FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

Increasing performance of a deep learning model for SLS printing defect detection

Bachelor Thesis

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science in Engineering

to the University of Applied Sciences FH Campus Wien

Bachelor Degree Program: Computer Science and Digital Communications

Author:

Matthias Schmid-Kietreiber

Student identification number:

1910475050

Supervisor:

DI Dr. techn. Mugdim Bublin
Klamert Victor, BSc MSc

Date:

05.06.2022

Declaration of authorship:

I declare that this Bachelor Thesis has been written by myself. I have not used any other than the listed sources, nor have I received any unauthorized help.

I hereby certify that I have not submitted this Bachelor Thesis in any form (to a reviewer for assessment) either in Austria or abroad.

Furthermore, I assure that the (printed and electronic) copies I have submitted are identical.

Date: 05.06.2022 Signature: 

Abstract

Additive Fertigung bewegt sich immer mehr vom Prototyping und der Forschung in die ernsthafte industrielle Produktion. Ein wichtiger Schritt zur Weiterentwicklung ist die Qualitätssicherung gedruckter Bauteile. Eine der möglichen Lösungen dafür ist der Einsatz von Deep Learning und Computer Vision, um Fehler während des Druckprozesses zu identifizieren. Aktive Forschung wird auch am FH-Campus Wien betrieben, wo Herr Klamert Victor, BSc MSc die classifikations Möglichkeiten des Curlings während eines Selective Laser Sintering (SLS) Prozesses erforscht.
Ein wichtiger Bestandteil von Deep Learning sind die Daten, die zum Trainieren des Algorithmus verwendet werden. Diese Arbeit beschäftigt sich daher mit der möglichen Leistungssteigerung einer VGG-16 CNN Architektur durch Aufbereitung der Trainings Daten. Es wurden mehrere Aufbereitungsschritte und deren Auswirkungen auf die Klassifizierungsleistung des Models untersucht. Dazu wurden aus drei verschiedenen SLS-Druckverfahren Video Aufnahmen erstellt, die von Studierenden des Studiengangs High Tech Manufacturing der FH-Campus Wien aufgenommen wurden. Diese Aufnahmen zeigten drei Reihen mit drei verschiedenen Geometrien. Aus welchen weitere Datensätze generiert wurden, indem diese einzelnen Komponenten ausgeschnitten wurden. Durch die Anwendung von unterschiedlichen Bearbeitungsschritten wie dem Entfernen von Duplikaten und dem Entfernen nicht identifizierbaren Bildern wurde ein zweites Set von Datensätzen erstellt. Ein Vergleich von CNN-Modellen, die auf diesen Datensätzen Batches trainiert wurden, zeigte deutlich, dass, solange der Datensatz groß genug ist, aufbereitete Daten Leistungssteigerungen von bis zu 20% erzielen können.

Kurzfassung

Additive Fertigung bewegt sich immer mehr vom Prototyping und der Forschung in die ernsthafte industrielle Produktion. Ein wichtiger Schritt zur Weiterentwicklung ist die Qualitätssicherung gedruckter Bauteile. Eine der möglichen Lösungen dafür ist der Einsatz von Deep Learning und Computer Vision, um Fehler während des Druckprozesses zu identifizieren. Aktive Forschung wird auch am FH-Campus Wien betrieben, wo Herr Klamert Victor, BSc MSc die Klassifikationsmöglichkeiten des Curling während eines Selective Laser Sintering (SLS) erforscht.
Ein wichtiger Bestandteil von Deep Learning sind die Daten, die zum Trainieren des Algorithmus verwendet werden. Diese Arbeit untersucht eine Vielzahl von vorverarbeiteten Datensätzen und deren Einfluss auf die erreichbare Leistung einer vortrainierten VGG-16 CNN Architektur. Dazu wurden aus drei verschiedenen SLS-Druckverfahren Video Aufnahmen erstellt, die von Studierenden des Studiengangs High Tech Manufacturing der FH-Campus Wien aufgenommen wurden. Diese Aufnahmen zeigten drei Reihen mit drei verschiedenen Geometrien. Weitere Datensätze wurden generiert, indem diese einzelnen Komponenten ausgeschnitten wurden. Durch die Anwendung von unterschiedlichen Bearbeitungsschritten wie dem Entfernen von Duplikaten und dem Entfernen nicht identifizierbaren Bildern wurde ein zweites Set von Datensätzen erstellt. Ein Vergleich von CNN-Modellen, die auf diesen Datensätzen Batches trainiert wurden, zeigte deutlich, dass, solange der Datensatz groß genug ist, vorverarbeitete Daten Leistungssteigerungen von bis zu 20% erzielen.

List of Abbreviations

AM	Additive Manufacturing
SLS	Selective Laser Sintering
ML	Machine Learning
DL	Deep Learning
NN	Neuronal Networks
MLP	Multi Layer Perceptron
DNN	Deep Neuronal Networks
CNN	Convolutional Neuronal Networks
DCNN	Deep Convolutional Neural Networks
CV	Computer Vision

Key Terms

Additive Manufacturing
Selective Laser Sintering
Deep Learning
Convolutional Neuronal Networks
Computer Vision
VGG-16

Contents

1. Introduction

After another Artificial Intelligence (AI) winter, AI is back and a hot topic once again. One reason for its awakening is its sub field called Deep Learning (DL). DL combined with the increase of computational resources made it possible to train computers to carry out tasks which were thought only possible for humans. With its ability to automate tasks previously only possible for humans, manufactures are busy integrating systems into their businesses. One of them being the field of additive manufacturing (AM). The main characteristic of an AM process is the layer wise manufacturing of a component by melting raw material. This layer wise production method makes detecting defects inside the component difficult and is still a hurdle that has to be overcome to make AM a more viable option in multiple industry settings. One possibility to solve this problem might be the visual detection of defects with the help of DL.

1.1. Selective Laser Sintering (SLS)

While there are many forms of AM, and many companies offer 3D printing services, selective laser sintering (SLS) is one of the most offered printing services according to a survey from 2019 conducted by AMFG [1]. SLS belongs to the group of powder bed fusion processes, which uses a energy source to melt the specified regions of a powder bed [2]. The main components of a SLS machine can be seen in figure 1.1 and are the laser which melts the powder, the powder feed chamber which feeds the raw material into the build chamber. The build chamber is where the component is printed layer by layer and a roller/sweeper or recoater spreads the powder above the platform.
The SLS printing procedure starts, as many other AM procedures, with a computer aided design (CAD) model of the component. This design contains the color, density, gradient and other data necessary to construct the component and is saved as a STL (stereolithography) [3]. The component is then sliced into layers and fed to the SLS machine. According to this data, powder is then spread onto the platform and preheated below its melting temperature. By preheating the material, the necessary laser power can be reduced and its absorption capability and wettability are increased. The laser is then guided though a computer controlled scanning system to fuse the powder particles [4]. After completion of the layer the platform is lowered through a piston and the roller spreads a new layer of powder material on top. The procedure is then repeated until the component is completed. [5].

1.1.1. Advantages of SLS

A major advantage of SLS and AM in general are the possible complex structures that can be manufactured. This makes geometries, especially inside the component, possible which would be impossible through traditional manufacturing methods like turning, milling and others. The high digitalization and the resulting possible automation are other strong points of AM which reduces the tooling necessary. Because of this production is not dependent on the volume that is produced, which in turn reduces costs. Additionally, the powder bed serves as

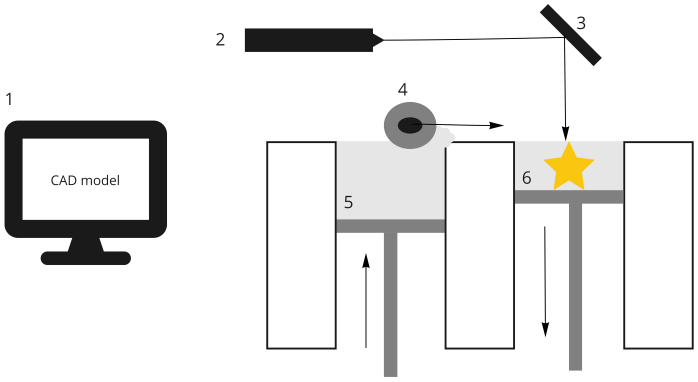


Figure 1.1.: Basic components of an SLS machine: 1) Computer; 2) laser; 3) scanning mirrors; 4) roller/sweeper; 5) material platform; 6) building platform; Adaped from [4]

support so that no additional support structures are needed during printing. Furthermore, the material can be used again with little to no processing. [4, 5, 6, 7]

1.1.2. Challenges of SLS

One of the challenges of SLS is the necessity to preheat the materials just below or up to the melting point to reduce to energy consumption of the laser. This leads to oxidation and degradation, therefore injecting inert gas to reduce oxygen becomes necessary [8, 9]. Temperature is also the reason for another challenging problem of SLS. Directly after fabrication of a layer the solidified material is still very hot. By applying a new layer of powder through the recoater, temperature differences between the comparatively cold powder and hot component can lead to residual stress which in turn leads to distortions of the component. One manifestation of this is called curling. As Curling is the situation of the component breaking through the powder bed due to the component's distortions. At the more advanced stages this can lead to the recoater ripping the component out of the build chamber [10, 11]. Not to be forgotten is also the fact that the powder used for SLS needs to be of uniform sizes. This limits the available choice of materials and requires additional processing steps which in turn increase costs [4].
Challenges related to heat are especially interesting topics of research for SLS operators due to the fact that parameterization makes it possible to mitigate it. But this also leads to a high complexity of adjusting the diverse parameters which take time and effort. This topic was also researched by Johannes Picker, in his bachelor thesis. Through the installation of a infrared camera in a SLS machine belonging to the High Tech Manufacturing department he was able to record images of five different build phases. He then evaluated these recordings with the help of excel and python to classify the temperature of the printing surface [12]. His work is mentioned here because the first part of this thesis as well as this thesis are based on his work. While he was concerned about the temperature and possible parameterization, this work (and the one before) are more concerned about recognizing curling and correctly classifying it as such through computer vision (CV) and DL.

1.2. Computer Vision and Deep Learning

Computer Vision (CV) is, in my opinion, one of the most interesting fields of deep learning. As the name suggests computer vision is all about making it possible for a computer to see and interpret the world as we humans do. The importance of CV is highlighted by the fact that it brought deep learning, which already had been around for some time [13], back into the foreground. It was a convolutional neural network (CNN) which was able to correctly classify handwritten numbers implemented by Yann LeCun and his team [14] and the increased computational power which made deep learning relevant again.
DL makes it possible to "train a algorithm" on a (for now) specified task and use it then for example to identify if the object on a image is a dog or a cat. The astonishing thing about this is that no expert is needed to define specific rules but that the algorithm learns them on its own. The algorithm only needs enough input data on which it can learn these rules and it is good to go. This opens the door for many new applications and automation of tasks that where dependent on human vision until now.
One big part of machine learning is defining the features on which a algorithm should be trained on. While this is still feasible for some data it becomes really difficult for larger and unstructured data like images. This is one of the many advantages of deep learning that the algorithm can find these features on its own. For this purpose, a neural network (NN) consists of two parts: the feature extractor and the classifier [14]. While in a basic NN both parts consist of fully connected Neurons/Units, networks used for CV use so called convolutions.

1.2.1. Convolutional Neural Networks

A NN consists of many perceptron's, a perceptron 1.2 calculates a weighted sum which is then used in a activation function and outputs the result [15]. If many of these are connected and stacked it is called a multi-layer perceptron or most of the time referred to as fully connected layers 1.3. The problem with fully connected layers is that the input must be one dimensional, therefore the spatial information of 2 dimensional images is lost [16]. Because of this the feature extractor consists of so-called convolutional layers which extract the features which are then fed into the classifier that consists of fully connected layers. Convolutions are filters, or kernels, that are applied on its input. The input begins as an image and is then deeper in the network called a feature map. These kernels multiply and sum the pixel values of the input with the values of the kernel. This results in an output where some pixels are now more pronounced (depending on the filter) then others. This leads to the before mentioned features that can be anything from edges, lines, abstract forms to eyes, legs, mouths and more. Another layer of the feature extractor often placed right after a convolutional layer is the pooling layer. This layer emphasizes the most important feature. In max. pooling for example this is done by only selecting the highest value in the area of the pooling matrix [14, 17, 18, 19, 20]. Multiple of these layers build the feature extractor part of the CNN followed by multiple stacked fully connected layers that build the classification part of the CNN 1.4.

1.3. Related work

This thesis uses a predefined architecture, called VGG-16, for the feature extraction part and will be explained a bit more in depth later on. For now, we will revisit the question about how quality control of AM can be increased with the help of machine learning. A pretrained ResNet50 was used by Zeqing Jin et al. to classify three different classes. By integrating it, already trained, directly into their printing process they were able to feed it a continues

stream of images. Their model was then able to classify the printing quality with an accuracy of 98%. Furthermore, it was able to detect defects which were even difficult for humans to detect [21]. To monitor melt-pool images, produced by a 350W laser, [22] used a 10HL-DNK2 architecture, where the number of units were reduced after each layer. The mean squared error (MSE) was used as a loss function, with which they reached a classification failure rate of 0.09%. [23] used a bi-stream deep convolutional neural network (DCNN) architecture to analyze a selective laser melting (SLM) process. One stream of the DCNN was fed images of the powder bed in the build chamber, while the other stream got slices of the component as input. The patterns from both streams were fused and classified by the model. This way they were able to reach a accuracy of 99.4%. A balanced accuracy of 96.80% was reached by Baumgartl et al. They accomplished this with a CNN which used depth wise separable convolutions. This model was trained on thermographic images from the printing process. A visualization of the exact defect position was also done by generating heat maps with a GradCam algorithm [24]. Leopold Le Roux et al. compared the performance of the CNN architectures DenseNet, AlexNet, SqueezeNet and ResNet. The defects they tried to classify were pores and bulging which occurred during an electron beam melting (EBM) process. By using a pretrained AlexNet model they were able to reach a accuracy rating of 95% and where able to show that a pretrained model performs better than a model trained from scratch [25]. Erik Westphal and Hermann Seitz used pretrained VGG-16 and Xception architectures to classify defects during an SLS process. By comparing the performance of both models, they concluded that the VGG-16 achieved the best accuracy result with a value of 93.9% [26]. Multiple points make their research special in context of this work. 1) They are also using the AM process SLS. 2) They explain in detail how their CNN architectures are constructed and what hyperparameters they used for training. 3) Also, a link to their used data set is offered. These points made it possible to start first investigations based on their implementations and quickly adapt it to the use case of this thesis.

