Automated PCB defect detection

- Using Machine Learning to automate defect detection in PCB production

LUND UNIVERSITY
Campus Helsingborg

LTH Faculty of Engineering at Campus Helsingborg
Department of Computer Science

Bachelor thesis:
Hugo Rolf
Carl Kjäll

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2024

Abstract

This thesis explores the application of machine learning techniques, specifically convolutional neural networks (CNNs) and Mask R-CNN, for automated defect detection in printed circuit boards (PCBs). The study addresses the need for efficient and cost-effective quality control methods in PCB production, which is critical for ensuring the reliability of electronic devices. By developing an AI model capable of identifying and classifying defects in PCBs, the research aims to reduce the resources required for manual inspection and improve the accuracy of defect detection.

The methodology involved the creation of a comprehensive dataset comprising images of PCBs with differing defects, followed by extensive preprocessing, including data augmentation to enhance model robustness. The machine learning models were trained and evaluated on this dataset, achieving a somewhat stable accuracy in identifying and classifying defects. Notably, the implementation of Mask R-CNN allowed for good localization and categorization of faults.

The findings suggest that the integration of advanced machine learning techniques can improve quality control processes in the electronics industry, offering scalable solutions for defect detection. Future research may explore the adaptation of these models to even more different types of PCBs and further optimization to enhance real-time processing capabilities.

Keywords: Machine learning, PCB, AI model, Object detection, Neural network, Dataset

Sammanfattning

Detta examensarbete undersöker tillämpningen av maskininlärningstekniker, specifikt konvolutionella neurala nätverk (CNN) och Mask R-CNN, för automatisk defektdetektering i kretskort (PCB). Studien adresserar behovet av effektiva och kostnadseffektiva metoder för kvalitetskontroll inom PCB-produktion, vilket är avgörande för att säkerställa tillförlitligheten hos elektroniska enheter. Genom att utveckla en AI-modell som kan identifiera och klassificera defekter i kretskort syftar forskningen till att minska de resurser som krävs för manuell inspektion och förbättra noggrannheten i defektdetekteringen.

Metodiken innefattade skapandet av en omfattande datamängd bestående av bilder på kretskort med olika defekter, följt av omfattande förbearbetning, inklusive dataaugmentering för att förbättra modellens robusthet. Maskininlärningsmodellerna tränades och utvärderades på denna datamängd, vilket resulterade i en relativt stabil noggrannhet i identifiering och klassificering av defekter. Särskilt implementeringen av Mask R-CNN möjliggjorde god lokalisering och kategorisering av fel.

Resultaten visar på att integrationen av avancerade maskininlärningstekniker kan förbättra kvalitetskontrollprocesserna inom elektronikindustrin och erbjuda skalbara lösningar för defektdetektering. Framtida forskning kan undersöka anpassningen av dessa modeller till ännu fler olika typer av kretskort och ytterligare optimering för att förbättra realtidsbehandlingskapaciteten.

Nyckelord: Maskininlärning, Kretskort, AI modell, Objektdetektion, Neuralt nätverk, Datamängd

Acknowledgements

In this part of the report we as the writers want to give our gratitude to the central people and parts that made this work possible. First of all we give our gratitude to Sigma Connectivity as a whole and all of the amazing people working there that always met us with a smile and kindness when in need of any help. Special thanks are given to Sven Young who believed in us and gave us the chance to start the work as well as Vishnu Hari who was our main mentor at Sigma. Mr. Hari was always quick to help at any given day and took out time from his busy schedule to check in with us daily. We would not have gotten as far as we did without his help. Finally we also want to thank our mentor at the university Roger Henriksson who continuously helped us stear the theoretical and report part of this thesis work. When doing a practical thesis work it can quickly be forgotten that the main focus of the project still is a theoretical evaluation and report.

List of contents

1. Introduction

This chapter introduces the thesis by outlining its purpose, background, goals, and challenges. It begins by explaining the aim of developing an AI model for automated PCB production control. The background section details the collaboration with Sigma Connectivity AB and provides context on Automated Optical Inspection (AOI) systems used in PCB quality assurance. The chapter then outlines the thesis's specific objectives, such as achieving dynamic adaptability, accuracy, and efficiency in the AI model. This is followed by the formulation of key research questions addressing the methods and algorithms necessary for the model's development. The motivation for the work is discussed, highlighting the relevance of AI in image recognition and its potential impact on the electronics industry. Finally, the chapter addresses the thesis's limitations, focusing on the model's capability to handle various PCB models and its application scope.

1.1 Purpose

The purpose of this work is to provide small businesses and individuals with an easily accessible product for efficient and simple automated control of PCB production. By developing an AI model that is usable and adaptable for various environments and equipment, it opens up opportunities for electronics manufacturers to produce in new sustainable ways. As more companies and individuals gain access to a simple and adaptable model for quality control, there is potential for new sustainable approaches to product manufacturing. The hope is that the model can be used as a foundation for others to build and implement in new sustainable ways.

1.2 Background

This thesis work was completed in collaboration with Sigma Connectivity AB Lund. Sigma Connectivity AB, based in Lund, Sweden, specialises in developing connected products and systems, offering expertise in areas such as IoT, wireless communication, software development, and product design. It is part of Sigma Group and aids companies in enhancing their technology products. In early discussions with Sigma they presented the idea of a called “off the shelf” product for automated PCB detection and the thesis work in itself was intended to be used as a base for this project or as a completed product if the time would suffice.

AOI (Automated Optical Inspection) is utilised within the electronic industry to ensure product quality in PCB circuits. This technology employs optics and image processing for inspection purposes. AOI systems constitute a pivotal component within industries involving production lines. The primary objective is to detect defective surfaces or other types of deviations from the correct board. This digital inspection apparatus utilises cameras or scanners, along with artificial intelligence, to emulate the human eye but at a more precise level.

The challenges within this domain today lie in the substantial resources required to train individuals tasked with analysing each specific PCB model to be tested. It is costly to develop pattern recognition code for each separate individual PCB model to be tested, often necessitating specific equipment such as customised cameras for accurate measurements. Consequently, automating PCB inspection during production is presently an expensive and complex undertaking for companies. Furthermore, there are currently no straightforward solutions for companies to utilise software that can dynamically adapt to work with multiple models of PCBs and be easily utilised with various equipment.

The solution to this would be to create an AI model that can serve as a template to be trained with various models of PCBs. This solution is based on developing two models that together can visually identify defects in a specific PCB and determine the type of faults present. The defect detection model is trained by compiling a dataset with example images of how a particular model of a PCB should look when correctly assembled, as well as example images of the same model with various faults. In this way, the model is trained to determine for an image of a specific PCB whether it has regions that do not appear to be correctly assembled and this requires further examination.

The second model should be designed to classify the type of fault detected on a specific PCB. The classification model will receive compressed, small images of the regions that the Detection model has identified as appearing incorrect, it will then analyse each image and try identifying which of the 6 predetermined faults the particular image has. This will allow the second model or the Classification model to be trained similarly and alongside the Detection model, using its outputs as the Detection models inputs.

1.3 Goal

To meet the expectations set forth in the purpose, the model needs to achieve a number of sub-objectives: dynamic, flexible, accurate and efficient. If the AI model can accept a dataset for three or more different PCB models without requiring any code modifications, it will be considered dynamic. Three was chosen since a variation of at least two examples are needed for the model to differentiate between differing models and three is thereby chosen for marginal reasons. Assuming that the AI model correctly identifies defects in 95% of tested PCBs for a given test, it will be regarded as accurate. This metric was discussed with the mentor at Sigma as a needed accuracy metric to reach for a marketable product. If the AI model can be used with two different cameras from two different manufacturers, it will be considered flexible. This metric was chosen as a baseline for the finished product so that the application is completely free from hardware limitations in the camera choice. Should the model achieve the same accuracy in testing with only a schematic image of a new PCB model as it does with the trained PCB model, the AI model can be deemed efficient. This needs to be achieved if the finished product hopes to be usable for an off the shelf implementation that should be usable on any given PCB.

1.4 Problem description

The following questions will be addressed through this work:

1) What methods can be used to make the AI model dynamic and flexible to accommodate multiple PCB models?
2) How can the accuracy of the AI model for defect detection be measured and validated?
3) What methods can be employed to tailor the data set for training the model?
4) Which algorithms are most effective for classifying different types of faults in a given PCB?
5) How large a data set will be required to train and adapt the model?

1.5 Motivation of work

As discussions about the thesis topic commenced, the relevance and emerging opportunities of AI in today's technological landscape quickly made it a priority. Further exploration into the field of AI revealed image recognition as particularly compelling due to its wide range of applications. The thesis was chosen due to its direct relevance to the company's activities in the electronic industry, especially in

the area of Automated Optical Inspection (AOI). The specific interest in this thesis work stemmed from its use of AI in image recognition and product development. The thesis provides an opportunity to explore and address complex technical challenges within AOI systems, while identifying new possibilities for improving and streamlining the production process using neural networks and machine learning. It also holds relevance for individuals and small businesses as it offers them a tool to further develop defect detection beyond just PCB production. One of the aims and objectives of this work is for the model to serve as a flexible and dynamic template that can be built upon and adapted for various applications, a practice often encouraged and pursued across various online forums.

1.6 Limitations

The most comprehensive aspect included in the thesis will be the model's performance and its ability to handle various PCB models, including variations in angle and positioning in front of the camera. This means that the model should be able to identify and classify defects when the PCBs are placed at different angles and positions, but within reasonable limits—specifically, the entire model must remain within the camera's view. The model will not be used for defect analysis of circuit boards other than PCBs

1.7 Work distribution

The following table is an approximation of how the work was divided between the two thesis workers Carl Kjäll and Hugo Rolf. Since the majority of the work was done in collaboration at the office of Sigma Connectivity AB in Lund the estimations of the divided labour has a degree of uncertainty. The majority of coding was done by pair-programming.

	Hugo Rolf	Carl Kjäll
Analysis	50	50
Design	50	50
Implementation	50	50
Testing/Evaluation	55	45
Thesis paper	40	60

Presentation	50	50
Poster	60	40

Table 1: Estimated resulting distribution of work

2. Technical background

This chapter explores the foundational technologies in modern electronics and computational systems, focusing on printed circuit boards (PCBs) and machine learning (ML). It begins with an overview of PCBs, their construction, common defects, and the importance of quality control. The discussion then shifts to ML, covering its basic principles, including supervised and unsupervised learning, preprocessing techniques, and neural network architectures. Finally, the chapter delves into advanced applications such as object detection and image classification, highlighting key models and algorithms used in these fields.

2.1 PCB

PCB stands for printed circuit board and are some of the most basic components of modern electronics. They are used in computers, smartphones, TVs and are adapted for most products they are intended to be used for, because of this PCBs vary greatly and do not have one standard approach or design. However, some primary features are the same across PCBs. Zacharia Petersson describes the basic construction of PCBs as all being built from alternating layers of conductive copper with layers of electrically insulating material [1]. The board features copper traces, pads, and planes for connectivity, an example of this is shown in figure 1. These are laminated between insulating layers and topped with a solder mask and silk screen for component labels. After initial fabrication, the board undergoes PCB assembly where components are soldered on and tested. This etching and laminating process creates the final multilayer structure of the PCB, ready for assembly and testing.

