

**Bachelor's thesis
15 credits, ground level**

Pharmaceutical Packaging Defect Detection Using Machine Learning-based Image Processing Algorithms

Farmaceutisk förpackningsfelsökning med hjälp av maskininlärningsbaserade bildbehandlingsalgoritmer

Anas Al-Saleh
Gabriella Pantic

Abstract

Due to the increased interest in deep learning in recent years, there has been a significant upsurge in research in regard to the implementation of machine learning-based systems in production environments. The main points of interest has been in defect detection, categorisation and full automation.

We collaborated with Metaqlix AB, a company interested in enhancing quality assurance processes for their clients using image recognition technology, to create a product for a pharmaceutical production line. The technology uses deep learning, specifically the Faster R-CNN architecture, to find packaging defects in real-time while moving along a conveyor belt. To train the model, a dataset of images is created and manually annotated to highlight various packaging defects. The system is then evaluated for its accuracy and efficiency in detecting these defects. The results became increasingly more efficient with the latest accuracy rate reaching 88.33%, which in turn shows that the proposed system can effectively automate the defect detection process, which improves the overall quality assurance workflow in pharmaceutical production.

Further discussion includes the details around some of the challenges that arose during the development process, including questions regarding data quality and how well the system performs under different conditions. By highlighting these aspects, the study provides valuable insights for future improvements and enables Metaqlix to better tailor its solutions to meet customer demands. In summary, this study shows a method in which machine learning and image recognition can change the quality assurance process in the pharmaceutical industry.

Sammanfattning

På grund av det ökade intresset för djupinlärning har det skett en betydande uppgång i mängden forskning när det gäller implementering av maskininlärningsbaserade system i produktionsmiljöer, där defektdetektering, kategorisering och full automatisering är huvudambitionerna.

Projektet genomfördes i samarbete med Metaqlix AB, ett företag som vill förbättra kvalitetsäkringsprocesserna för sina kunder genom att använda bildigenkänningsteknik. Systemet som utvecklades använder en avancerad djupinlärningsarkitektur, specifikt faster R-CNN, för att noggrant upptäcka defekter i förpackningar. För att träna modellen skapades en databas av bilder, där varje bild markerades manuellt för att indikera förekomsten av olika typer av defekter. Denna annotering är avgörande för att säkerställa att modellen kan lära sig att känna igen och klassificera defekter korrekt. Resultaten blev alltmer effektiva med den senaste noggrannhetsgraden som nådde 88,33%, vilket i sin tur visar att det föreslagna systemet effektivt kan automatisera defektdetekteringsprocessen och kan leda till en minskning av felaktiga produkter som når marknaden.

Vidare diskuterar arbetet de utmaningar som har uppstått under utvecklingsprocessen, inklusive frågor kring datakvalitet och hur väl systemet presterar under olika förhållanden. Genom att belysa dessa aspekter ger studien viktiga insikter för framtida förbättringar och möjliggör för Metaqlix att anpassa sina lösningar för att möta kundernas krav ännu bättre. Sammanfattningsvis visar denna studie en metod kring hur maskininlärning och bildigenkänning kan förändra kvalitetsäkringsprocesserna inom den farmaceutiska industrin.

Glossary

Faster R-CNN: A deep learning architecture used for object detection and segmentation in images. It is built on the R-CNN model and provides an efficient way to detect and locate objects in real time.

Bounding Box: Rectangular outline drawn around a detected object, in an image or video frame, used to visually indicate and locate the object within the frame.

Annotation: The process of labeling or identifying objects in images to create a training dataset for machine learning. This often involves drawing bounding boxes around objects and assigning them labels.

Epochs: The number of complete passes through the entire training dataset during the training process of a machine learning model.

Weights: Numerical values that represent the strength of connections between neurons in various layers of the network.

Steps: Number of steps per epoch, where the network updates its weights.

Algorithm: A set of steps or rules that follow a sequence to solve a problem or perform a task.

Batch size: The number of samples processed together in one iteration.

Hyperparameter tuning: Changing the parameters that control a machine learning models learning process in order to enhance its performance.

Acronyms

AI - Artificial Intelligence

CNN - Convolutional Neural Network

ML - Machine Learning

TF - TensorFlow (abbreviation)

SDRM – Systems Development Research Methodology

XML - Extensible Markup Language

R-CNN - Region-based Convolutional Neural Network

JSON - JavaScript Object Notation

OpenCV - Open-Source Computer Vision Library

CONV – Convolutional

ROI – Region of Interest

RPN – Region Proposal Network

CPU – Central Processing Unit

GPU – Graphics Processing Unit

Table of Contents

1	Introduction	1
1.1	Background	1
1.2	Problem.....	1
1.3	Research Questions	2
1.4	Scope and Limitations	2
1.5	System Requirements	3
2	Theoretical Background.....	4
2.1	Deep Learning.....	4
2.1.1	Image Processing	4
2.1.2	Supervised Learning.....	5
2.1.3	Anaconda Distribution.....	6
2.1.4	Evaluation Metrics	6
2.1.5	Loss Function.....	7
2.1.6	Nvidia and CUDA.....	7
2.2	Deep Learning Architecture.....	7
2.2.1	Convolutional Neural Network (CNN)	8
2.2.2	Faster Region-based Convolutional Neural Network (Faster R-CNN).....	9
2.2.3	Edge Impulse	9
2.3	Machine Learning Libraries.....	10
2.3.1	OpenCV	10
2.3.2	TensorFlow	10
2.3.3	Annotation Tool	11
3	Related Work	12
3.1	A Real-time Object Detection Algorithm For Video	12
3.2	Fruit Classification Using Optimized CNN	12
3.3	ARES: An Automated Rotten Egg Sorter Utilizing the Egg's Physical Properties and Artificial Neural Network	13
3.4	Embedded System Implementation for Material Recognition Using Deep Learning	13
4	Methodology.....	15
4.1	Construct a Conceptual Framework	15
4.2	Develop a System Architecture	16
4.3	Analyse and Design the System.....	18
4.3.1	Model Comparison.....	18
4.3.2	Versions and Compatibilities.....	19

4.4	Build the Prototype System	20
4.4.1	Use of Model.....	20
4.4.2	Number of Images and Image Collection	20
4.4.3	Image Preprocessing.....	20
4.5	Observe and Evaluate the System	21
5	Results and Analysis.....	22
5.1	Construct a Conceptual Framework	22
5.1.1	Data Collection.....	22
5.2	Develop a System Architecture	24
5.2.1	Datasets.....	24
5.3	Analyse and Design the System.....	25
5.3.1	Training.....	25
5.3.2	Data Analysis	33
5.4	Build the Prototype System	34
5.4.1	Graphical User Interface (GUI)	35
5.5	Observe and Evaluate the System	37
5.5.1	Evaluation.....	37
6	Discussion.....	45
7	Conclusion	46
8	Future Work	47
	References.....	48

1 Introduction

1.1 Background

Pharmaceutical products are something used by many people on a daily basis. There are different types of pharmaceutical products ranging from ingestible, injectable and topical. The packaging of these products serves as a shield against contamination, tampering and degradation, and provides a sense of security for the customer. However, despite having rigorous regulations and quality control procedures in place, packaging errors continues to be noticeable in the end-market and affects a customer's willingness to purchase the product [1].

Seeing as there exists some problems in this area, the authors of this thesis have entered a collaboration with a company called Metaqlix AB. The company has recently taken an interest in image recognition and wants to, on behalf of one of their customers, create a system that is able to perform image recognition in conjunction with quality assurance of the customer's products. This customer operates within production of a certain pharmaceutical product and who want to improve the quality assurance process by implementing an image processing algorithm that detects faults in the product packaging while it moves on a conveyor belt. Quality assurance is important to ensure that the products are in sellable condition, considering customers will show apprehensiveness towards noticeably faulty products [1].

Artificial Intelligence (AI) as a concept has existed for decades but it has recently received a significant increase in attention [2]. This due to the promising outlook of increased efficiency, which may in turn lead to an increase in revenue for companies who invest in this technology. The way to increase the revenue may be through shifting around the competencies of the workforce and identifying unnecessary competencies. In continuation, Machine Learning (ML), a branch of AI, is used to develop systems that can mimic the way a person thinks and works [3]. If correctly trained, an ML-based system can achieve greater accuracy in environments that require specific accuracy rates to be considered viable. With reference to the pharmaceutical environments, an ML backed by image recognition may show promise in identifying defects and irregularities in production lines.

1.2 Problem

Metaqlix has requested the development of a system that is able to increase the productivity of the quality assurance process in one of Metaqlix's customers' production lines. The system utilises concepts such as AI, image recognition and others to accurately detect defective products. Metaqlix provided a set of products, both non-defect and some with common defects (see Figure 1), with which a dataset is created to train the system's algorithm.

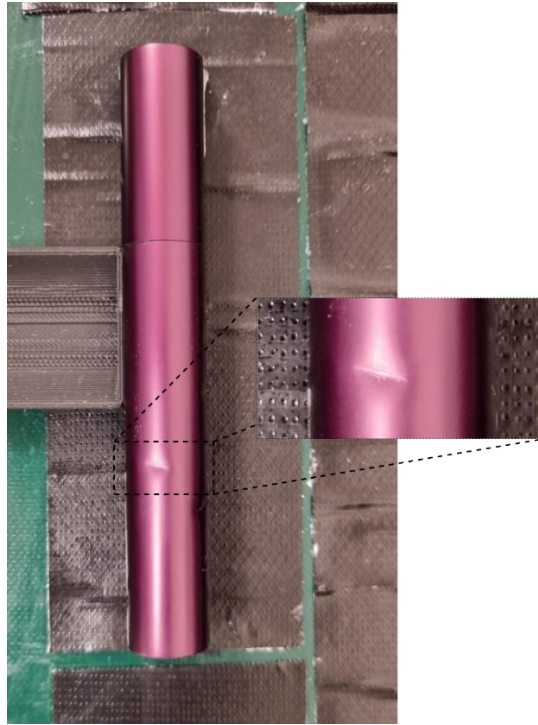


Figure 1. Example of defect on product

1.3 Research Questions

This subsection specifies the research questions addressed in this study.

RQ1: What is a suitable AI solution for the proposed package image recognition program?

RQ2: How can the performance level of the existing algorithms for package defect recognition be improved?

1.4 Scope and Limitations

This section addresses the limitations imposed upon the project and the details in this study.

L1: The trained model is specifically designed to detect a certain type of defect (see Figure 1).

L2: The trained model has only been trained on a product of a certain colour, see Figure 1. The product line has multiple colours for different kinds of products, but uses the same type of package.

1.5 System Requirements

The following points are the system requirements provided by Metaqlix to ensure the system's performance:

Functional requirements

- The system must be able to identify dents in the product's packaging.
- The system must be portable and function as intended in any of the company's production environments.
- The system must have a screen in which the collected data, specified in the non-functional requirements, and live feed is displayed.

Non-functional requirements

- The system displays the collected data pertaining to the following:
 - Recognition accuracy expressed in percent (%) while detecting an object in front of the camera.
 - The recognised defect.
 - The number of defective products found within an adjustable range.
 - The total number of scanned products within an adjustable range.
 - The following time variables:
 - Start date and time of one production batch.
 - End date and time of one production batch.
 - Total time from start to end of one production batch.



Figure 2. The product to be inspected for defects

2 Theoretical Background

A dynamic and quickly developing area of artificial intelligence called "machine learning" aims to create systems that can learn from data and make accurate estimations or decisions without having to be manually programmed for each unique activity [4]. Machine learning is generally divided into multiple types, and each one is crucial to the development and use of AI methods.

Advanced neural networks are used in deep learning to simulate the learning process. Using architectures such as Convolutional Neural Networks (CNNs) and Region-based Convolutional Neural Networks (R-CNNs), these algorithms possess the capability to identify patterns and estimate outcomes [5] [6].

2.1 Deep Learning

Deep learning is a type of machine learning that focuses on using neural networks and understands data. A large amount of data is used to train deep learning models, which then gradually enhance the models' performance. Traditional machine learning, meaning "non-deep", features a very simple neural network containing one or two computational layers [7]. Whereas deep learning models utilise three or more layers, more layers being upwards of thousands of layers, to train the model [8].

2.1.1 Image Processing

Image processing is an important area within computer science that focuses on analysing and improving images to extract useful information or enhance the visual representation of data. To effectively analyse and extract information from images, several key techniques and operations are employed. Among the most fundamental of these are [9]:

- **Binarization:** To simplify and speed up image processing, a colour or grayscale image can be converted into a binary image. This process reduces the image to just two-colour levels – black and white.
- **Smoothing:** A method used to reduce the sharpness and fine details of objects in an image, resulting in a blurring effect.
- **Sharpening:** A technique that enhances the edges and fine details of objects in an image, making them clearer and more defined.
- **Noise Removal and De-blurring:** Techniques that reduce unwanted noise and correct blurriness in images, improving overall image quality and sharpness.
- **Edge Extraction:** A method used to identify the boundaries of objects within an image, aiding in image segmentation and detailed analysis.
- **Segmentation:** This technique involves dividing an image into multiple segments or regions. It makes it possible to isolate particular parts or regions within an image, allowing faster processing of regions and easier defect detection.

After explaining the key image processing techniques, it's useful to understand how digital images are structured. Digital image processing converts images into a digital format, which can be represented as a two-dimensional function, $f(x,y)$, where x and y are spatial coordinates. Spatial coordinates can include fractions, allowing for more precise positioning even in between the pixels [10].

2.1.2 Supervised Learning

Supervised learning, also referred to as supervised machine learning, involves the use of labeled datasets to train algorithms for classification or result prediction. [11]. Labels are used to identify information about the piece of data and indicate its purpose [12]. It is a method that is done manually and called data labeling [13]. Supervised learning is divided into two categories, regression and classification. The goal of supervised learning involves creating a model that generalises well to new data. To achieve this, the model is evaluated on a separate validation dataset not used during training. This helps assess how well the model performs in real-world scenarios [14].

2.1.2.1 Classification

To accurately classify test data into distinct categories, classification uses an algorithm. It identifies specific objects in the dataset and attempts to define or label them appropriately [11]. The classification technique used in this project utilises binary class which divides the data into binary values 0 and 1, which in human terms can be translated to pass and fail [15].

