX 1080 Ti GPU used in the specialization project [20]. One A100 [38] has 3.6 times larger memory than the GeForce [37]. By having ten of these GPUs, the memory would be 36 times larger, making a huge difference. An in-depth explanation on how to connect to the Idun cluster can be found in Appendix B.

The increase in processing power allows for an increase in the number of images in the training set, an increase in batch sizes and the number of epochs. In addition, the increased power would decrease the network's training time. On the other hand, the A100 GPUs are expensive. The price lie around 100000 NOK, raising the question of whether it is reasonable to assume companies have these GPUs at hand. If networks that require this amount of processing power are suitable for industry implementation is discussed further in Section 6.3.

4.3. Dataset Splitting and Configuration

Both datasets were divided into two different parts: a training set and a validation set. The training sets were the largest, and were used to train the model, while the validation sets were used to assess the model's correctness throughout training. The simulated dataset was used for validation when training on the simulated dataset. Afterward, the real-world dataset was used for validation when training on the real-world dataset. The models were tested on the validation sets every second epoch of training, and this gave a clear overview of the models development while saving some computational power.

Due to the increased processing power provided by Idun, explained in Section 4.2.1, it was possible to utilize the entire repository of simulated images generated by Alstad in his thesis [4]. This resulted in a dataset consisting of 4000 simulated images with a matching ground truth set. 200 images were taken for the real-world dataset, 100 of these being aluminum and 100 steel. Afterward, a matching ground truth set was created as explained in Section 4.1.2.

The transfer learning was executed by first training a model on the simulated images for a given amount of epochs, followed by a given amount of epochs of training on the real-world images. When splitting the simulated and real-world datasets into their respective training and validation sets a specifically chosen splitting fraction was used for both.

The training set fraction was set to 0.8 and the validation set fraction was set at 0.2 to split the dataset. In the field, a split based on this proportion is common, it provides a large amount of training data while yet allowing for enough unique data in the validation set. The number of images in the training and validation sets are shown in Table 4.1. Using the 0.8 training fraction, the simulated training set included 4000 images multiplied by 0.8, for a total of 3200 images passing through the model per epoch. As mentioned earlier, the models were tested every second epoch against a validation set. This consisted of a 0.2 fraction times 4000 images, resulting in a validation set of 800 images running through the model per validation round. The images of welds in various angles, materials, and image-brightness were included in Alstad's dataset. As explained in the preliminaries in Section 2.5.1, these images displayed various forms of reflections, including specular, spread, and diffuse reflections.

To build the groundwork for an authentic model, a given dataset would be shuffled at random before being split into training and validation sets. An example as to why this is important could be when using the real-world dataset, this would ensure that a model did not only train on steel and only validate on aluminum. Instead, a random amount of steel and aluminum for both the training and validation set ensures that a model learns both spectral and diffuse reflections. The random distribution was held for the entire training phase, not only per epoch. This ensured that the images in the validation set stayed in the validation set. Meaning they were never used during training, this guaranteed that the validation score was based on images the model had not adjusted itself after.

	Dataset	Length of Dataset
Initial learning	Total	4000
	Training set	$0.8 \times 4000 = 3200$
	Validation set	$0.2 \times 4000 = 800$
Transfer learning	Total	200
	Training set	$0.8 \times 200 = 160$
	Validation set	$0.2 \times 200 = 40$

Table 4.1.: The datasets and splitting used during training.

4.4. Transfer Learning

Traditional learning is isolated and performs solely on specific tasks and datasets, training individual models in their regard. There is no knowledge that can be transferred and used afterward by other models. Transfer learning on the other hand, is utilizing knowledge acquired from one task to solve related ones, as briefly explained in the preliminaries in Section 2.6. It is typically used when the target dataset contains insufficient data to train a full-scale model from scratch. Transfer learning allows models to use features, weights and more from previously trained models. This allows for more flexible and faster learning, and resulting in lower generalization error. With possibly a more robust model that can make sound predictions even when having less data for the newer assignment. The reason being certain low-level properties such as edges, intensity, corners and forms can easily be used across tasks in the computer vision domain giving a solid foundation before tuning the model on the last dataset or task. However, it is essential to notice that transfer learning should be used to improve rather than deteriorate target task performance. Transfer learning does not always help a model to make better predictions, it is possible for transfer learning to make a model worse, this is known as negative transfer.

In this thesis the models were first trained on the simulated dataset for a given amount of epochs. After this, the models' hyperparameters and learnable parameters, such as weights, bias and more, were saved as a *.pt* file. PyTorch convention is to save models using either a *.pt* or *.pth* file extension [22]. This allowed for saving and loading of the files, either for testing locally or as a full-scale run on Idun. The next step was the transfer learning over to the real-world dataset. Before learning could take place on the real-world dataset, the *.pt* file needed to be loaded so that the object files could be deserialized to memory. Then the loaded model would have the same structure and learnable parameters as the previously saved model, and learning on the real-world dataset could start. A visualization of the process can be seen in Figure 4.8. This type of transfer learning is known as inductive transfer learning, and it relates to a learning mechanism's ability to improve performance on a current task after acquiring a distinct, but related, skill on a previous task.

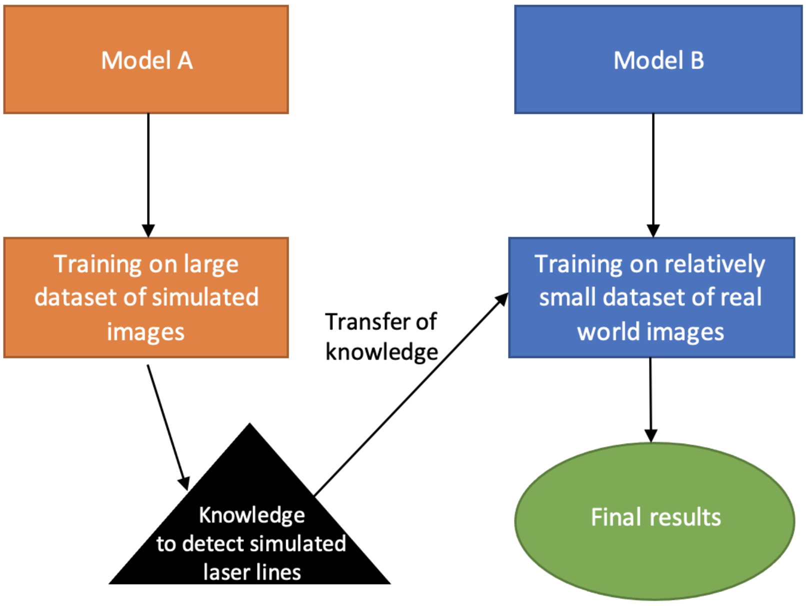

Model A
Model B
Training on large dataset of simulated images
Transfer of knowledge
Training on relatively small dataset of real world images
Knowledge to detect simulated laser lines
Final results

Figure 4.8.: Visualization of how transfer learning is used in this thesis.

When performing transfer learning, it is also an optional step to freeze or delete some layers of the loaded model before proceeding with the learning. Freezing a layer means disabling learning on the specific weights and biases in that layer. Freezing a layer reduces the computing time required for training while maintaining a high level of accuracy. It is most common to freeze the earliest layers of the network since this removes the need to use backpropagation for the gradient and update the weights in these layers. Suppose one were to freeze one of the deepest layers, one would still have to use backpropagation through the layer to get to the first layers, even if the deep layer does not need it. By deleting layers, the risk of overfitting is minimized by reducing the complexity of the network. It has been shown to allow highly deep networks to perform exceptionally well even when only a small target dataset is supplied [61]. For this thesis, none of the layers were frozen or deleted. This was decided to focus on the model's hyperparameters and to better understand how the U-Net architecture would behave. U-net's depth and complexity make it interesting to examine where the models would start to overfit when going from a large dataset to a much smaller one.

Instead of freezing layers, it is possible to adjust the learning rate transferring from the large dataset to the smaller. For example, looking at Figure 4.8 model A could use Adam as an optimizer with a learning rate set to 0.0003, while model B could use a lower learning rate set to 0.0001. The learning rate should be low because the U-net architecture is deep and complex, while the real-world image dataset is small. This brings the increased risk of overfitting and a low learning rate has the possibility of minimizing this outcome, as explained in Section 3.2.

4.5. Agile Collaboration

The emphasis on collaboration, communication and feedback is one of the fundamental contrasts between agile and plan-driven software development [34]. Retrospectives, pair programming and code reviews are all examples of collaboration. This thesis was developed through agile collaboration, both during software development, planning and implementation.

Regular updates with an academic advisor, ongoing team communication, presentation, code sharing, pair programming, and task distribution contributed to the thesis's natural evolution.

4.5.1. Software Development Collaboration

It was decided to use GitHub to provide seamless collaboration for the software development. GitHub includes capabilities like continuous integration, bug tracking and distributed version control [43], which make it ideal for collaboration and the reason behind the choice as a host for the software development in this thesis. Including a website, GitHub offers a desktop application that is simple to use and integrates with the source code editor Visual Studio Code [35]. This, in combination with the previous familiarity with the frameworks and that the development of the specialization project [20] was completed with this combination, made GitHub and Visual Studio Code a natural choice. As with the specialization project [20], the development was done in the popular high-level and open-source programming language Python [44], which is known for its extensive machine learning tools.

The code developed during this thesis can be obtained in the following GitHub repository.

The code was self-written, with guidance from Ola Alstad and resources from the available online community of machine learning tools.

Chapter 5.

Results

The following sections present the results produced during this thesis. Numerous variations of U-Net models were used to conduct transfer learning to study and compare their results. Complete tables of the results can be found in Appendix A.

5.1. U-Net

As explained in Section 3.1.1, the weights on positive and negative masks were decided based on the results found in the specialization project [20] done in advance of this thesis. The optimal weighting of cross-entropy according to [20] was found to be 0.7 for the positive masks and 0.3 for the negative masks. During the project different epochs were also evaluated. This gave a solid foundation for this thesis to choose the number of epochs based on empirical analysis. However, the simulated dataset was more extensive in this thesis than in the previous project. In addition, transfer learning was now being implemented. Therefore some initial trials were executed to find the optimal amount of epochs for the simulated and real-world dataset in this thesis. It was decided to run models on the simulated dataset for 4 epochs. This was fewer epochs than in the previous project, but the number of training steps the models would run through would be about the same as the best model of the prior project since the dataset was more extensive this time. U-Net has several layers and as Section 2.4.13 explains, the architecture is quite complex. This is reflected in the time it takes to train the models made with the network. Furthermore, the complexity of the network means one must be cautious to avoid overfitting. Especially since the goal of the thesis is to get the best real-world prediction the network should learn the fundamentals from the simulated dataset, not get too specialized. This could lead to the transfer learning degrading the pre-adjusted weights in the model instead of developing them further towards the global optimum. Since the real-world dataset only consists of 200 images, the deep neural network is especially prone to overfitting.

As for the transfer learning, it was chosen to train for 6 epochs on the real-world images. Beyond this point, the models would become very unstable, jumping between good and poor predictions from one epoch to the next. These tendencies can also be seen by examining the plot of the loss function in Figure 5.1. This model was first trained on 3

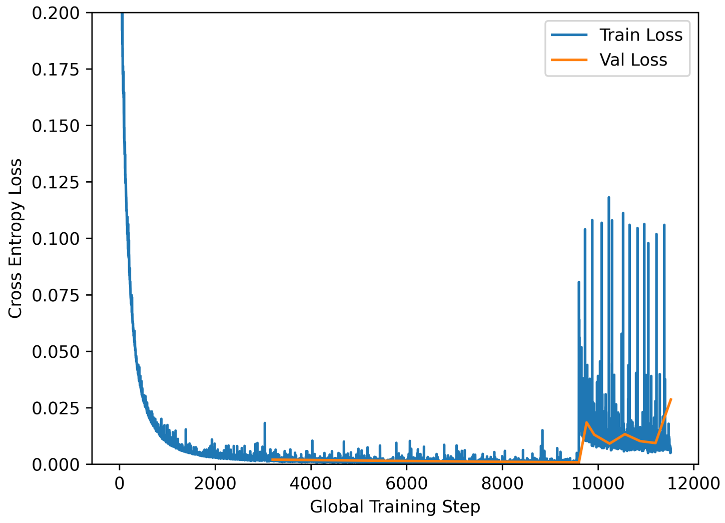


Figure 5.1.: Plot of the loss function to a test-model showing the validation loss diverging from the training loss towards the end of training.

epochs with 3200 training images on the simulated dataset. This corresponds to the first 9600 global steps since each image is seen as a step during the model learning. Independent of whether the weight-updating happens for every single image or the given batch size. Then the model started transfer learning on the real-world dataset with 160 training images for 12 epochs, a total of 1920 global steps. This corresponds to the global steps between 9600 and 11520. Be aware that both the training and validation were executed with the simulated dataset during initial learning, while during transfer learning, both training and validation were executed with the real-world dataset. Towards the end of the training on the real-world images, the orange validation loss diverges up and away from the trending blue training loss, which is a clear sign of overfitting. Since this is a sign that the model is learning the individual images in the training set and not learning generalization, which is needed to make good predictions on unseen images, such as the images in the validation set. The individual high peaks for the training loss during the real-world dataset will be further discussed in Section 6.2.3 as these often are correlated with the batch size and were seen in some other models as well. Additionally, potential improvements to the algorithm and the training runs are discussed in Section 6.4. Dice score was used as the primary accuracy metric to compare the results since it counters the problem of class imbalance that occurs in pixel accuracy, as explained in Section 3.6. The different accuracy metrics and their use during the thesis are further discussed in Section 6.2.1.

5.2. Different Optimizers

Models were made using all four optimizers presented in Section 3.5: the non-adaptive SGD and the adaptive, Adagrad, RMSProp and Adam. To better study the behavior of the different optimizers, models were made with learning rates varying from 0.1 to 0.0001. Table 5.1 shows the Dice scores of the different models. From the table, it is interesting to observe that the more adaptive the optimizers were, the lower learning rates achieved the highest Dice score. The non-adaptive SGD performed best at the highest learning rate, 0.1, while the two most adaptive being RMSProp and Adam performed best at the lowest learning rates. RMSProp reached the highest dice score with a learning rate of 0.0001, and Adam with a combination of a learning rate of 0.0003 during initial learning and 0.0001 during transfer learning.