As with all production, defects occur during production of PCBs which can result in faulty operations or completely unusable boards. When specifically looking at bare single layer PCBs there are 14 known defects as listed from "Printed circuit board defect detection using mathematical morphology and MATLAB image processing tools,"[2]. These include breakout, pin hole, open circuit, under etch, mouse-bite, missing conductor, spur, short, wrong size hole, conductor too close, spurious copper,

excessive short, missing hole and over etch. In general, these defects can occur on virtually any implementation of a PCB and are therefore a central part in any quality control of a produced PCB.

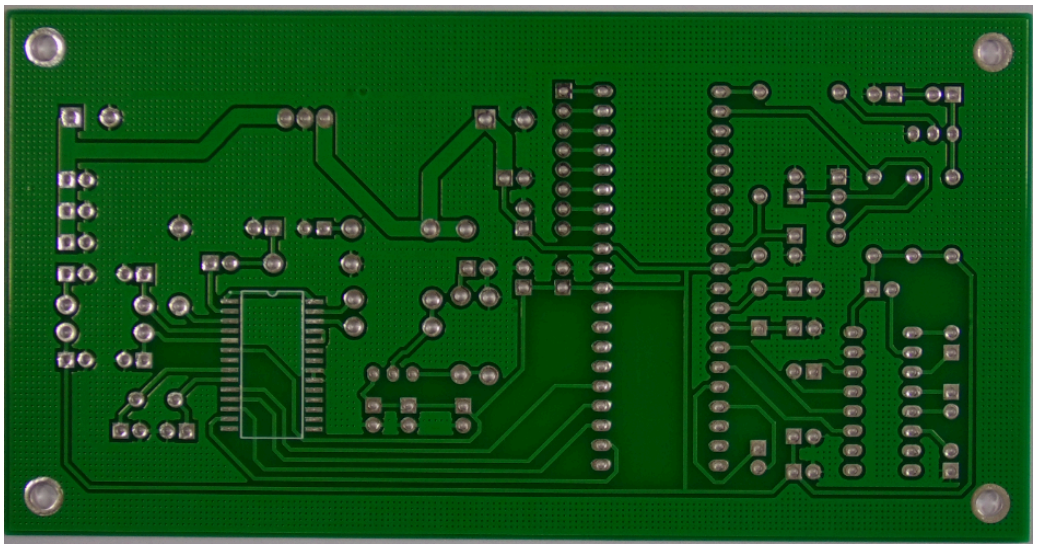

Figure 1: Example of PCB board

2.2 Machine Learning

Machine learning (ML) represents a subset of artificial intelligence where algorithms improve automatically through experience and by using data. The first real encounter with machine learning can be traced back to experiments in the mid-20th century such as Arthur Samuel's work with checkers 1950s[3]. This program used ML techniques to improve over time by playing against itself and learning from its success. This capability lies at the core of machine learning, enabling systems to autonomously enhance their performance and adapt over time. This improvement occurs in tasks ranging from simple games to complex systems, whether the outcomes are known—typical of supervised learning—or unknown, as in unsupervised learning scenarios.

2.3 Supervised learning

In supervised learning the focus is pointed towards input and output pairs from the dataset. These pairs are used to form a function that maps these values from input to output[4]. This concept is often used when dealing with classification problems when there are predefined classes. In this case, the input could be an image of a cat and the output could be different types of predefined animals. By training on such labelled data, the algorithm learns to recognize patterns or features such as the shape of the ears, size of the eyes for example. A more theoretical approach, as described in Russel and Norvig's article "Artificial Intelligence: A Modern Approach," involves creating a hypothesis, h, that serves as a model of the

data.[4]. This is essentially a function that the learning algorithm believes to be a close approximation of the true function, which is generated by the training data.

2.4 Unsupervised learning

In unsupervised learning there is a known input value also known as the dataset, however there is no corresponding output to the input[4]. Instead the goal is to create a context and relationships between these input values, eventually the unsupervised algorithm will be able to find out a pattern during the training of the data.

2.5 Preprocessing

Preprocessing in the context of machine learning is the process of preparing data by different configurations before it can be used to train an AI model. When gathering data to be used in machine learning the most common method is to create a dataset that can be fed into an AI model during training of the model[12]. A dataset is a general term for a collection of data to be used for machine learning. It can contain texts, images, videos or any other form of digital media. A part of preprocessing is labelling the data which means to assign each data point with a label, often a text, to distinguish every data point from the rest within the dataset. An example of this could be having a dataset with a hundred images of cats and dogs. In this case each image would need a label like “dog3” or “ cat14”.

Apart from this another common practice when it comes to preparing visual data like images and videos is to reformat the pixel size so that all data points in the entire dataset have the same size. For example, in a dataset of images it can be beneficial to resize all the images to a specific pixel size for height and width. Lastly, if the dataset originally only contains a small batch of data points or very similar data points, augmentation can be used to create artificial data points based on their original versions with tweaked features. With the example again being images in a dataset, augmentation could potentially flip, rotate, distort or move an object within the image thus creating a bigger and more diverse dataset for the model to train on.

2.6 Artificial Neural Networks (ANN)

Artificial Neural Networks (ANN) or Neural Networks (NN) is a way of trying to extract the complex functions of the human brain by mimicking its structure of interconnected neurons, using layers of artificial neurons that process information through weighted connections and nonlinear transformations to solve a variety of tasks, running from pattern recognition to decision making. As previously explained when training neural networks there are two approaches, one is via supervised learning and the other is unsupervised learning[4].

Supervised learning has become popular when it comes to image classification and object detection. The NN consists of different layers: Input layer, hidden layer and output layer. Each of these layers consists of neurons or nodes that could hold a relevant value for the given context. All of these nodes are then connected to the neurons in the next layer. The links between the node in the first layer and next layer have an associated weight. These weights modify the strength of the signal that is passed from one node to another. Each node in the next layer receives inputs from its predecessors which is the sum of the weights as shown in figure 2. Bias is an adjustable parameter that enables a more flexible decision threshold for each neuron. It is not exactly the same as the threshold in biological neurons, but it serves a similar role by determining when a neuron should activate in an artificial neural network. This threshold is managed by what's called an activation function. Some popular activation functions are Sigmoid function and ReLu function[5]. The activation function takes the sum of all these weights and transforms it into an output value. If this output value is above a certain level the node should give a response.

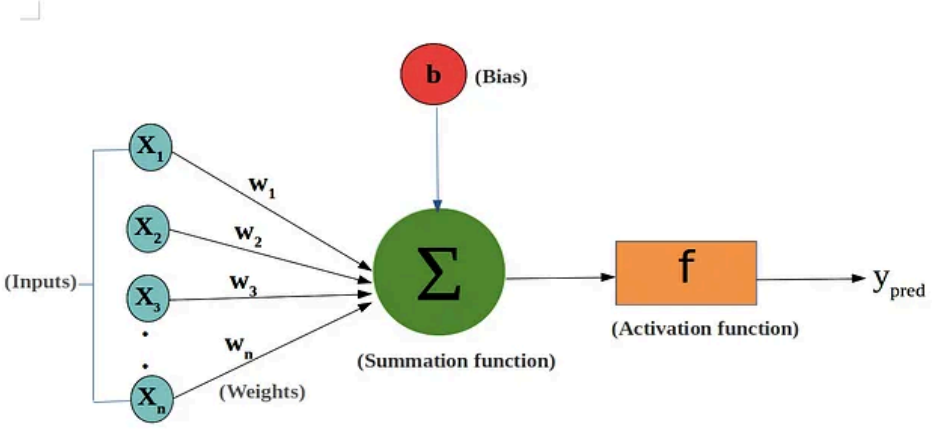

Figure 2: Example of a neuron structure

2.7 Convolutional Neural Networks (CNN)

Convolutional Neural Network (CNN) is a type of ANN and is primarily used for image recognition and processing tasks. It excels in handling pixel data, making it a cornerstone for computer vision applications such as image segmentation. The base architecture consists of three layers: Convolutional Layer, Pooling Layer and Fully Connected Layer[6]. The Convolutional Layer utilises filters and kernels to extract features into feature maps, capturing essential visual details. The Pooling Layer downscales the feature maps by summarising features within specific patches, an example of this is shown in figure 3. The Fully Connected Layer links every neuron in one layer to every neuron in another, facilitating the final decision-making process in the network.

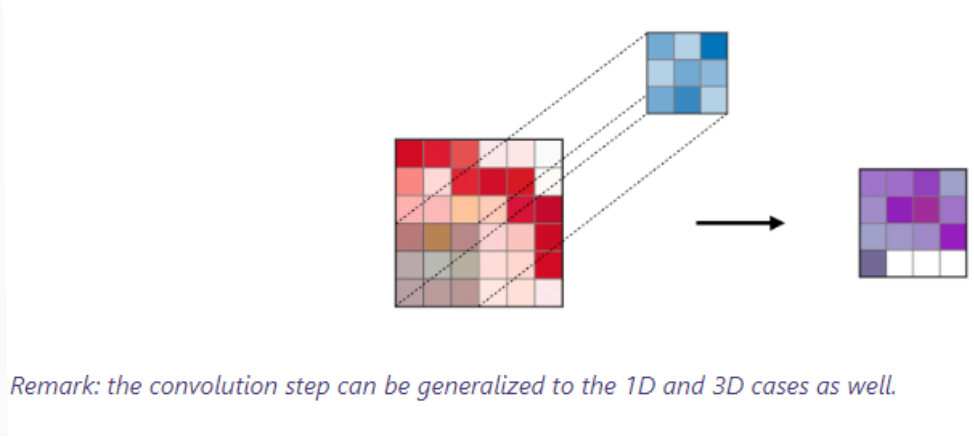

Figure 3: Example of Max Pooling

2.8 Object detection

Object detection is an area within the computer vision field that has been growing in popularity and use cases in recent years. Use cases for object detection can range from visual segmentation of traffic for a self-driving car to defect detection of products in factories. Different algorithms and approaches have developed over the years but has in recent time generally been categorised as either one stage detectors or

two stage detectors. One-stage detectors directly predict object classes and bounding boxes from full images in a single step, which generally allows for faster processing and simpler implementation. Two-stage detectors first generate region proposals (potential object locations) and then run a classifier on these regions to refine the predictions. This typically results in higher accuracy at the cost of increased computational complexity and slower processing speeds. Some of these different models and algorithms are analysed and explained in “Object Detection in 20 Years: A Survey”[7] such as the YOLO model and faster R-CNN model.

The You Only Look Once (YOLO) algorithm, introduced by Joseph et al. in 2015, marked the first one-stage detector in the deep learning era, known for its high speed and the innovative approach of applying a single neural network across the full image. While YOLO dramatically increases detection speed, achieving up to 155 frames per second, it has faced challenges with localization accuracy, particularly for smaller objects. Improvements and new versions like YOLOv7 have focused on enhancing both speed and accuracy through optimised network structures. Introduced by Ren et al. in 2015, Faster R-CNN(Regional Convolutional Network) is a two stage detector that advances the capabilities of its predecessor Fast R-CNN by incorporating a Region Proposal Network (RPN), making it the first near-real-time deep learning detector. It integrates crucial object detection processes—like proposal detection, feature extraction, and bounding box regression—into a cohesive, end-to-end framework. Despite its speed and integration advancements, Faster R-CNN still encounters some computational redundancy.