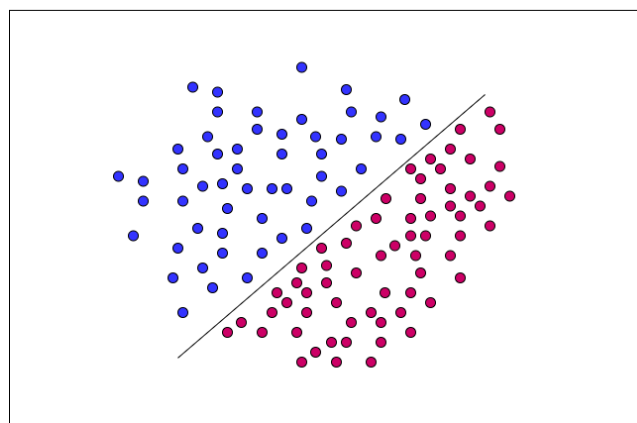


Figure 3. The process of classification involves categorizing input data into predefined labels, image inspired by [16].

2.1.2.2 Regression

To understand the interaction between dependent and independent variables, regression is used. The primary objective of regression is to find the best-fitting line or curve that describes the relationship between the variables. This line, often referred to as the regression line, minimizes the difference between the predicted values and the actual values in the dataset. It is used to anticipate things like whether a machine will break down. Regression analysis frequently makes use of logistic, polynomial, and linear regression algorithms [11] [15].

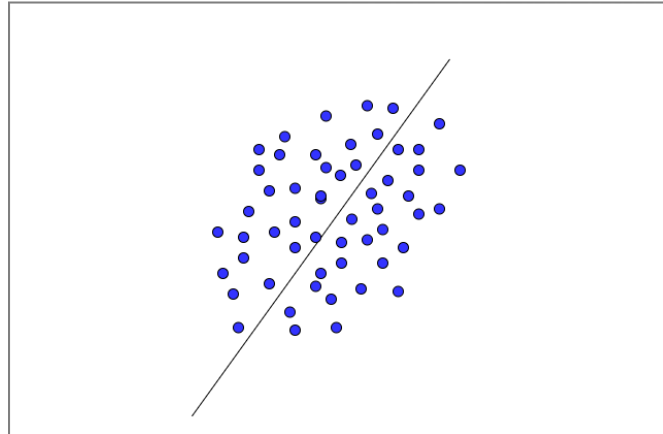


Figure 4. Regression predicts numerical outcomes based on patterns in training data, image inspired by [16].

2.1.3 Anaconda Distribution

Anaconda is a free, open-source platform for the Python programming language that has been designed to make working with data science easier to structure. It provides a comprehensive toolkit, including the ability to create and manage multiple environments. These environments, often referred to as integrated development environments allow you to work with Python in a controlled setting. Each environment can be customized with specific packages and tools required for different projects, ensuring that dependencies do not conflict [17].

2.1.4 Evaluation Metrics

In order to improve the performance of the created system, it is of great importance that the results are analysed with appropriate factors. The most common factors to evaluate include recall, precision, accuracy and something called F1 score [18].

Recall shows the model's capabilities in finding all true defects, reporting the percentage of defects that the model was able to detect. A high recall value indicates that the model is successful in finding defects, in other words, that it misses few defects [19].

$$\text{Recall: } \frac{TP}{TP+FN}$$

Precision measures how many of the cases the model has classified as defects, that are in fact true defects. The higher the precision score, the more of the predicted defects are actually true defects [19].

$$\text{Precision: } \frac{TP}{TP+FP}$$

Accuracy shows how much of all the predictions made by the model are true. It counts both TP (True Positive), where the model correctly predicted a defect, and TN (True Negative), where the model correctly predicted no defect. Accuracy is expressed as a percentage of the total number of predictions [19].

$$\text{Accuracy: } \frac{TP+TN}{TP+TN+FP+FN}$$

F1 score is a measurement that combines both precision and recall. It provides a balanced view of the model's performance by considering both how accurate the model is (Precision) and how well it finds the defects (Recall). A high F1 score signifies that the model is both accurate, and is able to predict true defects well [19].

$$\text{F1 score Precision: } 2 \times \frac{P \times R}{P+R}$$

2.1.5 Loss Function

A loss function is a concept in machine learning and deep learning that measures how well a model is performing. It calculates the difference between the model's expected output and the actual desired outcome. By calculating this difference, the loss function guides the model's learning process. A higher value of the loss function indicates a greater error in the model's predictions, signalling the need for adjustments to the model's parameters [20].

One of the simplest and most fundamental loss functions in classification problems is the 0–1 loss function. This loss function operates on a binary principle: if the model's predicted label matches the actual label, the loss is 0, indicating no error. Conversely, if the predicted label does not match the actual label, the loss is set to 1, representing a misclassification. The 0–1 loss function is straightforward and easy to understand, as it directly reflects whether a prediction is correct or incorrect [21].

2.1.6 Nvidia and CUDA

Nvidia is a company that develops graphics cards for, among other things, artificial intelligence and for making various complex calculations. It is particularly useful in object detection, as it can perform segmentation and classification. Nvidia is also compatible with various libraries and frameworks such as OpenCV and TensorFlow. It is also compatible with the CUDA (Compute Unified Device Architecture) library, which is a programming model and enables parallel processing on their graphics cards [22].

2.2 Deep Learning Architecture

Deep learning is helpful in solving complex issues as it makes use of advanced algorithms and neural network structures. Convolutional neural networks (CNN) and Faster Region-based Convolutional Neural Network (Faster R-CNN) are two of the many different types of architectures used in deep learning. These architectures are able to recognise complex patterns and predict results [23].

Among deep learning architectures, CNNs and Faster R-CNNs are particularly noteworthy. CNN are designed to process and analyse visual data by applying a series of convolutional layers that automatically learn and extract features such as edges, textures, and shapes from images. This hierarchical feature extraction process allows CNNs to perform exceptionally well in tasks like image classification and object recognition [23].

2.2.1 Convolutional Neural Network (CNN)

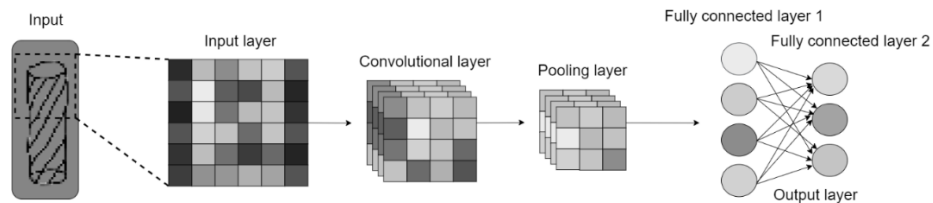


Figure 5. Convolutional Neural Network structure illustrating layers for image classification. Image inspired by [24].

Convolutional Neural Network (CNN) is a deep learning architecture that specialises in recognition and has been designed to detect predefined objects in images. The architecture makes use of algorithms which in turn makes it possible to identify patterns.

What CNN does is break down the desired features, which are specified by the user in an earlier step, into smaller elements and from there the algorithm analyses every part of the element separately. In the next step of the process, the CNN reduces the information using pooling layers [25]. Pooling is an operation used to reduce the quantity of data the network needs to learn. There are two different methods of pooling that can be used in the CNN process, Max Pooling and Average Pooling.

The two methods both use something called a feature map, which is a two-dimensional array of numbers that provides image properties [26]. But where the methods differ is how the elements are calculated and chosen. Max pooling chooses the maximum element whereas Average Pooling will the average of the elements. The properties that are most apparent from the previous map are therefore included in the feature map that is created by the max pooling layer. Average pooling calculates the average of the elements. As a result, average pooling provides the average of the features. Both of these methods aim to focus on important elements while reducing the amount of data [27].

2.2.2 Faster Region-based Convolutional Neural Network (Faster R-CNN)

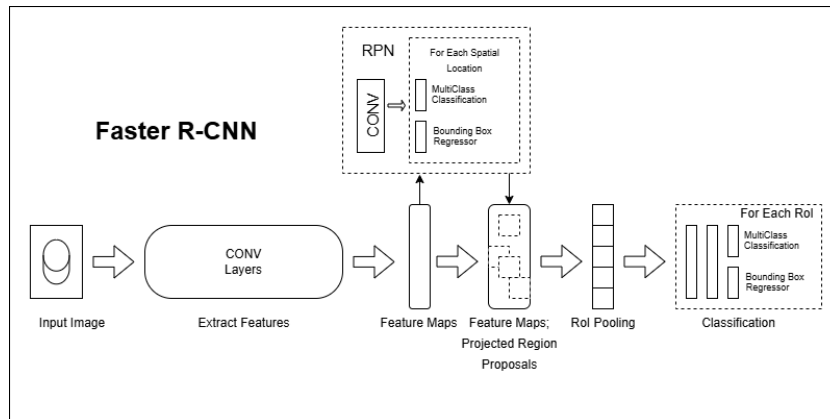


Figure 6. Faster R-CNN overview, image inspired by [28].

An enhanced deep learning model called Faster R-CNN (Faster Region-based Convolutional Neural Network) is used to identify objects in images. It improves upon previous methods by combining two main parts: a Region Proposal Network (RPN) and a Convolutional Neural Network (CNN). The RPN generates potential bounding boxes or regions, where objects might be located in an image, and the CNN analyses these regions to classify the objects [29].

These deep learning models consist of multiple processing layers, each capable of learning different levels of data abstraction. By stacking these layers, deep learning models can capture the subtle and complex structures within large datasets [30].

The rapid growth of deep learning is mainly due to three key factors [23]:

- the big boost in chip processing power, such as GPU
- the drop in the cost of computing hardware
- major improvements in machine learning algorithms.

These advances have allowed deep learning models like Faster R-CNN to reach impressive accuracy in difficult tasks such as object detection, image segmentation, and identifying objects across various conditions and environments.

2.2.3 Edge Impulse

Edge impulse is a platform for Tiny Machine Learning (TinyML) that simplifies the development process. TinyML is a technology that allows machine learning models to operate on low-power devices with limited resources. The platform allows developers to work with TinyML algorithms through a simple interface. Data can be collected from any compatible device, which allow users to gather relevant information from input sources. A machine learning algorithm can be created, or an existing one can be used, and a model can be trained. Once the model is ready, it can be exported as a C++ library, which can be easily used on a supported embedded device. The platform also provides useful information such as memory usage and processing time for the chosen development board [31].

2.3 Machine Learning Libraries

The development and implementation of machine learning models and algorithms depend on machine learning libraries. These libraries provide pre-built tools, allowing developers to focus on building effective solutions. Machine learning is a popular tool for tasks like complex data classification and making decisions. It involves developing algorithms that let systems learn from data. This technology is used in various areas such as image recognition. To use machine learning models, libraries are necessary. It offers a collection of pre-build functions and tools that simplify complex tasks, making it simple for developers to build, train, and improve models [32].

2.3.1 OpenCV

OpenCV is a computer vision library. It is developed by intel and is used in image recognition and other technologies. The essence of computer vision involves the processing and analysis of visual data, such as images and videos to extract meaningful information and make informed decisions This can include tasks like identifying objects within an image, tracking movement in video sequences, and even understanding complex scenes. Computer vision systems can perform functions such as facial recognition, automated surveillance, and real-time object classification [33].

The library supports CPU (central processing unit) and GPU (graphics processing unit) acceleration to further improve performance when dealing with large datasets, which is crucial for applications requiring real-time processing. The difference between using the CPU and the GPU is significant when handling large amounts of image data. While the CPU can perform tasks with high precision, it may struggle with speed when processing big datasets in real-time applications, such as video analysis or object detection. In contrast, the GPU, designed to handle parallel processing, can process multiple image tasks simultaneously, making it much faster for these types of operations [34].

2.3.2 TensorFlow

Tensorflow is an open-source software library that has a particular focus on training. Tensorflow is compatible with a large number of programming languages such as Python, JavaScript, C++, and Java, which makes it useful for a wide range of application. TensorFlow's ability to use the GPU to speed up processes is one of its most important features. This is useful for training deep neural networks on big datasets [35].

2.3.2.1 *Tensorboard*

A visualisation tool called Tensorboard helps the understanding of Tensorflow algorithms. Tensorboard allows the visualisation of a model computation graph, training metrics, and parameter values for users. Training metrics like accuracy and loss may be shown over time, giving information about how well the model is learning from the data. Additionally, by comparing various training runs, the graphs facilitate the process of fine-tuning hyperparameters and making improvements to enhance model performance [36].

2.3.3 Annotation Tool

In order to produce high-quality datasets for machine learning, annotation tools are needed. Researchers and developers can train their algorithms more effectively and accurately by labeling photos [37]. Two tools were used in the creation of this project.

2.3.3.1 *LabelMe*

LabelMe is used in machine learning to produce labelled datasets. Compared to using just rectangular bounding boxes, LabelMe offers a more precise and easy way for users to manually annotate images by drawing polygons around objects. This method is especially helpful for tasks requiring a high level of precision, like object segmentation and image classification. These polygons can also be given specific names by users, resulting in a structured dataset that can be used to train machine learning models [37].

2.3.3.2 *LabelImg*

LabelMe is used to create labelled datasets in machine learning. By creating bounding boxes around the object and labeling it with tags, users can manually annotate images. Machine learning models are then trained using this annotated data for tasks such as object detection, segmentation, and classification. Machine learning models are then trained using the data collected by LabelMe for tasks such object detection, segmentation, and classification. [38].

3 Related Work

This section serves to provide an overview into relevant work that has been performed in relation to the proposed subject of this paper. Considering the fast progression of technological advancements, special notice has been taken to find papers that has been published within the last 10 years

3.1 A Real-time Object Detection Algorithm For Video

L. Shengyu *et al.* [39] published a paper in 2019 on the studies of real-time object detection algorithms for video processing. The authors discusses the evolution and currents state of object detection in computer vision. With the advent of deep learning, object detection has significantly advanced and different deep convolutional neural networks (CNNs) has been developed. These CNN have different abilities that are better utilised than other architectures for different purposes depending on the usage area. The mentioned CNN architectures struggle with slow detection speeds and high computational demands.

The paper proposes a real-time object detection method called Fast YOLO, which builds on the YOLO algorithm by incorporation efficient convolution operations inspired by GoogLeNet. With Fast YOLO the authors aum to enhance detection speed while maintaining accuracy.