Optimizer	Learning Rate	Batch Size	Epochs	Dice Score
SGD	0.1	4	4, 6	0.8057
SGD	0.01	4	4, 6	0.7634
SGD	0.001	4	4, 6	0.6109
SGD	0.0001	4	4, 6	0.4023
Adagrad	0.1	4	4, 6	0.8104
Adagrad	0.01	4	4, 6	0.8151
Adagrad	0.001	4	4, 6	0.8328
Adagrad	0.0001	4	4, 6	0.7723
RMSProp	0.1	4	4, 6	0.7643
RMSProp	0.01	4	4, 6	0.8225
RMSProp	0.001	4	4, 6	0.8364
RMSProp	0.0001	4	4, 6	0.8551
Adam	0.1	4	4, 6	0.7073
Adam	0.01	4	4, 6	0.7994
Adam	0.001	4	4, 6	0.8303
Adam	0.001, 0.0001	4	4, 6	0.8409
Adam	0.0003, 0.0001	4	4, 6	0.8426

Table 5.1.: Dice scores for models with different optimizers and different learning rates. The batch sizes, epochs and cross-entropy loss function were consistent. The models reaching the highest Dice score for their respective optimizers are outlined in blue.

The results presented in Table 5.1 are illustrated visually in Figure 5.2. The models with SGD as optimizer have Dice scores that deviate most compared to the other optimizers. An increase in learning rate increases the Dice score, resulting in twice the Dice score at learning rate 0.1, in purple, compared to the Dice score at learning rate 0.0001, in blue. RMSProp and Adam illustrate the opposite behavior, achieving their highest Dice score for the lowest learning rate, though their difference in overall Dice score is smaller. RM-SProp has a difference of less than 0.1 between the highest and the lowest score. Adagrad, on the other hand, illustrates different behavior than the other adaptive optimizers by increasing until it achieves the best Dice score at the learning rate of 0.001, in green, before decreasing. Overall, RMSProp reaches the highest Dice score among the optimizers with a Dice score of 0.8426.

Figure 5.3 illustrates the cross-entropy loss functions for the four models that achieved the highest Dice score for their respective optimizers, the models marked in blue in Table 5.1.

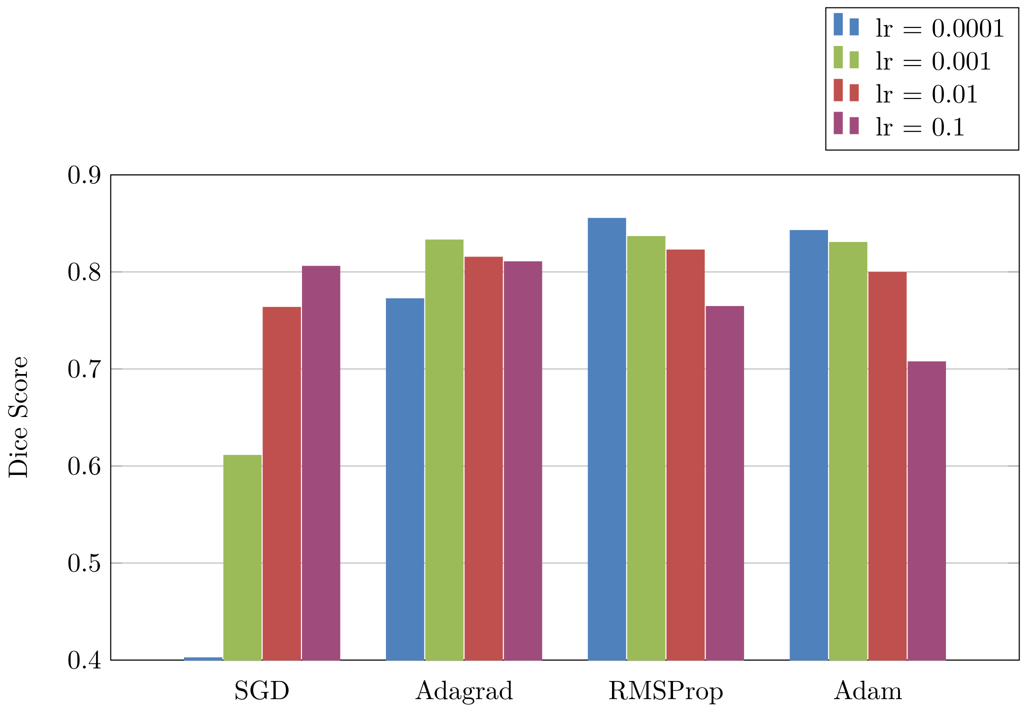
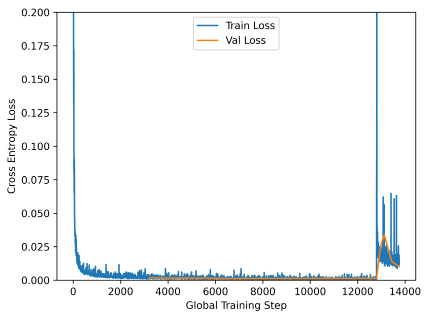
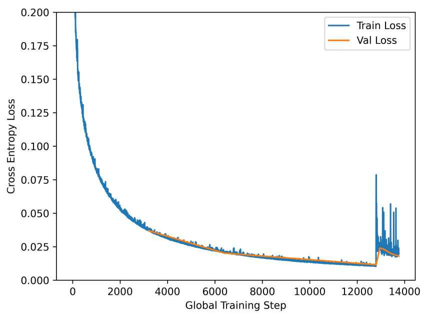


Figure 5.2.: Graphs illustrating the Dice scores for different optimizers and learning rates.

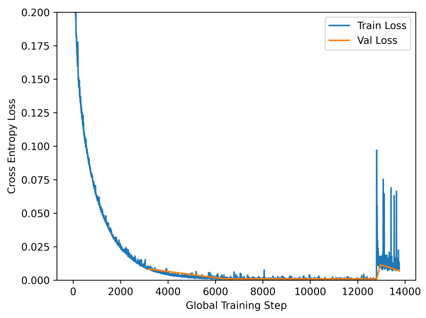
From the loss functions it is a clear difference between the adaptive optimizers and the non-adaptive optimizer, SGD. The cross-entropy loss of SGD, seen in Figure 5.3(a), quickly plateaus at a slight loss during the initial training process on the simulated dataset. When the transfer learning begins, the loss spikes upwards before decreasing, but at a higher loss than the adaptive optimizers. In addition, the validation loss uses more training steps to decrease than the others. This could be a sign of overfitting, which will be discussed further in Section 6.2. The adaptive optimizers also experience an increase in their loss at the beginning of the transfer learning, but this is a much lower increase than SGD and it decreases over fewer training steps. The low learning rate for the models with the adaptive algorithms causes them to use more steps to converge. The adaptive optimizers RMSProp in Figure 5.3(c) and Adam in Figure 5.3(d) have very similar cross-entropy loss plots, while Adagrad in Figure 5.3(b) displays a higher and slower declining loss, which is also reflected in a lower Dice score in Table 5.1, this might be due to Adagrad being the least adaptive of the three adaptive optimizers. As explained in Section 3.5 Adagrad uses more minor updates and one of the drawbacks is the rapidly decreasing learning rates as a result of this. RMSProp and Adam are built to address this issue by dividing the learning rate by average squared gradients.



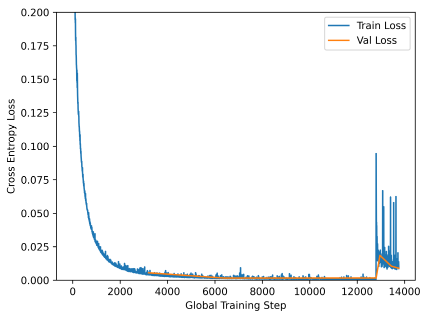
(a) SGD.



(b) Adagrad.



(c) RMSProp.



(d) Adam.

Figure 5.3.: Plots of the loss functions for the models with the highest Dice score for the different optimizers, the models marked in blue in Table 5.1.

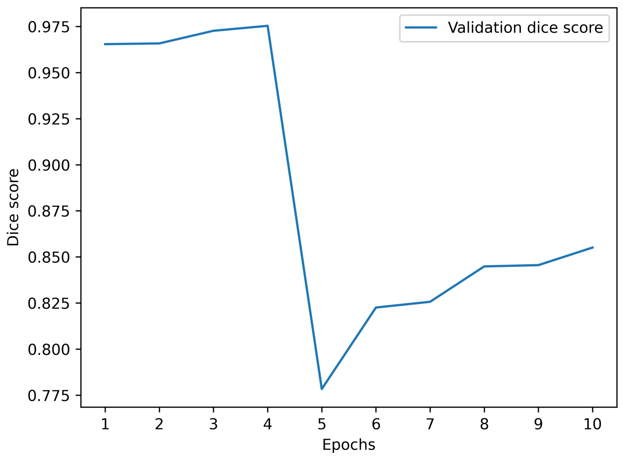
5.3. Different Batch Sizes

As discussed in Section 3.3 smaller batch sizes often converge quickly toward a good, but not always the best, solution. To study this phenomenon the models reaching the best Dice scores when studying the different optimizers were run with batch sizes of 2, 4 and 16. The Dice scores of these models are presented in Table 5.2. From the table, where the best models for each optimizer are marked in blue, it is clear that there was a correlation between smaller batch sizes and higher Dice scores. For all the optimizers, the model with a batch size of 2 reached the highest Dice score, confirming that the models with the smallest batch size achieved the most accurate predictions.

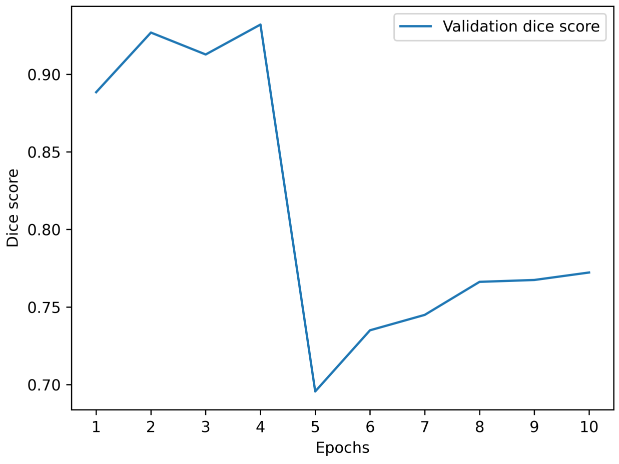
Optimizer	Learning Rate	Batch Size	Epochs	Dice Score
SGD	0.1	2	$4, 6$	0.8443
(DL) SGD	0.1	2	$4, 6$	0.8176
SGD	0.1	4	$4, 6$	0.8057
SGD	0.1	16	$4, 6$	0.7074
Adagrad	0.001	2	$4, 6$	0.8551
(DL) Adagrad	0.001	2	$4, 6$	0.8421
Adagrad	0.001	4	$4, 6$	0.8328
Adagrad	0.001	16	$4, 6$	0.7742
RMSProp	0.0001	2	$4, 6$	0.8637
(DL) RMSProp	0.0001	2	$4, 6$	0.8609
RMSProp	0.0001	4	$4, 6$	0.8551
RMSProp	0.0001	16	$4, 6$	0.8269
Adam	$0.0003, 0.0001$	2	$4, 6$	0.8541
(DL) Adam	$0.0003, 0.0001$	2	$4, 6$	0.8508
Adam	$0.0003, 0.0001$	4	$4, 6$	0.8426
Adam	$0.0003, 0.0001$	16	$4, 6$	0.8120

Table 5.2.: Dice scores for models where the hyperparameters of epochs and learning rates are optimal, and the batch size vary between 2, 4 and 16. The models reaching the highest Dice score for their respective optimizers are outlined in blue. The gray rows where the optimizers are marked with (DL) use Dice loss as their loss function.

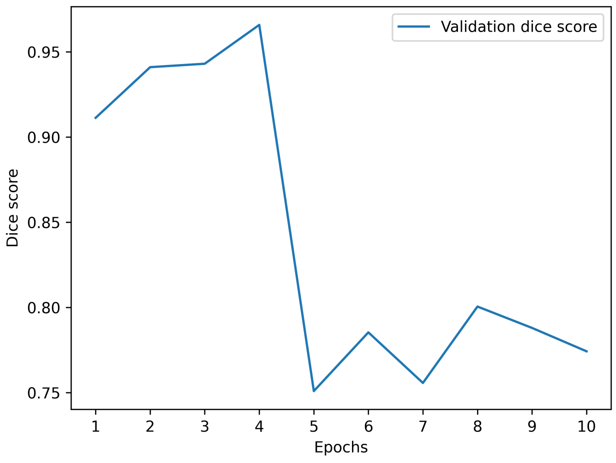
Figure 5.4 illustrate the Dice scores during the training of models, with Adagrad as optimizer and varying batch sizes, for each epoch. All models reach a similarly high Dice score during the initial training process, the first 4 epochs, though the model with batch size 2 starts with the highest score. The Dice score plots gave results based on their current validation set, similarly to the cross-entropy loss plots already mentioned. Meaning that the first 4 epochs used the simulated dataset for validation, and the last 6 used the real-world dataset for validation. The models differed when the transfer learning was initiated. All models experienced a significant drop in Dice score, and the models with smaller batch sizes continued to increase in Dice score after the fall. The model with a batch size of 16, seen in Figure 5.4(c) increased for one epoch before decreasing again in epoch number 7. The models with smaller batch sizes only experienced a less rapid increase at epoch number 7 and continued to improve before they landed at a high Dice score. On the other hand, the model with the batch size of 16 continued to differ from the others. After the sequential increase and decrease for 4 epochs, the model decreased at the last epoch to a significantly lower Dice score than the other models. This might be due to the weights being updated far fewer times, or poor generalization. Possibly as a result of converging to a local optimum on the real-world dataset, which is one of the disadvantages of larger batch sizes as presented in Section 3.3.



(a) Bach Size 2.



(b) Bach Size 4.



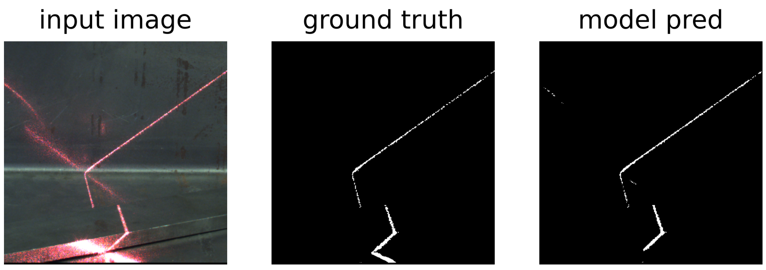
(c) Bach Size 16.