2.9 Image classification

Image classification is another field within computer vision, this field of computer vision has another core function than object detection, namely classifying objects within a picture rather than localising them. In practice this means that the AI model working with the task needs to analyse an input image to extract features from the image and with the help of deep learning try to understand patterns that match to predefined categories[8]. With other words, an image classifier that has been trained to classify dogs and cats will not be able to label an image of a rat since its features do not match any of the learnt patterns. Something to carry in mind is that with any image classifier, it will never yield a label with a 100% accuracy. The model itself will calculate the likelihood of an image being a specific label for every defined category and then choose the output label based on which of the categories has the highest chance to be correct.

There are a number of models and implementations created for image classification like for object detection. One of these is the ResNet (Residual Neural Network) architecture that was introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian sun in their paper titled “Deep residual Learning for Image Recognition “ in 2015[9]. ResNet is a type of CNN that uses skip connections or shortcuts to jump over some layers. Typical CNNs become increasingly complex as more layers are added, which can lead to higher training errors—ResNet addresses this by enabling training of much deeper networks through these skip connections. The key idea is to add the input of the current layer to the output of a layer deeper in the network, which helps to prevent the vanishing gradient problem and improve convergence during training.

2.10 Mask R-CNN

Today there also exist models that try to incorporate both the principles of object detection as well as image classification. One of these models is called Mask R-CNN and is built on the Faster R-CNN model while adding a feature called masks. Normally Faster R-CNN can detect and localise objects in an image with so called bounding boxes around the objective within the image, Mask R-CNN on the other hand does this as well as adding a branch to the model which can localise objects at a pixel level. The mask classification branch then predicts binary masks for each region of interest within the image[10]. The Mask R-CNN model uses the ResNet architecture as its backbone, shown in figure 4, for feature extraction as well as implementing FPN(Feature Pyramid Network) which is an architecture that enhances object detection across various scales by constructing a pyramid of feature maps from a single input image. FPN improves detection models by integrating low-resolution, semantically strong features with high-resolution, semantically weaker features, leading to better localization and classification of objects of different sizes.

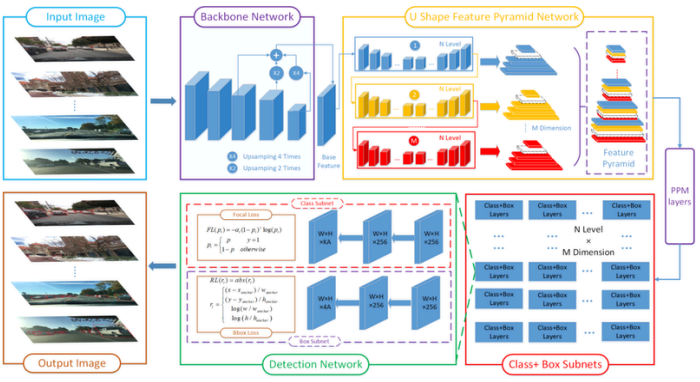

Figure 4: An example of a model working similarly to mask-R-CNN

3. Method

This chapter will outline the phases of the thesis and progression to the decision on which object detection algorithm to use. Each phase was built upon the insights and results of the previous one. The workflow included initial research, planning and design, dataset gathering, software and hardware selection, as well as the libraries and frameworks used. This chapter will explain the methods employed in creating the AI model for PCB defect detection in detail.

3.1 Workflow

The thesis was completed by the two authors of this paper Carl Kjäll and Hugo Rolf working together throughout the entire process. Since this thesis work was done in collaboration with Sigma Connectivity deskspace was provided at the office in Lund. Apart from deskspace, a desktop PC as well as 2 laptops were provided for working both at the office as well as remote. Early on during the thesis it was decided that working together at the office during most days of the week would be the most optimal way to progress the work. This plan of work made it simple to divide the work since the same amount of time was given from each person each day. Apart from this the work itself was mostly done together and work tasks were mostly divided when needed. When writing the report different sections were divided for more effective writing.

3.2 Initial Research

The initial research phase was primarily during the first 2 weeks of the thesis although continuing research was performed throughout the entire process. The first area to understand for this specific thesis work was computer graphics and neural networks. Gathering information on these topics were made through many different mediums such as reading past reports in similar topics, watching educational youtube videos from both personal developers as well as other educational institutions and having conversations with the mentor of this thesis work at Sigma Connectivity Vishnu Hari. Mr. Hari is a machine learning engineer and has worked in multiple AI and specifically machine learning projects in the past. During the initial research phase as well as further into the thesis a logbook was written for gathering the information, links and references. This logbook also provided a great way of keeping track of the progress as well as remembering information from different sessions of information gathering. Through these methods a base understanding of what different parts were needed for the completion of the work was acquired.

3.3 Planning and Design

When the base understanding of the needed parts were laid out, planning of gathering and developing these different parts were started. The first topic was the design of the dataset creation pipeline as well as the overall design of the model. Although the initial design of the model would be completely redone during a later time in the work, the design of the dataset creation pipeline for the most part stayed the same. Figure 5 shows a picture of a whiteboard used during the first week planning and researching.

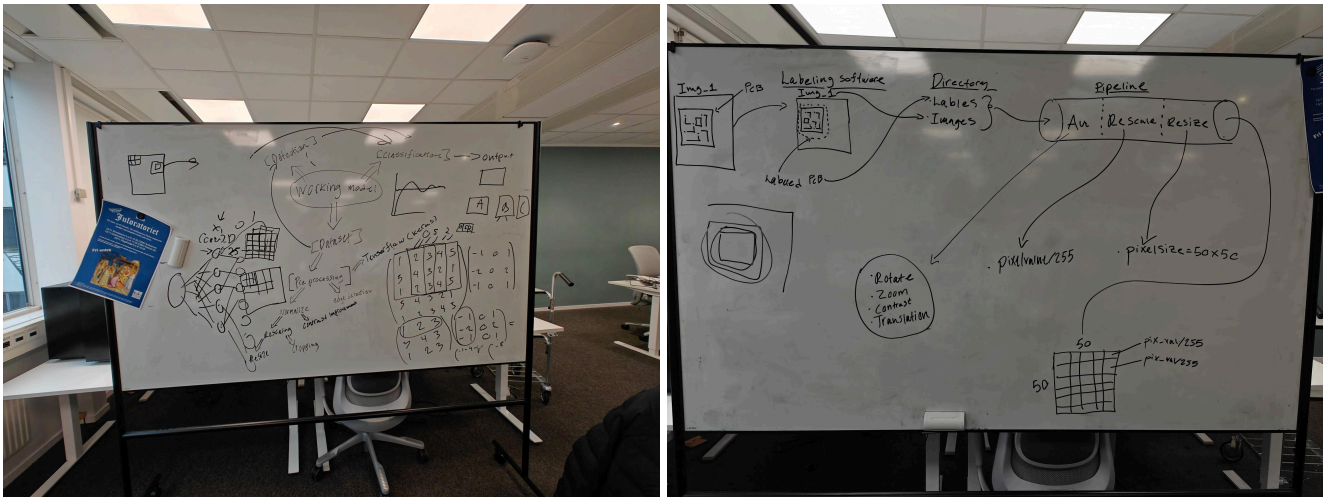

Figure 5: The two pictures shows examples of early design drawing and information gathering

3.4 Dataset gathering

Collecting the dataset was the first step in the process of building the AI model. This could be done without deciding what object detection algorithm to use or how the pipeline for the pictures should be designed. For this thesis the pictures were gathered online from the popular platform kaggle where data scientists and researchers can access public datasets[11]. The PCB dataset was created by Weibo Huang and Peng Wei at the Open Lab on Human Robot Interaction of Peking University[12]. Weibo Huang and Peng Wei wrote an article “A PCB Dataset for Defects Detection and Classification”, where the authors and creators address a significant gap in the field of digital object detection. The dataset provides a standardised resource that allows researchers and other types of developers to compare different classification and object detection algorithms more accurately.

The dataset includes 1386 images with six types of defects, missing hole, mouse bite, open circuit, short, spur and spurious copper. These are annotated and classified to facilitate machine learning algorithms effectively, making it easier to recognise and differentiate these flaws effectively. This type of a resource

is important for developing an AI-based AOI system that can operate under the complex conditions presented by modern PCBs.

The main factor in choosing which dataset to use was the variety of PCBs presented in the dataset collection. There were 12 different PCB models in the selected dataset which is crucial for a varied understanding of PCB design in general for the AI model. Training with a diverse set of PCB models and defect types helps the AI model to learn, identify and distinguish between different kinds of flaws. This variability in the training data prevents the model from overfitting to specific defect characteristics and ensures it can generalise well to new unseen PCB models.

3.5 Software and Hardware

TensorFlow was used as the primary framework for developing our machine learning model due to its robust, versatile and highly scalable nature. As described in the abstract of its paper, TensorFlow provides an interface for expressing a wide range of machine learning algorithms and an implementation for executing such algorithms across diverse systems[13]. TensorFlow's ability to express a variety of algorithms is crucial for the thesis, which involves complex neural network architectures for image processing and pattern recognition. Its flexibility allows for the adaptation of training.

The computer used for developing and training the AI model was a stationary PC with as of 2024 powerful parts. The GPU used for this thesis was the NVIDIA GeForce RTX 4090 GPU which is a part of NVIDIA's latest generation of graphics cards. This is a powerful tool for deep learning and scientific computations. Deep learning models, particularly those applied in fields like computer vision, require substantial computational resources not only due to the complexity of the models but also because of the large volumes of data they process, especially images. The GPU architecture is specifically optimised for deep learning. With thousands of CUDA cores, the Tensor Cores specifically designed for matrix operations, the RTX 4090 reduces the time required for training deep learning models, which will eventually improve the accuracy of the model[14]. Apart from the powerful GPU the PC also had 32Gb of RAM, sufficient cooling for using the most of the PC's performance as well as in today's standard high end AMD Ryzen 9 CPU.

3.6 Libraries and Frameworks

In the beginning of the thesis the initial plan was to create an AI model from scratch, however, when continuing to research and plan the design of the model it quickly became apparent that the amount of work and time needed to create a model with the complexities needed for this projects task would be out of reach for the scope of the work. After discussion with the practical mentor of the thesis, Mr. Hari, it was agreed upon that the most practical approach would be to search for an open source implemented model that could be trained and adjusted for the thesis's specific dataset and task. After researching different models the Mask R-CNN model was chosen for object detection and segmentation When creating the augmentation pipeline the imgaug library was used to augment and resize the images[16].

Mask R-CNN is a model used for object detection, making it highly suitable for our thesis which requires precise identification and localisation of objects within images[15]. This model is based on the Faster R-CNN by adding a branch of predicting segmentation masks on each region of interest. The model is made by using Python 3, Keras and TensorFlow, which are all popular amongst researchers and data scientists when it comes to AI and machine learning. The key features when using this framework is that it gives both bounding boxes around the defect areas as well as generates a pixel-level mask for each object detected in the image, which greatly enhances accuracy when it comes to defect detection. This Mask R-CNN model can both perform object detection and segmentation within a single model. This dual capability ensures that the system can not only locate defects on PCBs but also classify them directly. By using a pre-built and trained model the focus could be pointed towards refining the model for the specific use-case rather than building from scratch.