The authors continue with providing an overview of the advancements in object detection by specifying the different architectures that is used in the comparison. The paper also outlines a comprehensive approach to improving object detection with the proposed architecture, Fast YOLO.

The experiments were a success, and the authors provided with data specifying the precision and recall rates for all CNNs, including the improved Fast YOLO. The last-mentioned architecture outperformed the other architectures in both categories.

Lastly, the authors proposed possible applications and future work within IoT Cloud services, where the idea is to store massive datasets on the cloud for training.

3.2 Fruit Classification Using Optimized CNN

In 2023, A. Kushwaha *et al.* [40] published a paper on the subject of fruit classification using optimized CNN. The authors claim that fruit classification is vital in many different industries and for people with dietary restrictions. Fruit classification powered by machine learning has previously shown a lot of success and the authors aim to continue upon this research by using a deep learning approach.

The authors perform their studies by using a preexisting dataset called “The Fruit Recognition” for both training and testing of the model, and the deep neural network created to perform this task utilises Keras and TensorFlow.

Firstly, they describe their proposed work in detail, mentioning data processing and dataset augmentation to name a few. They also go into detail on the tools and frameworks used in their work process.

They report successful results, boasting an accuracy of 96.88% after 40 iterations. Their CNN used five layers, followed by pooling layers and fully linked neural network layers.

This paper is of relevance to our thesis considering the authors usage of TensorFlow and Keras in their work, in addition to the usage of image recognition techniques and CNN algorithms.

3.3 ARES: An Automated Rotten Egg Sorter Utilizing the Egg's Physical Properties and Artificial Neural Network

Dela Cruz et al. [41], published a paper in 2023 discussing how to automate classification of rotten and non-rotten eggs using the egg's physical properties as the variable and utilising artificial neural network to perform the task. The authors describes that the egg market currently uses spectroscopy in industrial setting to sort the eggs. Very little research has been done in regards of other ways to identify rotten eggs, but one strategy is based on the weight of the egg being the physical variable for whether the egg is spoiled or not. The authors proposes a system containing two subsystems that works parallel to each other where one is focused on classifying and the other on sorting the eggs based on its quality.

To perform the task, the authors uses both hardware components for the sorting process. They bought 40 eggs a month before they would perform the data collection and was then soaked in water to make sure that the eggs were rotten, and an additional 40 eggs were purchased in time for the data collection to ensure the egg's freshness. The system was programmed in Python, with TensorFlow and Keras being used to develop their ANN (Artificial Neural Network).

Both the hardware mechanism and the software, meaning the classification system, showed successful results. The classification system was tested using three different deep learning algorithms, but the one who showed the best result was the ANN where at a split ratio of 80:20, the system boasts an accuracy of 93.75%. Considering that this research contained such a small dataset, they were surprised that this deep learning algorithm yielded the best result, but are attributing the results to the type of kernel used in the project.

They finish the study by recommending possible ways to improve upon their studies to achieve full optimisation.

This study utilises similar "image recognition concepts" to our study where they have a classification system that sorts in terms of approved and not approved, as well as utilising Python, TensorFlow and Keras in the development of their system.

3.4 Embedded System Implementation for Material Recognition Using Deep Learning

In 2017, Khaled S. Younis et al [42], published a paper detailing a way to implement embedded system in material recognition while using deep learning. The authors claim that this subject is important in many fields, but especially in the industrial sector. They propose that automating the process of separating and packaging different materials can save money, time and improve efficiency.

The authors has sourced two datasets, where the first one is called Flickr Material Dataset (FMD) and consists of ten common material categories. The second dataset is called University of Jordan Inspection System (UJIS) and consists of the same ten material categories as FMD. For the application they utilise Keras and TensorFlow.

Their system showed some successful results where the FMD trained system displayed an accuracy of 79.25%. They also tested the system trained on the local database (UJIS) where they also mounted a camera on a Raspberry Pi 3 embedded system, with an accuracy of 90.5%. They stress that the created system is self-contained and is portable with minor adjustments to the new environment.

Noticing as this paper utilises the same concepts as in our project, this study is closely related to what we want to achieve. We take special note to the portability aspect of this study as it's also one of our concerns.

4 Methodology

To reach the goal of creating a suitable system for Metaqlix’s customer, Systems Development Research Methodology (SDRM) (Nunamaker’s design and creation approach) is used as the chosen methodology for system development, encompassing an iterative cycle of problem identification, goal definition, design and development, results demonstration, and results evaluation [43].

Nunamaker is involved in both designing and programming the system. This approach consists of five main steps within the iterative cycle of the system development research process. This approach is explained in subsections. Iterative development is used to build the system, with the initial step focusing on collecting the information and data required for solving the issue [43].

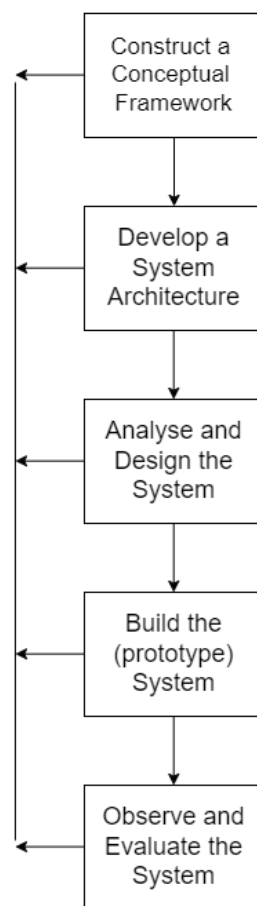


Figure 7. A diagram illustrating the research process for system development [39].

4.1 Construct a Conceptual Framework

The system requirements identified in Chapter 1.5 are analysed and categorized into functional and non-functional requirements. This clarifies the goals and specifications that the system must fulfil. Following that, different design options and strategies to satisfy these requirements are explored.

4.2 Develop a System Architecture

The two main parts of the system are training and classification. These parts, as well as its operation and available functionalities, are shown in Figure 8 below. Additionally, the data flow between each process or system components is represented by lines connecting the boxes. This indicates that data is changing from one state to another. The boxes with dashed lines indicate which step of the training or classification phase the data is in. The top dashed box includes “Label data, Model training, Validation, and Evaluation,” and represents the training phase. “Capture Images, Implementation of Model, and Detection” are included in the bottom dashed box.

- *Image Database*

Images of packages are stored for use in model training and defect detection. The system accesses the contained data.

- *Data Preprocessing*

The images that are used must be prepared to train the system. The images are optimized by removing the background and focusing on the important details in the image such as the appearance of the package.

- *Label Data*

The initial step in the training process involves data about each image where it is associated with a label indicating if the package is defective or not. These labels are used to train the model to detect defects.

- *Model Training*

Training the model using the labelled images to learn to detect patterns and features of both defective and non-defective packages.

- *Validation*

The process is to evaluate the performance of the model during the training phase. This means that part of the data not used for training, is used to assess the model’s ability to evaluate data.

- *Evaluation*

This step ensures that the model performs well on new data. About 20% of the dataset was chosen as test data in order to evaluate the ability of the model to make predictions to new data. The system was also tested in a live environment by connecting a camera and streaming video in real time, in addition to using static pictures.

- *Capture Images*

The system enters the classification phase, where new pictures of packages are taken.

- *Implementation of Model*

Implementation of the trained model for detection of defects on new images of packages.

- *Detection*

Using the implemented model to identify defects in packages based on the new images, the result can be either defective or non-defective. The algorithm also specifies the location of the defect.

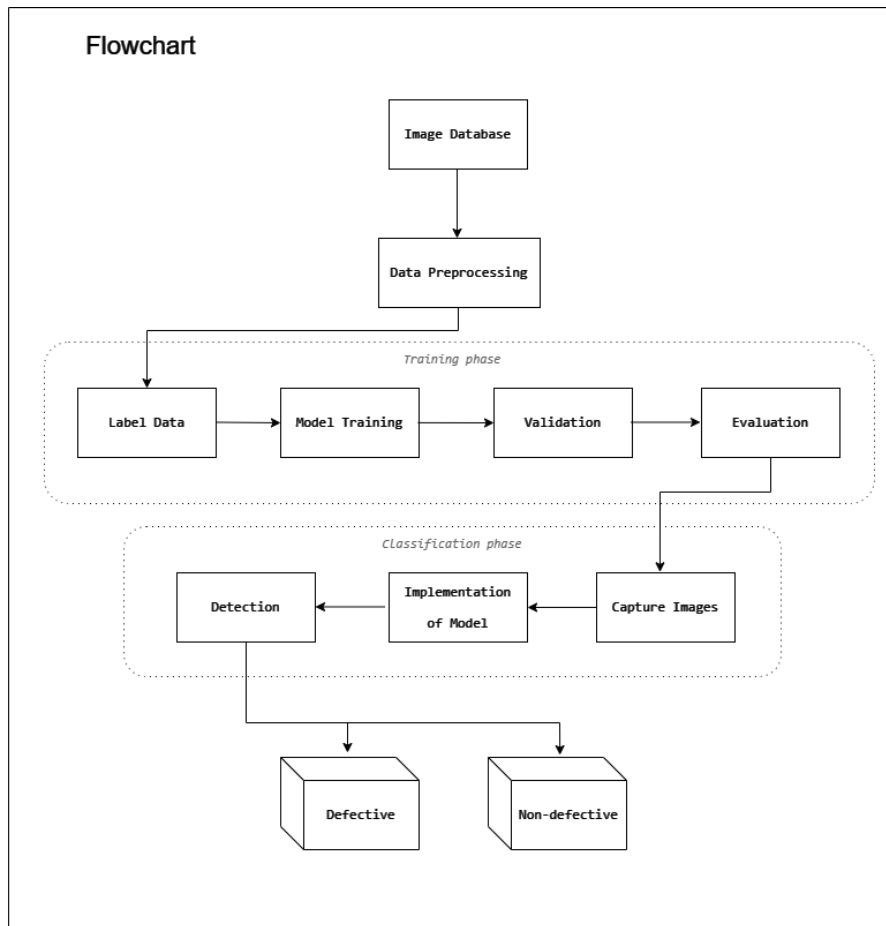


Figure 8. Visualization of project workflow.

- *GUI to Camera*

The Graphical User Interface (GUI) initiates

- *Camera To OpenCV*

Once the Camera receives the request, it begins capturing and streaming live video.

- *OpenCV to Model*

The OpenCV module receives the live video frames from the camera and performs preprocessing tasks.

- *Model to GUI*

The Model analyses the processed frames to detect specific features or objects. After completing the analysis, the Model sends the detection results back to GUI displaying these results to the user.

- *GUI to Log*

The GUI logs relevant metrics into a log system.

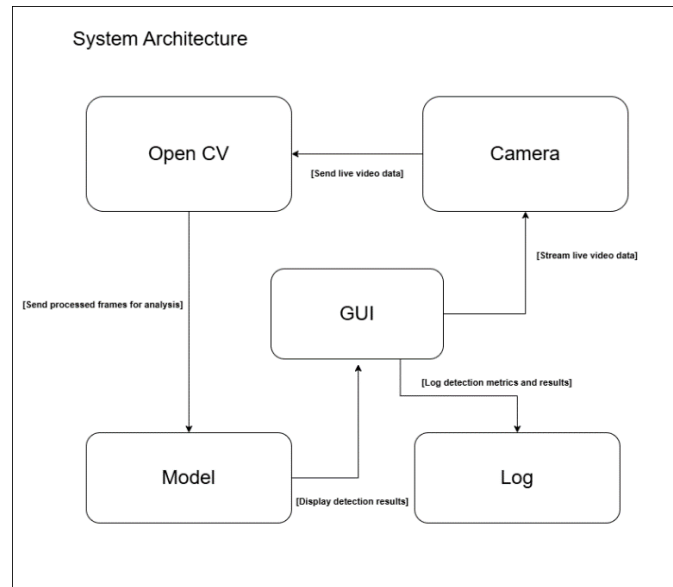


Figure 9. Visualization of system architecture.

This system architecture is designed to handle live video streaming, real-time processing, and detection, while also managing the display and logging of results. It includes several components to ensure a seamless flow of video data from capture to analysis. The key components of this system include a Graphical User Interface (GUI), a camera, OpenCV for image processing, a model for analysis, and a logging system.

4.3 Analyse and Design the System

This section describes how the system was analysed and designed. It covers the selection of an appropriate model for object detection, the requirements used to evaluate different models, and why the final choice was made. In addition, it discusses which versions and tools were chosen to ensure that they worked well together. The goal of the analysis was to find a good balance between accuracy and speed and system requirements in order to effectively detect defect on the objects.

4.3.1 Model Comparison

The method started with an intensive search for different suitable deep learning models for object detection. Convolutional Neural Networks (CNN), Faster R-CNN, and YOLO were among the models that were the focus of the search. Accuracy, the precision that the model could identify defects, and the technical efficiency required for real-time processing were the main parameters. The process included both theoretical analysis and data collection. Reviewing different image processing methods, model architectures, and hyperparameter tuning techniques were all part of the theoretical research [44].

This helped identify which approach would most effectively meet the projects requirements by putting focus on each model's positive and negative aspects.

Factor	CNN	R-CNN	YOLO
Precision and recall	Good for simple object recognition.	High precision, but slow processing.	Lower precision for small objects.
Processing speed	Slow because of many layers.	Very slow (analyse each region separately).	Very fast (processong the entire image at once).
Model Complexity and training time	Easy to train.	Very complex, takes a long time to train.	Fast.
Use case	Image classification and basic object detection.	Useful for high precision.	Suitable for real-time detection in video streams.
Architecture features	Uses layers to classify the whole image.	Region-based (first finns regions, the classifications).	Analyse the entire image in one step.

Table 1. Overview of the advantages and disadvantages of different machine learning models for object detection.

Table 1 shows strengths and weaknesses in regard to the sought after requirements. Boxes with a red colour indicates unsuitable characteristics, yellow indicates acceptable characteristics and green indicates sought after characteristics.

The main reason Faster R-CNN was chosen for this project was because of its high object detection accuracy, which is required for detecting small defects in small objects. Accuracy was given a higher priority than speed because the project's focus was defect detection. The performance of the system is directly impacted by the precision of defect identification, even though live detection capabilities are crucial.