1.3.1. Related work on the FH-Campus Vienna

As already mentioned, Johannes Picker, a student of the High Tech Manufacturing (HTM) course, also investigated this topic in his bachelor thesis. He researched the temperature distribution which occurs during the SLS printing. By experimenting with different heat sensors and thermal cameras and their possible integration into the SLS machine. He chooses to integrate a FLIR T420 thermal camera. But for the camera to fit onto the SLS machine he first had to design and produce a mounting device which made it possible to install the camera. He then studied five different printing phases closer by taking 15 infrared pictures

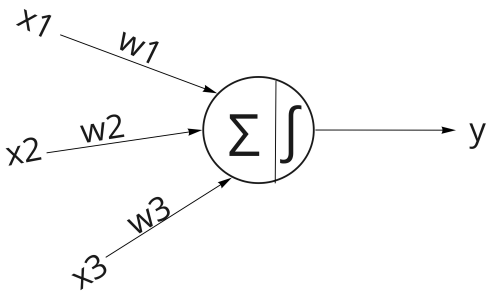


Figure 1.2.: Perceptron, Adapted from [15]

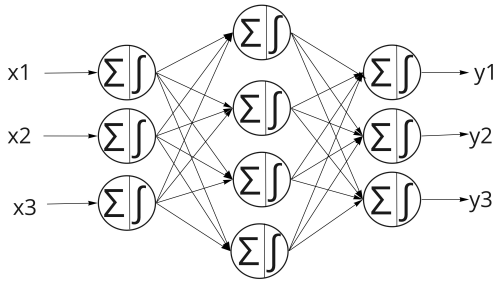


Figure 1.3.: Multi layer perceptron (MLP), Adapted from [15]

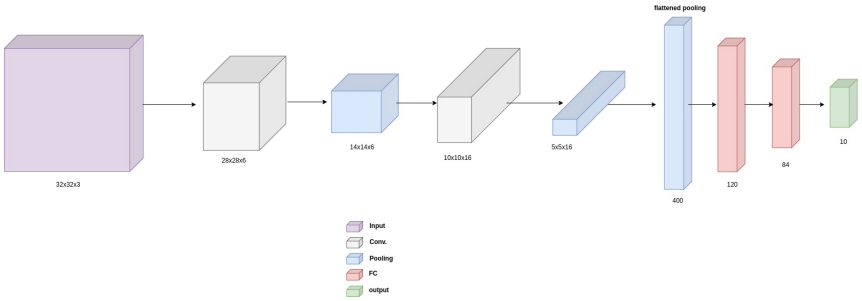


Figure 1.4.: Basic Convolutional Network Architecture

of the powder bed during a printing process. By processing these images further in a python program, he programmed, he was able to save the minimum and maximum temperature in a table. With additional statistical figures he was then able to analyse the temperature of the different printing phases. His final result was that, to his surprise, that only a minor temperature difference of the powder was noticeable. On the other hand, it was the component that radiated a high amount of heat [12]. Based on his work the first part of this work, bachelor thesis 1 (BA1), was built upon. For this computer vision and deep learning was used to classify curling of a component during an SLS printing process. Three different CNN architectures were built, guided by the results of [26], and their performance was compared. The training data was provided by colleagues of the HTM study course and consisted of infrared recordings of the powder bed during a printing process. From these recordings images were extracted and used for binary image classification. After various pre-processing of the data set the CNN architectures VGG-16, Xception and ResNet50 were trained and evaluated on their performance to correctly recognize an image as OK or defect. While VGG-16 reached an accuracy of 99.09%, Xception and ResNet50 only reached an accuracy of 16.58% [27]. While the high accuracy of the VGG-16 architecture was a pleasant surprise it's high value also cast some suspicion if this results could be trusted. To further investigate its results, a so called GradCam algorithm was used to produce heat maps to see what part of the image lead to its prediction. Even after the positive result of the analysis of these heat maps many questions were still left open but had to be addressed in future works because of time constraints of the BA1. Some of these questions are tackled in this thesis

1.4. Goal of this thesis

As mentioned before BA1 was concerned with constructing different CNN architectures and comparing their performance. This work now uses the best performing architecture, consisting of a VGG-16 pretrained feature extractor and a custom binary classifier, to investigate questions left open from BA1. One of these questions is how the data set can be responsible for any false performance indications and how these can be mitigated. The goal is to reduce doubts generated through possible bias of the data set and with it reduce false classifications, by increasing the quality of the data used for training and evaluation. The original classification problem remains as known from BA1, to detect curling on an infrared image of a SLS printing process. But increased focus will be put on the data used and possible pre-processing methods. In the end a comparison of multiple model performances will be given. For this purpose, section 2 will describe the feature extractor and classifier architecture in a bit more detail. Furthermore, the purpose and benefits of transfer learning are explained, and the thesis methodology will be stated. Section 3 gives an overview of the data set. Most importantly the various pre-processing steps will be explained and how it led to the categorization

of the data for training. In section 4 an explanation of the practical implementation is given followed by section 5 in which the results will be presented. These will then be discussed in section 6, in which also an outlook for possible future investigations will be given.

2. Project background and model architecture

2.1. Background

This thesis is part of a larger project, which encompasses BA1, this work and multiple master theses of multiple study courses held on the FH-Campus Vienna. All these are under the umbrella of Mr. Klamert Victors doctor thesis, how to use deep learning for real time process monitoring and curling defect detection in SLS.
Two students of the High Tech Manufacturing course investigate the topic of using thermographic image processing to improve the additive sintering process. For this purpose, they improved the setting of Mr. Pickers work [12] and captured infrared images of the SLS printing process. Without these recordings any machine learning process would not be possible, they build the raw data necessary to train a deep learning algorithm.
With this raw data, besides this thesis, two students of the Software Design and Engineering master course, also try to detect curling by using machine learning. While one of them is using methods similar to this work, the other student is investigating multiple possibilities like, clustering and anomaly detection using generative adversarial networks (GANs). All these have in common that the goal is to increase the quality of the SLS process.

2.2. Model architecture

While it is perfectly fine to build a deep learning network from scratch. It can become tedious to tune all hyperparameters to get the best results. Luckily some CNN architectures have become known to perform extremely well and have become something of a standard. These models all got known through the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The purpose of this challenge is to use a computer vision algorithm to classify the enormous number of classes and images contained in the ImageNet data set [28]. AlexNet which improves upon the LeNet architecture brought DL back into the focus of computer vision by creating a much deeper model then LeNet. It also used the Rectified Linear Unit (ReLU) function as a activation function [29]. A research team from Oxford called the Visual Geometry Group won the ILSVRC in 2014, with their VGG-16 CNN architecture. With their relatively simple design they managed to reach a top-5 error rate of 6.8% [30]. VGG-16 showed the world that deeper CNN accomplish better results, and so the much deeper ResNet architecture was created by Kaiming He et al. [31]. Another important CNN model, which introduced new architectural elements, is the Xception model created by Franc¸ois Chollet, a researcher currently at google [32].
While there are many more well performing CNN algorithms, the before mentioned models all brought CV and DL where it is today. This thesis will take advantage of their accomplishments and use a VGG-16 pretrained model as a feature extractor.

2.2.1. Transfer learning and pretraining

Many CNN architectures can be found online, especially the before mentioned models. While this reduces the need to build a CNN model from scratch, training a model, on the ImageNet

data set for example, can take days or might not even be impossible if the necessary hardware is not accessible. Here the term pretraining or pretrained comes into play. Not only is it possible to find CNN models online but also their weights, which were calculated during their training. This makes it possible to initialize a model with its weights, pretrained, ready for classification or retraining [16, 33].
Another great thing about pretrained models is that it can be used on other domains by retraining layers of the CNN. This is called transfer learning and has many advantages. For example is it possible to fit a model to ones need even if only a small amount of domain data is available? Furthermore, by exchanging the classifier part of the model customized classifications are possible. Depending on the differences of the source domain and the new domain multiple layers are frozen during training to preserve their trained features. This is also called fine tuning [15, 16].

2.2.2. Custom model

Following is the description and explanation of the custom model used. The full model architecture can be found in Appendix A

Feature Extractor

With all its advantages this thesis also uses pre- and transfer training. As a feature extractor a VGG-16 architecture will be used, including its weights calculated by being trained on the ImageNet data set. To use VGG-16 the input images have to be a fixed size of 224 x 224 x 3, 3 being the red, green and blue (RGB) channels of the image. 13 convolutional layers with a 3x3 kernel generate so called feature maps. To preserve the spatial resolution a stride of one pixel and padding is used. For regularization max-pooling layers are used after convolutions with a 2x2 window and stride 2 [30].

Classifier

As mentioned before transfer and pretraining is used to make binary classification possible for this work. For this the original VGG-16 classifier part was removed and a custom classifier was put on top to make binary classification possible. Since the classification part consists of a MLP the feature maps have to be transformed into vector form to be compatible with the FC layers. This is accomplished through a flatten layer after which a FC layer, consisting of over 1400 units and an ReLU activation function, is placed. Following it is a dropout layer and a batch normalization layer. To be able to binary classify the two classes DEF or OK on SLS printing images, the output layer consists of a dense layer with one unit and a sigmoid activation function.

2.3. Methodology

To determine the results of this thesis, multiple experiments in two stages were carried out. For the experiments in the first stage, different pre-processed batches of training data were used to determine on which batch which pre-processing yielded the best result.
In the second stage of the experiments this pre-processed batch of data was then used for fine tuning of the model. Multiple training runs were carried out, with progressively lower rate of frozen layers. All results were documented and compared to find the best performing, fine-tuned model.

3. Data

3.1. Acquisition

The raw data was produced by students of the High Tech Manufacturing study course, during their research during their master theses. This in-situ recordings were created with a FLIR T420 infrared camera manufactured by the company FLIR. The T420 makes it possible to measure temperatures from -20°C to +650°C, by using micro-bolometers to convert the electromagnetic radiation into temperature values [34]. The included software was used to record the SLS printing process and makes adjusting recording parameters and the camera focus possible [12]. The SLS machine in which the recording was created was an EOS Formiga P110 manufactured in the year 2013. The build chamber of the P110 makes it possible to build components with a volume of up to 200mm x 250mm x 330mm. The thermoplastic PA2200 is used as a material and is applied through the roller/sweeper onto the building platform. 160mm above the platform a heating element is located. This heating element is separated into 4 configurable heating zones and bring the temperature close to the melting point of the PA2200. The powder is then melted by a 30W CO2 laser applied through a F-theta lens. This way a printing speed of up to 5m/s and layer thicknesses of 0,06mm, 0,1mm or 0,12mm can be reached. Also, to prevent explosions the building chamber is filled with nitrogen [35].

3.2. Data sets

As described above multiple recordings of SLS printing processes were created. But for this work, recordings of three different dates were used for model training and comparison. From these recordings frames were extracted and separated first by their recording time and second if curling is recognizable or not. For this separation domain experts were needed to distinguish an image without curling from an image with curling and for now has to be

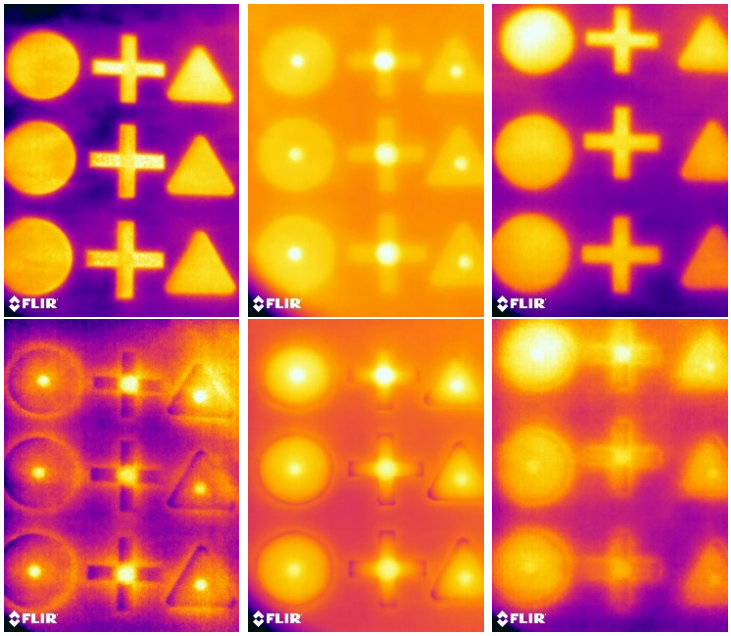
name/date	**total size (frames)**	**size of class OK (frames)**	**size of class DEF (frames)**
06.04.2022 - multiple geometries	76.450	75.763	687
23.03.2022 - multiple geometries	95.357	84.397	10.960
11.03.2022 - multiple geometries	144.019	141.800	2.219
06.04.2022 - patches	688.050	681.867	6.183
23.03.2022 - patches	858.213	759.573	98.640
11.03.2022 - patches	1.296.171	1.276.200	19.971

Table 3.1.: Number of frames of the raw data set

done by manually. This tedious task is one of the downsides of supervised learning in which labelled data is needed to train a DL algorithm. In this case Klamert Victor, BSc MSc took over this important job. In the end the raw data consisted of three folders named after the recording times and each containing a folder called OK and DEF, each containing the images without and with curling respectively. More information about the raw data size and names can be seen in table 3.1

3.2.1. Data set depictions

One important difference to the data set from BA1 is that these new frames depict multiple different geometries. Besides the already known cross geometry a circle and a triangle geometry is depicted on one frame. With these three rows of a circle, cross and triangle create the structure seen on one frame. As can be seen in 3.1. The color scheme for this data set was also changed in comparison to BA1. With this more emphasis is supposed to be put onto the single components and not on the colors of the background. Another noticeable element is that this differs for the frames from 23.03.2022. Here through accidental covering of the left corner the temperature difference was so big that the FLIR camera adjusted the color scheme in a more yellow tone. With this the components become more difficult to be distinguishable which also means that other factors besides curling should be less recognizable.