The library that was used in the process of creating the dataset was as previously mentioned the imgaug library[15]. This library was used in combination with the Python language, the guide “Augment Images and Multiple Bounding Boxes for Deep Learning in 4 Steps” by Asset Karazhay[17] on how to convert the provided XML-files from the kaggle dataset into one CSV file and a pandas dataframe to create the augmentation pipeline. This dataset could then be fed into the deep learning model together with a guide on how to effectively augment the pictures from our dataset which is crucial for not overfitting the model.

3.7 Source criticism

This section evaluates the reliability of the references cited in the thesis. Due to the substantial number of references, they have been categorised by source type and reviewed collectively.

3.7.1 Technical Documentation

Several references in this thesis fall under the category of technical documentation. Such sources are typically created to instruct users on how to use a particular product. As a result, these documents are generally reliable and free from biassed or malicious intent. However, it is important to note that the creators of these products are also the publishers of the documentation, which may introduce potential biases due to their vested interest in presenting their products favourably. This could affect the content and objectivity of the documentation

The following references relate to technical documentation: [11, 14, 18]

3.7.2 Articles and papers

The articles and papers referenced in this thesis provide a solid foundation of knowledge on various topics such as PCB design, defect detection, and machine learning. They offer valuable insights, with some providing historical context and others reflecting recent advancements. While these sources are generally reliable and well-researched, their relevance can vary based on their publication date and focus. Overall, they effectively support and enrich the thesis's findings, though users should consider their context and applicability to current research.

The following references relate to articles: [1, 2, 3, 7, 9, 10, 12, 20]

3.7.3 Frameworks & Libraries

Some of the sources referenced in this thesis fall into the category of frameworks and libraries. It is important to exercise caution when using these resources, as their quality and reliability can vary significantly. For instance, while some frameworks and libraries are well-maintained and rigorously

tested, others may have poor documentation, be outdated, or even abandoned by their developers. Relying on unreliable sources can lead to increased development time, system vulnerabilities, and potential errors in your work.

The following references relate to technical documentation: [13, 15, 16]

3.7.4 Tools

During the development of the thesis, several resources were utilised to support and enhance the work of the thesis team. These resources primarily served as references to provide additional explanations or guidance, rather than directly influencing the facts and conclusions presented in the thesis. Their main purpose was to offer sources where further information could be accessed.

The following references relate to tools or frameworks: [5, 6, 8, 17, 19, 21, 22]

3.7.5 Course Literature

Course literature was only minimally referenced in this thesis. This literature, which has been utilised in university courses, is generally considered reliable and was primarily used to reaffirm and refresh the knowledge previously acquired by the thesis team.

The following references relate to course literature: [4]

4. Analysis

This chapter will outline the final technical choices made for the model. Each choice was built upon the insights and results of the previous one. This chapter will explain the methods employed in addressing the dependency issues that had a significant impact on the model selection and implementation. Following the discussion on dependency issues, the chapter will delve into the process of dataset creation, detailing the approaches considered and the final method adopted. It will then cover the considerations and adjustments made to match the batch size with the GPU capabilities, ensuring efficient training. Finally, the chapter will discuss the selection of optimizers and activation functions, as well as the implementation of callbacks to enhance the training process and model performance.

4.1 Dependency issues

Before analysing the different technical choices that were made regarding the dataset as well as the training and testing of the model, firstly it's important to bring up the dependency issue phase since this had a large impact on the final technical choices made for the model to be used. When considering training an AI model from scratch with an own dataset and with the help of a GPU in 2024 there are plenty of choices that need to be made before the first line of code will work as intended. The major parts that come into play when trying to execute this are as follows: operating system, Python version, GPU-brand, GPU-dependency-versions, developing package version and the last model to be used.

When choosing an operating system there are 3 major options to choose from being Windows, Mac and Linux. Within Linux there are also different distributions with Ubuntu being the standard distribution for Linux users. Apart from there being multiple operating systems to choose from, there exists multiple programming languages as well. Python is the standard programming language in the area of machine learning since most libraries are built to be used with it, but since Python is continuously being updated there exists multiple versions of Python. In the current GPU market NVIDIA and AMD are the standard options to choose from. Apart from this, differing GPU versions within these brands will have different amounts of computational power, driver versions and software dependencies. Depending on the GPU used, there are multiple different needed software to make them usable for machine learning training. For example if using a modern NVIDIA GPU both a cuDNN and CUDA package needs to be installed as well. Two libraries being used for machine learning in 2024 are TensorFlow and PyTorch of which multiple versions and iterations have been made. Lastly a model needs to be either created from scratch or downloaded as a package and configured and implemented in a development environment.

During the dependency phase the problem of choosing and matching the different versions of all of the above mentioned software took a greater amount of time than anticipated. The process of choosing how to manage these different versions of packages came down to two main parts, the model and the GPU. Since the company Sigma had bought a Nvidia RTX 4090 to be used for training, the model itself as well as the GPU became the shared first priority to work around. After solving issues and problems with the code and differing package version for multiple weeks the restrictions and needed packages for using the Mask R-CNN model with a Nvidia RTX 4090 in 2024 can be summarised here: The original Mask r cnn is built using TensorFlow and Keras and should be used with these libraries. The original implementation was built on Keras earlier than 2.0 which was the first problem since the current Python version is 3.11/3.12 which neither support TensorFlow versions earlier than 2.0. The second problem was that a NVIDIA GPU needs a package called CUDA to be used for training and different CUDA versions support different versions of TensorFlow as well as different models of a GPU. The RTX 4090 needs to be used with a CUDA version of 11.8 or later, a CUDA version of 11.8 only works with a TensorFlow version of 2.12-2.14 and finally a cuDNN version of 8.6-8.7 is needed for the 11.8 CUDA version[18]. All of these need to match a correct version of Ubuntu or Mac (Mac was not an evaluated option for this thesis and not discussed in this thesis) and does not work with any version of Windows as of TensorFlow 2.0 and later when using GPU support.

The only way to get around these issues if the thesis was going to use a RTX 4090 with the Mask R-CNN model was to update the code in the model itself to work with a newer version of TensorFlow. Trying to correct the code to work with a newer version of TensorFlow was a much harder task than initially anticipated. After trying to update the code for a few days a fork to the original github was found that had updated the code to work with TensorFlow 2.14. Since TensorFlow 2.14 supported CUDA 11.8 which the GPU supports it could be made to work with a 4090. The final packages and specific version used were as follows: Ubuntu 22.0, Python 3.10.12, TensorFlow 2.14, CUDA 11.8, cuDNN 8.7. Following is a table found at TensorFlow’s website that was used for matching the different package and distribution versions.

GPU

Version	Python version	Compiler	Build tools	cuDNN	CUDA
tensorflow-2.17.0	3.9-3.12	Clang 17.0.6	Bazel 6.5.0	8.9	12.3
tensorflow-2.16.1	3.9-3.12	Clang 17.0.6	Bazel 6.5.0	8.9	12.3
tensorflow-2.15.0	3.9-3.11	Clang 16.0.0	Bazel 6.1.0	8.9	12.2
tensorflow-2.14.0	3.9-3.11	Clang 16.0.0	Bazel 6.1.0	8.7	11.8
tensorflow-2.13.0	3.8-3.11	Clang 16.0.0	Bazel 5.3.0	8.6	11.8
tensorflow-2.12.0	3.8-3.11	GCC 9.3.1	Bazel 5.3.0	8.6	11.8

Table 1: Table of compatible version

4.2 The dataset

When choosing how to create the dataset needed three main approaches were discussed among the thesis workers. The first was creating the dataset entirely from scratch by taking pictures of a PCB as well as creating physical or artificial alterations of the PCB images to create production error examples and after that using a software to label every object in the picture. The second approach would be to find a complete dataset that would have a large pool of examples with different types of production errors, different augmentations on these images as well as different PCB models with accompanying files for labels of all the objects in the images. Since a complete dataset matching our needs could not be found, a third approach based on a combination of the first two were then adopted.

This approach involved finding a relatively small dataset with example images of a few different PCB models with 6 different examples of manufacturing errors as well as XML annotation files for every image and then augmenting this dataset with a custom augmentation pipeline which firstly resizes all the images to a set format and then augments each image with 3 random augmentations from a fixed list. Examples from this list are gaussian blur, rotation, zoom, and affine transformation. The main challenge with this was to make sure that all the annotations for the images would be correctly moved with the augmented images. An annotation for this dataset and thesis means an array of 4 values, xmin, ymin, xmax and ymax. These four coordinates together create a so-called bounding box around an object within the image. The reason for this is so that the model can have a correct and exact answer of where the

objects it is supposed to identify actually is within the image. An example of how this looks for the model is shown as in figure 6. When augmenting an image it is possible for the PCB in the image to get moved to another place within the image and or get zoomed in closer or farther away to it. When this happens the defections on the PCB also move within the image and the bounding boxes for these defections remain in the same place unless handled.

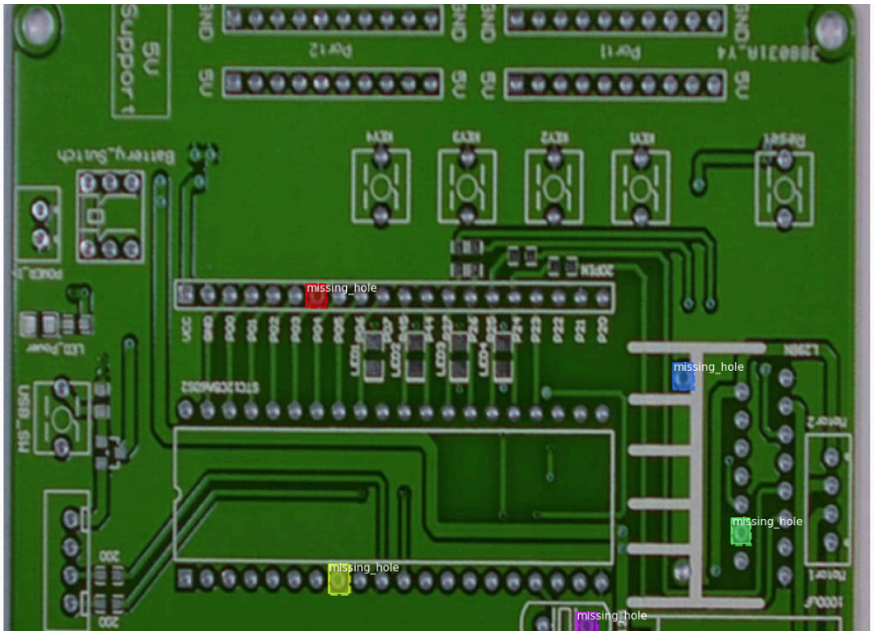

Figure 6: Example of a cropped image with bounding boxes and masks

As previously mentioned the imgaug library was utilised for these augmentation and resizing tasks in combination with the library of pandas dataframe that was used for managing the annotations[15]. The augmentation pipeline works through firstly loading the images into a local array and then loading and categorising all of the annotations for each image into a pandas dataframe from their respective XML files. When the data frame has all the necessary annotations for the images the images get resized to 1024x1024 pixels and their new bounding boxes get updated in the data frame. The next step in the process is for each image to get augmented and their new bounding boxes to get added in the data frame. When this is complete the dataset is ready to load for training from any file or folder in the repo since the dataset is a combination of a single folder of images and a single csv file. Since the dataset only contains 697 images and the pipeline generates one augmented image per image presented it generates an equally large dataset everytime the pipeline is run. Since a very big dataset was desired for the thesis the pipeline was run a few times until the complete dataset consisted of about 10000 images.