High Precision: Faster R-CNN is especially suited to our requirements because it performs very well in accurately classifying and recognising small objects. This precision is needed because the defects that aims to be detected are frequently subtle and could be undetected by less accurate models.

Region Proposal Network (RPN): The architecture involves a Region Proposal Network (RPN) that efficiently produces object proposals, allowing the model to focus on the areas of interest. This feature improves the performance of detection, especially for small objects where details are important.

Balance between speed and accuracy: Faster R-CNN provides an acceptable compromission between detection speed and accuracy, despite the fact that it might not be as quick as certain single-stage models. Improving accuracy is crucial because it can help reduce the number of undetected defects.

4.3.2 Versions and Compatibilities

TensorFlow 2.10.0 and Python 2.10.0 were used in the implementation, with TensorFlow providing the frameworks and tools required for model construction, training, and assessment. Because of their reliability and compatibility with the project's chosen libraries and algorithms, these specific versions were chosen. Incompatibilities with other project-related libraries, such as OpenCV for image recognition were also avoided. Python 2.10.0 was chosen as the programming language due to its large library support, and its integration with Tensorflow 2.10.0.

4.4 Build the Prototype System

Building a prototype is an important part of the development phase. The chosen deep learning model serves as a working prototype that fulfils the goal of this study. While following the chosen methodology, it allows for opportunities in:

- *Early Testing*

The prototype allows the testing of the system in real-life conditions or simulation to assess its performance of intended tasks. For example, it can be used to verify if the model accurately detects and classifies defects in images as expected.

- *Performance Evaluation*

To assess how well the model is working, important metrics including accuracy, precision, recall, and F1-score can be tested. This helps in identifying strengths and weaknesses early on.

- *Iterative Improvement*

Based on the results from early testing.

The system, programmed in Python, utilises libraries and frameworks such as OpenCV, Keras and TensorFlow [45].

4.4.1 Use of Model

The system was created by using the Faster R-CNN model architecture, but the model used was not pre-trained on the required dataset. Instead, a customised dataset was used to train the model. The faster R-CNN architecture was chosen for its proven effectiveness in object detection tasks [46], and the training process was carefully followed to ensure the model was properly optimized for the specific requirements of this project [47].

4.4.2 Number of Images and Image Collection

A dataset containing 300 images was collected for the development of the prototype. Subsequently, an additional 500 images were collected in the company's real working environment. The purpose of this large-scale image collection was to make sure the model could identify defects in different kinds of situations and environments.

4.4.3 Image Preprocessing

A python script was programmed to resize the images and make sure they met the required specifications. The python code used in this project is designed to preprocess images before the training process. It sets up directories for input and output, and resizes images to specific dimensions of 450 pixels in width and 800 pixels in height.

4.5 Observe and Evaluate the System

During this phase, the system is observed and evaluated to assess its performance in defect detection on objects. The system passes through several tests to measure processing speed and effectiveness. The system was tested with 60 images first, 20% of the first 300 images, that was not included in the testing phase of the system. The system was later tested with an additional 100 images, 20% of 500. Live detection was also tested. The system's performance is evaluated in order to identify whether there is opportunity for improvement. Following the comparison and analysis of the results, the results of the tests determine whether they align with the established goals of the system. If it is discovered that there are areas where the system fails to meet the established requirements, these weak points can be identified and improved. After the implementations of the improvements, it is important to test and ensure that the new changes effectively solved the identified problems [48].

5 Results and Analysis

Results from the various research methodological areas are given in the sections that follow. The specifics of our data collection procedure, the features of the datasets we chose, along with the methods used to obtain the entire datasets, the training parameters and training results, and evaluation metrics that were utilised to select the problem's top-performing weights are covered in each subsection.

5.1 Construct a Conceptual Framework

In order to simplify the process of performing this study, the conceptual framework has been constructed based on Nunamaker's design and research approach specified in chapter 4.

5.1.1 Data Collection

Within the deep learning sphere, training data is one of the most important things to gather for the work to be viable. The quality of such data is also important to keep into consideration, since it may greatly impact the outcome of the trained model. Our product is something that is currently sold online among other channels, and thus has no readily available collection of data in regard to defects. Therefore, we had to perform our own data collection, which was performed in two rounds.

The first round was performed in a home environment, where staging had been prepared. The staging consisted of a green background, chosen because the colour did not in any way resemble the colour of the product. Proper lighting conditions was also thoroughly contemplated to make the defects on the products show up properly on the pictures. That is why two table lamps were directed in a favourable direction. A metallic stand was also used to make sure that all pictures were taken with the same parameters. The default camera of a Samsung Galaxy S21 was utilised through all data collection phases.

When taking the pictures, 6 products with manufactured defects and 3 products without any defects were used. For each product, 30-40 pictures were taken, rotating the product degrees after each picture, to ensure the entire surface of the product was covered. The defects on the product were randomly spread out on the surface and multiple defects existed on the same product. Everything was taken into consideration for the dataset creation phase. A total of 300 pictures were taken during the first data collection phase.

At a later stage in the project execution, an additional 500 pictures were taken in the production environment. A setup similar to the first data collection phase was set up but utilising the unique environmental focal points of the company, such as the lightning and the conveyor belt. The same strategy of turning the product a few millimetres was used and this time, between 50-60 pictures per product was taken.

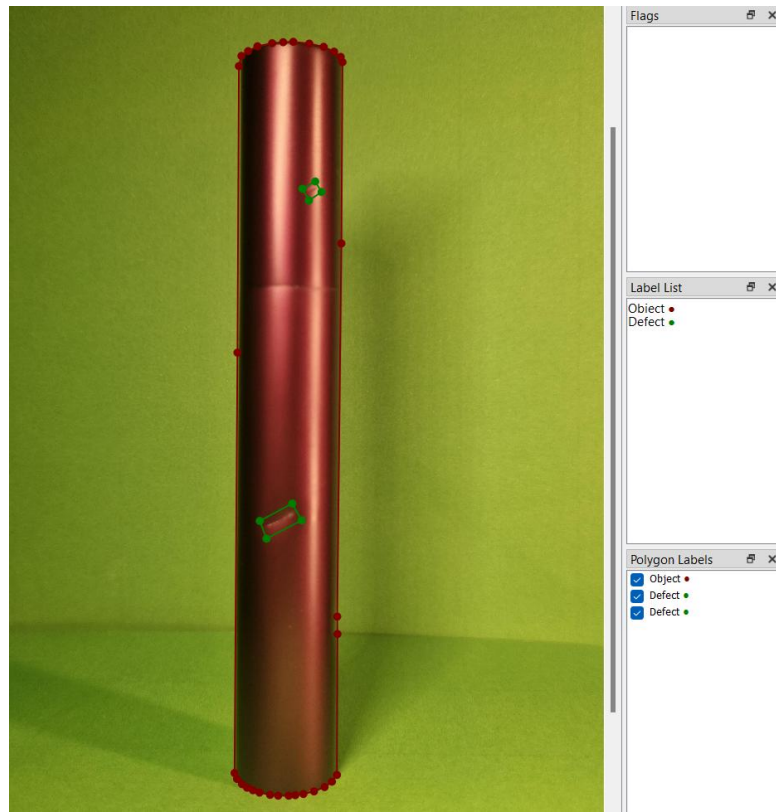


Figure 10. GUI of the LabelMe tool with our product and corresponding labels processed.

In order to train our Faster R-CNN model in an effective way, we had to annotate all the taken pictures with labels, as shown in Figure 10, which displays the GUI of the LabelMe tool. These annotations contain coordinates of the marked areas and are divided into specific chosen categories. With the first annotation tool, LabelMe, we used two different categories of labels. The object, meaning the whole product, and the defect. If multiple defects were visible in the picture, all defect areas were marked for annotation. LabelMe creates annotations based on polygonal points and was saved in a JSON file format in the same folder as the original pictures. They were then manually divided into a training folder and a test folder, where 80% of the pictures were used for training and 20% for testing. The JSON files were first converted into XML format since the created python program did not fully understand the coordinate system in the JSON format, and then fed into the R-CNN layers for training and testing.

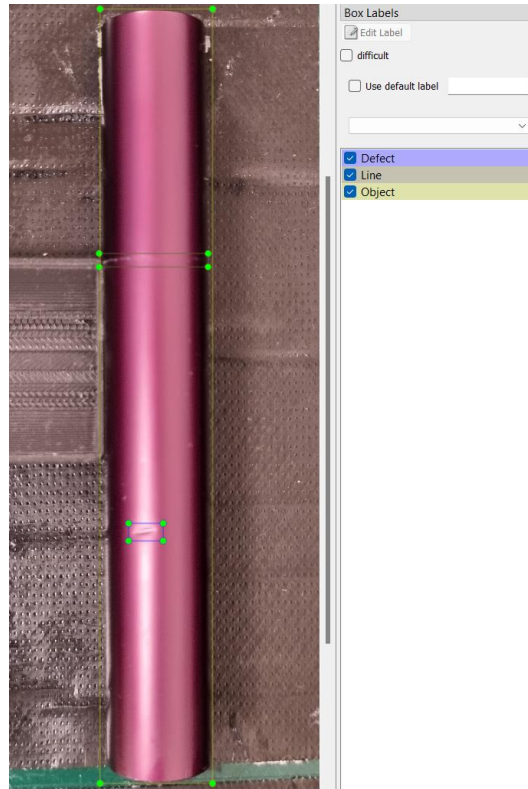


Figure 11. GUI of the Labellmg tool with our product and corresponding labels processed.

Later, another annotation tool was used called Labellmg, as shown in Figure 11. This tool immediately saves the files as XML instead of having to go through the process of processing the annotations unnecessarily, as well as utilising bounding boxes instead of polygonal points. During this phase another label category was added since it had been noticed during testing in the previous phase, that the line dividing the body and head of the product was mistakenly considered a defect. Therefore, a total of three label categories is now used, namely: object, defect and line.

5.2 Develop a System Architecture

In chapter 4 of this thesis a system architecture was developed to showcase the different components of the system, see Figure 9. This to facilitate the implementation of the conceptual framework.

5.2.1 Datasets

During the labeling process, each label is described, and the relevant labels are applied, such as potential defects on the object, the object itself, and the line between the lid and the lower part of the object. The labeling information was preserved by storing the annotated data in JSON files. These JSON files were then converted to XML files so that the labelled data could be used in machine learning models and processed further. The XML files contain the coordinates of the bounding boxes marked on the images, along with information about the labelled objects or defects. Bounding boxes are rectangular borders drawn around the objects or defects.

Training Process (rounds)	Total Images	Images containing defects	Images not containing defects
First	300	172	128
Second	500	345	155

Table 2. An overview of the training process that displays how many images, both with and without defects, are used in each training session.

The training process across two rounds is shown in Table 2, which also shows the total number of pictures and the number of images with defects compared to pictures without. The model's ability to detect defects is enhanced by the second round's larger dataset compared to the first round's, which contains additional images with defects.

5.3 Analyse and Design the System

5.3.1 Training

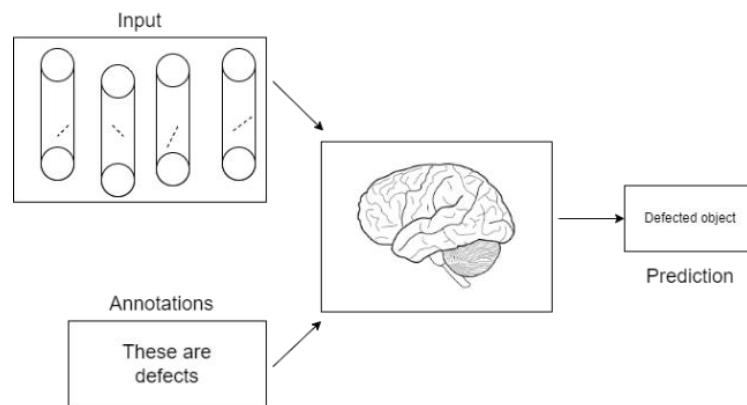


Figure 12. Supervised Learning Model. Image inspired by [15].

During the training phase, as shown in Figure 12, a supervised learning approach was applied. The collected images makes up for our datasets, which has been annotated to mark existing defects, and will be used to train the machine learning model. The annotations are crucial for the success of the training.

During training, the model took the annotated images as input, where it processed the bounding boxes and labels to learn how to differentiate between defective and non-defective objects. The model designed to handle object detection tasks, gradually improved its understanding of what constitutes a defect through multiple iterations of training on the annotated data.

Once the initial phase of training was completed, a second round of training was conducted using images collected in the production environment of the collaborative company. This second training phase was essential for fine-tuning the model and ensuring its predictions would adapt well to real-world conditions. By the end of the training, the model was able to identify and predict defects on new images with high accuracy, successfully separate between normal and defective objects.

5.3.1.1 Evaluation of Edge Impulse Model Performance

The performance metrics from Edge Impulse did not meet the projects requirements, indicating that this approach may not be ideal for the needs of the system. The specific results from the Edge Impulse model are summarized in Table 3.

Metrics			
Dataset	Recall	Precision	F1-score
D1	1.000	0.667	0.800

Table 3. Edge Impulse metrics of performance for dataset D1, including F1-score, recall, and precision.

These metrics are crucial for evaluating the model's effectiveness in identifying defects. Recall measures the model's ability to correctly identify all relevant instances, while precision indicates the accuracy of the model's positive predictions.

Upon further analysis, it became evident that the model was generating bad results. While the model demonstrated high accuracy, it frequently identified defects that were not present in the images, as can be seen in Figure 13.