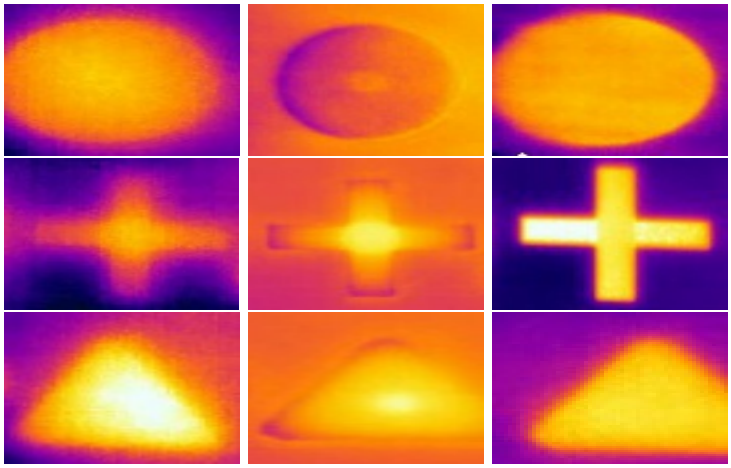
(a) Frames from 06.04.2022 (b) Frames from 23.03.2022 (c) Frames from 11.03.2022

Figure 3.1.: Example frames of the data sets - 1st row OK, 2nd row DEF

3.2.2. Patches

The multiple geometries made it also possible to create new data sets out of the existing ones. By extracting only, the geometry and thus creating frames only depicting a single geometry, even larger data sets could be generated 3.2. Their size is also documented in table 3.1 of the data sets. One advantage is as already mentioned the increase of the data set. But the main

reason for cropping the frame into its single geometries is to further narrow down the area of interest. By training a CNN only on these patches it should become possible to classify curling correctly not only on patches but also on frames with multiple geometries. With this the performance of the model to generalize should increase.



(a) Patches from 06.04.2022 (b) Patches from 23.03.2022 (c) Patches from 11.03.2022

Figure 3.2.: Example patches of the data sets

3.3. Data preparation

For a CNN to be able to use the data for training some preparation is needed. One constraint of a pretrained CNN architecture is that the shape of the input is defined by the architecture creators and has to be adhered by. In case of the VGG-16 algorithm the input must be 224x224x3. The first two numbers represent the height and width of the input in pixels and the last number tells the number of color channels. Therefore, the input of the model must consist of images with a height and width of 224 pixel and the three color channels red, green and blue (RGB). Besides this another important task is to split the data into a training-, validation-, and test-set. The train set is used to train the model to learn the data representations necessary for classification. To validate the results during training the validation set is used. After the model is finished training the test set is used to evaluate the performance again on data not seen during training. This is supposed to generate a unbiased evaluation of the model. But here one of the most important things to watch out for, is to never let the algorithm come into contact with the test set during training. This is important to follow rigorously. Then if not, the evaluation of the trained model results in high performance values across the whole model. Which leads to wrong assumptions and is, in the worst case, only recognized when the model is used in production. Wrong classifications and a complete retraining and tuning of a model can be the result. This costs time, money, and the trust of the customers, which had certain expectations to be fulfilled. These reasons have given enough justification to analyze the data, used for training and evaluation, in more detail. By doing so some potential problems could be identified.

3.3.1. Down sampling

One of the easiest to spot, is the imbalance of the two classes of the data. This class imbalance occurs if one of the classes has a significant higher number of examples [36, 37]. A quick look in table 3.1 makes it clear that in all three data sets the OK class contains a much larger number of examples. A problem of this imbalance is that the result of the model is dominated by the class with more images. Take the data set from 06.04.2022 for example. Out of 76.450 images only 687 images are in the class DEF, that is not even one percent. If the model now classifies everything as OK, its accuracy performance would be close to 100%. There are multiple possibilities to mitigate class imbalances, ranging from adjusting the CNN algorithm, processing the data or combinations of both [36]. In this work the simplest approach was used, down sampling. Down sampling means deleting random images from the majority class (the class with the most examples) till both classes hold the same number of images.

3.3.2. Removing unidentifiable frames

While down sampling reduced the sizes of the data sets dramatically, it made it easier to analyze it visually. One important realization was the amount of frames which did not belong to either of the two classes was quite significant. There are two moments during the SLS printing process in which it becomes impossible for the infrared camera to capture curling. One is the moment the laser of the SLS machine is melting the powder material. The high temperature of this process reduces the recorded frame to a mostly blue colored image. Next is the coating of the component with new powder. Not only are the recoater and the new powder obstructing view to the component, the temperature difference between new powder and the component also changes the color scheme of the frame 3.3. While it was known these images existed their number was quite honestly underestimated. This was especially noticeable in the data set from the 06.04.2022. After down sampling suddenly almost all of the frames in the class OK consisted of unidentifiable frames. Of course, this data set was not acceptable and had to be fixed. One solution to this problem would be to train another classifier on this data and use it to remove these frames. Especially in consideration of future work in this topic, this approach becomes a necessity for the preparation of future data sets. The student of the master course mentioned before, who is investigating this topic with the help of clustering and anomaly detection, implemented this classifier through a "simple" CNN. With it a big portion of frames can be removed, and comparatively little manual effort has to be spent later to remove the remaining ones. Nevertheless because of the complexity of building a new classifier and training it and maybe bad timing for this work all unidentifiable images were removed by hand. The astonishing end result was that the removed images amounted almost to halve of the data set as can be seen in table 3.2. The reason the frames in class DEF stayed the same is that they were already cleaned by Mr. Klamert Victor beforehand.

3.3.3. Removing duplicates

Through this a relatively clean data set was created and down sampling no longer left unidentifiable images in the classes. But there was still one major area of concern. As mentioned before one problem of the model trained in BA1 was the high evaluation score on the test set but the poor performance on new data. Even though this new data contained the same cross geometry the model was trained on. Since the model performed well on the test set which should have been decoupled from training, suspicion arouse that it did in fact come into contact with the model during training. One explanation could be that since the FLIR infrared camera records in a 30 frames per second rate, the differences between frames taken

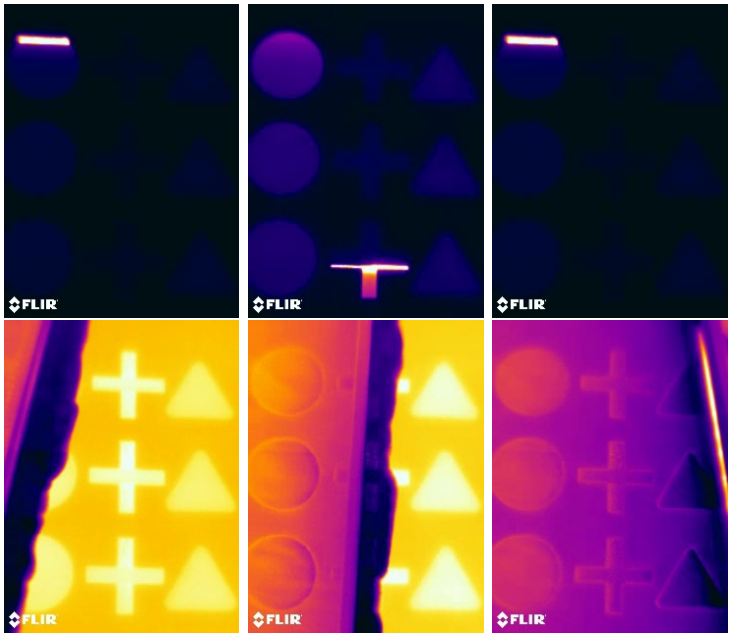


Figure 3.3.: Examples of frames containing the laser and recoating process - 1st row laser, 2nd row recoater

right after another could be almost non-existent. With almost no differences between them they could as well be duplicates. While it is common to add duplicates to increase the number of examples in a class, also called up sampling [36]. This only has to be done after the data set was split into train-, validation-, and test-set. If duplicates exist beforehand and are then put into the train and test set, the test set then contains examples which were also used to train the model. When the model is then evaluated on the test set it correctly classifies them since it was trained on them. Which in turn leads to an evaluation of the model which does not translate if used on different data.
To solve this problem a hashing algorithm was implemented. This algorithm creates a hash out of a frame and then compares them with each other. Is the same hash discovered one of them is deleted from the data set. This can be adjusted by setting the length of the to be created hash. The longer it is the more the frames have to be the same and vice versa.

name/date	**total size (frames)**	**size of class OK (frames)**	**size of class DEF (frames)**
06.04.2022 - multiple geometries	44.452	43.765	687
23.03.2022 - multiple geometries	57.020	46.060	10.960
11.03.2022 - multiple geometries	77.422	75.203	2.219
06.04.2022 - patches	400.068	393.885	6.183
23.03.2022 - patches	513.180	414.540	98.640
11.03.2022 - patches	696.798	676.827	19.971

Table 3.2.: Number of frames of the raw data set - after removing unidentifiable frames

Data set		Dirty & Clean		
	Name	Total Size	Size of class OK	Size of class DEF
06.04	multiple geometries 06.04 downsampled	1.374	687	687
	multiple geometries 06.04 no duplicates	708	354	354
	patches 06.04 downsampled	12.366	6.183	6.183
	patches 06.04 no duplicates	10.992	5.496	5.496
23.03	multiple geometries 23.03 downsampled	21.920	10.960	10.960
	multiple geometries 23.03 no duplicates	2.566	1.283	1.283
	patches 23.03 downsampled	197.280	98.640	98.640
	patches 23.03 no duplicates	87.426	43.713	43.713
11.03	multiple geometries 11.03 downsampled	4.438	2.219	2.219
	multiple geometries 11.03 no duplicates	1.320	660	660
	patches 11.03 downsampled	39.942	19.971	19.971
	patches 11.03 no duplicates	25.262	12.631	12.631

Table 3.3.: Dirty and clean data sets

3.4. Final data set composition

Through the before mentioned steps, multiple data sets of different composition were created. They were categorized and named after the steps which were performed on them. Firstly, they were categorized into "dirty" or "clean" depending on if they contain unidentifiable images or not. And secondly if they were only down sampled or if first the duplicates were removed and then they were down sampled. This resulted in 24 different data sets and since they were down sampled to fit the already cleaned DEF class the number of examples is the same for "dirty" and "clean" 3.3. Each of them was also split into train-, validation- and test-set and then used to train the same CNN architecture and compare the results. Also, further data augmentation was performed on the train set during the architecture training and will be explained in more detail in the next chapter.

4. Implementation

Deep learning projects are often heavily domain based. Besides research the goal is not the DL algorithm itself but a solution to a problem which DL hopefully solves better than previous methods. Because of this many domain experts found them self in the need to be able to program. They needed a programming language which was easy and fast to learn. This is one of the reasons that today python is the most used programming language for DL. Python is a high-level scripting language [38] and has been the main language for data scientists for a long time. It makes it easy to manipulate and move data on the computer and its many users driven libraries simplify many tasks. Moreover, the support and resources that can be found online are staggering. To take advantage of the whole python ecosystem this work also uses python for the practical implementation.

4.1. Libraries

While the numerous freely accessible libraries are one of the strong points of python, management of them can often become complicated and tedious. But the python community also build solutions around this problem. One of the simplest solutions is to use a cloud solution like google colab. It is accessible from every computer with internet access and most importantly no packages have to be installed by the user. Still, for more serious development there is almost no way around using a local machine. Here the python standard package manager pip [39] or Anaconda [40] can be used to manage libraries and its dependencies. For the project in this work a combination of both was used. Anaconda was used to create a virtual environment and to install the main libraries. Additional libraries which are not available through anaconda were then installed via pip. These management tools make sure all libraries are compatible with each other and by saving the virtual anaconda environment it becomes easy to share a .yaml file with witch all libraries can be installed through anaconda.

4.1.1. Most important libraries

Because of their importance, not only in this project, but to DL in general the most important libraries used will be briefly mentioned and described. Foundation for all necessary operations during the algorithm is the library tensorflow [41]. This library uses the concept of tensors to represent data in n-dimensional arrays and makes DL even possible to program. On top of tensorflow and inherent part of tensorflow since version 2.0 is the high-level API Keras [42]. These two libraries make development and optimization of DL feasible. Another major benefit is the GPU support through tensorflow which reduces training time dramatically. Alternatively, the framework Pytorch [43] can be used, it offers many of the same features and functions as tensorflow. This work uses tensorflow for its implementation of the CNN algorithm. A library similar to tensorflow concerned with n-dimensional array manipulation is numpy [44]. Since data is represented as n-dimensional arrays, numpy becomes a necessary tool for DL and data science in general. Another important aspect of DL is visualization of diverse metrics through graphs and the like. For this the library matplotlib [45] was used in this thesis.

4.2. Project setup

Before going into more detail, the basic project setup will be explained here. As described above this project was developed in the python programming language and setup in a virtual Anaconda environment. The repository of the source code can be found on GitHub under the url `https://github.com/Chrono666/ba2`. The included README.md contains instructions on how to install and use the project and all necessary dependency.
In BA1 mostly jupyter notebooks [46] were used for development, which made it easy to experiment with CNN algorithms. Since the main purpose of this work was primarily on how a single architecture performance increases by changing the data set "normal" python scripts were used. Each of the main scripts handles an important DL workflow.

4.2.1. Raw data preparation

The first step of the workflow was to prepare the raw data into a format usable for training. These steps were already described in detail in chapter 3 and consist of down sampling, removing unidentifiable frames and removing duplicates. In the end it was then split into train-, validation-, and test-set with a 0.8 to 0.1 to 0.1 split ratio through the splitfolders library [47]. All these steps, besides the removal of unidentifiable images, is done automatically by the script preprocess_raw_data.py. Through various command line arguments different stages of preparation can be executed.

4.2.2. Data loading and augmentation

With this fully prepared data sets can be created, ready for further use in the project. These data sets are then loaded with the help of keras methods designed to load image data sets for CNNs. These functions are not only able to load the data but are also able to apply data augmentation. Data augmentation refers to the manipulation of the images and has the advantage of mitigating over fitting and making the model more robust [48]. The data augmentation in this project was done through tensorflows ImageDataGenerator and consists of the following augmentations: 1) max shift image width by 2%, 2) max shift image height by 2%, 3) re scale the pixel values into a range between 0 and 1 by dividing 1/255 4) cut up to 15% from the image 5) in- or decrease the zoom of the image up to 15% 6) randomly flip the image horizontally 7) fill lost pixels with the nearest values. Important is that these augmentations are only applied to the train-set.

```
def preprocess_config(rotation_range=20,
                      width_shift_range=0.2,
                      height_shift_range=0.2,
                      shear_range=0.15,
                      zoom_range=0.15,
                      horizontal_flip=True,
                      fill_mode='nearest'):
   return ImageDataGenerator(rotation_range=rotation_range,
                             width_shift_range=width_shift_range,
                             height_shift_range=height_shift_range,
                             rescale=1. / 255,
                             shear_range=shear_range,
                             zoom_range=zoom_range,
                             horizontal_flip=horizontal_flip,
                             fill_mode=fill_mode)
```

Listing 4.1: Code snipped of function which returns a configured ImageDataGenerator

Loss Function	Learning rate	Optimizer	Max Epochs	Early Stopping Patience	Units of Dense layer
Binary cross entropy	$1 \times 10^{-}3$	Adam ß1=0.9 ß2=0.999	100	20	1400

Table 4.1.: Hyperparameters

4.2.3. Model architecture

With the data loaded and augmented it is then further passed to the CNN architecture. In chapter 2 the composition of the custom architecture was explained in detail. The VGG-16 feature extractor part is directly available through the tensorflow library initialized with the trained ImageNet data set weights. Adding the customized classifier was easily done by adding layers on top through Keras. Before training, the model needs to be compiled with the chosen loss function, optimizer and performance metrics which should be tracked. Since the problem to be solved was a binary classification the binary cross entropy was used as a loss function. Further the adam optimizer with a learning rate of 0.0001, ß1 of 0.9 and ß2 of 0.999 was used. These parameters are also known as hyperparameters and are summarized in table 4.1.

4.2.4. Training loop

The model was then trained with the hyperparameters listed in table 4.1 through the training loop constructed in the train.py python script. Various command line arguments can be passed to the train script to adjust the before mentioned hyperparameters, as well as the batch size, data location and how many convolutional layers should be frozen. A example on how the train.py script can be invoked is displayed in listing 5.1.

```
1 python train.py --data-dir cross_geometry --epochs 100 --batch-size 64 --
    learning-rate 0.0001 --beta-1 0.9 --beta-2 0.999
```

Listing 4.2: Example of starting the train.py script

By invoking the train.py scrip the data is loaded, augmented, and resized to 224x224x3 to fit the VGG-16 input layer. The custom model is built and compiled with the parameters passed as arguments. While the VGG-16 feature extractor is already initialized with the pretrained weights, the weights of the classifier are initialized randomly. This of course leads to a high loss calculated by the loss function. Normally this would be no problem and is after all how a DL algorithm works. Now the purpose of transfer learning is to use already trained weights which were fine-tuned through training on large data sets like ImageNet for example. These weights would now be rendered less effective if the high loss is propagated back to them. To mitigate this the feature extractors layers are frozen, by setting the trainable property to false. This means that the weights will not be updated during training and stay the same. If the domain would be close to the domain of the pretrained feature extractor, the whole training loop could be done like this and is also called fine tuning. Since this will also be used later in this thesis, this process will be described in more detail at that point. After training the model for five epochs, only the classifier weights are updated and reduce the range of the loss. After these initial five epochs all layers are set back to be trainable. The model is compiled again with the same hyperparameters, and training is continued. The maximum epochs the model can be trained was set to 100, but an early stopping call-back

with a patience of 20 watches the loss and stops training if it has not changed after 20 epochs. With this overfitting due to over training is prevented. After 100 epochs or depending on the early stopping call-back sooner, training is completed, and the model is saved. A saved model makes it possible to initialize it again later without the need of building it again. But even more importantly the calculated weights are also preserved. This makes it possible to fully load the model with its trained weights and use it for classification in other applications. A reduced form of the training loop is show in listing 4.3.

```
preprocess_config = dataset.preprocess_config()
train_data, val_data, test_data = dataset.load_dataset(args.data_dir,
                                                       target_size=(224, 224),
                                                      batch_size=args.batch_size,
                                                     class_mode='binary',
                                                   configuration=preprocess_config)

model = build_model(dropout_rate=args.dropout_rate)
model = compile_model(model, args.pre_learning_rate, args.beta_1, args.beta_2,
                      ['accuracy', 'Recall', 'Precision', 'AUC'])
set_base_model_layers_trainable(model, False)
history = train_model(model, train_data, val_data, epochs=args.pre_epochs)
set_conv_layers_trainable(model, True, args.freeze)
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=
    args.early_stopping)
model = compile_model(model, args.learning_rate, args.beta_1, args.beta_2, ['
    accuracy', 'Recall', 'Precision', 'AUC'])
history = train_model(model, train_data, val_data, epochs=args.epochs, callbacks
    =[early_stopping])
loss, accuracy, recall, precision, auc = model.evaluate(test_data)
save_model_data(model, file_path='saved_models',  model_name='vgg16')
```

Listing 4.3: Training loop

4.2.5. Model evaluation

At the end of the training loop the model was evaluated on the test set. It is one of the most important steps and informs about the performance of the model, on data it has not seen before. Here the importance of the data preparation mentioned in chapter 3 is once again highlighted. By making sure no duplicates and no unidentifiable images are in the test set the result of the evaluation is actually what it promises. Nevertheless, to further investigate the performance of trained models on data the test script test.py was implemented. The main purpose of this script is to load a model, test it on several images and document its performance. Additionally, the tested images are sorted depending on their classification result and heat maps of the activation of the last convolutional layer are created through a GradCam algorithm, which will be explained later.

4.3. Reports

A look at table 3.3 hints already to the number of models that have to be trained. If a model is then later used to classify images through the test.py script, keeping track of the accumulated information becomes a challenging task. It is important to be able to distinguish with what data set a model was trained and how it performed on the evaluation on the test set. Also, if a specific model is then used to classify images through the test.py scrip it is also of importance to document what model was used, what data was used to train the model, what number of images were used to test the model on, and most importantly the

performance metrics the model reached. The amount of data can soon become difficult to manage and information might get lost or become uninterpretable. To solve this problem a report generation functionality was implemented and added to the training and testing of a model. A report is a collection of HTML files that, when opened, display meta information, results, and visualizations.

4.3.1. Train report

The report generated after training separates the information onto three HTML pages, which makes it possible to open the report in a web browser and navigate through it like on a web page. The information page displays the used optimizer and its configured parameters. Below is information about the batch size used, the maximum number of epochs and the value of the early stopping patience. Next a graphical representation of the CNN architecture is displayed with this it is possible to be perfectly clear in the future what CNN architecture was trained. At the end of the information page information about the data used for training can be found. Depending on the directory name a useful data set name is shown followed by the total size of the data, the split ratio, and the resulting sizes for train-, validation-, and test-set. Given this information it should be possible to infer all necessary information about a model used at a later point of time.
Next is the results page on which the results of the evaluation of the model on the test set are summarized. They consist of the metrics defined during model compiling and are as followed:
1) the loss, 2) accuracy, 3) recall, 4) precision, 5) area under the curve (AUC) and the 6) F1 score. Also, the development of these metrics compared to the validation-set is given through line graphs and give another view of the training process. To get a feeling of how long this particular training run took the total run time and epochs are also documented.
The last page only contains the images which were used throughout the report.

4.3.2. Test report

It was already mentioned that a test script was implemented to test a specific model. While it was explained what it generally does its main purpose is also to document the results in an structured form. This test report is one of the main tools to analyze trained models. Once again, the first page is for displaying general information. This information offers important data about what model was used to classify what data. This makes later research possible without cumbersome investigations about what data and model were used at what point of time. For this, the first section displays the name of the model, the name of the data set used to train the model and its size. Below the evaluation results of the model generated through the keras evaluate method are displayed. Followed by the classification results of the model on the images. The results provide the following information: 1) the total number of classified images, 2) how many of them were classified as true positives, 3) how many of them were classified as true negatives, 4) how many of them were classified as false positive and 5) how many of them were classified as false negatives. Included are example images of each classified category. One disadvantage of CNNs is that it is quite difficult to infer what lead to the output of a specific prediction. Why did it classify an image as a false positive? One way to investigate its decision making is through a Gradient-weighted Class Activation Mapping (Grad-Cam) algorithm. This mapping uses gradient information from the last convolutional layer to create a heat map, visualizing what region of the image lead to the predicted classification [49]. With this it becomes possible to deepen investigation on what part of a image lead to the classification. Examples of these images can be found on the second page of the test report and in figure 4.1. While Grad-Cam uses the last convolutional

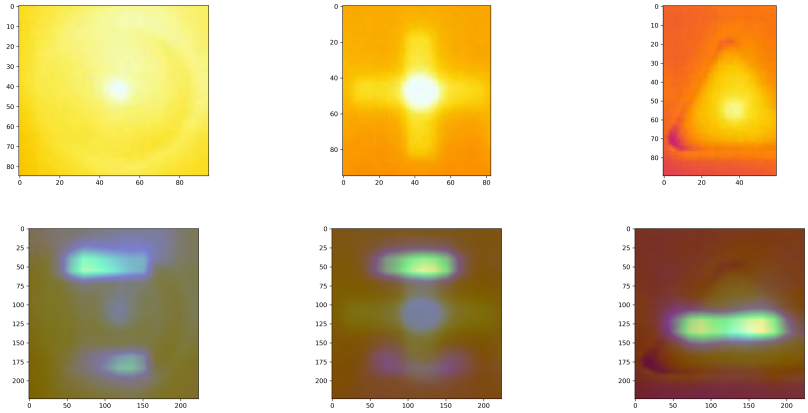


Figure 4.1.: Examples of original and grad-cam images

layer it might also be interesting what kind of features all other convolutional layers were extracting. These so-called feature maps can be seen on the last page of the report. There a random image was chosen and its feature maps, during classification, were extracted. To keep the report short and concise only example images are shown but especially in the case of the Grad-Cam images all images are of interest. All of the images classified can therefore be found as part of the test report folder in an image folder. This image folder contains sub folders named after the classification results like true positive, true negative, false positive and false negative. In each of these folders the respective classified images can be found. Also, the Grad-Cam images of them are located inside a grad_cam folder. Examples of both reports can be found in Appendix B.

4.4. Experiment procedure

With the reports in place the management of experiment data becomes feasible and build a strong foundation for documentation of the executed experiments. There are four broad categories in which the experiments can be divided into:

1. First the CNN architecture was trained with all 12 "dirty" data sets, resulting into 12 trained models.

2. Next the architecture was trained on the 12 "clean" data sets and the resulting models, and their performance could be compared to its dirty counterpart.

3. After the second step future experiments were done in order to try and increase the performance of models, trained on clean data sets without duplicates, with the help of fine tuning.

4. In the end the best models of each fine tuning process were tested through the test.py script to generate a test report and GradCam images to investigate their performance on other data.

All experiments were executed on a Dell Mobile Precision Workstation 7760 CTO, borrowed from the FH Campus Vienna. This high-performance laptop was powered through a 11th Gen Intel Core i9-1190H with 8 cores from 2.6 GHz to 5GHz. Memory consisted of 4x16 RAM and a local hard drive with a size of close to 1000GB. The GPU installed was a NVIDIA

RTX A5000 with 16GB working memory. The operating system used was the Windows 10 enterprise edition.

5. Results

In this chapter the results of the experiments are presented. As mentioned in section 4.4 the experiments can be separated into four categories and are therefore also presented in that order. At first the results of the trained models on the "dirty" data set are shown. Followed by the training results on the "clean" data set. Next the results of model fine tuning are presented. Important is that for these first three sections the performance metrics were each generated on the test set of the corresponding data set. To further investigate the trained models, ability to generalize, each model which accomplished the highest accuracy and F1 score during fine tuning was then used to evaluate the test sets of the other data sets.
Also important to mention is that while multiple performance metrics are extracted during evaluation, presented are only the accuracy and F1 score of the models. This was done to keep the tables clear and concise. All other metrics can be found in the corresponding train and test reports. And are as follows:

Accuracy: describes how well it performs across all classes.

Loss: Is the value of the loss that is tried to be minimized during training.

Precision: describes the ratio between positive samples correctly classified in respect to all positive classified samples.

Recall: describes the ratio between actual positive samples correctly classified in respect to all positive classified samples.

AUC: is the area under the ROC curve, the ROC curve depicts the rate of true positives and false positives and auc is the area below this curve which gives insight into the ability of a classifier to distinguish between classes.

F1: is the harmonic mean of the combination of precision and recall.

Also to get a feeling of how long each model took to train on each data set, the total train time and number of epochs are also shown. All models were trained with the same hyperparameters which can be seen in table 4.1.

5.1. Results of model trained with "dirty" data sets

Table 5.1 displays the results of the CNN architectures trained on the "dirty" data sets. Each orange colored row describes the pre-processing of the data set used to train the models. The first column further informs what data set was used described by their time of recording. Followed by the columns containing the performance metrics accuracy, F1 score, and the epochs and time taken to train each separate model.
It can be seen that generally performance is higher for data sets only down sampled. Also, except the very small data set from 06.04.2022 the data sets containing multiple geometries have higher accuracy's then data sets containing patches. The last section of the table shows especially that the models trained on the data set containing multiple geometries which were only down sampled had the best performance on their test set. But as mentioned in section

3.3.3 these performances are probably a result of duplicates which could have ended up in both the train split and the test split of the data set. This of course increases the evaluation result of the models.

Patches - no duplicates "dirty"				
	Accuracy	F1 score	Epochs	Time
06.04.2022	0.7388	0.646	39	~00:54:00
23.03.2022	0.821	0.8466	71	~11:54:00
11.03.2022	0.72	0.77	78	~03:25:00
Multiple Geometries - no duplicates "dirty"				
	Accuracy	F1 score	Epochs	Time
06.04.2022	0.527	0.679	39	~00:04:00
23.03.2022	0.821	0.846	50	~00:15:00
11.03.2022	0.939	0.942	35	~00:06:00
Patches - down sampled "dirty"				
	Accuracy	F1 score	Epochs	Time
06.04.2022	0.852	0.827	39	~00:54:00
23.03.2022	0.7535	0.7915	78	~27:30:00
11.03.2022	0.667	0.730	71	~05:00:00
Multiple Geometries - down sampled "dirty"				
	Accuracy	F1 score	Epochs	Time
06.04.2022	0.728	0.993	27	~00:05:00
23.03.2022	0.979	0.980	74	~03:00:00
11.03.2022	0.993	0.993	57	~00:29:00

Table 5.1.: Performance parameters of models trained and evaluated on "dirty" data sets

5.2. Results of model trained with "clean" data sets

The table 5.2 in this section now displays the results of the CNN architectures trained on the "clean" data sets. As before, but this time in blue, the first rows describe the pre-processing applied on the data sets, while the columns hold information about data sets and their accuracy, F1 score, epochs and train time.
Once again performance of models trained with only down sampled data is quite high. The models trained on the data sets from 06.04.2022 and 23.03.2022 containing multiple geometries with no duplicates performed quite badly with an accuracy of only 50%. One reason for this poor performance is probably the small size of the data sets as can be seen in table 3.3. As described in the work of Halevy Alon and his team more data increases performance and reduces overfitting [50], based on this more data is almost always better. This becomes more evident when the models trained on multiple geometries - down sampled are compared to the ones trained on multiple geometries - no duplicates. Without removing the duplicates the data set from 06.04.2022 isn't that much bigger then without them and its performance is also only slightly better. On the other hand, the data from 23.03.2022 is almost ten times as much than without duplicates and reaches close to 97% accuracy. But once again another factor could be the duplicates located in both the train and test split of the data set which further increases these metrics.

Patches - no duplicates "clean"				
	Accuracy	F1 score	Epochs	Time
06.04.2022	0.9019	0.9102	100	∼02:00:00
23.03.2022	0.9053	0.9114	100	∼16:28:00
11.03.2022	0.9359	0.9386	67	∼03:08:00
Multiple Geometries - no duplicates "clean"				
	Accuracy	F1 score	Epochs	Time
06.04.2022	0.5	0.6666	48	∼00:05:00
23.03.2022	0.5	0.6666	30	∼00:10:00
11.03.2022	0.9848	0.9846	100	∼00:16:00
Patches - down sampled "clean"				
	Accuracy	F1 score	Epochs	Time
06.04.2022	0.9862	0.9864	45	∼01:00:00
23.03.2022	0.8630	0.8788	47	∼17:38:00
11.03.2022	0.7437	0.7951	93	∼06:40:00
Multiple Geometries - down sampled "clean"				
	Accuracy	F1 score	Epochs	Time
06.04.2022	0.6785	0.7513	24	∼00:05:00
23.03.2022	0.969	0.970	51	∼02:10:00
11.03.2022	0.997	0.9977	94	∼00:49:00

Table 5.2.: Performance parameters of models trained and evaluated on "clean" data sets

5.2.1. Comparison

The comparison of all models trained on "dirty" and "clean" data can be seen in table 5.3. There it can be clearly seen that besides the already mentioned models, all architectures trained on the cleaned data sets have higher accuracy and a higher F1 score then their "dirty" counterpart. Besides this, the results from models trained on the data sets without duplicates are much more believable by making sure that no images used for training contaminate the test split.

5.3. Results of model fine tuning

Fine tuning is as already mentioned in section 2.2.1 a common step when using pretraining and transfer training. It refers to the procedure of freezing layers so that the weights do not get updated during training. The main idea behind this is that by using the trained feature maps from the pretrained model the new custom model, first of all does not have to train them itself and secondly might even extract features which it would not have learned on its own. Based on the different domains more or less layers can be frozen and might lead to cases were only the classifier has to be trained at all, if even [16].
This procedure was also taken advantage of in this work. For this only the cleaned data sets without duplicates were used, since on one hand their results were better and more importantly more believable without duplicates. The VGG-16 architecture has 13 convolutional layers which are responsible for creating the feature maps and can be frozen. Because of this, 13 models were trained on one data set each while successively freezing one convolutional layer after another. This could easily be accomplished by adding the command line argument "–freeze" to the train.py script.

		Accuracy		F1 Score	
		dirty	clean	dirty	clean
patches - no duplicates	06.04.2022	0.7388	0.9019	0.646	0.9102
	23.03.2022	0.821	0.9053	0.8466	0.9114
	11.03.2022	0.72	0.9359	0.77	0.9386
multiple geometries - no duplicates	06.04.2022	0.527	0.5	0.679	0.6666
	23.03.2022	0.821	0.5	0.846	0.6666
	11.03.2022	0.939	0.9848	0.942	0.9846
patches - down sampled	06.04.2022	0.852	0.9862	0.827	0.9864
	23.03.2022	0.7535	0.8630	0.7915	0.8788
	11.03.2022	0.667	0.7437	0.730	0.7951
multiple geometries - down sampled	06.04.2022	0.728	0.6785	0.993	0.7513
	23.03.2022	0.979	0.969	0.980	0.970
	11.03.2022	0.993	0.997	0.993	0.9977

Table 5.3.: Comparison of performance parameters of models trained and evaluated on "dirty" and "clean" data sets

```
python train.py --data-dir data/patches_06.04_no_duplicates --freeze 1
```

Listing 5.1: Example of training a model with the first conv layer frozen

5.3.1. Summary of fine-tuning results

In table 5.4 the summarized results of the fine tuning are given. The complete results of each model can be found in Appendix C. Displayed are the models which reached the highest accuracy and F1 score on their respective test split. The number of frozen convolutional layers is displayed in the first column, the second column shows the name of the deepest frozen convolutional layer while the other columns display the already know metrics like accuracy, F1 score, epochs and time. As the table shows, accuracy and F1 could be increased for all data sets except the data set from 11.03.2022. Especially the performance from models trained on multiple geometries increased from 50% up to 98% for the 06.04.2022 and to 97% for the 23.03.2022.

patches - no duplicates - clean						
	number of conv layers frozen	Frozen conv layers	Accuracy	F1 score	Epochs	Time
06.04.2022	2/13	block1_conv2	0.9891	0.9892	52	~01:02:00
23.03.2022	10/13	block4_conv3	0.9758	0.9760	100	~14:29:00
11.03.2022	1/13	block1_conv1	0.8924	0.9015	95	~04:20:00
multiple geometries - no duplicates - clean						
06.04.2022	2/13	block1_conv2	0.9861	0.9859	41	~00:04:00
23.03.2022	5/13	block3_conv1	0.8992	0.8879	35	~00:11:00
11.03.2022	10/13	block4_conv3	0.8409	0.8292	49	~00:08:00

Table 5.4.: Best results of fine tuning procedure

5.4. Results of testing model on different test sets

While each model performed well when tested on their own test set, a question left open is, how well a model performs on the test sets of different data sets. To answer this question the best performing models, as seen in table 5.5, were selected and used in the test.py script to classify the test sets of all data sets (its own included). Each model was therefore evaluated on 11 "clean" test sets. The data sets were reduced to 11 because the down sampled test set from 23.03.2022 was too big for grad cam conversion and stopped the test.py script unexpectedly. Since it basically consists of the same data as its no duplicate counterpart it was removed from evaluation. The down sampled data sets only differed in their size, which led to almost the same results. But for the sake of completeness, they are also listed in table 5.6.

patches - no duplicates - clean						
	number of conv layers frozen	Frozen conv layers	Accuracy	F1 score	Epochs	Time
06.04.2022	2/13	block1_conv2	0.9891	0.9892	52	∼01:02:00
23.03.2022	10/13	block4_conv3	0.9758	0.9760	100	∼14:29:00
11.03.2022	0/13	none	0.9359	0.9386	67	∼03:08:00
multiple geometries - no duplicates - clean						
06.04.2022	2/13	block1_conv2	0.9861	0.9859	41	∼00:04:00
23.03.2022	5/13	block3_conv1	0.8992	0.8879	35	∼00:11:00
11.03.2022	0/13	none	0.9848	0.942	100	∼00:16:00

Table 5.5.: Best performing models

5.4.1. Comparison of evaluation results

In table 5.6 the results of the evaluation can be seen. In the blue row it is described with what data set the model was trained and what the last frozen convolutional layer was during training. Bellow the test splits and their corresponding accuracy and F1 score values, which were reached through the model, are displayed. Also, on the left side the models trained on patches are listed, while models trained on multiple geometries can be found on the right side of the table. For better visual differentiation the data set with the higher values were colored in orange.
This makes it first of all easy to notice that a model trained on patches generally has higher accuracy and F1 on test sets containing only patches and vice versa. Another insight the table provides, is that, as already mentioned, the rather poor performance on other test sets. Other than the test set belonging to each model, performance on other test sets were generally between 40% to 70% and sometimes even below 5%.

5.4.2. Testing models with no frozen layers during training

During further analysis of table 5.6 it was noticed that the evaluation results achieved through the model trained on the data from 11.03.2022 were over all much better. This was especially surprising since the data from 06.04.2022 and 11.03.2022 is very similar. Prior to evaluation it was therefore assumed that the results would be similar. What differentiated the model trained on data from 11.03.2022 to the other models was that no better results were reached during fine tuning so the model without it was used. This led to the suspicion that while fine tuning increased performance on the specific data used for training and evaluation, lead to worse generalization of other data. To investigate this the models trained on patches

without fine tuning, were used again for classification and evaluation on all other test sets. For comparison the tables of the fine tuned and not fine tuned models are put next to each other in table 5.7. And once again to make it easier to see which model had a better F1 score, it was marked orange. Since the model trained on data from 11.03.2022 was already not fine tuned both sides hold the same values. As suspected almost all models trained without fine tuning perform better on other test splits than the ones with fine tuning.

Model trained on 06.04.2022 - patches no duplicates - frozen conv layer block1_conv2			Model trained on 06.04.2022 - multiple geometries no duplicates - frozen conv layer block1_conv2		
Test Split	Accuracy	F1 Score	Test Split	Accuracy	F1 Score
06.04.2022 - patches no duplicates	0.9891	0.9892	06.04.2022 - patches no duplicates	0.6261	0.4080
06.04.2022 - multiple geometries no duplicates	0.5694	0.6990	06.04.2022 - multiple geometries no duplicates	0.98611	0.9859
06.04.2022 - patches down sampled	0.9870	0.9872	06.04.2022 - patches down sampled	0.5799	0.2896
06.04.2022 - multiple geometries down sampled	0.5786	0.7035	06.04.2022 - multiple geometries down sampled	0.9857	0.9855
23.03.2022 - patches no duplicates	0.6024	0.6975	23.03.2022 - patches no duplicates	0.4979	0.6626
23.03.2022 - multiple geometries no duplicates	0.4961	0.3689	23.03.2022 - multiple geometries no duplicates	0.5968	0.3246
23.03.2022 - multiple geometries down sampled	0.1719	0.2077	23.03.2022 - multiple geometries down sampled	0.6446	0.4486
11.03.2022 - patches no duplicates	0.5490	0.6876	11.03.2022 - patches no duplicates	0.6443	0.4794
11.03.2022 - multiple geometries no duplicates	0.5227	0.6769	11.03.2022 - multiple geometries no duplicates	0.8030	0.8333
11.03.2022 - patches down sampled	0.5332	0.6805	11.03.2022 - patches down sampled	0.6656	0.5159
11.03.2022 - multiple geometries down sampled	0.5067	0.6696	11.03.2022 - multiple geometries down sampled	0.7959	0.8305
Model trained on 23.03.2022 - patches no duplicates - frozen conv layer block4_conv3			Model trained on 23.03.2022 - multiple geometries no duplicates - frozen conv layer block3_conv1		
Test Split	Accuracy	F1 Score	Test Split	Accuracy	F1 Score
06.04.2022 - patches no duplicates	0.5317	0.5699	06.04.2022 - patches no duplicates	0.6479	0.4581
06.04.2022 - multiple geometries no duplicates	0.1805	0.3058	06.04.2022 - multiple geometries no duplicates	0.6111	0.3636
06.04.2022 - patches down sampled	0.4499	0.4538	06.04.2022 - patches down sampled	0.6260	0.4056
06.04.2022 - multiple geometries down sampled	0.1357	0.2091	06.04.2022 - multiple geometries down sampled	0.5928	0.3132
23.03.2022 - patches no duplicates	0.9757	0.9760	23.03.2022 - patches no duplicates	0.5131	0.0533
23.03.2022 - multiple geometries no duplicates	0.4147	0.2176	23.03.2022 - multiple geometries no duplicates	0.8992	0.8879
23.03.2022 - multiple geometries down sampled	0.1646	0.2021	23.03.2022 - multiple geometries down sampled	0.8503	0.8244
11.03.2022 - patches no duplicates	0.5102	0.5972	11.03.2022 - patches no duplicates	0.4319	0.5568
11.03.2022 - multiple geometries no duplicates	0.4090	0.3606	11.03.2022 - multiple geometries no duplicates	0.6666	0.5000
11.03.2022 - patches down sampled	0.5385	0.6242	11.03.2022 - patches down sampled	0.6106	0.4536
11.03.2022 - multiple geometries down sampled	0.3811	0.2886	11.03.2022 - multiple geometries down sampled	0.6345	0.4240
Model trained on 11.03.2022 - patches no duplicates - frozen conv layer None			Model trained on 11.03.2022 - multiple geometries no duplicates - frozen conv layer None		
Test Split	Accuracy	F1 Score	Test Split	Accuracy	F1 Score
06.04.2022 - patches no duplicates	0.9664	0.9667	06.04.2022 - patches no duplicates	0.6905	0.7256
06.04.2022 - multiple geometries no duplicates	0.8611	0.8648	06.04.2022 - multiple geometries no duplicates	0.6666	0.7500
06.04.2022 - patches down sampled	0.9491	0.9501	06.04.2022 - patches down sampled	0.6849	0.7198
06.04.2022 - multiple geometries down sampled	0.8142	0.7651	06.04.2022 - multiple geometries down sampled	0.7714	0.8139
23.03.2022 - patches no duplicates	0.6834	0.6997	23.03.2022 - patches no duplicates	0.6355	0.4719
23.03.2022 - multiple geometries no duplicates	0.4263	0.1777	23.03.2022 - multiple geometries no duplicates	0.7364	0.7914
23.03.2022 - multiple geometries down sampled	0.1021	0.0725	23.03.2022 - multiple geometries down sampled	0.6040	0.7148
11.03.2022 - patches no duplicates	0.9359	0.9386	11.03.2022 - patches no duplicates	0.7068	0.6998
11.03.2022 - multiple geometries no duplicates	0.5454	0.6703	11.03.2022 - multiple geometries no duplicates	0.9848	0.9846
11.03.2022 - patches down sampled	0.9399	0.9426	11.03.2022 - patches down sampled	0.6626	0.6595
11.03.2022 - multiple geometries down sampled	0.5112	0.6614	11.03.2022 - multiple geometries down sampled	0.9865	0.9863

Table 5.6.: Comparison of model evaluation

5.4.3. Results of architectures trained on combined data

Besides training CNN architectures on the single data sets, three models were trained on combinations of all data sets. Specifically one architecture was trained on a data set containing all patches from 06.04.2022, 23.03.2022 and 11.03.2022, one architecture was trained on all multiple geometries of these dates and the third architecture was trained on all patches and multiple geometries including the one from 19.11.2021 which were used for BA1. Only the cleaned and no duplicate versions were used in these combinations. Also no fine tuning was done because of the size of the data sets, which took often more than 20 hours to train. With an F1 score of 76.72% the model trained on the patches had the highest score followed by the one trained on multiple geometries with an F1 score of 76.11%. The architecture