4.3 Matching batch size with GPU

Batch size is an important factor to consider when tuning the configurations for training efficiency. When choosing the right batch size to use for the training it needs to be matched with the resources of the PC that's running the training[20]. There are three major factors that influence this choice: image size, GPU memory, and the computational time required. Batch size defines the number of samples that propagate through the network before the model parameters are updated. A higher batch size means more images from the dataset are used in each iteration.Using more images per iteration requires more computational power. In other words, a larger batch size increases memory usage. When for example using a RTX 4090 with 24 gigabytes of video ram (VRAM), only that amount of memory can be used at a specific time which in turn means that the data needed for all images combined from a batch needs to be lower than 24 gigabytes and here is where the image size matters. The larger the image used the greater the memory usage. For an example, a square image with a size of 512 would yield: 512×512=262,144 pixels while a size of 1024 would yield: 1024×1024=1,048,576 pixels, this means that twice the image size would need 4 times the memory usage. In the specific configuration file used for the model the batch size was calculated from: gpu_count*images_per_gpu. Since only one GPU was used and the images used were the size of 1024×1024 pixels these values were set to 1 making the batch size 1 as well.

Taking all of the different parameters into account, choosing the image size, batch size and GPU needed will come down to two principles, the problem the model is tasked with and the amount of time that can be used. The problem that this paper researches is very tiny object detection and classification. Since the objects that need to be detected are very small and often hard to distinguish in comparison to the images they are placed, large images are needed for this problem to maintain as much clarity of the objects as possible. When prioritising image size, the batch size then needs to be matched to the GPU memory by multiplying the memory needed per image with the batch size and making sure it's below the total VRAM.

4.4 Activation functions

In machine learning there are a plethora of important terms to understand in order to fully control the creation and performance of an AI model. These are all important for both the building and training of the model but there is one more deeper layer to discuss in order to fully understand the complexities of any AI model based on a neural network. The primary parts here are the mathematical functions that are being used to actually make the model “learn”. These are called activation functions and are generally not

developed by programmers, instead being developed by mathematicians and physicists which the programmers later implement and deploy.

Activation functions are some of the most basic building blocks for neurons within neural networks. The type of activation function to be used is defined by the developer when creating each new layer of neurons within a network. The activation function's purpose is to provide a mathematical function for checking values passed through the neuron to see if the neuron needs to be activated or not. This process is a major part of how the model learns during training by constantly turning off and on neurons within the layers of the network based on mathematical calculations. This is also what happens when a deployed AI model makes a prediction, depending on which neurons are activated the final prediction of the model will turn out differently. Two of the most commonly used activation functions in machine learning today are ReLU and Sigmoid functions.

The Sigmoid function takes any real number as input and compresses it to a value between 0 and 1. This function is particularly useful in classification problems where a probability value is needed as output. For example, if a neuron receives an input sum of 2, the output from the Sigmoid function will be $\sigma(2) = \frac{1}{1+e^{-2}} \approx 0.88$. This means that the neuron is activated with a high probability, as the output is close to 1. If the output is negative, for example -2 the output will be $\sigma(2) = \frac{1}{1+e^{2}} \approx 0.12$ and here the neuron is activated with a low probability as the output is close to 0.

The ReLU function returns 0 if the input is negative and returns the input value if it is positive. If the input value is 2, the output of the function will be 2 and the neuron is fully activated. If the input is -2 then the ReLU function will return 0 and the neuron will remain inactive as shown in figure 7.

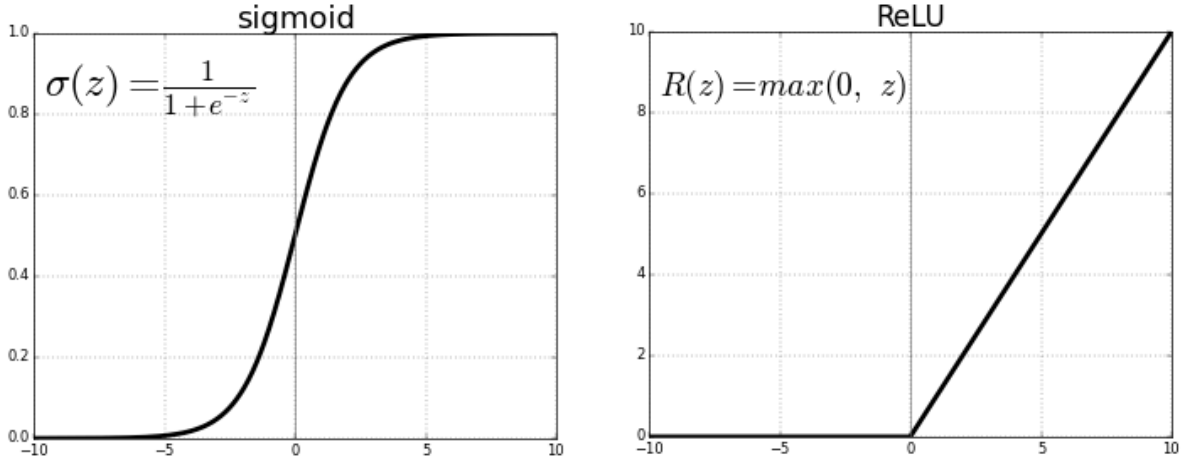

Figure 7: Example of Sigmoid function and ReLU

As displayed on the images these are two completely different mathematical expressions that handle values very differently with the biggest difference being of values under 0. The biggest difference this makes is computational power. Every activation of the Sigmoid function will take much more time and resources to complete than ReLu. This makes the time training and using a model about twice as long when using Sigmoid over ReLu. In the article “ReLu vs Sigmoid function in deep neural networks”[19] the results of a small comparison shows that the same amount of training was done more than 2 times faster by the ReLu function rather than the Sigmoid function. Apart from the longer time needed to use the Sigmoid function it often gets worse results as well in some scenarios because of two mathematical factors called vanishing gradient and convergence speed that may be a bit higher level to dive into than this paper intends. In most situations the ReLu function is both faster and yields better results than the Sigmoid function. Because of this the ReLu function was primarily used in the main body of the network while the Sigmoid function was utilised at the ending layers of the network for its higher precision when making final predictions.

4.5 Optimizer

Another important term and part of the model is its optimizer. Optimizers are only used for training and are generally used as one fixed optimizer for the entire model compared to activation functions that can differ for different layers. The similarities between optimizers and activation functions are they are both mathematical expressions used for optimising the model. To explain more easily, the activation functions mostly operate on a neuron layer while the optimizers operate at all the layers of a model using the information provided by the neurons in combination with the information from the validation to make adjustments in the network to better the performance. The basic idea of optimizers is to iteratively update the model weights in the direction that reduces the overall loss. By making continuous predictions on a training dataset and comparing the predictions to the correct answers a curve is formed for the loss based on how far off the predictions are from the true answer and by using the gradient of this curve the model can adjust and improve. This general idea is called gradient descent. Two of the most common optimizers today are SGD(Stochastic Gradient Descent) and Adam(Adaptive Moment Estimation) of which the Adam optimizer was used in this paper[19].

SGD updates model parameters by calculating the gradient of the loss function with respect to each parameter and moving the parameters in the opposite direction of the gradient. It requires more manual tuning of the learning rate and can benefit from techniques like momentum to accelerate convergence. However, SGD can struggle with noisy gradients and varying learning rates across different parameters. In contrast, Adam builds on SGD with techniques from other optimizers. Adam computes individual adaptive learning rates for each parameter by estimating both the first moment (mean) and the second moment (uncentered variance) of the gradients. This allows Adam to handle sparse gradients and automatically adjust the learning rate during training, often resulting in faster and more stable convergence compared to SGD. While Adam is generally preferred for its efficiency and quick convergence, it can yield worse results than a well optimised SGD implementation in some cases.

4.6 Callbacks

Callbacks are most often used as extra tools when training and developing models and can often be used to automate tasks that otherwise need to be performed manually[23]. Three main callbacks were used in this thesis, early stop, modelcheckpoint and tensorboard while a fourth was going to be added being learning rate schedule. Early stop is a callback where the training is automatically interrupted based on parameters being set by the developer. An example of early stopping used in this thesis was monitoring the validation loss and stopping a training session early if the loss did not improve for a set amount of epochs. Modelcheckpoint is a callback used for saving either the entire model or its weights in a h5.file after a specified set of epochs. The parameters for how often the model or its weights need to be saved is primarily based on the size of the model as well as the free amount of storage space on the PC training the model. The more layers and neurons a model has the bigger and deeper it gets with the storage needed for its weights increasing with the size of the model. The size needed to save the weights of the model used in this paper is slightly below 250Mb which in turn would take about 1Gb of storage every 4-5 weight files. When only training a few hundred epochs or less the storage would not be a problem, but when training a session on 40000 epochs the storage needed to save the weights for each epoch would be in terabytes. For this reason modelcheckpoint is used to save the weights or model at specified intervals for managing space and save data.

Tensorboard is a tool which can display graphs of different parameters such as overall loss and validation loss at real time following the training. This greatly helps viewing the training of any model and is the primary tool used when evaluating and tuning the model, its configuration and overall performance.

Tensorboard graphs can also be loaded from saved weights outside of real time training and the primary results of this paper comes from this program. The fourth and last callback that was intended to be used but unfortunately forgotten is called learning rate schedule. This callback helps tune the base learning rate of the model during training without interrupting it based on set parameters and evaluation metrics such as validation loss. This is very important to add when using older optimizers such as SGD since they do not have this feature built into the optimizer itself. The Adams optimizer does adjust learning rates for different parameters during training but does not tune the overall base learning rate in the same way a learning rate schedule does.

5. Results

This chapter will present the results from the different training sessions and the corresponding configurations to each graph presented using the sample from Matterport's github [15]. For the presentation of the graphs the tool TensorBoard was used. TensorBoard is a tool that provides measurements and visualisations during the machine learning workflow.

5.1 TensorBoard graphs

When taken as a whole, the following graphs illustrate how the model behaved during the training and validation stages. They propose common patterns of convergence for training deep learning models, characterised by an initial sharp decline in loss values, succeeded by a phase of more gradual improvements or stabilising. The variance in validation losses, especially for certain tasks like bounding box positioning and mask prediction, highlights the difficulties and complexities of these kinds of tasks.

5.1.1 Epoch loss

In machine learning, an epoch is a thorough pass through the entire dataset. One pass means a completed forward and backward pass through the entire dataset. This could be done by creating a single batch or dividing the data into smaller batches. The loss in this context is a measure of how well the model's predictions match the actual target values. This is a numerical value that represents the difference between the predicted output and the true output that is calculated after each validation attempt. This numerical value does not have a unit but is instead a subtracting difference between the predicted coordinate values compared to the actual coordinate values for each object and classification. When combining these two concepts it's called epoch loss. This is the cumulative loss over all training samples after one complete pass through the dataset. Monitoring this helps in understanding how the model behaves and improves over time. The loss value is a part of the multi-task loss function which is given by $L = L_{box} + L_{cls} + L_{mask}$ where L is the total loss value accumulated from the loss values as shown according to the article "Mask R-CNN" written by Kaiming He et al [20]. During validation the L is computed in the same manner as in training but is applied to the model's given validation dataset to measure how well the bounding-box predictions align with the ground-truth boxes in the validation data.