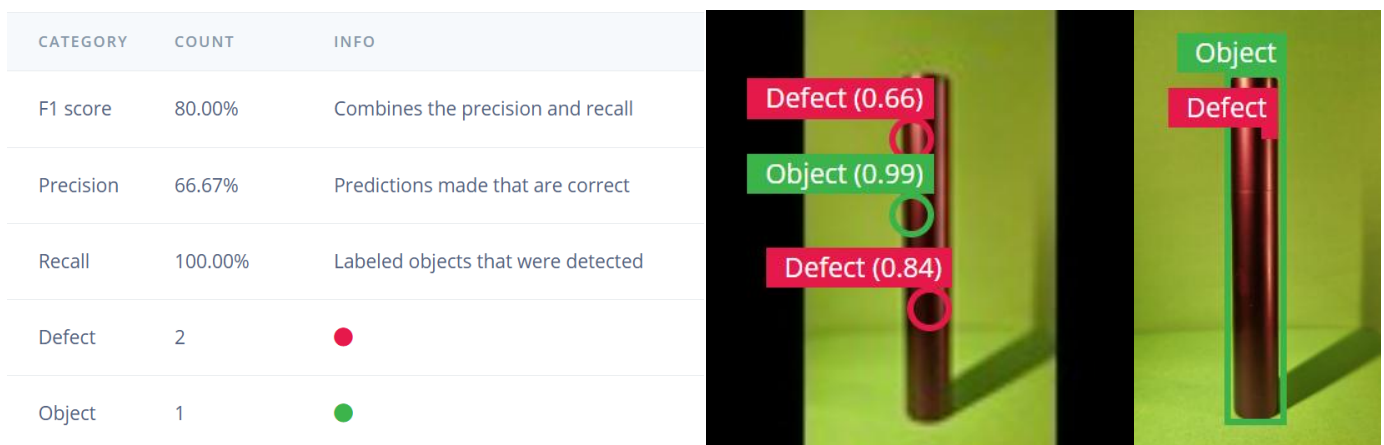


Figure 13. Results from the Edge Impulse model after training.

As a result, it was determined that a different deep learning model and libraries needed to be reviewed to identify a more suitable option for the described project. This involved assessing various models' performance and their ability to meet the project's specific requirements.

5.3.1.2 First Phase Training – 300 Images

In order to effectively train the model, careful preparation of both training data and training configurations was required. We started by navigating to the folder where the model and its configuration files were stored.

pipeline.config: This file contains the algorithm with all settings and hyperparameters that control the training process. Important parameters such as number of epochs (1), number of steps per epoch (10 000), batch size (1) were specified here.

The next steps involve working with the *train.record* and *test.record* files, which contain the training and test data in *TfRecord* format. The *train.record* file includes the data on which the model is trained, while the *test.record* file contains the data used to evaluate the model's performance during and after training. Additionally, the *label_map.pbtxt* file contains a translation table between the labels used in the dataset, as shown in Figure 14.

```
item {
  id: 1
  name: 'Object'
}

item {
  id: 2
  name: 'Defect'
}

item {
  id: 3
  name: 'Line'
}
```

Figure 14. This figure shows the contents of the *label_map.pbtxt* file, which is used in the machine learning model

To initiate the training process, we first navigated to the folder containing the model files in Anaconda. Then, the following command was executed:

```
python model_main_tf2.py --model_dir=models/faster_rcnn_resnet152_v1 --
pipeline_config_path=models/faster_rcnn_resnet152_v1/pipeline.config
```

In this command, *faster_rcnn_resnet152_v1* refers to the specific model architecture used for training. The model is based on Faster R-CNN (Region-based Convolutional Neural Network) with ResNet-152 as the backbone. The training was conducted using an NVIDIA GeForce RTX 3060 Ti GPU, which significantly enhanced the training speed and efficiency.

The following graphs display various aspects of the model's performance, focusing on the validation loss as the model progresses through training. In each graph, generated from TensorBoard, provides valuable insights into how well the model is learning to classify objects during training.

On the x-axis, the graph shows the number of steps, which represents the training iterations over the dataset. The y-axis displays the validation loss values. These values reflect how well the model's predictions match the ground truth labels, where lower numbers indicate better performance. The orange line in the graph represents the actual loss progression during training.

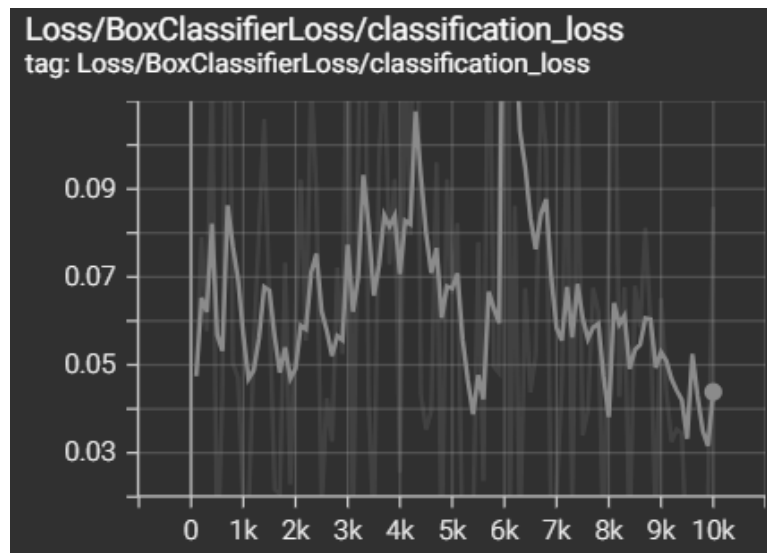


Figure 15. Graph showing the Classification Loss during first model training, illustrating how the model's ability to correctly classify objects improves over time.

Figure 15: The graph displays the validation loss, specifically assessing the model's performance in object classification. This loss measures the performance of the model in classifying objects within detection tasks. It reflects how accurately the model's predictions match the ground truth labels for object classification on unseen validation images. You might observe some instability in the loss values, indicating that the model is still learning and has not yet generalized well to the validation data. This is a common occurrence in the early stages of training.

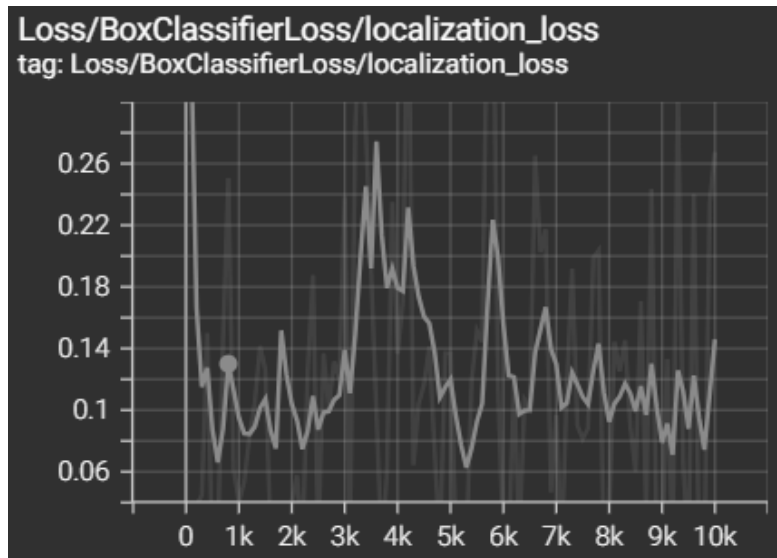


Figure 16. Graph displaying the Localization Loss during first model training, demonstrating the model's performance in accurately predicting the locations of objects over time.

Figure 16: The graph displays the validation loss for the model's object localization performance. This loss metric assesses how accurately the model predicts the bounding boxes or locations of objects within images. The orange line represents the localization loss, which is expected to decrease as the model becomes better at accurately identifying where the object and defects are located in the images.

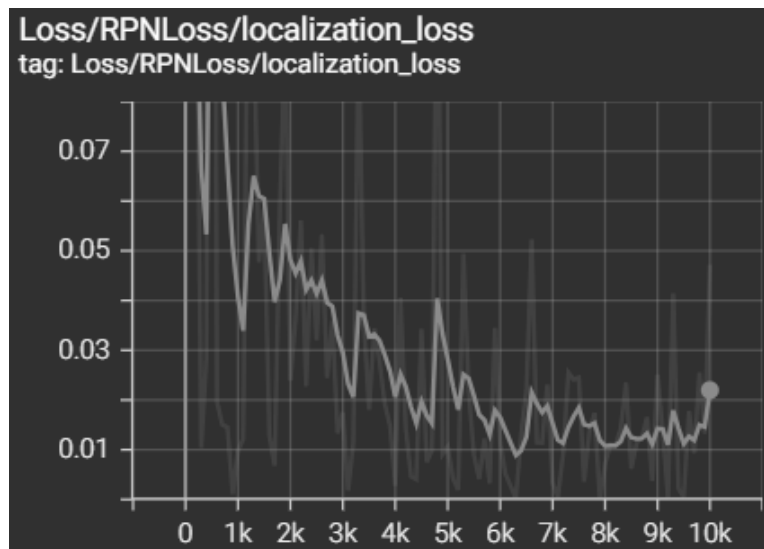


Figure 17. Graph illustrating the RPN Localization Loss during first training, highlighting the model's ability to accurately propose regions for object detection over time.

Figure 17: The graph illustrates the validation loss over the steps for the region proposal network (RPN), specifically focusing on the accuracy of object localization. At the beginning of training. The loss shows clear ups and downs at first, indicating that the model is having trouble accurately predicting bounding

boxes. As training continues, the loss gradually decreases, showing that the model is getting better at consistently generating precise bounding boxes for objects in the images.

5.3.1.3 Second Phase Training – 500 Images

During the second phase of training, we used 10 epochs, 10 000 steps and 1 batch size. In this phase of training, compatibility issues arose between the CUDA version and the TensorFlow version being used. The GPU could not be utilized effectively, and this led to a decision to uninstall CUDA entirely to avoid the problem. The training process was shifted to the CPU, using an Intel(R) Core(TM) i7-10700 processor. While this change significantly slowed down the training process, it ultimately succeeded, allowing the model to produce results.

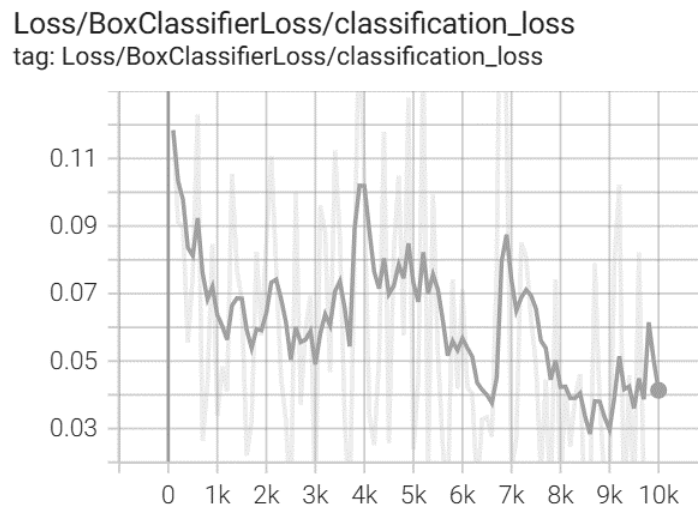


Figure 18. Graph showing the Classification Loss during the first model training, illustrating how the model's ability to correctly classify objects improves over time.

Figure 18 shows the differences in classification loss throughout the training process. The classification loss slowly reduces from its beginning high level, indicating that the model is improving in identifying the classes. We see the best results around 9,000 steps but if we take a look at the total Loss graph, Figure 21, we see that it doesn't affect the results too much.

Loss/BoxClassifierLoss/localization_loss
tag: Loss/BoxClassifierLoss/localization_loss

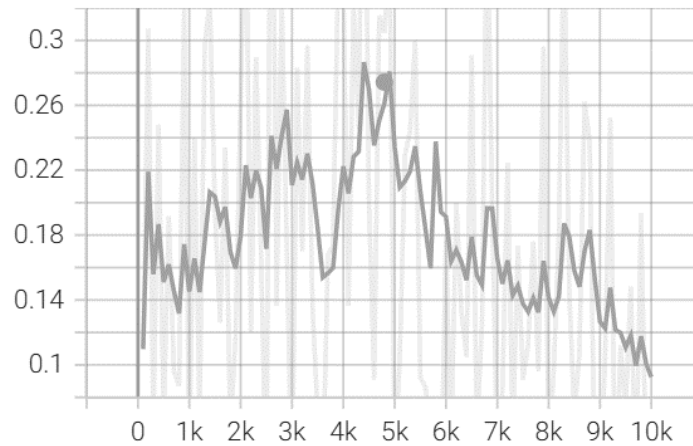


Figure 19. Graph displaying the Localization Loss during the second model training, demonstrating the model's performance in accurately predicting the locations of objects over time.

We see an improvement in localisation loss, a metric used to assess how well a model predicts the locations and bounding boxes of objects within images, as illustrated in Figure 19. The localisation loss steadily reduces throughout training, indicating the model's increased object location accuracy. The rate of loss reduction falls as the number of steps rises towards 10,000.

Loss/RPNLoss/localization_loss
tag: Loss/RPNLoss/localization_loss

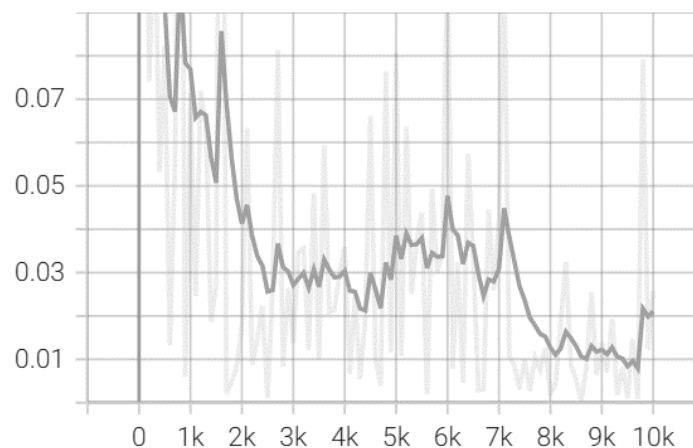


Figure 20. Graph illustrating the RPN Localization Loss during the second training, highlighting the model's ability to accurately propose regions for object detection over time.

As shown in Figure 20, the loss decreases less quickly after 7,000 steps, and the loss clearly increases at 10,000 steps. When a model's success on training data no longer moves into improved accuracy on unknown data, it seems to be overtrained.