Figure 5.4.: Dice scores for models using Adagrad as their optimizer, epochs of 4 on the simulated dataset and 6 on the real-world dataset and with cross-entropy loss function, varying the batch size between 2, 4 and 16.

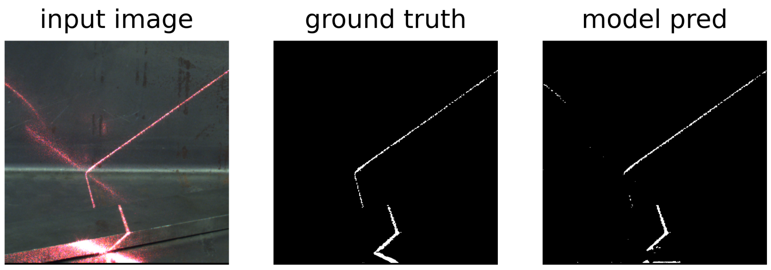
5.4. Different Loss Functions

Section 3.1 presented two different loss functions, cross-entropy loss and Dice loss. Cross-entropy was utilized as the loss function in the search for the best hyperparameters, with the weighting set at 0.7 on positive masks and 0.3 on negative masks. After the optimal parameters were found, the models were run with Dice-loss as loss function to compare the results. These results can be found in Table 5.2.

The models outlined in gray in Table 5.2 are models that used Dice loss as their loss function. They achieved a slightly lower Dice score compared to the models with identical hyperparameters, except that they used cross-entropy as their loss function, outlined in blue in the table. The difference in Dice score varies between the different optimizers. The difference was slightest for the models with RMSProp as optimizer algorithm, as small as 0.0028. The most significant difference appears for the models with SGD as optimizer with a difference of 0.0267, which is almost ten times bigger than the slightest difference. Figure 5.5 illustrates the comparison between the model predictions for the two models with the most significant difference. Even though they have the biggest difference, the model predictions only present a marginal difference to the human eye.



(a) Cross-Entropy Loss.



(b) Dice Loss.

Figure 5.5.: Comparison of the input image, the ground truth and the model predictions of two models using SGD as their optimizer algorithm, batch size of 2, trained on 4 epochs on the simulated dataset and 6 on the real world-dataset, but with either cross-entropy or Dice as their loss function.

5.5. Transfer Learning on Different Materials

The real-world dataset consisted of images of both aluminium and steel profiles, as explained in Section 4.1.2. Several models were made to study how the transfer learning was affected by the difference in reflections due to the different surfaces of the materials. Models were initially trained and validated on the simulated dataset before being transfer learned on a real-world image dataset, the results of these models can be found in Table 5.3. The models were made with the optimized hyperparameters found to be a batch size of 2, epochs of 4 and 6, cross-entropy as loss function and either RMSProp or Adam as optimizer algorithm.

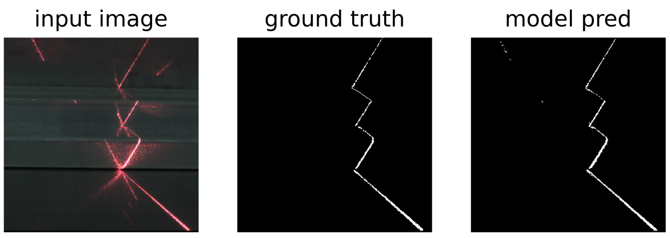
Optimizer	Training	Validation	Dice Score
RMSProp	Both	Aluminium	0.8361
RMSProp	Both	Steel	0.8861
Adam	Both	Aluminium	0.8317
Adam	Both	Steel	0.8985
RMSProp	Aluminium	Both	0.8332
RMSProp	Aluminium	Steel	0.8430
RMSProp	Steel	Both	0.8468
RMSProp	Steel	Aluminium	0.8041

Table 5.3.: Dice scores for models where the transfer learning was executed using different combinations of aluminium, steel or both in training and validation. All the models were made using a batch size of 2, epochs of 4 and 6 and cross-entropy as their loss function. The model reaching the highest Dice score is marked in blue.

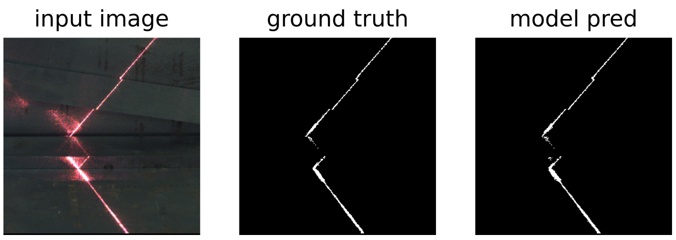
During the transfer learning the first four models in Table 5.3 were trained on both aluminium and steel, but only validated against one of them. From the table, it is clear that the model validated on steel reached the highest Dice score, for both models with Adam and RMSProp as optimizers. The model with Adam, marked in blue in the table, achieved the highest Dice score, as high as 0.8965. Adam was the optimal optimizer when the models were validated on steel with a Dice score of 0.0124 higher than the model with RMSProp. When the models were validated on aluminium, the model with RMSProp as optimizer reached a slightly higher Dice score, 0.0044 higher, than the model with Adam as optimizer.

From Table 5.3 it is clear that the models were more suited to handle diffuse reflections, which often occur in steel, while it was a tougher challenge to handle the specular reflections that aluminium is known for, which are presented in Section 4.1.2. Though the models achieved a higher Dice score when validated on steel, the difference between the Dice scores when validated on steel or aluminium was much lower than expected, only 0.0668. This was confirmed by Figure 5.6 which shows the comparison of the input image, ground truth and model prediction of a model validated on aluminium 5.6(a) and steel 5.6(b). The prediction on steel does not appear significantly superior to the prediction on aluminium, at least not to the human eye.

Models were also made by only training on either aluminium or steel, their results are



(a) Aluminium.



(b) Steel.

Figure 5.6.: The input image, the ground truth and the model predictions of two models with Adam as optimizer, where the transfer learning was trained on both aluminium and steel, but validated against either aluminium or steel.

presented in the last four rows in 5.3. Surprisingly, all Dice scores were above 0.80. It would be reasonable to expect that the models validated on steel would perform well due to the reflections produced by the nature of the material. However, the model trained on steel and validated on aluminium had a Dice score of 0.8041, which is more than expected. This reassured the robustness of the models produced by the initial learning on the simulated dataset, which is discussed further in Section 6.2.

5.6. Different Dataset Compositions

As mentioned in Section 4.1.2 it is a time-consuming job to take real-world images and generate a ground truth for every image in the dataset. It is more efficient to make a large quantity of simulated images. Therefore, investigating how many real-world images are needed to get satisfactory predictions from the network is highly relevant. Several models were trained using different amounts of simulated images and real-world images. The simulated dataset was trained using 1000 or 3200 unique images, while the real-world dataset was trained using 0, 20, 60, 100 and 160 unique images for training. These results can be found in Table 5.4. For these models, the batch size was set to 2. When training on the simulated dataset, the learning rate was set to 0.0003 over 4 epochs. At the same time, the learning rate after transferring to the real-world dataset was set to 0.0001 over 6 epochs. This does not include the models with 0 real-world images used for training, these had no training on the real-world images, only validation. Every model used cross-entropy loss, except the model marked in gray in Table 5.4 which used Dice loss.

Optimizer	Simulated images used for training	Real-world images used for training	Dice Score
RMSProp	3200	0	0.3807
Adam	3200	0	0.3147
(DL) Adam	3200	0	0.2342
Adam	3200	160	0.8541
Adam	3200	100	0.8376
Adam	3200	60	0.8223
Adam	3200	20	0.7842
Adam	1000	100	0.8152
Adam	1000	60	0.8135
Adam	1000	20	0.7818

Table 5.4.: Dice scores on the real-world dataset for models, with optimal hyperparameters, where both the number of images in the simulated dataset and the real-world dataset vary during training.

Only using the simulated dataset for training gave very poor results for validation on real-world images. Surprisingly using Dice-loss as the loss function and Adam as optimizer gave the worst performance, only achieving a Dice score of 0.2342 as seen in Table 5.4. The two are commonly seen coupled for deep neural networks. All other models in the table use cross-entropy loss. The best performance was when using RMSProp as optimizer, which resulted in a Dice score of 0.3807 with a sensitivity of only 0.1859 after the fourth and last epoch of training. These results were much lower than initially anticipated. The model struggled to differentiate weld lines and reflections. The model using Adam as optimizer was not far behind, with a Dice score of 0.3147. However, as seen in Figure 5.7 Adam reached a Dice score of 0.5875 on the second epoch. This was with a sensitivity of 0.7767. Notice that this specific figure plots the Dice score for both the simulated and the real-world images. The model was only trained on the simulated dataset with 3200 images and after every epoch of training the model was tested on a validation set of the simulated images, shown in blue, and tested on a validation set of the real-world images, shown in orange. The validation score of the simulated images can be used as a good indication of the model's improvements during its training. Since the thesis aims to get an insight into how the models perform on real-world images, the validation score of the real-world images is of most interest. Figure 5.7 shows that the model performed very well on its own validation set. However, the model struggled on the real-world dataset and only got a Dice score of half of the simulated dataset score. The fact that the Dice score improves for every epoch on the simulated dataset while it deteriorates on the real-world dataset is further discussed in Section 6.2.6.

Models trained solely on the simulated images gave poor results on real-world images, but not many real-world images were needed for the transfer learning to get satisfying results. Already when a model transfer learned on 20 real-world images after initial 3200

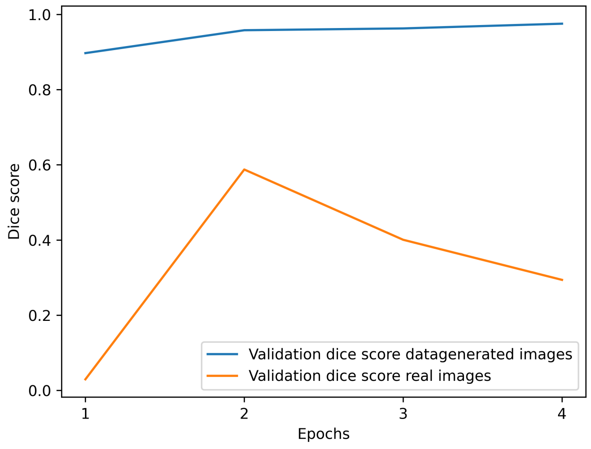


Figure 5.7.: Dice scores for validation on both simulated and real-world dataset during training on the simulated dataset.

simulated images the Dice score made a big leap up to 0.7842, a doubling of the Dice score. As seen in Figure 5.8 the model makes good predictions throughout the entire weld line with some minor false positive predictions on reflections with particularly high intensity. From here, the results from Table 5.4 steadily improved as models were trained on more and more real-world images. The best reaching a Dice score of 0.8541 when trained on 160 images. There were almost no false positive predictions of reflections at such a high Dice score, and many of the wrongly predicted pixels were from imperfect edge detection of the actual weld line. Either a couple of pixels too wide or too narrow compared to the ground truth of the weld line. These individual pixels and their effect on the numerical results are further discussed in Section 6.2.1. The difference between 20 and 160 images was 0.0699. This difference was smaller than expected and raised the question of how accurate predictions the models have to make to be implemented in the industry. This topic is considered in Section 6.3.

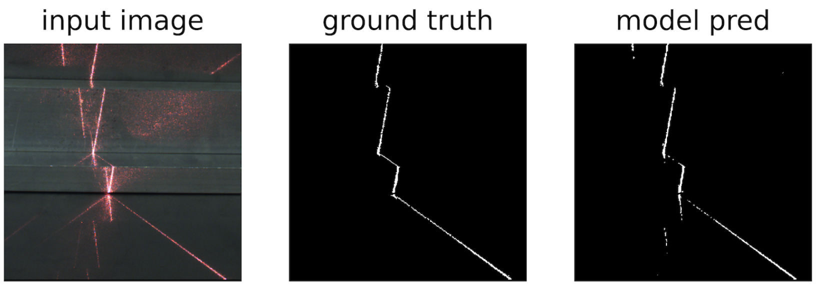


Figure 5.8.: Model prediction after training on 3200 simulated images and 20 real-world images, with optimal hyperparameters.

Training on fewer simulated images got a lower Dice score as expected, but the difference was small. Initially training on 3200 versus 1000 simulated images with transfer learning on 60 real-world images, gave a Dice score of respectively 0.8223 versus 0.8135. When using 100 real-world images, the results were 0.8376 for 3200 simulated and 0.8152 for 1000 simulated. The visual comparison revealed that all four models made good predictions on weld line and specular reflections. The slight difference was on diffuse reflections, where the models with the smaller simulated dataset had a few more false positives around the lightest intense corners. These results might show an upper limit of how many images are necessary after reaching a certain level of precision. The size of the datasets and their effect on the models' performance are discussed in Section 6.2.6.

Chapter 6.

Discussion

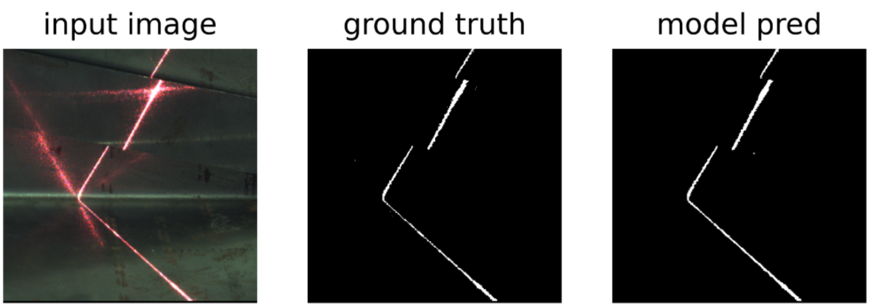
This section will present the thesis' expected results, and analyze and discuss the actual results. Furthermore, the potential errors, the suitability for industry implementation and future work that would be a natural continuation of the work already done in this thesis are all discussed.