Model trained on 06.04.2022 - patches no duplicates - frozen conv layer block1_conv2			Model trained on 06.04.2022 - patches no duplicates		
Test Split	Accuracy	F1 Score	Test Split	Accuracy	F1 Score
06.04.2022 - patches no duplicates	0.9891	0.9892	06.04.2022 - patches no duplicates	0.9972	0.9972
06.04.2022 - multiple geometries no duplicates	0.5694	0.6990	06.04.2022 - multiple geometries no duplicates	0.6944	0.7659
06.04.2022 - patches down sampled	0.9870	0.9872	06.04.2022 - patches down sampled	0.9983	0.9983
06.04.2022 - multiple geometries down sampled	0.5786	0.7035	06.04.2022 - multiple geometries down sampled	0.6071	0.7179
23.03.2022 - patches no duplicates	0.6024	0.6975	23.03.2022 - patches no duplicates	0.5892	0.6351
23.03.2022 - multiple geometries no duplicates	0.4961	0.3689	23.03.2022 - multiple geometries no duplicates	0.4323	0.1878
23.03.2022 - multiple geometries down sampled	0.1719	0.2077	23.03.2022 - multiple geometries down sampled	0.1113	0.0905
11.03.2022 - patches no duplicates	0.5490	0.6876	11.03.2022 - patches no duplicates	0.5929	0.7069
11.03.2022 - multiple geometries no duplicates	0.5227	0.6769	11.03.2022 - multiple geometries no duplicates	0.5530	0.6910
11.03.2022 - patches down sampled	0.5332	0.6805	11.03.2022 - patches down sampled	0.5615	0.6931
11.03.2022 - multiple geometries down sampled	0.5067	0.6696	11.03.2022 - multiple geometries down sampled	0.5156	0.6737
Model trained on 23.03.2022 - patches no duplicates - frozen conv layer block4_conv3			Model trained on 23.03.2022 - patches no duplicates		
Test Split	Accuracy	F1 Score	Test Split	Accuracy	F1 Score
06.04.2022 - patches no duplicates	0.5317	0.5699	06.04.2022 - patches no duplicates	0.7114	0.7561
06.04.2022 - multiple geometries no duplicates	0.1805	0.3058	06.04.2022 - multiple geometries no duplicates	0.2777	0.2972
06.04.2022 - patches down sampled	0.4499	0.4538	06.04.2022 - patches down sampled	0.7092	0.7537
06.04.2022 - multiple geometries down sampled	0.1357	0.2091	06.04.2022 - multiple geometries down sampled	0.3214	0.2016
23.03.2022 - patches no duplicates	0.9757	0.9760	23.03.2022 - patches no duplicates	0.9053	0.9114
23.03.2022 - multiple geometries no duplicates	0.4147	0.2176	23.03.2022 - multiple geometries no duplicates	0.3178	0.2903
23.03.2022 - multiple geometries down sampled	0.1646	0.2021	23.03.2022 - multiple geometries down sampled	0.5989	0.4411
11.03.2022 - patches no duplicates	0.5102	0.5972	11.03.2022 - patches no duplicates	0.5265	0.6661
11.03.2022 - multiple geometries no duplicates	0.4090	0.3606	11.03.2022 - multiple geometries no duplicates	0.6515	0.6406
11.03.2022 - patches down sampled	0.5385	0.6242	11.03.2022 - patches down sampled	0.5222	0.6649
11.03.2022 - multiple geometries down sampled	0.3811	0.2886	11.03.2022 - multiple geometries down sampled	0.6233	0.5778
Model trained on 11.03.2022 - patches no duplicates - frozen conv layer None			Model trained on 11.03.2022 - patches no duplicates		
Test Split	Accuracy	F1 Score	Test Split	Accuracy	F1 Score
06.04.2022 - patches no duplicates	0.9664	0.9667	06.04.2022 - patches no duplicates	0.9664	0.9667
06.04.2022 - multiple geometries no duplicates	0.8611	0.8648	06.04.2022 - multiple geometries no duplicates	0.8611	0.8648
06.04.2022 - patches down sampled	0.9491	0.9501	06.04.2022 - patches down sampled	0.9491	0.9501
06.04.2022 - multiple geometries down sampled	0.8142	0.7651	06.04.2022 - multiple geometries down sampled	0.8142	0.7651
23.03.2022 - patches no duplicates	0.6834	0.6997	23.03.2022 - patches no duplicates	0.6834	0.6997
23.03.2022 - multiple geometries no duplicates	0.4263	0.1777	23.03.2022 - multiple geometries no duplicates	0.4263	0.1777
23.03.2022 - multiple geometries down sampled	0.1021	0.0725	23.03.2022 - multiple geometries down sampled	0.1021	0.0725
11.03.2022 - patches no duplicates	0.9359	0.9386	11.03.2022 - patches no duplicates	0.9359	0.9386
11.03.2022 - multiple geometries no duplicates	0.5454	0.6703	11.03.2022 - multiple geometries no duplicates	0.5454	0.6703
11.03.2022 - patches down sampled	0.9399	0.9426	11.03.2022 - patches down sampled	0.9399	0.9426
11.03.2022 - multiple geometries down sampled	0.5112	0.6614	11.03.2022 - multiple geometries down sampled	0.5112	0.6614

Table 5.7.: Comparison of fine tuned and not fine tuned models

trained on all combined data sets only reached a F1 score of 64% as can be seen in table 5.8. While the results were not that great a evaluation on all other test sets was still done for the sake of completes. Their own test set was excluded since the size would have resulted in an out of memory error while creating grad cam images. In table 5.9 the reached accuracy and F1 scores of the model trained on all patches is compared with the model trained on all multiple geometries. While table 5.10 showcases the results of the model trained on all data.

	Accuracy	F1 score	Epochs	Time
multiple geometries combined - no duplicates	0.6861	0.7611	52	~28:57:00
patches combined - no duplicates	0.7012	0.7672	99	~21:16:00
all cleaned data sets with no duplicates including 19.11.2021	0.6828	0.6489	81	~80:15:00

Table 5.8.: Results of trained CNN architectures on combined data sets

Model trained on multiple patches combined			Model trained on multiple geometries combined		
Test Split	Accuracy	F1 Score	Test Split	Accuracy	F1 Score
06.04.2022 - patches no duplicates	0.8212	0.8481	06.04.2022 - patches no duplicates	0.6451	0.7201
06.04.2022 - multiple geometries no duplicates	0.5138	0.6391	06.04.2022 - multiple geometries no duplicates	0.7500	0.8000
06.04.2022 - patches down sampled	0.8271	0.8518	06.04.2022 - patches down sampled	0.6357	0.7077
06.04.2022 - multiple geometries down sampled	0.5214	0.6417	06.04.2022 - multiple geometries down sampled	0.7357	0.7909
23.03.2022 - patches no duplicates	0.6542	0.7368	23.03.2022 - patches no duplicates	0.5223	0.5424
23.03.2022 - multiple geometries no duplicates	0.4806	0.6436	23.03.2022 - multiple geometries no duplicates	0.7286	0.7865
23.03.2022 - multiple geometries down sampled	0.4835	0.6512	23.03.2022 - multiple geometries down sampled	0.9197	0.9256
11.03.2022 - patches no duplicates	0.7444	0.7946	11.03.2022 - patches no duplicates	0.5537	0.6606
11.03.2022 - multiple geometries no duplicates	0.5151	0.6666	11.03.2022 - multiple geometries no duplicates	0.7196	0.7810
11.03.2022 - patches down sampled	0.7124	0.7759	11.03.2022 - patches down sampled	0.5595	0.6643
11.03.2022 - multiple geometries down sampled	0.5044	0.6646	11.03.2022 - multiple geometries down sampled	0.6524	0.7420

Table 5.9.: Comparison of evaluation results of models trained on all patches and all multiple geometries

5.5. Deeper analysis of the results

To get a feeling of how an accuracy and F1 score percentage correlate to classified images the models were analyzed more in depth with the help of the test report. Since a lot of data was generated through the evaluation of the models on the separate test splits only a selected few were chosen to be represented in this work. Since the models trained on the data from 11.03.2022 have one of the highest and consistent results across all models the two models trained on patches and on multiple geometries were selected.

5.5.1. Analysis of test data from 06.04.2022 patches no duplicates

Model trained on patches from 11.03.2022

The evaluation of the model trained on patches from 11.03.2022 on the test set of the data from 06.04.2022 resulted in an accuracy of 96.64% and a F1 score of 96.67%. This means that from the 1102 images classified 537 were classified as true positives, 515 were classified as true negatives, 10 images were classified as false negatives and 24 images were classified as false positives.

Model trained on multiple geometries from 11.03.2022

Contrary the model trained on multiple geometries on the same test set resulted in an accuracy of 69.05% and a F1 score of 72.56%. This means that from the 1102 images classified 448 were classified as true positives, 259 were classified as true negatives, 78 images were classified as false negatives and 234 images were classified as false positives. From these numbers it can easily be seen that this model trained on multiple geometries has a problem with recognizing curling on patches.

5.5.2. Analysis of test data from 06.04.2022 multiple geometries no duplicates

Model trained on patches from 11.03.2022

On the multiple geometries data from 06.04.2022 the model trained on patches reached an accuracy of 86.11% and a F1 score of 86.48%. Resulting in a classification of the 72 images into 32 classified as true positives, 29 as true negatives, 3 as false negatives and 6 as false positives. While it is a bit difficult to interpret the results on such little data it can still be seen that this model has again problems of recognizing curling.

Model trained on all data		
Test Split	Accuracy	F1 Score
06.04.2022 - patches no duplicates	0.5789	0.3151
06.04.2022 - multiple geometries no duplicates	0.9027	0.9066
06.04.2022 - patches down sampled	0.5420	0.2285
06.04.2022 - multiple geometries down sampled	0.8788	0.8874
23.03.2022 - patches no duplicates	0.5749	0.6337
23.03.2022 - multiple geometries no duplicates	0.5736	0.6892
23.03.2022 - multiple geometries down sampled	0.6861	0.7474
11.03.2022 - patches no duplicates	0.5257	0.1871
11.03.2022 - multiple geometries no duplicates	0.6212	0.7058
11.03.2022 - patches down sampled	0.5335	0.1831
11.03.2022 - multiple geometries down sampled	0.6143	0.7142

Table 5.10.: Evaluation results of model trained on all data

Model trained on multiple geometries from 11.03.2022

While generally models trained on multiple geometries perform better on images of multiple geometries, this model is an exception. The evaluation on the multiple geometries test set from 06.04.2022 resulted in an accuracy of 66.66% and a F1 score of 75.00%. From the 72 images 36 were classified as true positives, 12 as true negatives, 0 as false negatives and 23 as false positives. While the results were worse the trend that curling is not recognized is continued.

5.5.3. Analysis of test data from 23.03.2022 patches no duplicates

Model trained on patches from 11.03.2022

On the test split from the data of the 23.03.2022 patches no duplicates the model trained on patches from 11.03.2022 reached an accuracy of 68.34% and a F1 score of 69.97%. In numbers this means that from the 8744 images contained in the test set 3156 were classified as true positives, 2521 as true negatives, 875 as false negatives and 1575 as false positives. Leading once again to the conclusion that this model has problems recognizing curling.

Model trained on multiple geometries from 11.03.2022

The model trained on the multiple geometries reached an accuracy of 63.55% and a F1 score of 47.19% on the test set from 23.03.2022 patches. From 8744 images 1247 were classified as true positives, 2936 as true negatives, 2090 as false negatives and 148 as false positives. While this model's performance is comparable to the models before, interestingly this models seems to recognize curling even if it is not present.

5.5.4. Analysis of test data from 23.03.2022 multiple geometries no duplicates

Model trained on patches from 11.03.2022

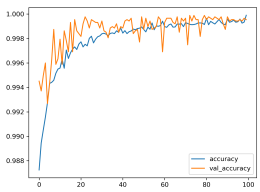
With an accuracy of 42.63% and a F1 score of only 17.77% this model is one of the worst performing models on this test set. From 258 images 15 were classified as true positives, 94 were classified as true negatives, 113 images were classified as false negatives and 35 images were classified as false positives.

Model trained on multiple geometries from 11.03.2022

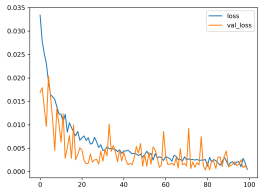
Contrary to the model trained on patches the one trained on multiple geometries reached an accuracy of 73.64% and a F1 score of 79.14%. This means that from the 258 images classified 129 were classified as true positives, 34 were classified as true negatives, 0 images were classified as false negatives and 63 images were classified as false positives. Interestingly this model classifies images wrong to contain curling as opposed to the model before which predicted curling on images that did not contain any.

5.5.5. Model trained on patches without duplicates from 23.03.2022 (without fine tuning)

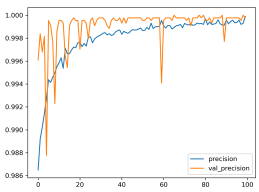
It was mentioned in 3.2.1 that the data from 23.03.2022 displays the frame in a different color scheme then the other data sets. Through this the component and the background are of similar color and therefore more difficult to distinguish from each other. The hypothesis was therefore that even more emphasis should be on curling and lead to CNN architectures that are even better at recognising it during classification. But contrary to this believe, expectations fell short as can be seen in table 5.7. To get a better understanding of what happened during classification the generated GradCam images will be presented.



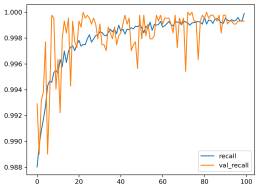
(a) accuracy behaviour during the training epochs of train and validation data



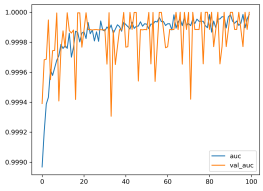
(b) loss behaviour during the training epochs of train and validation data



(c) precision behaviour during the training epochs of train and validation data



(d) recall behaviour during the training epochs of train and validation data



(e) auc behaviour during the training epochs of train and validation data

Figure 5.1.: Training performance metrics on non fine tuned model

Training details

But before that some more details of the accomplished metrics during model training are shown first. The model was trained on the data set from 23.03.2022 which was cleaned, had its duplicates removed and was down sampled. While the fine tuned version reached an accuracy of 97.57% and F1 score of 97.70% on its test set, the model without fine tuning reached only 90.53% in accuracy and 91.14% on its F1 score. But despite this the model without fine tuning performed better on the test sets of the other data sets. The behavior of the model during training and its resulting metrics can be seen in figure 5.1. As can be seen on the graphs that during training accuracy increased with each epoch while the loss was reduced until it was close to 100%. On the other hand, the evaluation after training on the test set only resulted in an accuracy of 90.53% as can be seen in figure 5.2. This behavior is an indication for overfitting. Which occurs when a model performs very well on the training's data in contrast to its test data [16].