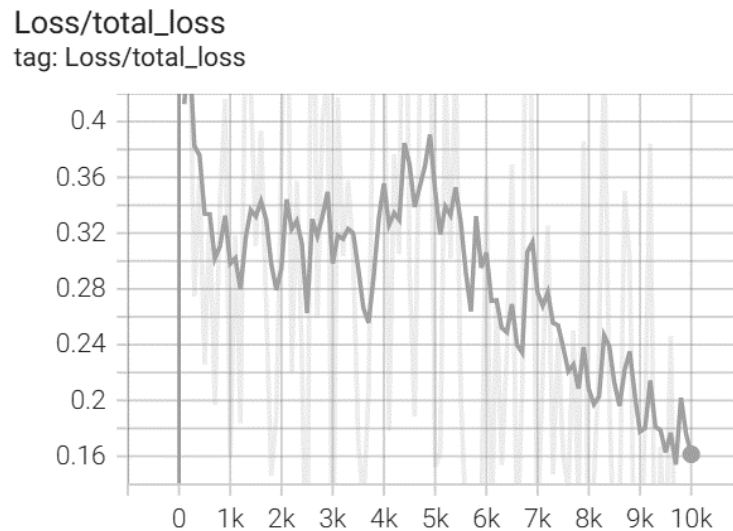


Figure 21. Graph showing the overall training loss.

Classification, localisation, and RPN localisation are the three forms of loss that are combined in the total loss graph, Figure 21, to give an overall assessment of the model's performance throughout training.

The graphs in figures: 18, 19, 20, and 21 provide insight into how the model performed in the second phase of training concerning various aspects like object classification, object localization, region proposal network and total loss. The line in all graphs concerning loss goes down throughout the 10,000 steps. That means the model is getting better in detect objects and defects with minimal mistakes.

Although there may be ups and downs at the beginning, in general, the trend looks very great; the model learns very well, and is getting more accurate gradually in the classification of the objects and their places.

5.3.2 Data Analysis

In this section, the aim is to analyse the dataset to identify patterns that separate between good packages and defective packages in a reduced-dimensional space based on their features. The 'Projector' tool in TensorBoard is designed to help visualize high-dimensional data by reducing its dimensions. By displaying patterns that are important in classification tasks such as identifying defects and separating between good and defected packages, this tool aids in a better understanding of the structure of the data [49].

5.3.2.1 First Phase Analysis – 300 Images

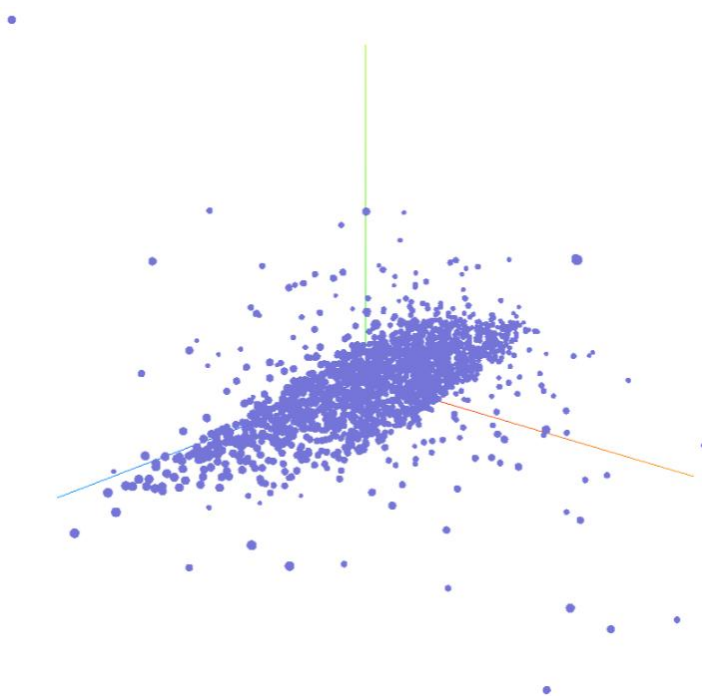


Figure 22. Graph representing data vectors projected into a lower-dimensional space using TensorBoard's 'Projector' tool.

The graph in Figure 22 shows data vectors projected into a lower-dimensional environment using the 'Projector' tool from TensorBoard. Data vectors refer to numerical representations of data points that capture various features or attributes of the objects being analysed. Lower dimensional space refers to reducing data from many dimensions (features) to fewer dimensions (e.g., 2D or 3D) to make patterns easier to visualise. This visualisation provides information that would be difficult to find in the original high-dimensional form and is a useful tool for understanding patterns and relationships within the dataset. TensorBoard was used to create the graph as part of the model training process.

Points in this graph represent vectors of data for individual objects or images in the dataset. The closer these points are, the more similar objects from the same semantic class share some features. This close grouping shows that the model has effectively learned to recognize and classify these objects. Some of these points in this graph represent outliers, which are data points that are located outside the main

cluster. These points reflect errors or misclassifications, indicating that the model still has difficulty accurately handling some unusual situations.

5.3.2.2 *Second Phase Analysis – 500 Images*

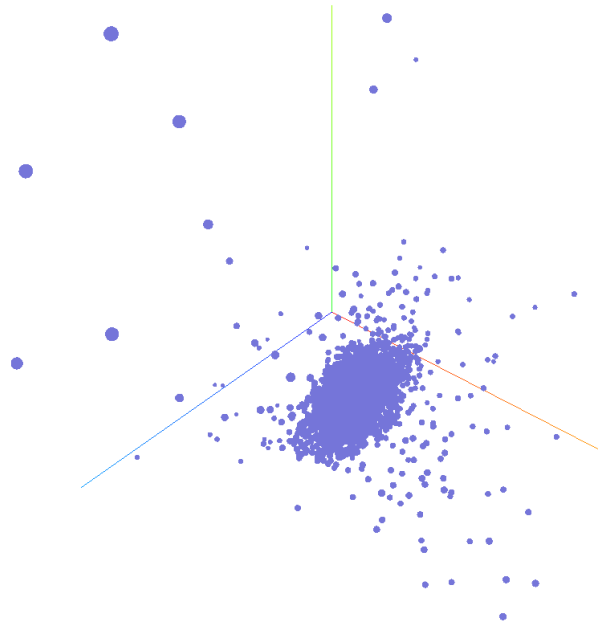


Figure 23. Graph illustrating data vectors in a lower-dimensional space, showing closer clustering of points in the second phase of analysis.

In the second phase analysis, the graph in Figure 23 shows that the points are closer together compared to the first phase. This closer clustering indicates that the model has improved in its ability to recognize and classify similar objects. As discussed in the analysis of Figure 22, some of these points are a bit far from the main cluster. These points are errors or model mistakes, indicating that the model still struggles with some unusual situations.

5.4 Build the Prototype System

The process of building the prototype system is discussed, focusing on the essential components and functionalities that contribute to its overall performance. The prototype was designed to fulfil specific requirements, including efficient defect detection and user interaction through a graphical interface.

5.4.1 Graphical User Interface (GUI)

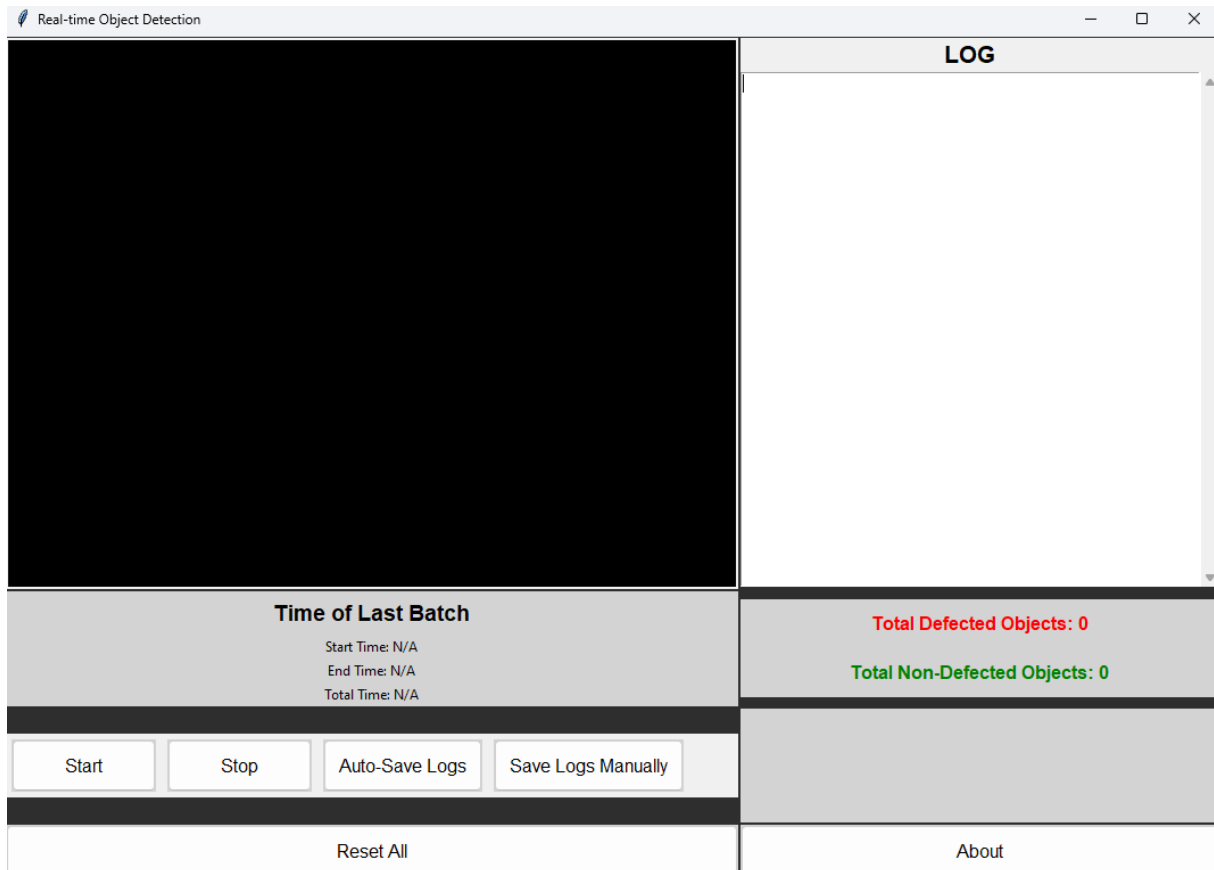


Figure 24. GUI displaying system monitoring and control functions.

The Graphical User Interface (GUI) created for this project is currently just a shell based on the requirements received from Metaqlix. The GUI was designed using Python with the Tkinter library. serves several key purposes:

Display of Important Information: It displays the important data such as the total number of defective and non-defective objects along with the total number of objects scanned. This information is always visible to the user, while the real-time counts of objects and defects detected appear in a dedicated window that is only shown while the program is running.

Monitoring Production Metrics: The GUI displays important timestamps for each production batch, including the start date and time, end date and time, and the total duration of the batch process. It provides real-time accuracy for each object and defect detected in front of the camera.

Logging Capabilities: The application has integrated logging features, allowing users to save logs automatically or manually.

Process Control: The prototype can start, stop or reset all values of the defect detection process.

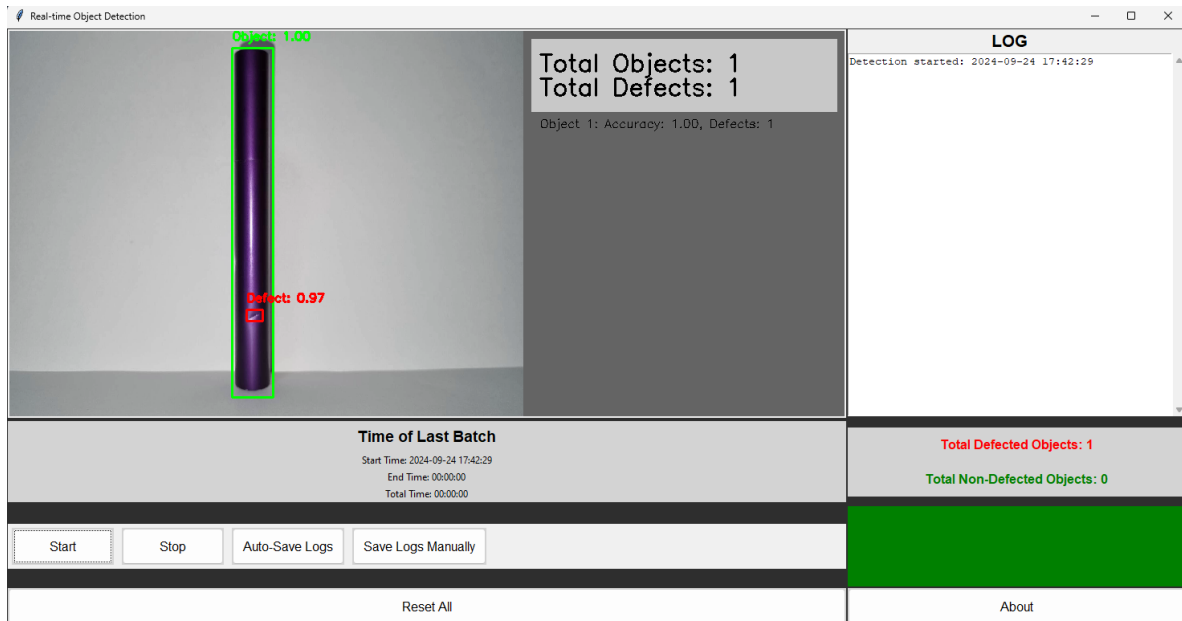


Figure 25. GUI displaying an object with a detected defect, demonstrating real-time detection.

Figure 25 shows an example of how the system can identify a defective object in real time. A green rectangle identifies the object, and a red rectangle identifies the defect in it. The GUI continually updates with the total number of faults found, and the number of defective objects is shown on the right side. As the process continues, this enables the user to view the defect detection status directly.