Figure 8 shows the training loss over 500 epochs. The loss starts from a high value about 5 and quickly decreases, stabilising around a lower value after about 200 epochs, after this the loss continues to decrease very gradually.

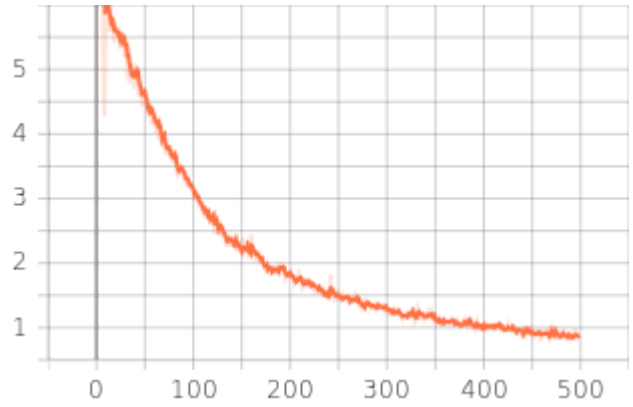

Figure 8: Epoch loss for results one

Figure 9 shows the epoch loss for the training dataset over 40000 epochs. It shows significant spikes in loss at regular intervals, rising up to 4.5 at these points, with most other values clustering near the bottom of the graph around 0.5 or less.

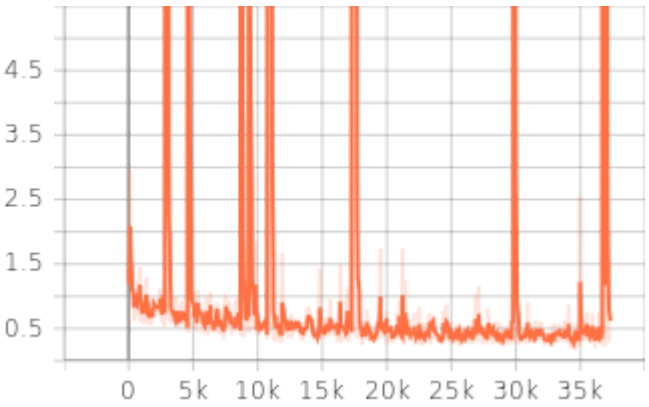

Figure 9: Epoch loss for results two

5.1.2 Epoch val loss

Epoch val loss refers to the validation loss measured after each complete pass through the validation dataset. Validation loss is calculated on a separate set of data that is not used for training but is used to tune hyperparameters and to monitor how well the model generalises to new, unseen data. Monitoring this value is crucial as it helps to identify issues like overfitting, where the model performs well on the given training data but poorly on the unseen data from the validation dataset. Figure 10 shows the

validation loss over 500 epochs. It starts at a relatively high value, about 6 and then stabilises around 200 to 300 epochs to a value of 2. However the validation loss shows higher variability and noise compared to the training loss, especially beyond epoch 200. This increased variability and noise in the validation loss can indicate the model's performance on unseen data fluctuates, highlighting the importance of considering both training and validation loss when evaluating model performance.

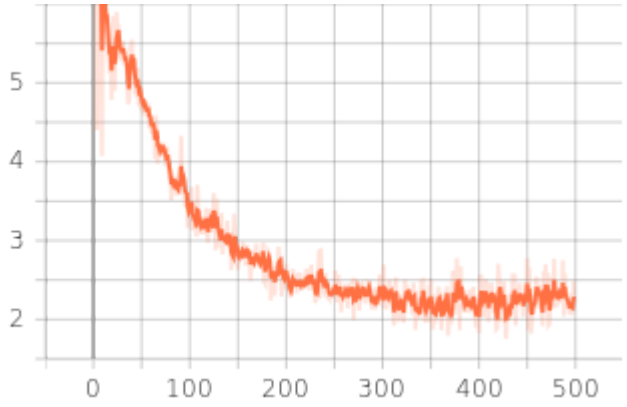

Figure 10: Epoch val loss for results one

For the second experiment the validation loss across the epochs is shown in figure 11. The loss is somewhat volatile, with several sharp peaks reaching up to around 2.2. Most values remain below 1.4, showing a recurring pattern of spikes throughout the training process.

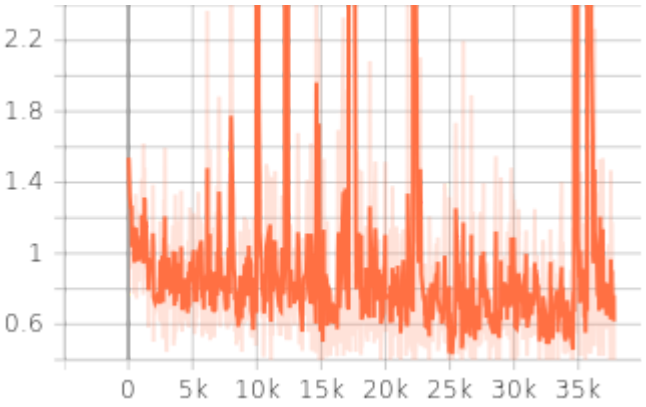

Figure 11: Epoch val loss for results two

5.1.3 Epoch val mR-CNN mask loss

Epoch val mR-CNN mask loss specifically refers to the validation loss associated with the Mask R-CNN part of the model, which is responsible for generating masks for the object detection tasks. This measures how accurately the predicted masks match the ground truth masks in the validation dataset. Figure 12 shows the validation loss for the Mask R-CNN part of the model. It starts around 0.36, drops sharply, and the levels off around 0.24 by epoch 200. Post-epoch 200, the graph exhibits substantial fluctuation with spikes. This pattern indicates that while the model improves significantly, its performance on the validation data becomes more variable after epoch 200. These fluctuations could suggest sensitivity to certain validation samples from the validation dataset or possible overfitting.

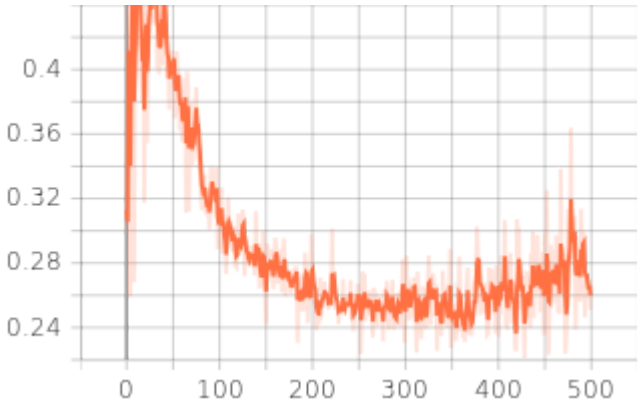

Figure 12: Epoch val MR-CNN mask loss for results one

Figure 13 shows the validation for the mask predictions of the Mask R-CNN. Loss values show a high degree of variation, with numerous peaks around 0.32, while maintaining a general lever closer to 0.24, going down during around 15000 and 25000 epochs to a value lower than this.

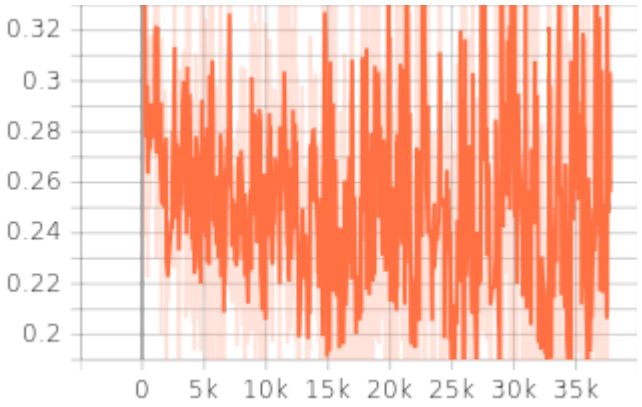

Figure 13: Epoch val MR-CNN mask loss for results two

5.1.4 Epoch val rpn bbox loss

Epoch val rpn bbox loss refers to the validation loss associated with the bounding box predictions of the Region Proposal Network (RPN) in the model. The RPN is responsible for proposing candidate object bounding boxes which are then refined by subsequent network stages. The bounding box loss measures how well these predictions are compared to the true bounding boxes in the validation dataset. Figure 14 keeps track of the loss associated with the bounding box predictions of the RPN on validation data. It starts around 0.75, decreases and settles into a more erratic pattern post epoch 100, hovering around 0.25 with considerable noise.

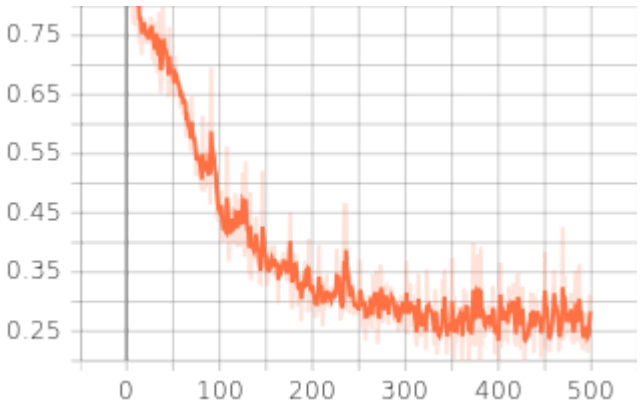

Figure 14: Epoch val rpn bbox loss for result one

Depicted in figure 15 validation loss for bounding box predictions from the RPN component. The graph is characterised by frequent and high spikes, with loss values peaking around 1.4 and a baseline fluctuation between 0.6 and 0.8.

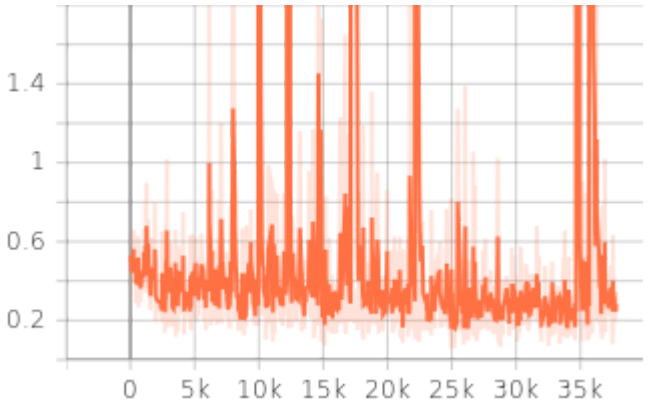

Figure 15: Epoch val rpn bbox loss for result two

5.1.5 Epoch val mR-CNN bbox loss

This loss refers to the bounding box loss during the validation phase of the Mask R-CNN model. This loss is evaluating how well the model is predicting the bounding box coordinates for objects in the validation dataset. Figure 16 represents the validation loss for the bounding box prediction made by the Mask R-CNN part of the model over 500 epochs. The validation loss starts at about 0.55 and then drops to a lower value of 0.25 by around epoch 100. It continues to decrease slowly and stabilises around 0.15 after about epoch 300.