Results

Results were generated on test set

LOSS:	ACCURACY:	RECALL:	PRECISION:	AUC:	F1:
42.47191619873047	0.9053065180778503	0.9748398661613464	0.855823278427124	0.9059813618659973	0.9114627698338345

Figure 5.2.: Evaluation of none fine tuned -model

GradCam Image Comparison

The difference of the fine tuned model and not fine tuned model is around 7%. The impact of these 7% can be seen by comparing their GradCam images. In figure 5.3 the generated GradCam images of the model with an accuracy of 90.53% can be seen. The images show frames correctly classified to contain curling. With the first row displaying the frame overlaid with the generated heat map and the original image in the second row bellow. Contrary to the expectation that the edges of the component where curling is prominent, would be highlighted, the model seems to classify based on the background, and even more surprisingly based on top of the image.

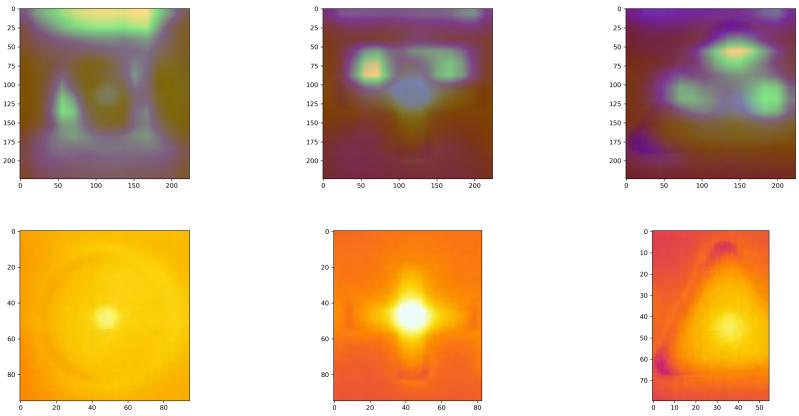


Figure 5.3.: Model without fine tuning generated GradCam images with curling present

This becomes even stranger when compared to GradCam images generated through the model with an accuracy of 97.57% due to fine tuning. Figure 5.4 once again displays im-

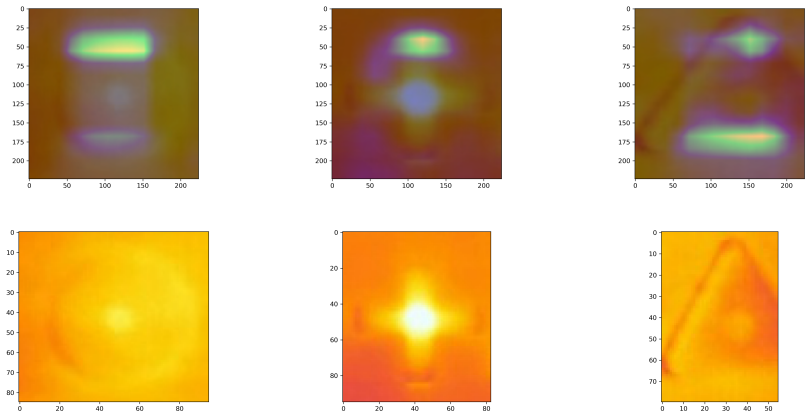


Figure 5.4.: Model with fine tuning generated GradCam images with curling present

ages classified correctly to contain curling. This time the edges which were expected to be highlighted are highlighted.

6. Discussion

From the results multiple conclusions can be drawn. First of all, as the comparison of models trained on "dirty" and "clean" data shows 3.3, models trained on clean data generally reach higher performance metrics. The exception were mainly data sets which were reduced to a very small number due to cleaning. Besides the performance of the models also the trustworthiness of the generated results was increased. By removing duplicates and unidentifiable frames from the data set, it can be assured that no frames the model was trained on are in the test split.
Another insight gained from the results were the results of the fine tuned models. Since the domain of this data set was pretty different compared to the domain of the ImageNet data set used for pretraining, it was unexpected that fine tuning would result in such good results. Beside models trained on the data from 11.03.2022 all other model performances increased by at least 7%. While surprising it was assumed that the weights trained through the multiple classes from the ImageNet data set were able to extract features better than when trained again. This assumption was then once again thrown overboard. While the performance of the model increased on its own test split, its performance on the test splits of the other data sets decreased. This was luckily found out because no fine tuning of the data from 11.03.2022 led to better results and made the better classification ability on other test splits noticeable. The comparison of fine tuned models and non fine tuned models in table 5.7 carried out after this discovery also confirmed this. This means, while fine tuning increased performance on its own test set it reduced the model's ability to generalize. Which is also what sums up the general discovery after analyzing the results. All trained models performed well on their data set which was confirmed by evaluation on the cleaned test set. But generalization on other test sets could not reach these results. This is problematic in two ways:

1. In the end a model trained on this data should be able to correctly classify curling during a SLS printing process on components it hasn't "seen" before. A model which is able to generalize with high accuracy is therefore needed.

2. The other thing is that the other data sets were not that different from each other. While the data set from 23.03.2022 displayed a different color scheme, the components displayed were the same. Even more so for the data set from 06.04.2022 and 11.03.2022 where even the color scheme was the same.

Given the similarity of these two the low evaluation result of one model on the others test split was not expected at all.
This led to the next experiments, where models were trained on all data sets combined. Not only did the models perform rather poorly on other test splits even their own test split didn't bring good results as can be seen in table 5.8. Also visible in the table is that this process was very time consuming and therefore leaving no time to further experiment with fine tuning. Deeper analysis of the results generated through the model trained on patches from 11.03.2022 gave further insight into the classification results of the model. There, a trend that the model usually has more difficulties recognizing curling, meaning that generally the number of false positives outweigh the number of false negatives.
The generated GradCam images gave more insight into how the fine tuned model trained

on patches from 23.03.2022 made its decisions compared to the model train on the same data without fine tuning. Even though the performance of both was not that far of (97.57% to 90.53%) the GradCam images clearly show that the fine tuned model was better able to classify based on curling which occurs on the edges of a component. Yet in terms of generalizing the model without tuning performed better.

6.1. Summary

This thesis continued the topic of the first bachelor thesis [27], of classifying if a infrared image of a SLS printing process contains curling or not. While BA1 was concerned with testing different CNN architectures to find the best one, the goal of this work was to further increase the performance through processing of the data sets. For this purpose, the CNN architecture which performed best during BA1 was used on new data which was acquired from students of the HTM study course. Various preprocessing steps were performed on the data sets, the most important being, removing duplicates and unidentifiable images. The original data consisted of frames containing multiple geometries and by cropping the image even bigger data sets containing only component patches could be generated. After undergoing these preprocessing steps multiple CNN models were generated on preprocessed and unprocessed data sets. A comparison of these models clearly displayed the performance increase of almost all models trained with preprocessed data. The only exceptions were the ones where preprocessing reduced the data set to a very small number. Fine tuning further increased performance of single models. But while performance increased for models trained on each data set, generalization on the test sets of other data sets did not reach expected levels. Especially considering the similarity of each data set. Here a interesting discovery was that models where no fine tuning was performed generally were able to generalize better then models which were fine tuned. Nevertheless, it could still be shown that the implemented data preprocessing increased the performance of CNN models and therefore the goal of the thesis can be seen as reached.

6.2. Future work

While the goal of increasing performance was reached, it is important that the poor generalization performance of the models is investigated. After all a model should be able to classify curling correctly on components which it was not trained on. One possible direction to fix this is to look deeper into the architecture of the used CNN algorithm. For example, is a limitation of using a pretrained feature extractor that one has to use the same input format. With a more customized approach layers could be added or removed to the feature extractor. Also, the input image itself could be passed in a different format. A reason for this is that instead of using images with the color channels RGB, converting them into gray scale images could emphasize curling even more. Another promising direction is the usage of generative adversarial networks (GAN). A big advantage of these networks is that training of the network can be done without labeled data also called unsupervised. In the last few years GANs have become one of the most important networks in computer vision and especially anomaly detection could fit the problem of classifying curling (the anomaly). An even more advanced topic for further research would be to classify curling before it even occurs. A possible direction for this problem could be the use of CNNs, GANs and long short term memory networks (LSTMs). By training such a hybrid network on sequences of data it could be possible to generate new frames that actually depict the future. Until now only curling

was mentioned but of course there are many other possible defects which can occur during a SLS printing process. Trying to detect these is another open topic for possible research.

Bibliography

[1] S. Dunham, J. Warbrick, N. V. Es, C. Perla, V. Akinsowon, and M. Nahirna, "State of the 3d printing industry survey 2019 am service providers," AMFG/Autonomous Manufacturing, Tech. Rep., 2019. 1

[2] M. Monzón, Z. Ortega, A. Martínez, and F. Ortega, "Standardization in additive manufacturing: activities carried out by international organizations and projects," *The international journal of advanced manufacturing technology*, vol. 76, no. 5-8, pp. 1111–1121, 2015. 1

[3] Z. Yin, "Direct generation of extended stl file from unorganized point data," *Computer-Aided Design*, vol. 43, no. 6, pp. 699–706, 2011. 1

[4] D. Jordi, S. Lidia, M. Karla, and C. Joaquim, *Selective Laser Sintering.* 111 River Street, Hoboken, NJ 07030,USA: John Wiley & Sons inc., 2020, pp. 481–499. 1, 2, 41

[5] R. Goodridge, C. Tuck, and R. Hague, "Laser sintering of polyamides and other polymers," *Progress in Materials science*, vol. 57, no. 2, pp. 229–267, 2012. 1, 2

[6] M. Korpela, N. Riikonen, H. Piili, A. Salminen, and O. Nyrhilä, *Additive Manufacturing—Past, Present, and the Future.* Cham: Springer International Publishing, 2020, pp. 17–41. [Online]. Available: https://doi.org/10.1007/978-3-030-46103-4_2 2

[7] I. G. Ian Gibson, "Additive manufacturing technologies 3d printing, rapid prototyping, and direct digital manufacturing," 2015. 2

[8] K. McAlea, "Materials and applications for the selective laser sintering," in *Process, Proc. 7ˆ< th> Int. Conf. on Rapid Prototyping*, 1997, pp. 23–33. 2

[9] J.-P. Kruth, "Selective laser sintering of wc-co'hard metal'parts," in *Proc. of 8ˆ< th> Int. Conf. on Production Engineering (ICPE), Japan*, 1997, pp. 149–156. 2

[10] U. S. Bertoli, G. Guss, S. Wu, M. J. Matthews, and J. M. Schoenung, "In-situ characterization of laser-powder interaction and cooling rates through high-speed imaging of powder bed fusion additive manufacturing," *Materials & Design*, vol. 135, pp. 385–396, 2017. 2

[11] T. Mukherjee, H. Wei, A. De, and T. DebRoy, "Heat and fluid flow in additive manufacturing–part ii: Powder bed fusion of stainless steel, and titanium, nickel and aluminum base alloys," *Computational Materials Science*, vol. 150, pp. 369–380, 2018. 2

[12] J. Picker, "Aufnehmen und auswerten der temperatur-verteilung im bauraum einer sls-anlage," BA thesis, FH Campus Wien, Favoritenstraße 226 1100 Wien, Austria, 2021. 2, 5, 7, 9

[13] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961. 3

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 3

[15] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems, 2nd Edition.* " O'Reilly Media, Inc.", 2019, ch. 10. 3, 4, 8, 41

[16] M. Elgendy, *Deep learning for vision systems.* Simon and Schuster, 2020. 3, 8, 24, 32

[17] K. Fukushima, "A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, pp. 193–202, 1980. 3

[18] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989. 3

[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989. 3

[20] Q. Zhao and Z. Shang, "Deep learning and its development," *Journal of Physics: Conference Series*, vol. 1948, no. 1, p. 012023, jun 2021. [Online]. Available: https://doi.org/10.1088/1742-6596/1948/1/012023 3

[21] Z. Jin, Z. Zhang, and G. X. Gu, "Autonomous in-situ correction of fused deposition modeling printers using computer vision and deep learning," *Manufacturing Letters*, vol. 22, pp. 11–15, 2019. 4

[22] O. Kwon, H. G. Kim, M. J. Ham, W. Kim, G.-H. Kim, J.-H. Cho, N. I. Kim, and K. Kim, "A deep neural network for classification of melt-pool images in metal additive manufacturing," *Journal of Intelligent Manufacturing*, vol. 31, no. 2, pp. 375–386, 2020. 4

[23] A. Caggiano, J. Zhang, V. Alfieri, F. Caiazzo, R. Gao, and R. Teti, "Machine learning-based image processing for on-line defect recognition in additive manufacturing," *CIRP Annals*, vol. 68, no. 1, pp. 451–454, 2019. 4

[24] H. Baumgartl, J. Tomas, R. Buettner, and M. Merkel, "A deep learning-based model for defect detection in laser-powder bed fusion using in-situ thermographic monitoring," *Progress in Additive Manufacturing*, pp. 1–9, 2020. 4

[25] L. Le Roux, C. Liu, Z. Ji, P. Kerfriden, D. Gage, F. Feyer, C. Körner, and S. Bigot, "Automatised quality assessment in additive layer manufacturing using layer-by-layer surface measurements and deep learning," *Procedia CIRP*, vol. 99, pp. 342–347, 2021. 4

[26] E. Westphal and H. Seitz, "A machine learning method for defect detection and visualization in selective laser sintering based on convolutional neural networks," *Additive Manufacturing*, vol. 41, p. 101965, 2021. 4, 5

[27] M. Schmid-Kietreiber, "In situ detection of printing defects during a sls manufacturing process using deep learning," BA thesis, FH Campus Wien, Favoritenstraße 226 1100 Wien, Austria, 2022. 5, 35

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. 7

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012. 7

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. 7, 8

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 7

[32] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258. 7

[33] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013. 8

[34] "Flir t420 & t440 high performance infrared camera with on-board visual camera, touch screen, wi-fi connectivity, & interchangeable lens, plus msx® image enhancement & auto orientation," *data-sheet*, 2014. [Online]. Available: www.flir.com 9

[35] *Kunststoff Laser-Sinter-System FORMIGA P 110 zur direkten Herstellung von Serien,Ersatzteilen und Funktionsprototypen*, EOS GmbH, Electro Optical Systems. [Online]. Available: https://www.eos.info 9

[36] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *Journal of Big Data*, vol. 6, no. 1, pp. 1–54, 2019. 12, 13

[37] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural Networks*, vol. 106, pp. 249–259, 2018. 12

[38] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. 15

[39] Python package index - pypi. [Online]. Available: https://pypi.org/ 15

[40] "Anaconda software distribution," 2020. [Online]. Available: https://docs.anaconda.com/ 15

[41] 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. Mané, 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. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/ 15

[42] F. Chollet *et al.*, "Keras," https://keras.io, 2015. 15

[43] 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," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf 15

[44] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, p. 357–362, 2020. 15

[45] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007. 15

[46] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87 – 90. 16

[47] J. Filter, "split-folders," https://pypi.org/project/split-folders/, 2018. 16

[48] X. Ying, "An overview of overfitting and its solutions," *Journal of Physics: Conference Series*, vol. 1168, p. 022022, 02 2019. 16

[49] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626. 19

[50] A. Halevy, P. Norvig, and F. Pereira, "The unreasonable effectiveness of data," *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8–12, 2009. 23

List of Figures

List of Tables

A. Model Architecture

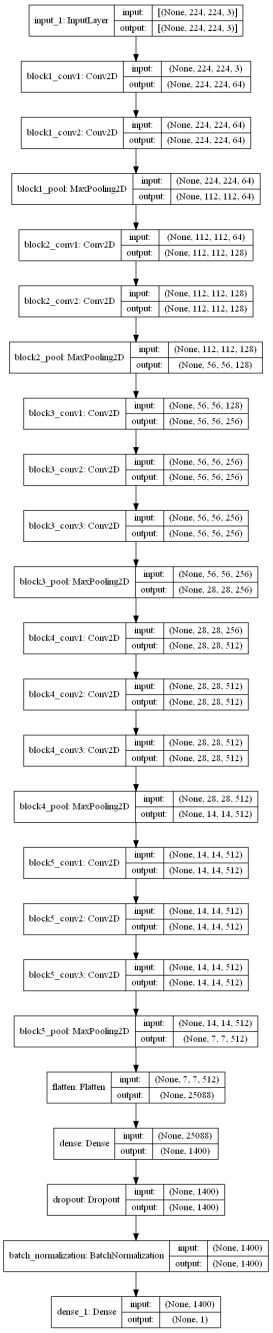
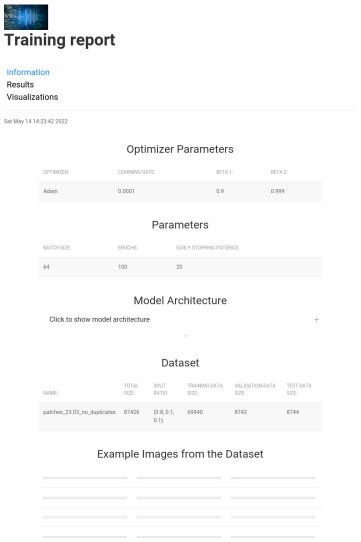
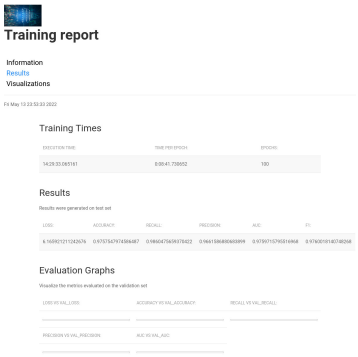


Figure A.1.: Custom model architecture

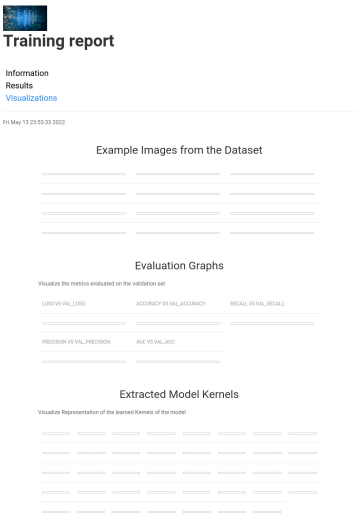
B. Example train and test report



(a) Info page of the train report

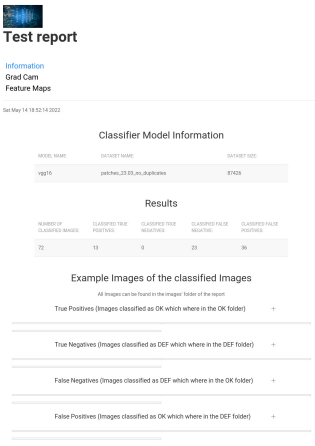


(b) Results page of the train report

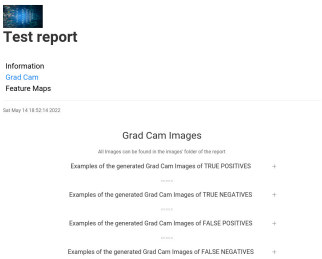


(c) Visualization page of the train report

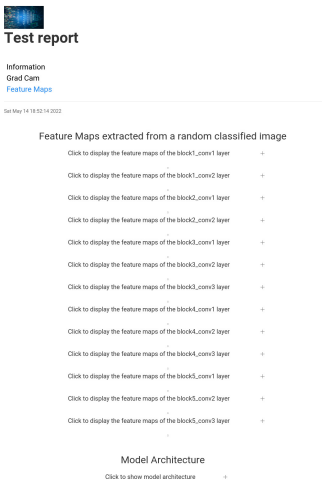
Figure B.1.: Example train report



(a) Info page of the train report



(b) Info page of the train report



(c) Feature map page of the train report

Figure B.2.: Example test report

C. Results of fine tuning

06.04.2022 - patches - no duplicates - clean					
number of conv layers frozen	Frozen conv layers	Accuracy	F1 score	Epochs	Time
1/13	block1_conv1 - block1_conv1	0.9752	0.9598	43	~00:55:00
2/13	block1_conv1 - block1_conv2	0.9891	0.9892	52	~01:02:00
3/13	block1_conv1 - block2_conv1	0.5	0.6666	43	~00:53:00
4/13	block1_conv1 - block2_conv2	0.5090	0.6707	51	~01:01:00
5/13	block1_conv1 - block3_conv1	0.5	0.6666	60	~01:10:00
6/13	block1_conv1 - block3_conv2	0.5362	0.6831	53	~01:02:00
7/13	block1_conv1 - block3_conv3	0.5099	0.6711	73	~01:23:00
8/13	block1_conv1 - block4_conv1	0.5081	0.6703	100	~01:51:00
9/13	block1_conv1 - block4_conv2	0.5181	0.6748	37	~00:45:00
10/13	block1_conv1 - block4_conv3	0.5027	0.6678	53	~01:03:00
11/13	block1_conv1 - block5_conv1	0.5009	0.6670	34	~00:42:00
12/13	block1_conv1 - block5_conv2	0.3756	0.4351	32	~00:39:00
13/13	block1_conv1 - block5_conv3	0.9655	0.9666	36	~00:43:00
06.04.2022 - multiple geometries - no duplicates - clean					
number of conv layers frozen	Frozen conv layers	Accuracy	F1 score	Epochs	Time
1/13	block1_conv1 - block1_conv1	0.5	0	33	~00:04:00
2/13	block1_conv1 - block1_conv2	0.9861	0.9859	41	~00:04:00
3/13	block1_conv1 - block2_conv1	0.5	0.6666	58	~00:06:00
4/13	block1_conv1 - block2_conv2	0.5	0.6666	68	~00:06:00
5/13	block1_conv1 - block3_conv1	0.5	0	78	~00:07:00
6/13	block1_conv1 - block3_conv2	0.5	0.6666	39	~00:04:00
7/13	block1_conv1 - block3_conv3	0.5	0.666	54	~00:05:00
8/13	block1_conv1 - block4_conv1	0.5	0.6666	39	~00:04:00
9/13	block1_conv1 - block4_conv2	0.9305	0.9253	28	~00:03:00
10/13	block1_conv1 - block4_conv3	0.5	0.6666	27	~00:03:00
11/13	block1_conv1 - block5_conv1	0.5138	0.6729	41	~00:04:00
12/13	block1_conv1 - block5_conv2	0.5	0.6666	41	~00:04:00
13/13	block1_conv1 - block5_conv3	0.5	0.666	24	~00:03:00

Table C.1.: Fine tuning results of models trained on 06.04.2022

23.03.2022 - patches - no duplicates - clean					
number of conv layers frozen	Frozen conv layers	Accuracy	F1 score	Epochs	Time
1/13	block1_conv1 - block1_conv1	0.9057	0.9086	51	~08:00:00
2/13	block1_conv1 - block1_conv2	0.5331	0.6817	57	~09:14:00
3/13	block1_conv1 - block2_conv1	0.5148	0.6732	40	~06:26:00
4/13	block1_conv1 - block2_conv2	0.8910	0.8934	75	~11:27:00
5/13	block1_conv1 - block3_conv1	0.9134	0.9189	39	~06:48:00
6/13	block1_conv1 - block3_conv2	0.9622	0.9634	82	~13:14:00
7/13	block1_conv1 - block3_conv3	0.9667	0.9659	60	~09:00:00
8/13	block1_conv1 - block4_conv1	0.9392	0.9419	79	~11:39:00
9/13	block1_conv1 - block4_conv2	0.9174	0.9236	100	~14:32:00
10/13	block1_conv1 - block4_conv3	0.9758	0.9760	100	~14:29:00
11/13	block1_conv1 - block5_conv1	0.9691	0.9683	99	~14:46:00
12/13	block1_conv1 - block5_conv2	0.9596	0.9579	100	~14:34:00
13/13	block1_conv1 - block5_conv3	0.8711	0.8856	48	~08:00:00
23.03.2022 - multiple geometries - no duplicates - clean					
number of conv layers frozen	Frozen conv layers	Accuracy	F1 score	Epochs	Time
1/13	block1_conv1 - block1_conv1	0.7093	0.5901	36	~00:12:00
2/13	block1_conv1 - block1_conv2	0.8759	0.8749	72	~00:22:00
3/13	block1_conv1 - block2_conv1	0.5	0.6666	35	~00:12:00
4/13	block1_conv1 - block2_conv2	0.7209	0.6170	25	~00:09:00
5/13	block1_conv1 - block3_conv1	0.8992	0.8879	35	~00:11:00
6/13	block1_conv1 - block3_conv2	0.7325	0.7876	27	~00:09:00
7/13	block1_conv1 - block3_conv3	0.8294	0.7939	32	~00:11:00
8/13	block1_conv1 - block4_conv1	0.7249	0.6203	30	~00:10:00
9/13	block1_conv1 - block4_conv2	0.5697	0.2448	36	~00:11:00
10/13	block1_conv1 - block4_conv3	0.7325	0.6349	32	~00:10:00
11/13	block1_conv1 - block5_conv1	0.5	0.6666	25	~00:08:00
12/13	block1_conv1 - block5_conv2	0.5	0.6666	27	~00:09:00
13/13	block1_conv1 - block5_conv3	0.5736	0.7010	94	~00:26:00

Table C.2.: Fine tuning results of models trained on 23.03.2022

11.03.2022 - patches - no duplicates - clean					
number of conv layers frozen	Frozen conv layers	Accuracy	F1 score	Epochs	Time
1/13	block1_conv1 - block1_conv1	0.8924	0.9015	95	~04:20:00
2/13	block1_conv1 - block1_conv2	0.7405	0.7813	93	~04:09:00
3/13	block1_conv1 - block2_conv1	0.5162	0.6739	70	~03:08:00
4/13	block1_conv1 - block2_conv2	0.6384	0.6398	97	~04:13:00
5/13	block1_conv1 - block3_conv1	0.5415	0.6854	87	~03:49:00
6/13	block1_conv1 - block3_conv2	0.5534	0.6911	63	~02:49:00
7/13	block1_conv1 - block3_conv3	0.5518	0.6905	85	~03:39:00
8/13	block1_conv1 - block4_conv1	0.5067	0.6696	92	~03:57:00
9/13	block1_conv1 - block4_conv2	0.5344	0.6823	72	~03:07:00
10/13	block1_conv1 - block4_conv3	0.5316	0.6803	81	~03:28:00
11/13	block1_conv1 - block5_conv1	0.8350	0.8239	100	~04:14:00
12/13	block1_conv1 - block5_conv2	0.5158	0.6737	100	~04:14:00
13/13	block1_conv1 - block5_conv3	0.6606	0.7221	100	~04:18:00
11.03.2022 - multiple geometries - no duplicates - clean					
number of conv layers frozen	Frozen conv layers	Accuracy	F1 score	Epochs	Time
1/13	block1_conv1 - block1_conv1	0.5	0.6666	100	~00:16:00
2/13	block1_conv1 - block1_conv2	0.6287	0.7292	99	~00:16:00
3/13	block1_conv1 - block2_conv1	0.5	0.6666	88	~00:11:00
4/13	block1_conv1 - block2_conv2	0.5	0.6666	66	~00:11:00
5/13	block1_conv1 - block3_conv1	0.5	0.6666	84	~00:13:00
6/13	block1_conv1 - block3_conv2	0.5075	0.6700	56	~00:09:00
7/13	block1_conv1 - block3_conv3	0.5	0.6666	85	~00:13:00
8/13	block1_conv1 - block4_conv1	0.6818	0.7558	100	~00:15:00
9/13	block1_conv1 - block4_conv2	0.5	0.6666	48	~00:08:00
10/13	block1_conv1 - block4_conv3	0.8409	0.8292	49	~00:08:00
11/13	block1_conv1 - block5_conv1	0.5	0.6666	21	~00:04:00
12/13	block1_conv1 - block5_conv2	0.5	0.6666	30	~00:05:00
13/13	block1_conv1 - block5_conv3	0.5	0.6666	69	~00:11:00

Table C.3.: Fine tuning results of models trained on 11.03.2022

19.11.2021 - crosses - no duplicates - clean					
number of conv layers frozen	Frozen conv layers	Accuracy	F1 score	Epochs	Time
0/13	none	0.9895	0.9894	24	~02:29:00
1/13	block1_conv1 - block1_conv1	0.9627	0.9612	85	~07:41:00
2/13	block1_conv1 - block1_conv2	0.9921	0.9921	24	~02:28:00
3/13	block1_conv1 - block2_conv1	0.5	0	32	~03:09:00
4/13	block1_conv1 - block2_conv2	0.9347	0.9303	21	~02:13:00
5/13	block1_conv1 - block3_conv1	0.5	0	33	~03:14:00
6/13	block1_conv1 - block3_conv2	0.5	0	29	~02:53:00
7/13	block1_conv1 - block3_conv3	0.9976	0.9976	48	~04:30:00
8/13	block1_conv1 - block4_conv1	0.5	0	51	~04:45:00
9/13	block1_conv1 - block4_conv2	0.5	0	36	~03:29:00
10/13	block1_conv1 - block4_conv3	0.5028	0.0114	44	~04:10:00
11/13	block1_conv1 - block5_conv1	0.5	0	21	~02:12:00
12/13	block1_conv1 - block5_conv2	0.5	0	24	~02:28:00
13/13	block1_conv1 - block5_conv3	0.9707	0.9699	22	~02:17:00

Table C.4.: Fine tuning results of models trained on 19.11.2021