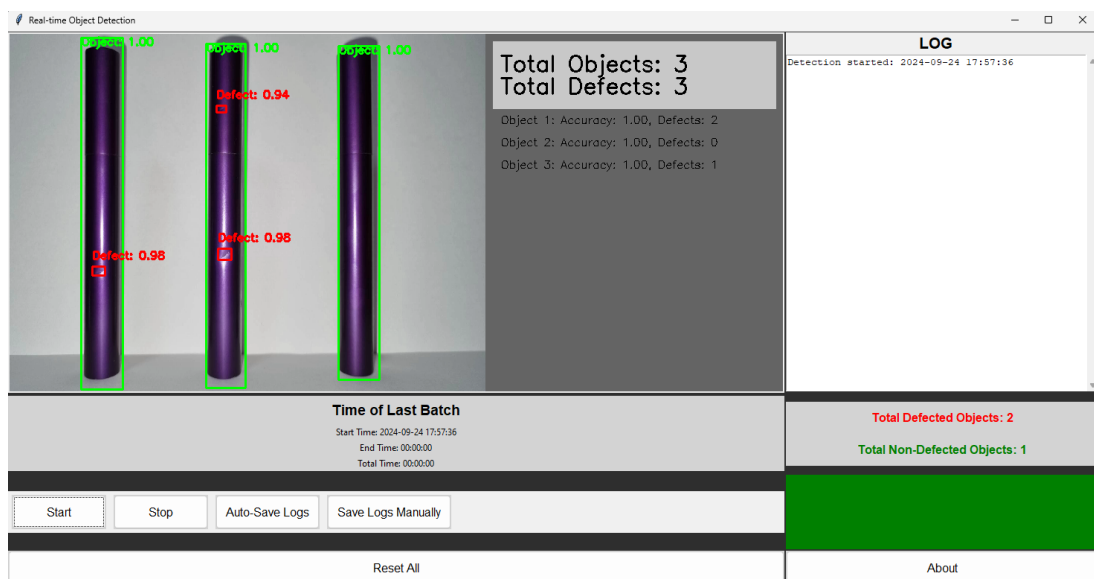


Figure 26. GUI displaying three objects: the left object has a detected defect, the middle object shows two defects, and the right object is without defects, demonstrating real-time detection.

In Figure 26, three objects, one with two defects, one with one defect, and one without any defects, are identified in real time by the system. The GUI keeps updating the number of defects found, and an additional panel on the right side shows the total number of faulty and non-defective items. On the right side of the GUI, the total number of defected objects is displayed in red, and the total number of non-defective objects is shown in green below it.

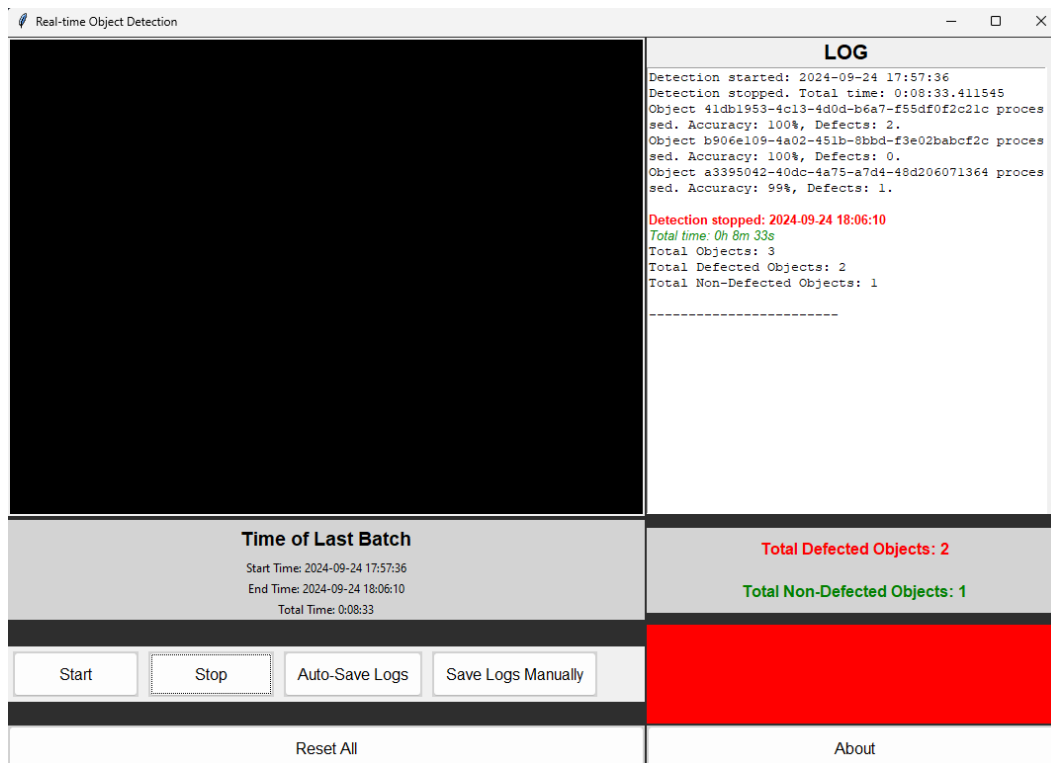


Figure 27. GUI displaying a log on the right side, lists all objects detected by the camera, including the count of defective and non-defective objects, as well as the total duration of monitoring.

As seen in Figure 27, an object is documented five seconds after it leaves the camera's field of vision. When the user presses the stop button, all things that have passed in front of the camera and are now visible in the live stream are logged. The following information are included in the logged information, which is kept in a plain text file: the timestamp of the detection, the total number of faulty and non-defective items identified, and the total number of objects scanned in the session. This logging feature helps the user keep a clear record of the detection process for later review.

5.5 Observe and Evaluate the System

5.5.1 Evaluation

In order to calculate the metrics, the following declaration was made with TP, FP, TN and FN.

- TP (True Positive): The model classifies image as containing a defect, and it does contain a defect.
- FP (False Positive): The model classifies image as containing a defect, and it does not contain a defect.
- TN (True Negative): The model classifies image as not containing a defect, and it does not contain a defect.
- FN (False Negative): The model classifies image as not containing a defect, and it does contain a defect.

In this evaluation, the focus was on assessing the defect detection performance of the model across different training steps, as we were unable to use more batch size due to technical constraints that resulted in errors during training. Instead, we trained the model over a range of steps, epochs, and evaluated its performance.

5.5.1.1 First Training Phase Evaluation

Metrics				
Dataset	Recall	Precision	Accuracy	F1-score
D1	0.833	0.714	0.700	0.768
D2	0.900	0.833	0.767	0.865
D3	0.929	0.907	0.883	0.918

Table 4. First phase performance metrics, tested on 60 images out of 300.

Table 4 presents the evaluation metrics for the models trained 10 000 steps. The dataset for this first phase involves testing of 60 images, 20% of the 300 images. D1, D2 and D3 represents stages of this first training phase. The metrics, as discussed in Section 2.1.4 (Evaluation Metrics), include recall, precision, accuracy, and F1 score, which offer insights into the model's performance in each stage.

Stage D1 achieved a recall of 0.833 and a precision of 0.714, with an accuracy of 0.700 and an F1 score of 0.768. While D1 demonstrates a commendable recall, indicating satisfactory performance in identifying defects, its precision and accuracy are relatively lower compared to the other datasets.

Stage D2 returned a recall of 0.900 and a precision of 0.833, thereby providing accuracy of 0.767 with an F1 score of 0.865. This stage has very balanced performance with high values for both recall and precision.

Stage D3 stands out with the highest overall performance. It achieved a recall of 0.929 and a precision of 0.907, with an accuracy of 0.883 and an F1 score of 0.918. This stage performs better to ensure that D3 is the most effective in detecting and classifying defects.

D1			
TARGET \ OUTPUT	Defect	No defect	SUM
Defect	30 50.00%	6 10.00%	36 83.33% 16.67%
No defect	12 20.00%	12 20.00%	24 50.00% 50.00%
SUM	42 71.43% 28.57%	18 66.67% 33.33%	42 / 60 70.00% 30.00%

Figure 28. Model 1, D1 confusion matrix

The confusion matrix for the model, which was trained with 10,000 steps using stage D1, is displayed in Figure 28. From this matrix, we can see that the model successfully identified 30 defective instances as defects (True Positives) and missed only 6 defects (False Negatives).

D2			
TARGET \ OUTPUT	Defect	No defect	SUM
Defect	45 75.00%	5 8.33%	50 90.00% 10.00%
No defect	9 15.00%	1 1.67%	10 10.00% 90.00%
SUM	54 83.33% 16.67%	6 16.67% 83.33%	46 / 60 76.67% 23.33%

Figure 29. Model 1, D2 confusion matrix.

For the model trained with 10,000 steps on dataset D2, the matrix indicates that the model successfully identified 45 defective items as defects (True Positives) and missed 5 defects (False Negatives). The matrix also shows that the model incorrectly classified 9 non-defective items as defects (False Positives). Additionally, it accurately recognized only 1 non-defective item (True Negative), according to Figure 29.

D3			
TARGET \ OUTPUT	Defect	No defect	SUM
Defect	39 65.00%	3 5.00%	42 92.86% 7.14%
No defect	4 6.67%	14 23.33%	18 77.78% 22.22%
SUM	43 90.70% 9.30%	17 82.35% 17.65%	53 / 60 88.33% 11.67%

Figure 30. Model 1, D3 confusion matrix.

The confusion matrix for the model trained with 10,000 steps on dataset D3 shows its performance in Figure 30. The matrix shows that the model accurately identified 39 defective items as defects (True Positives) and missed only 3 defects (False Negatives). This indicates a strong ability to detect actual defects, leading to a high recall rate. Additionally, the model incorrectly classified 4 non-defective items as defects (False Positives) and correctly identified 14 non-defective items (True Negatives).

In addition to the validation loss graphs, assessing the model's performance on specific test images provides deeper insights into its effectiveness. Figure 31 shows the model's detection results after 2000 training steps.

This image was input into the model, and the resulting output illustrates how the model performed. It shows that the object was not detected at all, with the green bounding box placed far from the actual object.

This initial performance is also reflected in the validation loss graph shown in Figure 17. After 2000 steps, the graph shows relatively high loss values, indicating that the model was not yet effectively learning to detect and localize objects. These high loss values match the inaccurate detection results seen in the test image.



Figure 31. The model failed to identify the object and defects after 2000 training steps.

Following this, the model underwent an additional 8000 steps of training, bringing the total to 10,000 steps. As shown in the updated test image, this extended training period led to significant improvements. The model was then able to accurately detect the object and correctly position the bounding box around it, with better handling of object defects and localization.

This improvement is presented in the graph of Figure 21, where the loss values decrease substantially with continued training, reflecting enhanced model performance and more accurate detection capabilities.



Figure 32. After 10,000 training steps, the model was able to detect the object and defects.

After the training had concluded, you could see how the system performed. Static images were fed to the system to see if the model could find the object and the defects. In Figure 32, it is shown that the model was able to identify two defects as well as the object. By marking it with green bounding boxes, we see that the model distinguishes between objects and defects.

5.5.1.2 Second Training Phase Evaluation

Metrics				
Dataset	Recall	Precision	Accuracy	F1-score
D1	0.81	1.00	0.9	0.9
D2	0.96	0.96	0.96	0.96
D3	0.89	0.9333	0.92	0.91

Table 5. Second phase performance metrics, tested on 100 images out of 500.

Table 5 presents the evaluation metrics for the models trained with 10 000 steps. Stage D1 achieved a recall of 0.81 and a precision of 1.00, with an accuracy of 0.9 and an F1 score of 0.9.

Dataset D2 returned a recall of 0.96 and a precision of 0.96, thereby providing accuracy of 0.96 with an F1 score of 0.96. This dataset has very balanced performance with high values for both recall and precision.

Dataset D3 stands out with the highest overall performance. It achieved a recall of 0.89 and a precision of 0.9333, with an accuracy of 0.92 and an F1 score of 0.91.

D1			
TARGET \ OUTPUT	Defect	Not Defect	SUM
Defect	43 43.00%	10 10.00%	53 81.13% 18.87%
Not Defect	0 0.00%	47 47.00%	47 100.00% 0.00%
SUM	43 100.00% 0.00%	57 82.46% 17.54%	90 / 100 90.00% 10.00%

Figure 33. Model 2, D1 confusion matrix.

The confusion matrix for the model, which was trained with 10,000 steps using dataset D1, is displayed in Figure 33. From this matrix, we can see that the model successfully identified 43 defective instances as defects (True Positives) and missed 10 defects (False Negatives).

D2			
TARGET \ OUTPUT	Defect	Not Defect	SUM
Defect	48 48.00%	2 2.00%	50 96.00% 4.00%
Not Defect	2 2.00%	48 48.00%	50 96.00% 4.00%
SUM	50 96.00% 4.00%	50 96.00% 4.00%	96 / 100 96.00% 4.00%

Figure 34. Model 2, D2 confusion matrix.

For the model trained with 10,000 steps on dataset D2, the matrix indicates that the model successfully identified 48 defective items as defects (True Positives) and missed 2 defects (False Negatives). The matrix also shows that the model incorrectly classified 2 non-defective items as defects (False Positives). Additionally, it accurately recognized 2 non-defective items (True Negative), according to Figure 34.

D3			
TARGET \ OUTPUT	Defect	Not Defect	SUM
Defect	42 42.00%	5 5.00%	47 89.36% 10.64%
Not Defect	3 3.00%	50 50.00%	53 94.34% 5.66%
SUM	45 93.33% 6.67%	55 90.91% 9.09%	92 / 100 92.00% 8.00%

Figure 35. Model 2, D3 confusion matrix.

The confusion matrix for the model trained with 10,000 steps on dataset D3 shows its performance. The matrix shows that the model accurately identified 42 defective items as defects (True Positives) and missed only 5 defects (False Negatives). This indicates a strong ability to detect actual defects, leading to a high recall rate. Additionally, the model incorrectly classified 3 non-defective items as defects (False Positives) and correctly identified 50 non-defective items (True Negatives), according to Figure 35.

6 Discussion

In this study a deep learning model was trained in several steps to identify and locate defect in images. The results from each step has shown a gradual improvement in the performance of the model, as evidenced by the reduction loss values (chapter 5.3) and improvements in the metrics obtained from the confusion matrices (chapter 5.5). After 2,000 training steps, the model showed significant difficulty in correctly identifying objects in the test images, as evidenced by both high loss values and poor performance in detecting defects on the object. This became particularly clear in Figure 31, where the bounding box was placed in the wrong place, and the object was not identified correctly. These results agree with the high loss values observed in the graph after 2000 steps of training. After another 8,000 steps, 10,000 steps in total, the models exhibited clear improvements.

After the testing had been performed for the first 300 pictures, three confusion matrices (D1, D2 and D3) were created to show how the model became increasingly more capable of identifying and locating the object, with increased recall and precision values. According to Figure 30, the testing phase D3 showed the best overall performance after the 10,000 steps, with high values for all metrics, especially recall and F1 score, indicating a more reliable detection of defects.

Analysing the final results for D3, we see that the model achieved high recall and precision, which is particularly valuable in contexts where it is critical to detect all possible defects, even if this means generating some false positives (FPs).

We redid the confusion matrixes for the next batch of pictures, namely the 500 pictures. Phase D1 (see Figure 33), of the second batch showed the highest precision of 100%, while the accuracy was 90%. Phase D3 had slightly higher recall, accuracy and F1-score while the precision value was lower caused by the model classifying two non-defective images as defective. In other words, two FP.

Looking at phase D2 in Figure 34, we can see that the model measures 96% through all measurement values, which shows the highest and most balanced results compared to the other testing phases. During the D2 phase, model 2 classified some non-defective images as defective and defective images as non-defective, causing FP and FN.

A principal factor that affected the performance of the model was the lighting in the test images. In some cases, it was difficult to detect defects due to poor, insufficient or uneven lighting, which affected the model's ability to correctly identify the defects. This highlights the importance of having a good light source both during image acquisition and live detection in the production environment. In an environment where lighting conditions has been carefully controlled and optimised to support defect detection, the model is expected to perform significantly better. It is interesting to see this relationship come into play considering the significance it has on the performance of the model, it is highly relevant to this study, but must also be taken into account as the limitation that it is.

It is also important to note the specific conditions that affected the training process. Although the model was not trained for several epochs due to technical limitations, it still managed to achieve significant results by increasing the number of training steps.

After 10.000 steps of training, the loss graph showed a slight increase in loss, as evidenced in Figure 18 and 20. This indicates that the model is beginning to become over-trained and will thus perform less accurately in detecting and classifying objects.

7 Conclusion

In this work, we conducted a study on suitable AI solutions for a package image recognition program, during which we collected, prepared, and trained datasets. We also developed a graphical user interface (GUI) that functions as an object detection application to facilitate real-time analysis. By reviewing papers that detailed systems similar to our proposed solution and comparing the positive and negative aspects of many potential AI methodologies, we were able to rule out a number of options. Since Faster R-CNN was the most popular AI solution for this type of job, we concluded that this AI solution would best suit our system. We ultimately achieved our goal, with our system reaching an accuracy of 96%.

RQ1: What is a suitable AI solution for the proposed package image recognition program?

During our research, we went through a different possible AI solution that could work with the proposed package image recognition program. Weighing the options and considering the different benefits and drawbacks, we had difficulties deciding on just one solution. We therefore continued our research by going through different papers that had created systems similar to our proposed system, thus eliminating a few more options. The most commonly used AI solution for the kind of projects we have executed was Faster R-CNN, which helped us finalise our decision.

RQ2: How can the performance level of the existing algorithms for package defect recognition be improved?

If we look at the results of the first and the second training phase, we can see a great improvement of the different values between the phases. There are many factors that played a role, partly the number of images, but also the number of epochs. The more images the model could train on, the better and more accurate the model became at detecting objects and any defects,

The algorithm can be further improved by adding additional functions that controls what is to be detected in the live feed when performing the quality assurance. Namely, more control in where the bounding boxes are placed and only processing what it contains, in the feed.

8 Future Work

While this work focused on training the model to identify defects in specific products, there are several improvements that can be made to enhance the system's capabilities. Future work could include the following:

- Expanding the dataset
- Increase the product catalogue
- Make a fully automated elimination process
- The system may have the ability to easily add more data models to expand the system's ability to identify other products.
- The system may have the ability to dynamically identify different products and save their data in a database related to the specific product group.

References

- [1] Metaqlix AB, *Personal communication via E-mail.*, info@metaqlix.com.
- [2] Y. Li and Z. Min, "Review on the Status and Development Trend of AI Industry," in *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, Chengdu, China, 2019.
- [3] P. Burggräf, J. Wagner and B. Koke, "2018 International Conference on Information Management and Processing (ICIMP)," in *Artificial intelligence in production management: A review of the current state of affairs and research trends in academia*, 2018.
- [4] X. Dong, E. liang and L. Zi, "Research on Machine Learning Methods for Intelligent Decision Problems," 2019.
- [5] T. Nakazawa and D. V. Kulkarni, "Wafer Map Defect Pattern Classification and Image Retrieval Using Convolutional Neural Network," *IEEE Transactions on Semiconductor Manufacturing*, vol. 31, p. 6, 2018.
- [6] M.-C. Chiu and T.-M. Chen, "Applying Data Augmentation and Mask R-CNN-Based Instance Segmentation Method for Mixed-Type Wafer Maps Defect Patterns Classification," *IEEE Transactions on Semiconductor Manufacturing*, vol. 34, no. 4, p. 9, 2021.
- [7] C. Janiesch, P. Zschech and K. Heinrich, "Machine learning and deep learning," vol. 31, pp. 685-695, 08 april 2021.
- [8] P. P. Shinde and S. Shah, "A Review of Machine Learning and Deep Learning Applications," in *IEEE*, Pune, India, 2018.
- [9] V. Tyagi, *Understanding Digital Image Processing*, Oxfordshire: Taylor & Francis Group, 2018.
- [10] T. Acharya and A. K. Ray, *Image Processing Principles and Applications*, Canada: John Wiley & Sons, Inc., Hoboken, New Jersey, 2005 .
- [11] "ibm," [Online]. Available: <https://www.ibm.com/topics/supervised-learning>. [Accessed 07 June 2024].
- [12] "jvatpoint," Free learning platform for better future, [Online]. Available: <https://www.jvatpoint.com/data-labelling-in-machine-learning>. [Accessed 07 June 2024].
- [13] T. Fredriksson, J. Bosch, H. H. Olsson and D. I. Mattos, "Machine Learning Algorithms for Labeling: Where and How They are Used?," in *Annual IEEE Systems Conference*, Montreal, QC, Canada, 2022.
- [14] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," *SN Computer Science*, vol. 2, p. 21, 2021.

- [15] V. Gupta, V. K. Mishra, P. Singhal and A. Kumar, “An Overview of Supervised Machine Learning Algorithm,” in *International Conference on System Modeling & Advancement in Research Trends (SMART)*, Moradabad, India, 2022.
- [16] N. Agnihotri, “Engineers Garage,” [Online]. Available: <https://www.engineersgarage.com/machine-learning-algorithms-classification/>. [Accessed 08 06 2024].
- [17] D. Rolon-Mérette, M. Ross, T. Rolon-Mérette and K. Church, “Introduction to Anaconda and Python:,” *The Quantitative Methods for Psychology*, vol. 16, p. 9, 2020.
- [18] H. Abu-Mariah and W. Ashour, “Moving Object Detection Based on Clustering and Event-Based Camera,” in *IEEE*, Gaza, State of Palestine, 2023.
- [19] H. Ju, C. Peng, J. Liu, J. Zhang and L. Liu, “Improved Metric-Learning-Based Recognition Method for Rail Surface State With Small-Sample Data,” *IEEE*, vol. 12, p. 12, 2024.
- [20] . A. Culotta, P. Kanani, R. Hall, M. Wick and A. McCallum, “Error-driven Machine Learning with a Ranking Loss Function,” *Department of Computer Science*, p. 6, 2018.
- [21] Q. Wang, . Y. Ma, K. Zhao and . Y. Tian, “A Comprehensive Survey of Loss Functions in Machine,” *Annals of Data Science*, vol. 9, pp. 187-212, 2022.
- [22] Nvidia, “Nvidia,” 2024. [Online]. Available: <https://www.nvidia.com/en-us/glossary/computer-vision/#>. [Accessed 2024].
- [23] Y. Guo, Y. Liu, A. Oerlemans, . S. Lao, S. Wu and M. S. Lew, “Deep learning for visual understanding: A review,” *Neurocomputing*, vol. 187, pp. 27-48, 2015.
- [24] S. Shah, “Alanalytics Vidhya,” 15 March 2022. [Online]. Available: <https://www.analyticsvidhya.com/blog/2022/01/convolutional-neural-network-an-overview/>. [Accessed 06 June 2024].
- [25] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie and L. Farhan, “Review of deep learning: concepts, CNN,” *Journal of Big Data*, vol. 8, p. 74, 2021.
- [26] S. Hesaraki, “medium,” 18 10 2023. [Online]. Available: <https://medium.com/@saba99/feature-map-35ba7e6c689e>. [Accessed 11 04 2024].
- [27] S. Khosla, “geeksforgeeks,” 21 04 2023. [Online]. Available: <https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/>. [Accessed 11 04 2024].
- [28] A. Makone, “Medium,” 10 12 2020. [Online]. Available: <https://ashutoshmakone.medium.com/faster-rcnn-502e4a2e1ec6>.
- [29] Z. Ning, F. Yiran and L. Eung-Joo, “Activity Object Detection Based on Improved Faster R-CNN,” *Journal of Korea Multimedia Society (한국멀티미디어학회논문지)*, vol. 24, no. 3, pp. 416-422, 2021.

- [30] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," pp. 436-444, 27 May 2015.
- [31] E. A. E. Arthur, F. A. Wulnye, D. A. N. Gookyi, K. O.-B. O. Agyekum, P. Danquah and R. Gyaang, "Edge Impulse vs TensorFlow: A Comparative Analysis of TinyML Platforms for Maize Leaf Disease Identification," in *IEEE*, 2024.
- [32] M. Somvanshi, P. Chavan, S. Tambade and S. V. Shinde, "A review of machine learning techniques using decision tree and support vector machine," in *IEEE*, Pune, 2016.
- [33] I. Culjak, D. Abram, T. Pribanic, H. Dzapo and M. Cifrek, "A brief introduction to OpenCV," in *IEEE*, Opatija, 2012.
- [34] B. Hangün and Ö. Eyecioğlu, "Performance Comparison Between OpenCV Built in," *International Journal of Engineering Science and Application*, vol. 1, no. No. 2, p. 8, 2017 .
- [35] H. R. Roth, I. Yang, Y. Cheng, Y. Wen, Z. Xu, Y.-T. Hsieh, K. Kersten, A. Harouni, C. Zhao, K. Lu, Z. Zhang, W. Li, A. Myronenko, D. Yang, S. Yang, N. Rieke, A. Quraini, C. Chen, D. Xu, N. Ma, P. Dogra, M. Flores and A. Feng, "NVIDIA FLARE: Federated Learning from Simulation to Real-World," arXiv, 2022.
- [36] B. Pang, E. Nijkamp and Y. N. Wu, "Sage Journals," 10 september 2019. [Online]. Available: https://journals.sagepub.com/doi/full/10.3102/1076998619872761?casa_token=tn2j0IwDqk4AAAA%3AuH-88NgvVBkbqeLDKA4jiCEs0ckZ6sjSDaYT0j561dmTs46jCWR7synu-zGx1NexCRQPtSxaw4Tn5Q. [Accessed 08 june 2024].
- [37] S. Prakiljačić, R. Grbić, M. Vranješ and M. Herceg, "Tool for image annotation in context of modern object detection," in *IEEE*, Novi Sad, Serbia, 2024.
- [38] B. C. Russell, A. Torralba, K. P. Murphy and W. T. Freeman , "LabelMe: A Database and Web-Based Tool for Image Annotation," vol. 77, p. 17, 2008.
- [39] L. Shengyu, W. Beizhan, W. Hongji, C. Lihao, L. Ma and Z. Xiaoyan, "A real-time object detection algorithm for video," *Computers & Electrical Engineering*, vol. 77, pp. 398-408, 2019.
- [40] A. Kushwaha, "Fruit Classification Using Optimized CNN," in *2023 International Conference on IoT, Communication and Automation Technology (ICICAT)*, 2023.
- [41] J. J. Dela Cruz, J. De Leon, C. A. Bundoc, M. Geroleo, M. L. Lazatin and M. Rosales, "ARES: An Automated Rotten Egg Sorter Utilizing the Egg's Physical Properties and Artificial Neural Network," in *2023 11th International Conference on Information and Communication Technology (ICoICT)*, 2023.
- [42] K. S. Younis, W. Ayyad and A. Al-Ajlony, "Embedded system implementation for material recognition using deep learning," in *2017 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, 2017.
- [43] M. Chen, T. D. M. Purdin, J. F. NUNAMAKER and JR, "Systems Development in Information Systems Research," vol. 7, p. 18, 1991.

- [44] R. Singh, “Choosing the Best ML Model: A Guide to Model Selection,” 3 June 2023. [Online]. Available: <https://www.linkedin.com/pulse/choosing-best-ml-model-guide-selection-ravi-singh>. [Accessed 10 April 2024].
- [45] G. Zaccane and M. R. Karim, *Deep Learning with TensorFlow*, Birmingham: Packt Publishing Ltd, 2017.
- [46] Y. Liu, “An Improved Faster R-CNN for Object Detection,” in *IEEE*, Shanghai, 2018.
- [47] “TensorFlow 2 Object Detection API tutorial,” 2020. [Online]. Available: <https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/index.html>. [Accessed 25 06 2024].
- [48] A. Tolk, “The Next Generation of Modeling & Simulation: Integrating Big Data and Deep Learning,” p. 8, 2015.
- [49] TensorFlow, “TensorBoard,” TensorFlow, 2024. [Online]. Available: <https://www.tensorflow.org/tensorboard>. [Accessed 2024].
- [50] E. Lisowski, “Deep Learning Architecture Examples,” 2020.
- [51] A. “Open-instruction,” 21 05 2021. [Online]. Available: <https://open-instruction.com/dl-algorithms/convolutional-neural-network/>. [Accessed 12 04 2024].
- [52] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” *CVPR*, p. 10, 2016.
- [53] K. Peffers, T. Tuure, M. A. Rothenberger and S. Chatterjee, “A Design Science Research Methodology for information Systems Research,” *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45-77, 2007.
- [54] D. Kreuzberger, N. Kühl and H. Sebastian , “Machine Learning Operations (MLOps): Overview, Definition, and Architecture,” *IEEE Access*, vol. 11, p. 14, 2023.
- [55] NVIDIA, [Online]. Available: <https://www.nvidia.com/en-us/glossary/tensorflow/>. [Accessed 09 June 2024].