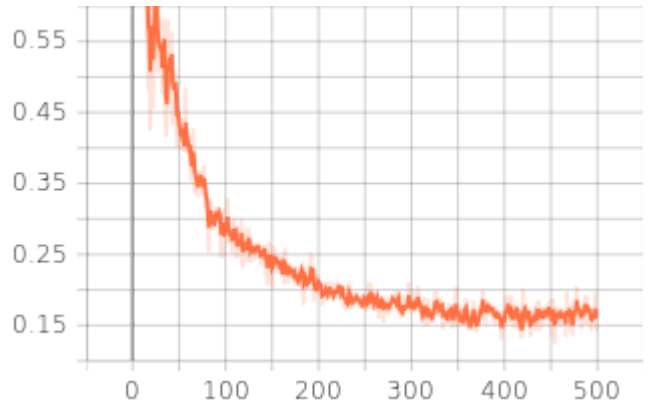

Figure 16: Epoch val MR-CNN bbox loss for result one

Figure 17 presents the validation loss for bounding box predictions made by the Mask R-CNN. The loss fluctuates substantially, with many sharp peaks that briefly reach values as high as 0.24, while the majority of the values lie around 0.16.

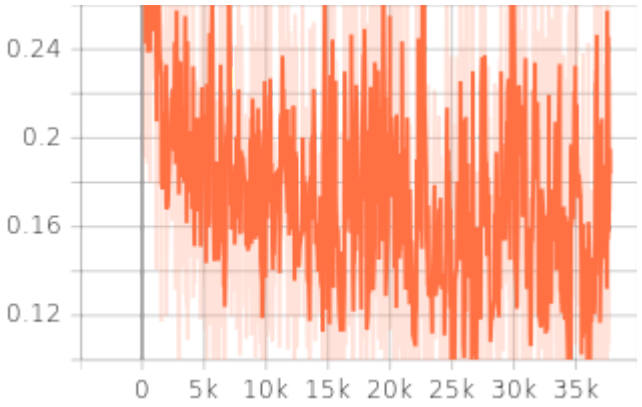

Figure 17: Epoch val MR-CNN bbox loss for result two

5.1.6 Epoch val mR-CNN class loss

The classification loss values are primarily influenced by the confidence score of the true class. Consequently, these losses indicate the model's confidence in predicting the class labels, or how accurately the model predicts the correct class. For this specific loss, all object classes are included in the classification [20]. Figure 18 keeps track of the validation loss related to the class prediction component of the Mask R-CNN. It starts from a relative low value of 0.55 indicating that there is low error at the start of the training. The loss exhibits a significant amount of noise and fluctuation. The noise and fluctuations in the validation loss might indicate challenges in consistently predicting class labels accurately, which could be due to the complexity of the classification task or a varying validation dataset.

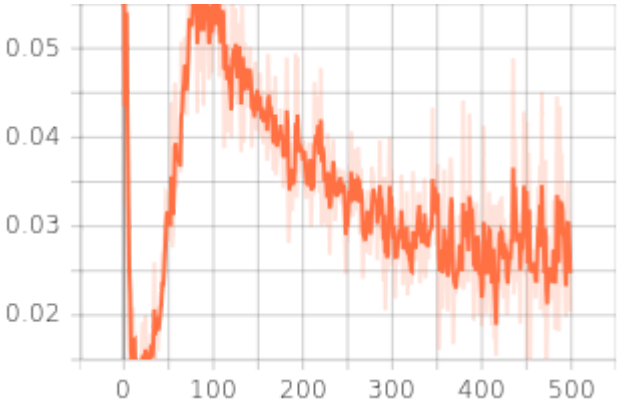

Figure 18: Epoch val MR-CNN class loss for result one

Figure 19 shows the validation loss associated with class predictions by the Mask R-CNN. This loss is very volatile, displaying a pattern of sharp and frequent spikes that vary significantly in height, ranging mostly between 0.01 and 0,024.

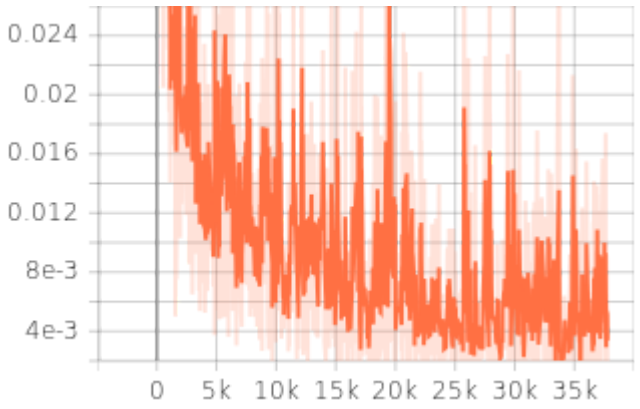

Figure 19: Epoch val MR-CNN class loss for result two

5.1.7 Epoch val rpn class loss

Figure 20 shows the loss for the class prediction component of the region proposal network on the validation set. The loss starts just below 0.055 and experiences a sharp decrease, settling to values between 0.01 and 0.03 after about 200 epochs. Similar to the Mask R-CNN class loss, this graph shows fluctuations that reflect challenges in consistently predicting class scores accurately in varying validations scenarios

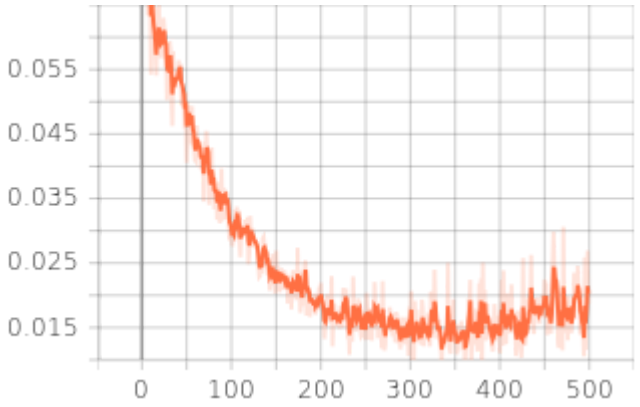

Figure 20: Epoch val rpn class loss for result one

Figure 21 shows the validation loss associated with the class predictions of the Mask R-CNN. The loss exhibits volatility, with sharp frequent spikes. The values vary significantly in height, mostly ranging between 0.005 and 0.035. This pattern suggests that the model's on the validation set fluctuates considerably during training, indicating potential instability or sensitivity to the data.

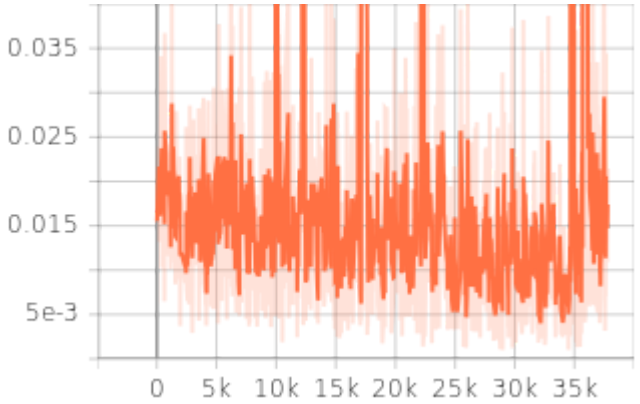

Figure 21: Epoch val rpn class loss for result two

5.2 Comparative analysis

When comparing these two training sessions with 500 epochs and 40000 epochs it is apparent that the first experiment, which were configured with 500 epochs and 50 steps per epoch, shows a more stable and gradual decrease in loss for both epoch loss and epoch val M-R-CNN mask loss as shown in section 5.2 and 5.4. The epoch loss for the first experiment decreases and stabilises around 400 epochs with moderate variation after significant training progression. This indicates a fairly typical learning curve where it starts from a high value and then curves out and eventually flattens out. The behaviour of this curve is fairly common with model's that effectively learn and generalise without major disturbances, the behaviour could be explained by the more detailed updates when configuring more steps per epoch for this experiment.

Conversely, the second experiment, which extends to 40000 epochs but only uses one step per epoch, presents a fundamentally different behaviour than the first experiment. The spikes in epoch loss could suggest that the model encounters more variability and possibly overfits. When both curves are separated by substantial gaps and the model performs significantly better on training data than on the validation data then the model is typically overfitting. Similarly, the validation mask loss shows degree of variability, indicating that while the model is learning it also encounters periods of instability that could affect the generalisation performance. When using one step per epoch the model receives less frequent updates which could contribute to the spikes in the graph. However, it is noteworthy that the second experiment receives lower minimum loss values compared to the first experiment, indicating that with enough training time, they might become more similar.

These two different experiments emphasise the need to watch both training and validation losses over many epochs and to consider how the number of steps per epoch affects the model's learning. The instability seen in the second experiment shows that strategies like adjusting the learning rate, early stopping and increasing the steps per epoch are important to prevent overfitting and keep performance stable during long training periods. Even though the model during some periods produced better results than the first experiment.

While both experiments show that the model is capable of learning and improving, the extended training in the second experiment with reduced steps per epoch contributes to additional complexities that need careful management to ensure consistent and reliable model performance. Adjusting the steps per epochs could be a crucial factor in achieving more stable and effective training outcomes, even the second

experiment demonstrates the potential for lower minimum loss values with extended training and lower time when training.

6. Conclusion

In the conclusion of this thesis four main areas will be discussed. The success of the thesis as a whole, discussing the answer found to the problem formulations, an ethical evaluation of this work's impact on the labour market as well as future improvements for the work.

6.1 Success of the thesis

When evaluating the success of this thesis there are two aspects to discuss, the practical success as well as the theoretical success. When evaluating the practical success of this thesis the goal formulation needs to be checked against the final results. In the goal formulation of chapter one four main goals were described as being dynamic, flexible, accurate and efficient to be used for evaluating a complete product. The product would be considered dynamic if it could be trained on at least three different PCB models without needing to change any of the code, during the training the dataset consisted of 6 different PCB models and in the way the dataset pipeline as well as the AI model was implemented it can be trained on any number of PCB models without needing to change the main code at all. This was with other words successful.

The next check was to see if the model could achieve an accuracy of 95% during a test which can be approximated at the validation step during training. Since there was a problem getting the model to calculate and display the accuracy correctly the validation loss was the main metric analysed since approximations of the accuracy can be made from it. Mister.Hari explained that when he previously had gotten around 95% accuracy for an earlier work his validation loss values had stabilised around 0.05. With that in mind he argued that the accuracy of the model could be approximated from subtracting the validation loss for a specific epoch from 1. Using that logic the best achieved accuracy could be approximated to around 80% from the epoch with the lowest value of just below 0.2. So when looking at the best results gotten from any of the training sessions the validation loss estimated to accuracy did not achieve 95%. In this metric the model did not meet the initial criteria and was therefore not successful.

An important consideration to address is whether the original metric goal was feasible within the parameters of the thesis work. Since an approximated accuracy of 80% could be achieved within about a month of training and tuning a conclusion can be drawn that with more time the model could potentially reach the intended metric. What is harder to approximate is how much more time would be needed to

achieve this since it seems that the training progress towards reaching lower validation loss values is not linear and heavily depend on expertise from the developers. With that in mind it can be argued that the demanded accuracy may not have been feasible for this work because of the uncertain time needed to achieve it.