6.1. Expectations

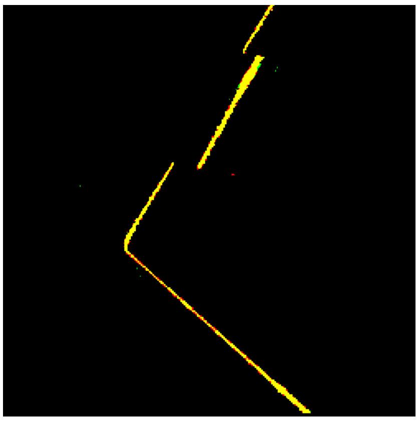
Based on the conclusion in the specialization project [20] and the fact that Adam is the most adaptive optimization algorithm, it would be reasonable to assume that Adam would be the optimal optimizer. By examining recent trends in the machine learning community, one would consider that models with Adam as optimizer combined with Dice loss as loss function would be the highest achieving models. Section 4.1.2 presents the difference in surfaces between steel and aluminum. Based on their differences, an expectation arose that the models would perform better when removing the diffuse reflections on steel than the specular reflections on aluminum. The differences between the simulated images and the real-world images decreased the expectations of the overall performance of the models. Especially the expectation that the models would perform well with a limited amount of real-world images included in the training process decreased, assuming that the models would require all 200 images in the real-world dataset.

The overall expectations of the transfer learning were that it would be mediocrely successful, that the predictions made by the models would precisely remove over half of the reflections, but not much more. In addition, the transfer learning on the real-world dataset was expected to require a lower learning rate than the initial learning performed on the simulated dataset. In the future work presented in the specialization project [20] the implementation of both Dice score and Dice loss into the convolutional neural network was proposed and expected to be completed as a part of the thesis. The future work of [20] also included accessing more processing power by connecting to the Idun cluster, a process expected to require both time and effort.

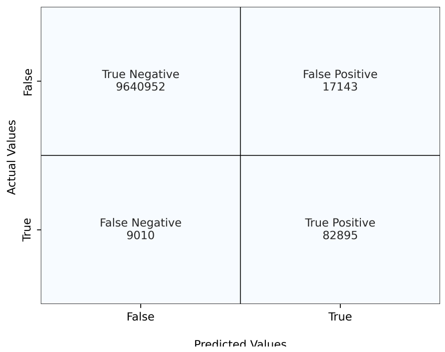
6.2. Comparing Results



(a) Comparison of the input image, the ground truth and the model prediction.



(b) The model prediction with the different possible outcomes of classification in different colors.



(c) Confusion Matrix for the entire validation set.

Figure 6.1.: Visual representation of a prediction made by the best performing model.

The model with the best performance used RMSProp as optimizer, cross-entropy as loss function, a batch size of 2, and a learning rate of 0.0001 for both the simulated and real-world dataset. As a final result, it got 0.8637 in Dice score, 0.9971 in pixel accuracy, 0.9019 in sensitivity, and 0.9982 in specificity. Overall the results were beyond expectations, an example of the predictions can be seen in Figure 6.1. It was quite interesting that RMSProp had the best performance, as it was thought Adam would perform the best out of the optimizers, this will be further discussed in Section 6.2.2. It is worth noting that, while this was the model that performed the best, numerous other models performed nearly as well. In fact six models got a Dice score above 0.85 when training with different optimizers, see Table A.2. Furthermore, three more models reached a Dice score within only 0.01 lower than the best. These were Adagrad with cross-entropy, Adam with cross-entropy, and RMSProp with Dice loss. All four models were quite similar when making predictions. The fact that all used different optimizers shows how different combinations of hyperparameters can be used to reach a global optimum. The robustness of the U-net architecture might be a factor that contributed to so many successful models. Because of the solid basis provided by U-Net, the models were able to train successfully, perhaps even when some hyperparameters were not optimal.

A model must handle both specular and diffuse reflections to achieve good predictions. As mentioned in Section 2.5.1, specular reflections are sharp and look similar to the original weld line, while the diffuse reflections light up a larger area around. Overall, the models performed slightly better on diffuse reflections than on specular reflections on both the simulated and the real-world datasets. Some of the specular reflection lines, particularly in the simulated dataset, had a striking resemblance to the weld line. Perhaps forcing the network to a more significant degree differentiate the weld line and reflections by the angles and connection of the lines to other lines in the image. One explanation could be that diffuse reflections generally had a lower intensity than the actual weld line. The true weld line was more visible, while the diffuse reflections were larger and less intense. Since the local differences are a function of the surrounding pixels, they are well suited for convolutional filtering. However, in some images, the diffuse reflections could have a powerful intensity, resulting in a large portion being bright red. An example of this can be seen in the blue square on the bottom of Figure 6.2. This particular model only obtained a Dice score of 0.6109. When a diffuse reflection had such high intensity as in the figure, it was difficult for the models to predict what part of it was an actual weld line.

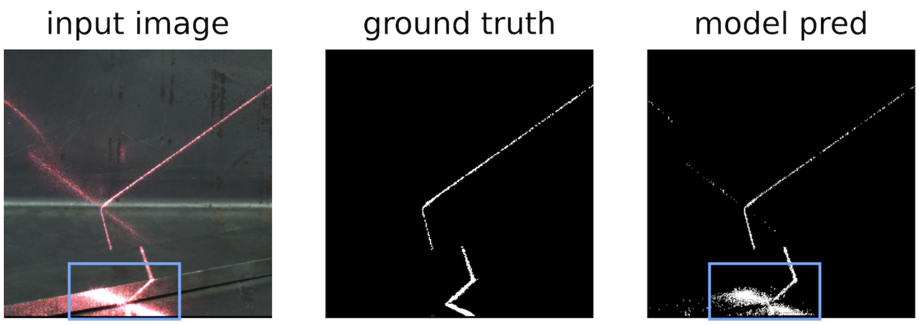


Figure 6.2.: Example of a model struggling to make predictions on very intense diffuse reflections in the bottom of an image, in the blue square. The model used cross-entropy loss as loss function, SGD as optimizer, a learning rate of 0.001 and batch size 4.

The reason behind why models performed so well in general, could be due to the large simulated training set. The models were exposed to many different scenarios by having many images, including specular and diffuse reflections on different kinds of metals, different laser intensities, and the angle at which the laser hit the surfaces. This was one of the many benefits of initial training on a simulated dataset before transfer learning to a smaller real-world dataset. Since U-net is so advanced it can learn incredibly fast, however this also makes it prone to overfitting. This was observed on a few of the initial training runs for the thesis before lowering the number of epochs, as mentioned in Section 5.1. If U-net is trained on too few images over too many epochs, it could easily memorize individual images instead of learning to generalize. By automating the process of generating random and unique images, it is possible to achieve a far larger dataset than a real-world dataset, and in a shorter time than it would take to do it by hand. The simulations enable the use of such a large dataset that it is possible to use very few epochs while still getting in enough training steps for the models. Another benefit of using a simulated dataset is minimizing possible errors on the ground truths since this is decided by true values and not on eyesight by humans. This is further discussed in Section 6.2.7.

As expected, connecting to Idun 4.2.1 proved to be a challenge. There was no problem connecting to the network drive to upload the necessary datasets and code. The tutorials for this on the NTNU web pages were thorough and well-written. However, the next step was unclear, specifically how the slurm-file should be written to run the code on the cluster. The tutorials were perceived as sporadic, only giving partial instructions for what was needed to run on the cluster and how to write the required slurm-file. As a result, the thesis used a combination of the NTNU web pages and a specialization project with an in-depth guide for connecting to and running code on Idun written by NTNU student Jonas Strand Aasberg [1]. See Appendix B for a short guide on how to access the cluster and how to write such slurm files.

6.2.1. Accuracy Metrics

The accuracy metrics obtained in this thesis were Dice score, pixel accuracy, sensitivity and specificity. As explained in Section 3.6, they all have a unique way of evaluating a model. Dice score was chosen as the primary accuracy metric to evaluate models. In the preceding specialization project [20], pixel accuracy was the only accuracy metric used to evaluate models trained on a dataset. However, it was argued that pixel accuracy alone was insufficient after evaluating the empirical data. Because the weld line is such a small part of an image, losing a few pixels or even the complete weld line could still result in a good pixel accuracy. This may give a misleading impression of a model's ability, as predicting partial weld lines is of no aid. When a model already has a good pixel accuracy result, an improvement will only increase the pixel accuracy percentage by a minimal amount. As a result, the significance of the alterations made to a model may be underestimated. As noted in the previous Section 6.2, the four best performing models were within 0.01 of each other in Dice score. These four models can be found in Table 6.1 with Dice score, pixel accuracy, sensitivity, and specificity. The models all used batch size 2, over 4 epochs on the simulated dataset and 6 epochs on the real-world dataset, but different optimizers with their optimal learning rate. The one marked in gray used Dice loss, while the rest used cross-entropy loss.

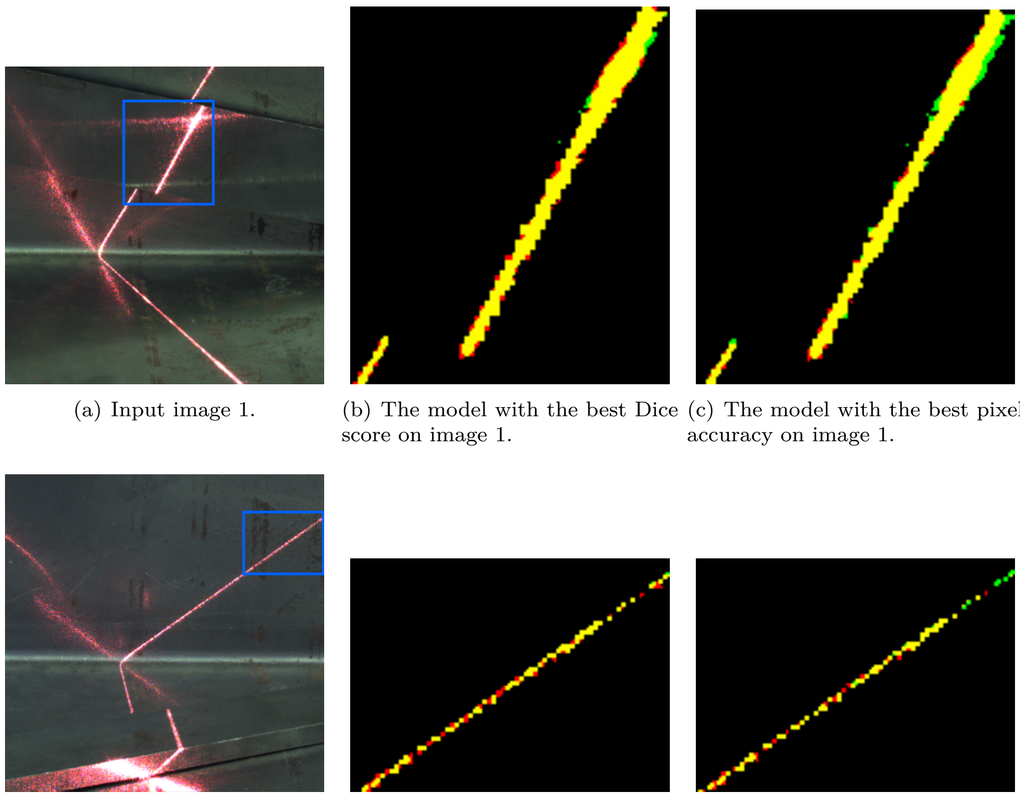
Optimizer	Dice score	Pixel Accuracy	Sensitivity	Specificity
RMSProp	0.8637	0.9973	0.9019	0.9982
(DL) RMSProp	0.8609	0.9974	0.8477	0.9988
Adagrad	0.8550	0.9971	0.8953	0.9981
Adam	0.8541	0.9971	0.8943	0.9980

Table 6.1.: The four best performing models and their results on the real-world validation set.

In comparison to the Dice score, the pixel accuracy for the four models only deviated by 0.003. It is also important to remark that the order would have changed if the models were ranged based on pixel accuracy instead of the Dice score. The model with RMSProp and Dice loss got a lower Dice score, but higher pixel accuracy, than the model with RMSProp and cross-entropy loss. The main distinction between pixel accuracy and Dice score is that pixel accuracy considers true negatives, whereas Dice score does not. The fact that the models performed well on the two different metrics raised the question of how the models made their predictions. Sensitivity and specificity provided further insight. A higher Dice score correlated with higher sensitivity, the true positive rate. In contrast, a higher pixel accuracy correlated with higher specificity, the true negative rate. As expected, sensitivity varied more than specificity since there are far fewer positive masks than negative masks in the dataset. Due to the modest range in Dice score and pixel accuracy outcomes, a visual inspection had to be done to detect any significant differences between the approaches. There was no distinct difference in predicting the real-world dataset reflections. When comparing the models based on the different accuracy metrics, it was expected that there would be a larger variation between them. Incorrect edge detection accounted for a considerable portion of the error. Small mistakes like this were found in all models. To highlight the difference, the predictions made by the two models were magnified and are shown in Figure 6.3. The top row consists of two different models making predictions on the same input image. The bottom row consists of the same two models making predictions on a second input image. The model to the left reached the highest Dice score in this thesis and is marked in blue in Table 6.1. The model to the right reached the second-highest Dice score, but the highest pixel accuracy and is marked in gray in Table 6.1. To give an understanding of how magnified the images are: Figure 6.3(b) is a magnified part of Figure 6.1.

As seen in Figure 6.3(b) and 6.3(e), the model with the best Dice score and second-best pixel accuracy had some false positives (red), predicting the weld line to be thicker than it was. This is especially clear in Figure 6.3(e), where the weld line was particularly narrow and with minor gaps. However, it has almost no false negatives. The model with the second-best Dice score and the best pixel accuracy seen in Figure 6.3(c) and 6.3(f) has very few individual false positives, but does have some larger areas that are false negatives (green). In Figure 6.3(f) it misses the last part of the weld line in the top right corner of the image. Both models performed exceedingly well, and choosing a single one would be difficult since they have different strengths and weaknesses. The question arose whether a complete, but too thick weld line should be prioritized over a thinner weld line with minor gaps. This must be considered for each unique task when used in the industry.

Overall the models performed better than expected. With so many well-performing models, it was essential to differentiate them with multiple accuracy metrics. Sensitivity and specificity gave a clearer insight into the performance of the models after initially inspecting the Dice scores. However, as previously stated, visual inspection was required to distinguish how the best-performing models made their predictions. For further clarification, mean subpixel error or pixel outlier fraction could have been implemented as accuracy metrics. Since wrong predictions of individual pixels on the weld line edge might not affect the actual welding as much as wrong predictions of pixels far away.



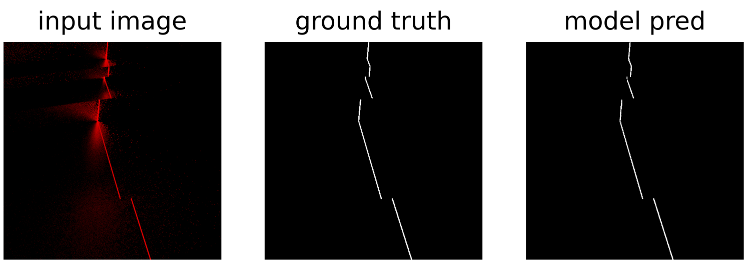
(a) Input image 1. (b) The model with the best Dice score on image 1. (c) The model with the best pixel accuracy on image 1.

(d) Input image 2. (e) The model with the best Dice score on image 2. (f) The model with the best pixel accuracy on image 2.

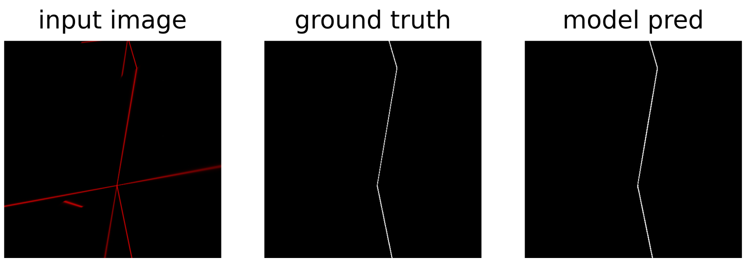
Figure 6.3.: Magnified visual representation of the predictions made by the two best performing models, on two different images to illustrate how individual pixels of the weld line are predicted. Yellow are true positive, red are false positive, and green are false negative of the weld line.

6.2.2. Optimizers

The specialization project [20] discovered results in line with a study performed by Wilson, Roelofs, Stern, Srebro and Recht at Berkeley University of California and Toyota Technological Institute in Chicago in 2017 [60], presented in Section 3.5. The models with non-adaptive algorithms such as SGD may generate as precise, sometimes more accurate, results as models with adaptive algorithms such as RMSProp and Adam. A similar discovery appears in this thesis as well, in the initial learning stage. The best scoring model with SGD as optimizer had a Dice score of 0.9728 at the end of the last training epoch on the simulated dataset. In comparison, the best scoring model with Adam as optimizer had a Dice score of 0.9767, which gives the adaptive optimizer only a 0.0039 higher score. In Figure 6.4 it is clear that both the model with SGD 6.4(a) and the model with Adam 6.4(b) produce precise predictions without clearly visible difference between their preciseness. The lack of difference agrees with the conclusions reached by both [20] and [60] that models can produce as precise predictions with SGD as with Adam as their optimizer, at least in the initial learning executed on the simulated dataset.



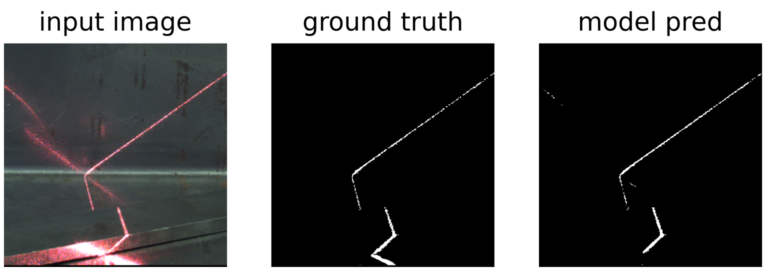
(a) Initial Learning - SGD.



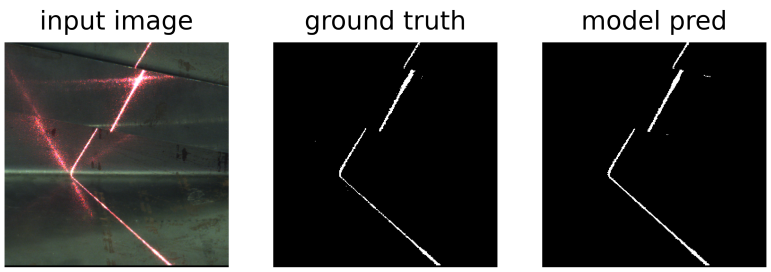
(b) Initial Learning - Adam.

Figure 6.4.: Comparison of a simulated input image, the ground truth and the model predictions of the best scoring models for optimizers SGD and Adam at the end of the last epoch of initial learning.

However, the difference between the non-adaptive and adaptive optimizers became more significant when transfer learning was conducted. At the end of the last epoch of transfer learning, the model with SGD had a Dice score of 0.8443 while the model with Adam had 0.8541, which was a difference of 0.0098. This was twice as high as the difference at the end of the initial learning. Illustrating an advantage of using an adaptive optimizer such as Adam when transfer learning, even though the differences were minor overall. The difference is visually presented in Figure 6.5, where the prediction made by the model with Adam in Figure 6.5(b) is more precise than the one made by the model with SGD in Figure 6.5(a), even though the differences are less prominent than expected, which coincides with their differences in Dice score.



(a) Transfer Learning - SGD.

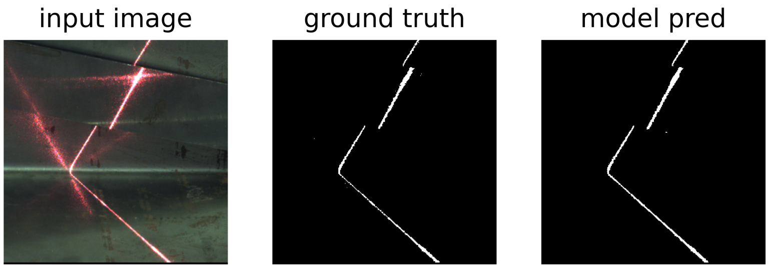


(b) Transfer Learning - Adam.

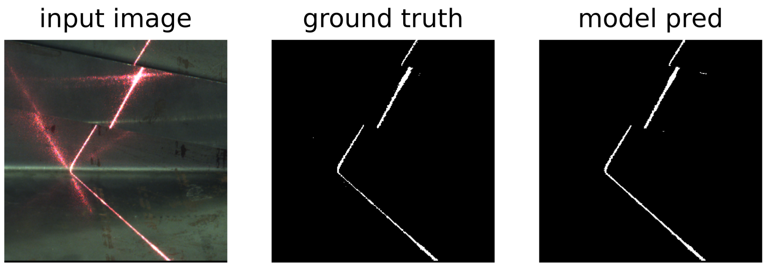
Figure 6.5.: Comparison of the input image, the ground truth and the model predictions of the best scoring models for optimizers SGD and Adam at the end of the last epoch of transfer learning.

New studies similar to this thesis underline that adaptive optimizers are more suited for transfer learning. Two different studies were published in 2021, the classification of tomato leaf disease [54], and volcanic rocks [41] from images using transfer learning with convolutional neural networks. Both studies reached the same conclusion when comparing SGD, RMSProp and Adam as optimizers in their networks. Their numbers were slightly different, but they both concluded that Adam was the optimal optimizer, with RMSProp following closely behind and SGD coming in last, achieving a lower accuracy than the adaptive optimizers. The study of tomato leaf disease experienced a more considerable difference between the two adaptive optimizers than the study of volcanic rocks. This difference might be due to the different convolutional neural networks used in the two studies. The tomato leaf disease study used Modified-Xception deep neural network, while the study of the volcanic rocks used Dense-Net121 and ResNet50. In addition, they used different functions to calculate accuracy, but neither used Dice score. Both studies present a very little difference between RMSProp and Adam, the accuracy differences were as small as 0.0054 and 0.0036. These results are in line with the results presented in Section 5.2, only that the difference was the other way around, with RMSProp achieving the highest Dice score, and a slightly higher pixel accuracy as well, as seen in Table 6.1.

Based on the specialization project and literature concerning both U-Net and transfer learning, for instance, the two studies presented in the previous paragraph, an expectation for Adam to be the optimal optimizer was natural. Yet, the results in this study contradicted this expectation. A possible reason for this contradiction might be that the two studies presented used different networks than U-Net. Most studies of transfer learning with U-Net only use Adam as optimizer. An interesting observation is that the results in this thesis have a slightly higher difference in Dice score than in pixel accuracy between RMSProp and Adam. RMSProp has a 0.0096 higher Dice score than Adam. However, when comparing the models' pixel accuracy, the difference is only 0.0002 in favor of RMSProp, which is even smaller than the accuracy differences detected in the two studies [54] and [41]. Comparing the severely low pixel accuracy difference with the higher Dice score difference is a reminder of the importance of the precision measuring method. This might argue that both RMSProp and Adam are suitable for transfer learning with deep convolutional neural networks. Figure 6.6 illustrate model predictions made on the same real-world image by a model with RMSProp in Figure 6.6(a) and a model with Adam in Figure 6.6(b). The visual differences between the model predictions are minimal, even though they have a Dice score difference of 0.0096 in favor of RMSProp.



(a) RMSProp.



(b) Adam.

Figure 6.6.: Comparison of a real-world input image, the ground truth and the model predictions of the two best scoring models using RMSProp or Adam as their optimizer algorithm.

Overall the differences in both Dice scores presented in Table 5.1 and the visual model predictions in Figures 6.5 and 6.6 are less significant than expected, which could be a confirmation of the robustness of the network. A likely reason that a lot the models made good predictions regardless of their optimizer and learning rate is the size of the simulated dataset used in initial learning. With a dataset of 4000 images, all the models could adjust their weights well before they proceeded to the transfer learning. Unexpectedly lowering the learning rate from the simulated dataset to the real-world dataset had little effect on the results. This may be because the changes in learning rate were to subtle. Therefor a lot of models were trained with equal learning rates in initial and transfer learning.

6.2.3. Batch Size

Section 5.3 presents how the models with a batch size of 2 achieved the highest Dice score compared to batch sizes 4 and 16, regardless of the optimizer used by the model. In addition, the Dice score decreased as the batch size increased, with the batch size of 16 achieving the lowest Dice scores. Smaller batch sizes allow for more updates per epoch and the possibility to learn and perform minor improvements more often and sooner than larger batch sizes, as explained in Section 3.3. This is possibly why the smallest batch size achieved the highest Dice score for transfer learning since the real-world dataset contains so few images. Additionally, the smaller batch sizes also allow for more random noise. The larger batch size is less exposed to noise and unpredictability, making it more vulnerable to local optimums.

The relatively small real-world dataset can explain why the model with the smallest batch size is the highest achieving, as the model with the highest batch size would need more images to update enough times to reach a high Dice score. A possible action to ensure that a model with a larger batch size has enough updates would be to increase the number of epochs, but this would increase the possibility of more overfitting. To avoid this a possibility could be to incorporate a break in the learning when the loss function converges, this is further discussed in Section 6.4. Section 3.3 also motions that a bigger batch size might be more efficient since it can handle more of the training set at once, which comes to an advantage when the training set is sufficiently large, which seems not to be the case for the real-world training set of 160 images. On the other hand, the simulated training set is extensive, with its 3200 images, resulting in the model with the larger batch size reaching high Dice scores at the end of the initial training. Figure 6.7 illustrates the Dice scores for models with varying batch size at the end of initial learning and transfer learning. The Dice scores reached by the transfer learning were as expected, with the model with a batch size of 2 achieving the highest score and then a declining score when the models increased the batch size. At the end of the initial learning it would be reasonable to expect that the model with the largest batch size would have the highest Dice score. That is because the dataset could be large enough to perform enough training steps per epoch. At the same time, the use of more images per batch could adjust the models towards an optimum of a more significant part of the dataset for every training step. However, this is not the case, the model with batch size 4 reached the highest Dice score at the end of initial learning. Notably, the overall differences between the scores at the end of initial learning were slight, as seen in the figure, all scores were above 0.96, with batch size 4 scoring 0.013 higher than batch size 16. The slight difference in initial learning compared to transfer learning illustrate that the batch size has more influence when the size of the dataset is smaller.

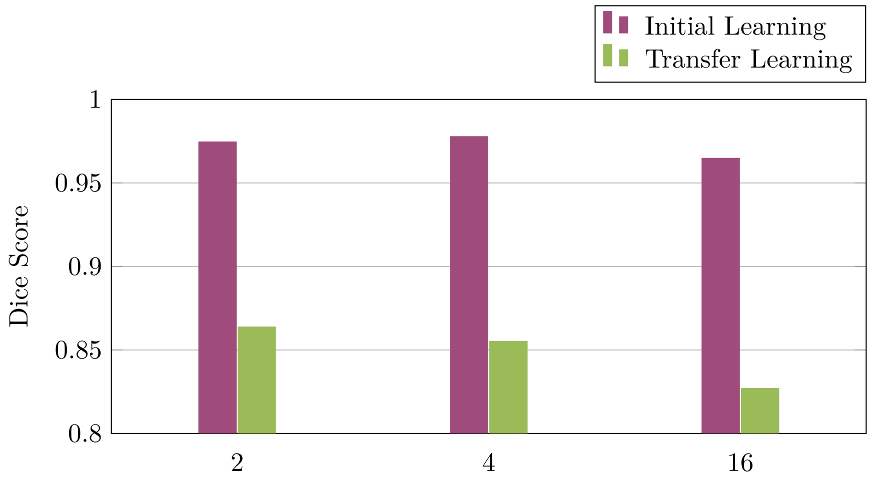


Figure 6.7.: Dice scores at the end of the initial learning and transfer learning for the best scoring model with RMSProp as optimizer and varying batch sizes between 2, 4 and 16.

Table 5.2 in Section 5.3 lists all the Dice scores for models with different optimizers and with varying batch sizes. By reading the table from top to bottom, gradually from the non-adaptive optimizer SGD to the most adaptive optimizer Adam, it is observed that the increase in batch size had less impact on the adaptive optimizers. SGD had the biggest difference, 0.1368, while Adam and RMSProp had the smallest, 0.0397 and 0.0368. This is another example of how these adaptive optimizers are a good fit for transfer learning models.

Figure 6.8 illustrates the spikes in the loss function of a model with batch size 2. The spikes are a consequence of mini-batch learning with such a small batch size. The spikes result from some mini-batches containing noise and unfortunate, unpredictable training data. It is possible that lowering the learning rate further could have minimized the spikes. However, the plots are generally trending evenly, making batch size a likely cause of the spikes. Smoother plots for the models with an increased batch size confirmed this possibility. These spikes are even more prominent when using SGD. The model has to use larger batch sizes to reduce or remove the spikes. A significantly larger batch size would require substantially more processing power and probably more epochs as explained in Section 3.3.

6.2.4. Loss Functions

As presented in Section 3.1.2 Dice loss is known for handling imbalanced class distribution, which is prominent in the dataset used in this thesis. Dice loss was expected to provide better learning than cross-entropy loss because the class imbalance advantage combined with Dice loss allows for more responsive adjustments to the true positive predictions. Dice loss has been superior in handling class imbalance for many similar studies. A study published in Computerized Medical Imaging and Graphics in 2022 on generalizing

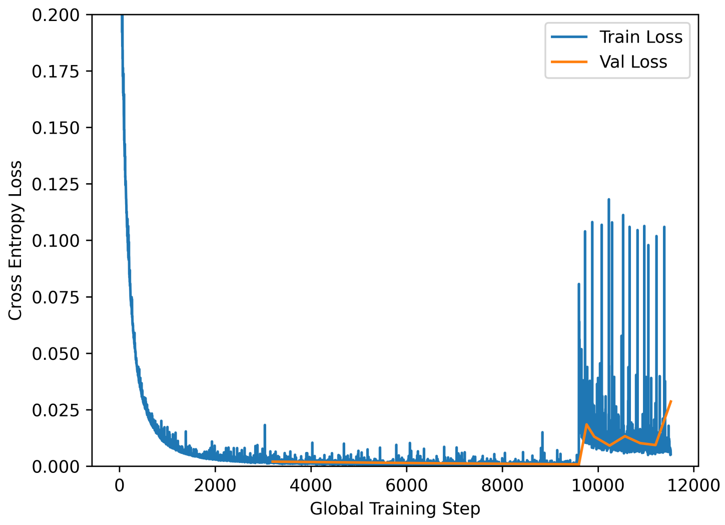
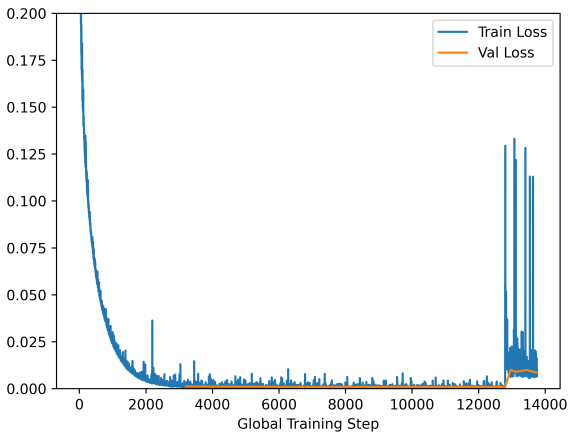


Figure 6.8.: Loss function plot illustrating the spikes in training loss of a model with batch size 2.

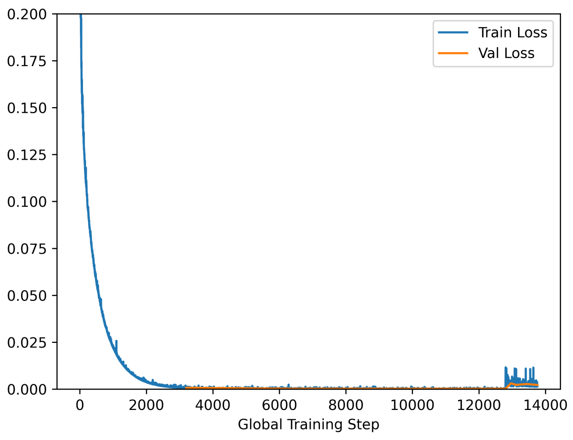
Dice and cross-entropy based losses to handle class imbalance [58] found that the Dice-based loss functions performed better with imbalanced data for all but one of their five experiments. This thesis is another exception where Dice loss is outperformed, as seen in Section 5.4. A possible reason for the deficient performance of Dice loss can be its vulnerability to hyperparameters. The different combinations tested may have caused the learning to move towards a local optimum and not being able to traverse out, instead of reaching the global optimum.

The study of class imbalance [58] found a clear visual difference between the segmentations generated by different loss functions. They observed an association between cross-entropy-based loss functions and a greater proportion of false negative predictions, which contradicts the observations from this thesis. Figure 6.3(c) and 6.3(f) in Section 6.2.1 use Dice loss and illustrate a greater proportion of false negative predictions than 6.3(b) and 6.3(e) which use cross-entropy loss and achieved the highest Dice score. This contradiction might be due to the heavy weighting ratio of classes used in the cross-entropy loss with 0.3 for the background and 0.7 for the weld line in this thesis. The heavy weighting could be a reason for cross-entropy performing better than Dice loss.

The validation loss, the orange function, is lower for the model with Dice loss as loss function, Figure 6.9(b), than for the model with cross-entropy loss, Figure 6.9(a). It was unexpected that Dice loss, with its lower loss, did not achieve the highest Dice score. Especially since Dice loss uses Dice score to calculate the loss, as seen in the mathematical composition of the Dice loss function in Section 3.1.2. This could be due to hyperparameter vulnerability, causing the model to fall into a local optimum from which it is unable to escape in order to reach the global optimum. If the model gets stuck in a local optimum, it gets harder to tune the weights continually. Another effect of this can be seen in Figure 6.9. The figure illustrates how the model with Dice loss in Figure 6.9(b)



(a) Cross-Entropy Loss.



(b) Dice Loss.

Figure 6.9.: Comparing the loss plots of two identical models with the single difference that one uses cross-entropy loss while the other uses Dice loss.

has notably fewer and less prominent peaks than the model with cross-entropy in both initial and transfer learning.

6.2.5. Effects of Different Materials

The Dice scores presented in Section 5.5 were much higher than expected. All models reached a Dice score over 0.831 when trained on both aluminium and steel before being validated on either material. This illustrated the robustness of the models established through the initial training. As previously mentioned, the expectation was that the different reflections on the different materials would have a greater impact. The models exceeded expectations when removing specular reflections, this is especially evident in the model trained on steel and validated on aluminium, reaching a Dice score of 0.8041.

Figure 6.10 show a prediction made by this model and how it successfully removed all the specular reflections.

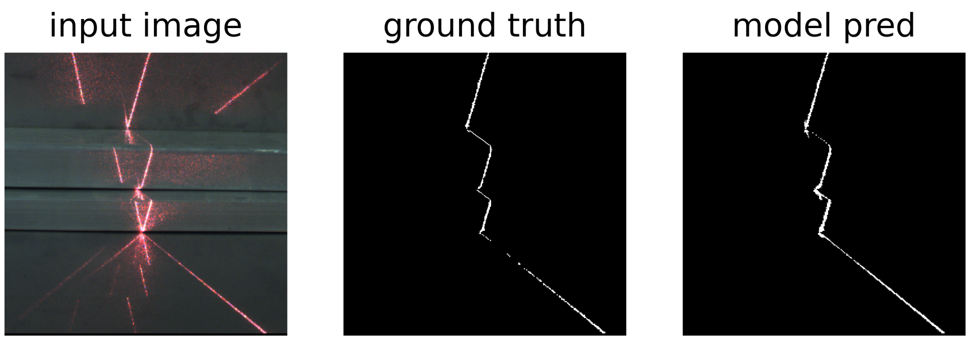
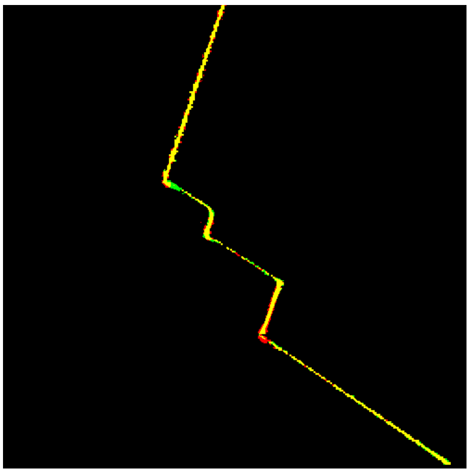


Figure 6.10.: Comparison of the input image, the ground truth and the model predictions of a model trained on aluminium and validated on steel during transfer learning.

Overall the models with RMSProp as their optimizer algorithm reached the best Dice score when hyperparameters were compared. However, when the models were tested on either aluminium or steel, as presented in Section 5.5, the results varied. When the models were validated on aluminium RMSProp achieved the best results, but when validated on steel Adam was superior, even though the differences were minor. The minor variations and the fact that the superior optimizer varied substantiated the arguments made in Section 6.2.2, that both Adam and RMSProp are suitable and well-performing optimizers for transfer learning tasks. Another reason for the variation may be the small validation dataset used. In order to validate the models on either steel or aluminium the validation set only contained 20 real-world images per material. The small validation dataset has the vulnerability that one difficult image could have a large impact on the total Dice score, altogether making the models more sensitive.

The observations contradicted expectations. The fact that the models had high Dice scores overall and the slight difference between the scores reached when validating on aluminium with specular reflections or steel with diffuse reflections can have several contributing factors. Firstly, the reflections produced on steel were less diffuse than expected, making the differences between the two materials less prominent. Secondly, the models seemed to generalize the difference between the actual weld line and the reflections in other directions, as mentioned in Section 6.2.1. The more eminent contributor is the pixels on the weld line. It seems that both aluminium and steel have relatively similar thickness and intensity of weld lines in addition to similar transitions from the weld line to the background. Therefore, it is probable that the U-Net models were able to produce equally precise model predictions regardless of the material being aluminium or steel. Figure 6.11 illustrates the similarities of the laser.


(a) Aluminium.

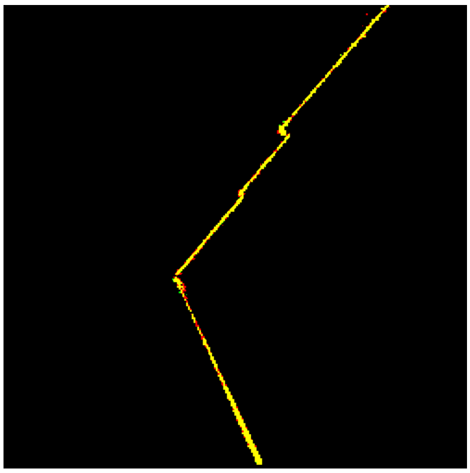

(b) Steel.

Figure 6.11.: Segmentation comparison of the predictions against the ground truths on aluminium and steel.

6.2.6. Effects of Dataset Composition

The results from Section 5.6 showed how using only simulated images for training gave poor predictions on real-world images. The loss function plot for the best model when only training on simulated images can be seen in Figure 6.12. This model only reached a Dice score of 0.3807. The figure contains the loss on the training and validation set similarly to the other loss plots throughout the thesis. In addition, this figure also incorporates the validation loss for the real-world images. The difference in validation loss on the simulated and real-world datasets provided information on how the Dice score could become so low for the model. It was expected that the two validation losses would be close to each other. However, the simulated validation loss stayed close to the training loss, and the real-world validation loss was far higher. Because the simulated training and validation loss had converged towards the end of the training, adding more epochs would be ineffective since the model had reached an optimum for this dataset. Another fact to support this thought can be seen from the Dice score plotted throughout the training for the same model. The figure was earlier shown in Section 5.6. Here, the Dice score of the real-word dataset peaks early before falling while the Dice score of the simulated dataset keeps climbing. Suggesting some critical learning points from the simulated dataset were not always applicable to the real-world dataset. This can be a consequence of the fact that the two datasets were so different and is further discussed in Section 6.2.7.

Already when the initial training was on 3200 images and transfer learning on as few as 20 real-world images, the performance of the models rose significantly. The fact that the models achieved good results after only transfer learning on 20 real-world images suggests that the models learned weld line patterns in the simulated dataset and could adjust the weights for properties more explicitly in the real-world dataset. Leaping from 0.3807 on zero real-world images to 0.7842 on 20 real-world images. As expected, the Dice score

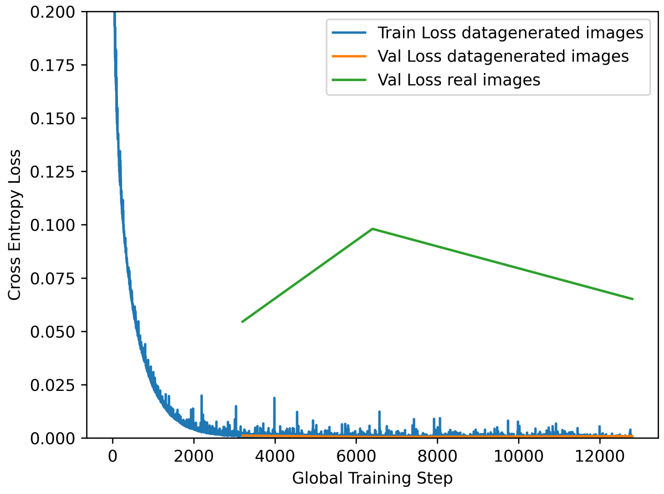


Figure 6.12.: Plot of the training and validation loss during training on the simulated dataset and the validation loss on the real-world dataset.

improved further by adding additional real-world images. A larger dataset is advantageous since the models can learn to generalize to a greater extent. This improvement kept going up, but slowed down after 60 real-world images scoring 0.8223, only 0.0312 lower than the model using 160 images which scored 0.8541. This is encouraging when looking at suitability for the industry since real-world images are time-consuming to produce. When the simulated dataset contained 1000 images and the real-world dataset contained 20 images, the Dice score was almost as high as the score reached by the model with 3200 simulated images. However, there was almost no improvement when adding a few more real-world images. Perhaps being a sign that using too few images in the simulated dataset limits the models understanding, since it has not experienced enough unique samples and scenarios.

6.2.7. Possible Errors

Machine learning should be trained on the same data that it will be working with to achieve the best outcomes. However, as earlier discussed, obtaining the necessary dataset can be both resource- and time-consuming. Transfer learning can be seen as a compromise, providing more samples to train on, though they are not specific for the task at hand. For this thesis, the simulated and real-world images were more different than expected. The simulated weld lines were approximately 3 pixels wide, while the real-world weld lines varied to a large degree and could be up to 12 pixels wide. The profiles used in the datasets had different modules, the notches and grooves, resulting in different reflection angles. Another aspect was the materials used, resulting in the reflections having a more extensive spread on the real-world images. These differences would have negatively affected the final result of the models since the weights would have different optimums.

The difference could be minimized by adjusting either of the datasets. For the simulated dataset, the parameters for the materials, such as roughness, could be adjusted. The laser thickness, or the size of the modules, could be set to match the real-world dataset better. For the real-world dataset, possible adjustments could be having different lighting in the room, changing camera settings, using a weaker laser, or performing some pre-processing on the images. Additionally, the camera could be fitted with a filter, known as red interference bandpass, only allowing the light of particular wavelengths equal to the laser through the lens. Modifying the simulated dataset could be more effective because all the data and corresponding ground truth could easily be located and managed.

As discussed in Section 6.2.1, many of the incorrectly predicted pixels came as a result of imperfect edge detection of the weld line. This was the case in both the simulated and real-world datasets. However, it was the most significant in the real-world dataset. One reason for the imperfect edge detection could be that the intensity of the laser was strongest in the middle and faded a little towards the edges. In this case, it might be possible to get a more precise cut edge by increasing the contrasts in pre-processing. However, there will always be some roughness and imperfections in natural materials, spreading a tiny amount of light around the weld line and its edges. Additionally, some of the incorrectly predicted pixels may result from poorly made ground truths for the real-world dataset. Since the ground truths were labeled by hand, there is always a possibility of minor differences in what masks a person would choose as positives or negatives. The ground truth is especially prone to faulty labeling when there is an intense diffuse reflection completely blocking the view of the weld line. In such circumstances, assumptions must be made since it is impossible to distinguish the actual weld line.

In this thesis, all images have a form of reflection, usually multiple. The first impression is that this is beneficial because the network can train on various scenarios with reflections. In the thesis, the models performed admirably. However, it may be possible that it could have a impact on the performance in industrial use. Not all images in real use will have reflections, but since the network is constantly trained to eliminate something from an image, the network may still want to predict some laser lines as negatives. This could lead to a higher rate of false negatives than seen in the thesis when utilized in the real world. In future work it would be interesting to compare the results of models validated on a dataset containing images both with and without reflections.

6.3. Suitability for Industry Implementation

One of the main points discussed in the suitability for industry implementation in the specialization project [20] that made the grounds for this thesis was the transition from a simulated to a real-world environment. This thesis solved this obstacle using transfer learning. The successful transfer learning exhibited in this thesis opens for more standardized and automatic use. The opportunity to use an already made model, including a large set of simulated images with corresponding ground truths, pre-tuned neural network architecture and a structured folder and file system, makes implementation possible and much more seamless. One crucial step to make the implementation seamless would be to automate the process of making ground truths. Section 4.1.2 explain how ground truths were made manually through Gimp, which requires time and knowledge of the editing program and is a possible source of manual errors. If this step were to be automated, the user would only be required to place a set of real-world images into a specific folder and have access to the processing power to run the model.

From the results with different dataset compositions in Section 6.2.6 the models achieved functional and even good results when transfer learning on as few as 20 real-world images. This is another positive property when discussing the suitability for industry implementation, that a user would only need to take and upload a couple of dozen images. Another promising result was that the same model with only 20 real-world images achieved almost the same Dice score with a simulated data set of only 1000 images as with a data set three times the size. The robustness illustrated with a relatively stable Dice score is promising and lowers the need for extensive processing power. This thesis trained models using the Idun cluster, with $A100$ and a training time of approximately 15 minutes. The same models had a training time of roughly 45 minutes when tested for comparison on the GeForce Ti that was used in [20]. Even though the GeForce needs more time to run the models, it is a more accessible and significantly less expensive GPU as explained in Section 4.2.1. Therefore, it might be more reasonable to assume a user could have access to a similar processing unit. However, whether these results and Dice scores are high enough to qualify the models for industry implementation would be up to the potential users. Investigating this would be a natural part of future work.

The implementation in the industry already discussed is based on the users receiving the fully computed network and folder structure similar to this thesis. If a user were to build a similar network from scratch, it would require the user to possess knowledge and understanding of both machine learning and computer graphics. In addition, the user would have to generate large enough simulated and real-world image datasets with corresponding ground truths. It would be significantly more time-consuming to build similar models from scratch than to receive an already complete network. However, it is still possible and suitable, even though the most effective implementation would be to obtain the already made and tuned network with folder and file structure.

To conclude, if the results presented in this thesis are adequate for implementation and the process of making ground truths is automated, transfer knowledge models similar to the once presented could most certainly be suitable for implementation and use in the industry.

6.4. Future Work

There are several subjects that would be interesting to examine further. It would be fascinating to train models with validation sets that include images without reflections, as already mentioned. It would also be interesting to use other cameras, study the effects of a regular camera or even a phone camera, or use images with more pixels. For example, images of size 1024×1024 might procure more accurate results, though this would require more training time. It could be interesting to make an algorithm that automatically stops the learning when the loss function plateaus to counteract overfitting during the training process. This type of algorithm would also allow for more flexible learning and potentially a larger number of epochs. The algorithm would only train for the necessary amount, which again could counteract overfitting and potentially reduce the training time and increase results. It would be interesting to implement such an algorithm to explore the consequences, if the results are significantly better it could improve the suitability for industry implementation. Another possibility to test in the future could be to freeze certain layers and place a linear classifier on top of the output probabilities. It would likely yield similar results, but minimizing the risk of overfitting. There are also even more combinations of hyperparameters that can be tested in future work to explore their impact on the results. For instance, studies as [58] achieve promising results using compound loss functions that use the benefits of both cross-entropy loss and Dice loss.

All the various computing possibilities presented would be interesting to study in the future. However, the more pressing question is whether users would implement the transfer learning models and the possible improvements to make them even more suitable for the industry. The only ones suited to answer this question are the potential users. Therefore it would be a natural next step to reach out to the industry and potential users to conduct research and get an insight into their needs and the steps needed to make the industry implementation a reality.

Chapter 7.

Conclusion

In this thesis, transfer learning was used to successfully transfer knowledge from a simulated to a real-world environment. Through semantic segmentation, the models were developed to estimate the pixel-wise position of the correct weld line. U-net was used with varying hyperparameters in order to produce an optimal model. Different combinations of batch sizes, epochs, loss functions, learning rates and optimizers were used to compare results and analyze their effect on the overall performance of the models.

In general, the models achieved better results than expected. The models with the highest Dice scores used adaptive optimizers and low learning rates, though the differences between the models were smaller than expected. The U-net architecture was the primary reason why many models with such diverse hyperparameters performed so well. The deep and complex architecture provided robustness, and it makes sense that U-net is so commonly seen in other semantic segmentation studies. However, the architecture comes with its disadvantages, the complexity requires a lot of resources. The access to significantly more processing power through the Idun cluster was pointed out as a contributing factor to the overall high Dice scores, since this allowed for a drastic increase in dataset size.

Models were made with varying ratios of different materials in the real-world images to further study the abilities of the transfer learning models and their suitability for industry implementation. Both specular and diffuse reflections were handled well regardless of material. Another important aspect when discussing the suitability for implementation was the size of the real-world dataset, and that the transfer learning exceeded expectations with small real-world datasets. The fact that the models performed well, after only training on a handful of real-world images, showed that they learned a lot about weld line patterns in the simulated dataset, and were able to further modify the weights for attributes more specifically in the real-world dataset.

The difference in laser thickness between simulated and real-world images and imperfect edge detection led to a slight decrease in Dice scores. Efforts should be made to achieve as similar datasets as possible. Specifically, the simulated dataset should be modified to match the real-world dataset. This is because the entire simulated dataset can be regenerated easily by only adjusting some parameters.

The results obtained in this thesis agree with similar studies presented throughout the thesis, that the U-Net architecture used with an adaptive optimizer is suitable for transfer learning tasks. The high Dice scores that these transfer models reach lay an exiting foundation for further work and industry implementation.

References

[1] J. S. Aasberg. Machine learning using 3d data on a high performance computing cluster. *NTNU*, 2021.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: large-scale machine learning on heterogeneous systems, 2015. URL: https://www.tensorflow.org/.

[3] D. Allman, A. Reiter, and M. A. L. Bell. A machine learning method to identify and remove reflection artifacts in photoacoustic channel data. *IEEE International Ultrasonics Symposium (IUS)*:1–4, 2017. DOI: 10.1109/ULTSYM.2017.8091630.

[4] O. Alstad. Convolutional neural networks for filtering reflections in laser scanner systems. *Department of Mechanical and Industrial Engineering, NTNU*, 2021. DOI: https://hdl.handle.net/11250/2787885.

[5] Ambientcg, 2021. URL: https://ambientcg.com/.

[6] A. Araujo, W. Norris, and J. Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019. DOI: 10.23915/distill.00021.

[7] A. A. Awan, H. Subramoni, and D. K. Panda. An in-depth performance characterization of cpu- and gpu-based dnn training on modern architectures. *Association for Computing Machinery*, 2017. DOI: 10.1145/3146347.3146356.

[8] Canon science lab: cmos sensors, 2022. URL: https://global.canon/en/technology/s_labo/light/003/05.html.

[9] CECI. Slurm quick start tutorial, 2022. URL: https://support.ceci-hpc.be/doc/_contents/QuickStart/SubmittingJobs/SlurmTutorial.html.

[10] S.-B. Chen. On intelligentized welding manufacturing. 363:3–34, 2015. DOI: 10.1007/978-3-319-18997-0_1.

[11] Common vision blox, 2022. URL: https://www.commonvisionblox.com/en/common-vision-blox-powerful-fast-modular/.

[12] computar. Mpz series machine vision lens v1226-mpz 1" 12mm f2.6, 2022. URL: https://computar.com/product/1450/V1226-MPZ.

[13] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[14] O. Egeland. Robot vision. *Department of Mechanical and Industrial Engineering, NTNU*, 2021.

[15] M. Ferguson, S. Jeong, K. H. Law, S. Levitan, A. Narayanan, R. Burkhardt, T. Jena, and Y.-T. T. Lee. A standardized representation of convolutional neural networks for reliable deployment of machine learning models in the manufacturing industry. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Volume 1: 39th Computers and Information in Engineering Conference, 2019. DOI: 10.1115/DETC2019-97095.

[16] J. G. D. M. Franca, M. A. Gazziro, A. N. Ide, and J. H. Saito. A 3d scanning system based on laser triangulation and variable field of view. *IEEE International Conference on Image Processing*, 2005. DOI: 10.1109/ICIP.2005.1529778.

[17] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[18] S. Grans and L. Tingelstad. Blazer: laser scanning simulation using physically based rendering. *arXiv preprint arXiv:2104.05430*, 2021.

[19] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Pres, 2nd edition, 2004. ISBN: ISBN: 0521540518. DOI: 10.1017/CBO9780511811685.

[20] A. B. Holm and K. Kallseter. Machine learning for weld line laser-reflection removal. *Department of Mechanical and Industrial Engineering, NTNU*, 2021.

[21] Image: laser triangulation. URL: https://imaging.teledyne-e2v.com/products/applications/3d-imaging/laser-triangulation/.

[22] M. Inkawhich. Pytorch, saving and loading models, 2022. URL: https://pytorch.org/tutorials/beginner/saving_loading_models.html.

[23] J. R. Jensen. *Introductory Digital Image Processing: A Remote Sensing Perspective. Chapter 8, Thematic Information Extraction: Image Classification*. Upper Saddle River, New Jersey: Prentice-Hall, second edition, 1996, pages 234–252.

[24] J. Jeong, T. S. Yoon, and J. B. Park. Towards a meaningful 3d map using a 3d lidar and a camera. *Sensors*, 18(8):2571, 2018. DOI: 10.3390/s18082571.

[25] E. S. Jonathan Long and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*:3431–3440, 2015. DOI: 10.1109/CVPR.2015.7298965.

[26] P. Kah, M. Shrestha, E. Hiltunen, and J. Martikainen. Robotic arc welding sensors and programming in industrial applications. *International Journal of Mechanical and Materials Engineering*, 10:13, 2015. DOI: 10.1186/s40712-015-0042-y.

[27] I. Kandel and M. Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, 6(4):312–315, 2020. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2020.04.010.

[28] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. *Published as a conference paper at the 3rd International Conference for Learning Representations*, 2015. DOI: 10.48550/ARXIV.1412.6980.

[29] M. Kumar, M. Shanavas, A. Sidappa, and M. Kiran. Cone beam computed tomography - know its secrets. *PMC: US National Library of Medicine, National Institutes of Health*, 7(2):64–68, 2015.

[30] Y. Ma, S. Soatto, J. Košecká, and S. S. Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer Science Business Media, New York, USA, 2004. DOI: 10.1007/978-0-387-21779-6.

[31] W. J. Matt Pharr and G. Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann Publishers Inc., third edition, 2016. ISBN: 9780128006450.

[32] P. Mattis and S. Kimball. Gimp: gnu image manipulation program, 1995. URL: https://www.gimp.org/.

[33] M. Mattsson. Laser line extraction with sub-pixel accuracy for 3d measurements. *Department of Mathematics, Linköping University*, 2020.

[34] A. Meier, M. Kropp, and G. Perellano. Python experience report of teaching agile collaboration and values: agile software development in large student teams. *IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*:76–80, 2016. DOI: 10.1109/CSEET.2016.30.

[35] Microsoft. Visual studio code, 2015. URL: https://code.visualstudio.com.

[36] NTNU. Idun: high performance computing group. URL: https://www.hpc.ntnu.no/idun/.

[37] NVIDIA. Geforce gtx 1080 ti, 2022. URL: https://www.nvidia.com/en-sg/geforce/products/10series/geforce-gtx-1080-ti/.

[38] NVIDIA. Nvidia a100 tensor core gpu, 2022. URL: https://www.nvidia.com/en-sg/data-center/a100/.

[39] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. DOI: 10.1109/TKDE.2009.191.

[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: an imperative style, high-performance deep learning library, 2019. URL: http://www.deeplearningbook.org.

[41] Ö. Polat, A. Polat, and T. Ekici. Automatic classification of volcanic rocks from thin section images using transfer learning networks. *Neural Computing and Applications, Springer*, 33:11531–11540, 2021. DOI: 10.1007/s00521-021-05849-3.

[42] H. Pottmann and J. Wallner. *Computational Line Geometry*. Springer-Verlag, Berlin, Germany, 2001. ISBN: 3-540-42058-4. DOI: 10.1007/978-3-642-04018-4.

[43] T. Preston-Werner, C. Wanstrath, P. J. Hyett, and S. Chacon. Github, 2008. URL: https://github.com.

[44] Python software foundation, guido van rossum. python, 1991. URL: https://www.python.org/.

[45] O. Ronneberger, P. Fischer, and T. Brox. U-net: convolutional networks for biomedical image segmentation. in: international conference on medical image computing and computer-assisted intervention (miccai). *Springer*. LNCS, 9351:231–241, 2015.

[46] M. Rossi, G. Belotti, C. Paganelli, A. Pella, A. Barcellini, P. Cerveri, and G. Baroni. Image-based shading correction for narrow-fov truncated pelvic cbct with deep convolutional neural networks and transfer learning. *MEDICAL PHYSICS: The International Journal of Medical Physics Research and Practice*, 48(11):7112–7126, 2021. DOI: 10.1002/mp.15282.

[47] A. Sarangam. Epoch in machine learning: a simple introduction, 2021. URL: https://www.jigsawacademy.com/blogs/ai-ml/epoch-in-machine-learning.

[48] J. G. Semple and G. T. Kneebone. *Algebraic Projective Geometry*. Oxford Classic Series. Claredon Press, 1952.

[49] P. Shirley and R. K. Morley. *Realistic Ray Tracing*. AK Peters, Ltd, Massachusetts, USA, second edition, 2008. ISBN: 9781568814612.

[50] E. Stevens, L. Antiga, and T. Viehmann. *Deep Learning with PyTorch*. Manning Publications Co., USA, 2020. ISBN: 9781617295263.

[51] S. A. Taghanaki, K. Abhishek, J. P. Cohen, J. Cohen-Adad, and G. Hamarneh. Deep semantic segmentation of natural and medical images: a review. *Artificial Intelligence Review,Springer*, 54:137–178, 2021. DOI: 10.1007/s10462-020-09854-1.

[52] Teledyne dalsa genie nano-gige, 2022. URL: https://www.stemmer-imaging.com/en/products/series/teledyne-dalsa-genie-nano/.

[53] M. Teulieres, J. Tilley, L. Bolz, P. Ludwig-Dehm, and S. Wagner. *Industrial robotics. Insights into the sector's future growth dynamics*. McKinsey and Company, 2019.

[54] R. Thangaraj, S. Anandamurugan, and V. K. Kaliappan. Automated tomato leaf disease classification using transfer learning-based deep convolution neural network. *Journal of Plant Diseases and Protection, Springer*, 128:73–65, 2021. DOI: 10.1007/s41348-020-00403-0.

[55] T. Tieleman and G. Hinton. Lecture 6.5—rmsprop: divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*:26–31, 2012.

[56] E. Tiu. Metrics to evaluate your semantic segmentation mode, 2019. URL: https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2.

[57] L. Torrey and J. Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[58] Unified focal loss: generalising dice and cross entropy-based losses to handle class imbalanced medical image segmentation. *Computerized Medical Imaging and Graphics*, 95:102026, 2022. ISSN: 0895-6111. DOI: https://doi.org/10.1016/j.compmedimag.2021.102026.

[59] A. Urbańczyk, J. Wright, D. Cowden, I. T. Solutions, H. Y. ÖZDERYA, M. Boyd, B. Agostini, M. Greminger, J. Buchanan, cactrot, huskier, M. S. de León Peque, P. Boin, W. Saville, B. Weissinger, Ruben, nopria, C. Osterwood, moeb, A. Kono, HLevering, W. Turner, A. Trhoň, G. Christoforo, just-georgeb, A. Peterson, A. Grunichev, A. Gregg-Smith, Bernhard, and D. Anderson. Cadquery/cadquery: cadquery 2.1.

[60] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The marginal value of adaptive gradient methods in machine learning. *Curran Associates, Inc.*, 2017. Berkeley University of California and Toyota Technological Institute at Chicago.

[61] W. Zhi, Z. Chen, Z. L. Henry Wing Fung Yueng, S. M. Zandavi, and Y. Y. Chung. Layer removal for transfer learning with deep convolutional neural networks. at the conference of international conference on neural information processing, 2017. URL: https://www.researchgate.net/publication/320631663_Layer_Removal_for_Transfer_Learning_with_Deep_Convolutional_Neural_Networks.

[62] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning. 109(1):43–76, 2021. DOI: 10.1109/JPROC.2020.3004555.

[63] K. H. Zou, S. K. Warfield, A. Bharatha, C. M. Tempany, M. R. Kaus, S. J. Haker, W. M. W. III, F. A. Jolesz, and R. Kikinis. Statistical validation of image segmentation quality based on a spatial overlap index1: scientific reports. *Academic Radiology 11.2*, 11(2):178–189, 2004. DOI: 10.1016/s1076-6332(03)00671-8.

Appendix A.

Accuracy Tables

A.1. Epochs

Optimizer	Learning Rate	Batch Size	Epochs	Dice Score
Adam	$0.0003, 0.0001$	4	$4, 6$	0.8426
Adam	$0.0003, 0.0001$	4	$2, 6$	0.8226
Adam	$0.0003, 0.0001$	4	$1, 4$	0.8095
Adam	$0.0003, 0.0001$	4	$3, 2$	0.7860
Adam	$0.0003, 0.0001$	4	$2, 10$	0.7717
Adam	$0.0003, 0.0001$	4	$4, 3$	0.7665

Table A.1.: Dice scores for different models with Adam as optimizer, the optimal learning rate for Adam and a batch size of 4, but varying in epochs. Ranked based on Dice score.

A.2. Optimization Algorithms

Optimizer	Learning Rate	Batch Size	Epochs	Dice Score
SGD	0.1	2	4, 6	0.8443
(DL) SGD	0.1	2	4, 6	0.8176
SGD	0.1	4	4, 6	0.8057
SGD	0.1	16	4, 6	0.7074
SGD	0.01	4	4, 6	0.7634
SGD	0.001	4	4, 6	0.6109
SGD	0.0001	4	4, 6	0.4023
Adagrad	0.1	4	4, 6	0.8104
Adagrad	0.01	4	4, 6	0.8151
Adagrad	0.001	2	4, 6	0.8551
(DL) Adagrad	0.001	2	4, 6	0.8421
Adagrad	0.001	4	4, 6	0.8328
Adagrad	0.001	16	4, 6	0.7742
Adagrad	0.0001	4	4, 6	0.7723
RMSProp	0.1	4	4, 6	0.7643
RMSProp	0.01	4	4, 6	0.8225
RMSProp	0.001	4	4, 6	0.8364
RMSProp	0.0001	4	4, 6	0.8551
RMSProp	0.0001	2	4, 6	0.8637
(DL) RMSProp	0.0001	2	4, 6	0.8609
RMSProp	0.0001	16	4, 6	0.8269
Adam	0.1	4	4, 6	0.7073
Adam	0.01	4	4, 6	0.7994
Adam	0.001	4	4, 6	0.8303
Adam	0.001, 0.0001	4	4, 6	0.8409
Adam	0.0003, 0.0001	2	4, 6	0.8541
(DL) Adam	0.0003, 0.0001	2	4, 6	0.8508
Adam	0.0003, 0.0001	4	4, 6	0.8426
Adam	0.0003, 0.0001	16	4, 6	0.8120

Table A.2.: Dice scores for different models with different optimizer algorithms, learning rates, batch sizes and loss functions. Arranged after the learning rate. The majority of the models utilized cross-entropy as their loss function with weighted classes of [0.3, 0.7], except the ones outlined in a light gray and marked with (DL) for Dice loss. The models reaching the highest dice score for their respective optimizers are outlined in blue.

A.3. Transfer Learning on Different Materials

Optimizer	Training	Validation	Dice Score
RMSProp	Both	Aluminium	0.8361
RMSProp	Both	Steel	0.8861
Adam	Both	Aluminium	0.8317
Adam	Both	Steel	0.8985
RMSProp	Aluminium	Both	0.8332
RMSProp	Aluminium	Steel	0.8430
RMSProp	Steel	Both	0.8468
RMSProp	Steel	Aluminium	0.8041

Table A.3.: Dice scores for models where the transfer learning was executed using different combinations of aluminium, steel or both in training and validation. All the models were made using a batch size of two and epochs of four and six. The model reaching the highest dice score is marked in blue.

A.4. Amount of Real-World Images vs Simulated Images used in the Transfer Learning Training

Optimizer	Simulated images used for training	Real-world images used for training	Dice Score
RMSProp	3200	0	0.3807
Adam	3200	0	0.3147
(DL) Adam	3200	0	0.2342
Adam	3200	160	0.8541
Adam	3200	100	0.8376
Adam	3200	60	0.8223
Adam	3200	20	0.7842
Adam	1000	100	0.8152
Adam	1000	60	0.8135
Adam	1000	20	0.7818

Table A.4.: Dice scores on the real-world dataset for models, with optimal hyperparameters, where both the number of images in the simulated dataset and the real-world dataset vary during training.

Appendix B.

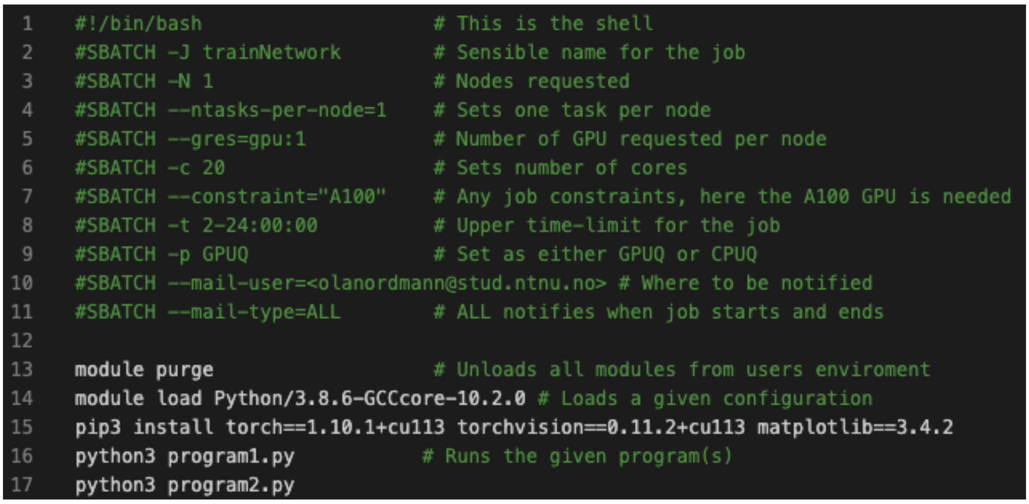
Connecting to the Idun cluster

As explained in Section 4.2.1, the cluster is a project created at NTNU. Therefore students at NTNU may ask their supervisor to approve access to the cluster resources that their department might have. Idun has a website on how to get access to Idun, where the contact persons for each shareholder are listed and how the procedure to get access is explained. All affiliations need to have a support agreement with NTNU IT in order to access the cluster, this is because Idun is a shareholder machine. They have a website explaining how to become a shareholder and partner in Idun.

After access has been granted, NTNU has websites with instructions to getting started on Idun, including how to login. For NTNU students, the login requires the *feide* username and password both during login in the terminal and when transferring data. For this thesis, the network drives called Samba were used for mounting Idun's home and work directory to a local machine. This is described in transfer data for Windows, macOS, and Linux users. The files were edited locally on a personal computer before being copied to the desired folder in the samba home directory. It is also possible to connect Github to the Samba drives, though this demands further configuration.

For a student to connect to Idun, the computer must be connected to the NTNU network. If the computer is outside of the school perimeters, it has to connect to a virtual private network, VPN. NTNU has a website that explains how to download and connect to a VPN to access the NTNU network outside of the school grounds.

NTNU student Jonas Strand Aasberg has written an in-depth guide for connecting to and running code on Idun as part of his student project [1]. A resource manager or job scheduler often organizes this kind of resource sharing on high-performance computing software such as Idun. Slurm is a Linux cluster management system that is open source. It facilitates resource allocation and distribution among users, as well as job submission queues in times of heavy resource demand. This job scheduler manages the scheduling of the jobs users commit and the allocation of resources to complete these jobs, such as GPUs and memory [9]. Slurm was the preferred job scheduler for this thesis. The slurm-files used in the thesis can be found in the Github repository. For example, the slurm-files allowed for detailed specifications of which programs to run, how many nodes to allocate, and how many tasks each node should execute, setting time constraints, and requesting notifications over email after each job was executed. Figure B.1 shows an example of a slurm-file with comments.



```
#!/bin/bash                         # This is the shell
#SBATCH -J trainNetwork             # Sensible name for the job
#SBATCH -N 1                        # Nodes requested
#SBATCH --ntasks-per-node=1         # Sets one task per node
#SBATCH --gres=gpu:1                # Number of GPU requested per node
#SBATCH -c 20                       # Sets number of cores
#SBATCH --constraint="A100"         # Any job constraints, here the A100 GPU is needed
#SBATCH -t 2-24:00:00               # Upper time-limit for the job
#SBATCH -p GPUQ                     # Set as either GPUQ or CPUQ
#SBATCH --mail-user=<olanordmann@stud.ntnu.no> # Where to be notified
#SBATCH --mail-type=ALL             # ALL notifies when job starts and ends

module purge                        # Unloads all modules from users enviroment
module load Python/3.8.6-GCCcore-10.2.0 # Loads a given configuration
pip3 install torch==1.10.1+cu113 torchvision==0.11.2+cu113 matplotlib==3.4.2
python3 program1.py               # Runs the given program(s)
python3 program2.py
```

Figure B.1.: Slurm-file example

A slurm-file must initially be uploaded to the network drive to run a job. Then the user must navigate to the correct folder in the directory containing both the slurm-file and necessary code files. The slurm-file would run by writing the following command in the terminal:

```
sbatch slurm-file.slurm
```

Then the user would receive a confirmation mail that the job was committed and when the job was finished. The outputs would be saved in a file called *slurm-job-number.out* in the same folder as the slurm-file.