For the last two metrics being testing the AI model with two different camera modules as well as testing the deployed model on a completely new PCB model, testing and evaluating of these metrics could not be performed because of time limitations in getting the accuracy as close to 95% as possible. Hence these two metrics were automatically unsuccessful. In analysis of the work done so far as well as in discussions with Mr. Hari, it is theorised that these two tests could become successful in the future. The reason for believing that the flexibility metric can be achieved has to do with the fact that the model itself can be deployed in multiple different ways and within different applications since it is a standard python file. Drawing from this fact it can be deduced that using the model with differing hardware should not be an issue. The goal of effectiveness is theorised to be able to be achieved since the model has been trained to find the 6 predefined defects on a given PCB no matter their shape or size, hence a new PCB schematic should not have a big impact on the models performance and ability to find it’s known defects.

With all of this in mind, the thesis did not succeed in practicality since a finished product was not achieved and the work thereby did not fulfil all of the main goals defined in the goal formulation. When looking at the theoretical aspect of the succession of this thesis the main topics of discussion will be if the thesis is believed to be able to finish with more time and if the results received point to a practical use case for the model.

As previously mentioned it is theorised that the thesis can be successful with more time and expertise since the model has shown steady improvements for all of the time it was being trained. In theory the thesis's work can be continued upon to reach the four formulated and discussed goals. The amount of extra time needed for this however varies widely depending on the expertise of the person or people picking up the work. The time needed for getting the accuracy to 95% is hard to determine since AI training and research is mainly based on probabilities. When the training and validation of the model performs as intended the rest of the work should take between 2-4 weeks to complete.

6.2 Meeting the problem formulations

In this chapter of the conclusion, the 5 problem formulation questions will be discussed and answered. The five questions from chapter one are as follows:

1. What methods can be used to make the AI model dynamic and flexible to accommodate multiple PCB models?
2. How can the accuracy of the AI model for defect detection be measured and validated?
3. What methods can be employed to tailor the data set for training the model?
4. Which algorithms are most effective for classifying different types of faults in a given PCB?
5. How large a data set will be required to train and adapt the model?

Three of these questions can be answered by explaining different parts of the dataset pipeline, these are questions 1, 3 and 5. The dataset chosen to use from Kaggle had as previously mentioned 6 different models of PCB layouts which varied widely in design. Since the complete dataset contained an equal amount of these different PCB-model designs the model did not overfit to one single PCB design but stayed consistent on all of them, this answers question one. The methods used to tailor the dataset for the task at hand as described in question three was primarily using the imgaug library for handling the bounding boxes around the defections within the images to make sure the model had complete and true answers for each training data. Lastly question five was solved by creating the augmentation pipeline for the original Kaggle dataset that could create an infinite amount of augmented data points. With this approach a set amount of data was not required since more data could always be provided if needed.
For the final two questions, two and four, they will be answered separately and a bit differently since they touch more on the model itself rather than the dataset or augmentation pipeline. As described in question two a way of measuring the accuracy of the model was needed and as previously mentioned a direct accuracy checking implementation was tried and unfortunately failed. However, this does not mean that no measuring of the accuracy could be made since estimations of accuracy can be drawn from the training validation loss. With future testing an exact accuracy will be able to be measured from deploying a trained model and running a passthrough of a new unseen example image.

As for question four, the best answer to that specific question would be that it's almost impossible to find one single algorithm that would be the most effective at classifying different types of faults on a given PCB. For the classification task many different algorithms are used together when creating the part of the model that handles these classifications. A better question would have been to ask which implemented

model is the most effective in the classification area where a comparison between for example mask R-CNN and yolo v8 could have been made.

6.3 Ethical dilemmas

In this thesis, the thesis workers explored the application of machine learning techniques in identifying defects in printed circuit boards, with the goal to enhance the efficiency and accuracy in manufacturing processes. While these techniques offer significant benefits they also introduce various ethical dilemmas that warrant careful considerations.

6.3.1 Job Displacement

One of the most significant ethical dilemmas associated with the implementation of AI-based Automated Optical Inspection systems in PCB manufacturing is the potential displacement of workers and working opportunities when it comes to manual inspection in the production line. The introduction of automated systems that can perform quality control tasks with high accuracy might reduce the need for human inspection, which could lead to job losses in these industries. It is essential to address this issue by considering the social responsibility of implementing such technologies. In this case both developers and companies could explore different strategies to retain and upskill affected workers enabling them to transition to new roles within the company.

6.3.2 Confidential Information and Data Security

The use of AI in manufacturing often involves handling sensitive data, including designs and manufacturing processes. It is crucial to ensure that this data is stored and managed securely to prevent unauthorised access or leaks. By following confidentiality principles according to the CIA[21] triad, sensitive information could be kept safe from unauthorised users. This could be achieved by implementing a robust data security system where encryption and secure data storage. In this case it’s essential to maintain confidentiality of sensitive information.

6.3.3 Conclusion

In conclusion, while machine learning offers substantial potential for improving PCB defect detection, it also highlights ethical considerations. Balancing different ethical dilemmas with technological advancements is crucial for the sustainable and responsible development of AI systems. By integrating

ethical principles and taking these dilemmas in consideration into the core of technological development we can ensure that future innovations AI or not serve the greater good.

6.4 Future work

The completion of this thesis marks and opens up opportunities in advancing automated defect detection for printed circuit boards using artificial intelligence. However there remain several promising directions for future research and development that could enhance such systems and their capabilities to broaden its applicability.

One of the next steps in improving the model is to make an extensive deployment and testing of the AI model across a diverse set of real-world scenarios. This phase is crucial and critical for validating the model's accuracy and reliability in various operational settings. Comprehensive testing will help identify any potential shortcomings and areas for improvement, ensuring the system is robust and reliable.

Also integrating the AI model with existing manufacturing systems presents another important avenue for future work. This integration would enable real-time defect detection, streamlining the production process and reducing reliance on manual inspection. Such a system would not only enhance operational efficiency but also contribute to higher quality standards in PCB manufacturing. Another possible improvement is to integrate this model into a smaller camera, not made for manufacturing, to see if the model can be used in a more dynamic way which could lead to lower costs in the manufacturing industry.

Further improvements to the model itself are also crucial in order to achieve better loss scores. Exploring alternative activation functions, such as Leaky ReLU[22], could offer better handling of non-linearities, potentially improving the model's performance. Additionally, investigating different backbone architectures beyond the current use, being ResNet, may enhance feature extraction capabilities, leading to more accurate defect detection. Experimenting with various optimisation algorithms, other than the commonly used Adam, could accelerate the training process and improve model generalisation, making the system more effective across different PCB types.

These enhancements will not only push boundaries of current technology but also facilitate its practical implementation in industrial environments where precision and efficiency are crucial.

7. Terminology

Adam Optimizer: A popular optimization algorithm in machine learning allowing efficient training of deep neural networks.

Bounding Box: In object detection, a bounding box is a rectangle drawn around an object in an image.

CNN (Convolutional Neural Network): A type of deep neural network commonly used for analysing visual imagery.

Epoch: One complete pass through the entire dataset in a machine learning training process.

Gradient Descent: An optimization algorithm used to minimise some function by moving towards the steepest descent as defined by the negative of the gradient.

Hyperparameter Tuning: The process of optimising the hyperparameters of a model to improve its performance.

Keras: An open-source library that provides a Python interface for artificial neural networks. Keras acts as a high-level API.

Mask R-CNN: An extension of Faster R-CNN that includes a branch for predicting segmentation masks of each Region of Interest.

Overfitting: A modelling error in machine learning where a model learns the training data too well.

PCB (Printed Circuit Board): A board used in electronics to mechanically support and electrically connect electronic components.

ReLU (Rectified Linear Unit): An activation function used in neural networks. The function returns 0 if it receives any negative input, but for any positive value, it returns that same value back.

SGD (Stochastic Gradient Descent): A variant of gradient descent where the model parameters are updated for each training example.

TensorBoard: A visualisation toolkit for machine learning experiments. Provides a graph with the loss values for each epoch.

TensorFlow: An open-source machine learning framework used for a wide range of tasks including deep learning.

Kaggle: A platform for data science competitions, datasets, and collaborative projects. Provides resources and a community for machine learning and data science practitioners.

8. References

[1] Z. Peterson, "What is a PCB and PCB Design?", Altium, 5 Oct. 2020. Updated 30 Jan. 2024. https://resources.altium.com/p/what-is-a-PCB

[2] S. Putera and Z. Ibrahim, "Printed circuit board defect detection using mathematical morphology and MATLAB image processing tools," in *Proceedings of the IEEE International Conference on Engineering, Technology and Computing.*, 2010, pp. 1-5. DOI: 10.1109/ICETC.2010.5530052.

[3] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development.*, vol. 3, pp. 210-229, 1959.

[4] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson Education, Inc., 2021.

[5] "Activation Functions in Neural Networks," *GeeksforGeeks*. https://www.geeksforgeeks.org/activation-functions-neural-networks/

[6] E. Odemakinde, "Mask R-CNN: A beginner's guide," *Viso.ai*, Jul. 2021. https://viso.ai/deep-learning/mask-r-cnn/

[7] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object Detection in 20 Years: A Survey," *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257-270, Mar. 2023. DOI: 10.1109/JPROC.2023.3238524.

[8] A. Rosebrock, "Image Classification Basics," *PyImageSearch*, 17 Apr. 2021. https://pyimagesearch.com/2021/04/17/image-classification-basics/

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," arXiv:1512.03385, 2015. https://arxiv.org/abs/1512.03385

[10] S. T. Russom, "AI-based Quality Inspection for Short-Series Production – Using synthetic dataset to perform instance segmentation for quality inspection," Master's thesis, *Department of Computer and Information Science.*, Linköping University, Sweden, 2022.

[11] A. Khatova, "PCB Defects Dataset," *Kaggle*, 2020. https://www.kaggle.com/datasets/akhatova/PCB-defects/data

[12] W. Huang and P. Wei, "A PCB Dataset for Defects Detection and Classification," *arXiv:1901.08204*, 2019. https://arxiv.org/abs/1901.08204

[13] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," *arXiv:1605.08695*, 2015. https://arxiv.org/abs/1605.08695

[14] "Deep Learning Resources," *NVIDIA Developer*. https://developer.nvidia.com/deep-learning

[15] W. Abdulla, "Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow," *GitHub repository*, 2017. https://github.com/matterport/Mask_R-CNN

[16] "Imgaug Implementation," *GitHub*. https://github.com/aleju/imgaug

[17] A. Karazhay, "Guide: Augment Images and Multiple Bounding Boxes for Deep Learning in 4 Steps with the Notebook," *Medium*. https://medium.com/@a.karazhay/guide-augment-images-and-multiple-bounding-boxes-for-deep-learning-in-4-steps-with-the-notebook-9b263e414dac

[18]"Install TensorFlow with GPU support," TensorFlow. https://www.TensorFlow.org/install/source#GPU

[19] A. Thakur, "ReLU vs. Sigmoid Function in Deep Neural Networks," *Weights & Biases*. https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks--VmlldzoyMDk0MzI

[20] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," arXiv:1703.06870, 2018. https://arxiv.org/abs/1703.06870

[21]VinciWorks, "What Are Confidentiality, Integrity and Availability in Information Security?," VinciWorks. https://vinciworks.com/blog/what-are-confidentiality-integrity-and-availability-in-information-security/

[22]J. C. Olamendy, "Understanding ReLU, LeakyReLU, and PReLU: A Comprehensive Guide," Medium. https://medium.com/@juanc.olamendy/understanding-relu-leakyrelu-and-prelu-a-comprehensive-guide-20f2775d3d64

[23] “Callback” *Tensorflow.* https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback