FH Vorarlberg
University of Applied Sciences

Evaluation of visual inspection tests via object detection.

A case study using the example of power transformers in the energy industry.

Master thesis
for the award of the academic degree

Master of Science in Engineering (MSc)

University of Applied Sciences Vorarlberg
Master computer science

Supervised by
Dr. rer. nat. Michael Hellwig

Submitted by
Frederick Zech, BSc

Dornbirn, July 2023

Abstract

Evaluation of visual inspection tests via object detection.

This thesis evaluates the feasibility of conducting visual inspection tests on power industry constructions using object detection techniques. The introduction provides an overview of this field's state-of-the-art technologies and approaches. For the implementation, a case study is then conducted, which is done in collaboration with the partner company OMICRON Electronics GmbH, focusing on power transformers as an example. The objective is to develop an inspection test using photographs to identify power transformers and their sub-components and detect existing rust spots and oil leaks within these components. Three object detection models are trained: one for power transformers and sub-components, one for rust detection, and one for oil leak detection. The training process utilizes the implementation of the YOLOv5 algorithm on a Linux-based workstation with an NVIDIA Quadro RTX 4000 GPU. The power transformer model is trained on a dataset provided by the partner company, while open-source datasets are used for rust and oil leak detection. The study highlights the need for a more powerful GPU to enhance training experiments and utilizes an Azure DevOps Pipeline to optimize the workflow. The performance of the power transformer detection model is satisfactory but influenced by image angles and an imbalance of certain sub-components in the dataset. Multi-angle video footage is a proposed solution for the inspection test to address this limitation and increase the size of the dataset, focusing on reducing the imbalance. The models trained on open-source datasets demonstrate the potential for rust and oil leak detection but lack accuracy due to their generic nature. Therefore, the datasets must be adjusted with case-specific data to achieve the desired accuracy for reliable visual inspection tests. The results of the case study have been well-received by the partner company's management, indicating future development opportunities. This case study will likely be a foundation for implementing visual inspection tests as a product.

Kurzreferat

Evaluierung von visueller Inspektionsprüfung durch Objekterkennung.

In dieser Arbeit wird die Durchführbarkeit von visuellen Inspektionstests an Konstruktionen der Energiewirtschaft mit Hilfe von Objekterkennungstechniken evaluiert. In der Einleitung wird ein Überblick über den Stand der Technik und die Ansätze in diesem Bereich gegeben. Für die Umsetzung wird dann eine Fallstudie durchgeführt, die in Zusammenarbeit mit dem Partner-Unternehmen OMICRON Electronics GmbH am Beispiel von Leistungstransformatoren durchgeführt wird. Ziel ist es, einen Inspektionstest zu entwickeln, der anhand von Fotos Stromtransformator und deren Bestandteile identifiziert, um dann vorhandene Roststellen und Öllecks in diesen Komponenten zu erkennen. Es werden drei Objekterkennungsmodelle trainiert: eines für Stromtransformator und deren Bestandteile, eines für die Rosterkennung und eines für die Erkennung von Öllecks. Der Trainingsprozess nutzt die Implementierung des YOLOv5-Algorithmus auf einer Linux-basierten Maschine, die mit einer NVIDIA Quadro RTX 4000 GPU ausgerüstet ist. Das Modell für den Stromtransformator wird anhand eines vom Partnerunternehmen bereitgestellten Datensatzes trainiert, während für die Erkennung von Rost und Öllecks Open-Source-Datensätze verwendet werden. Die Studie unterstreicht den Bedarf einer leistungsfähigeren GPU zur Verbesserung der Trainingsexperimente und nutzt eine Azure DevOps Pipeline zur Optimierung des Arbeitsablaufs. Die Leistung des Modells zur Erkennung von Stromtransformatoren ist zufriedenstellend, wird jedoch durch Bildwinkel und ein Ungleichgewicht bestimmter Unterkomponenten im Datensatz beeinflusst. Die Verwendung von Videomaterial mit mehreren Blickwinkeln ist eine vorgeschlagene Lösung für den Inspektionstest, um diese Einschränkung zu beheben, sowie die Vergrößerung des Datensatzes mit Schwerpunkt auf die Verringerung des Ungleichgewichts. Die Modelle, die auf Open-Source-Datensätzen trainiert wurden, zeigen das Potenzial für die Erkennung von Rost und Öllecks, sind aber aufgrund ihrer allgemeinen Natur nicht präzise genug. Daher müssen die Datensätze mit fallspezifischen Daten angepasst werden, um die gewünschte Genauigkeit für zuverlässige visuelle Inspektionstests zu erreichen. Die Ergebnisse der Fallstudie wurden von der Geschäftsleitung des Partnerunternehmens positiv aufgenommen, was auf zukünftige Entwicklungsmöglichkeiten hindeutet. Diese Fallstudie wird wahrscheinlich eine Grundlage für die Einführung von visuellen Inspektionstests als Produkt sein.

Foreword

I would like to thank the company OMICRON electronics GmbH for allowing me to work on this suspenseful project. I would especially like to thank the supervisors David Gopp (OMICRON), Simon Keckeis (OMICRON) and Michael Hellwig (University of Applied Sciences Vorarlberg) for their helpful feedback and reviews of the thesis.

Contents

List of Figures

List of Tables

1 Introduction

In recent years, the rapid advancement of artificial intelligence (AI) has propelled the field to new heights. AI has become increasingly powerful and versatile, with significant breakthroughs in machine learning algorithms and the availability of vast datasets. This thesis focuses on object detection, a crucial task within computer vision. The case study aims to evaluate the capability of performing visual inspection tests via object detection, using power transformers as a specific example.

The subsequent chapters delve into the problem statement, delimitation, and business objective, followed by an exploration of the state-of-the-art techniques, ranging from the fundamental concepts of neurons to the YOLO algorithm. The report further outlines the implementation and experimental setting, describing the technologies used and the conducted experiments for the implementation. Subsequently, the experimental results are presented, providing a detailed overview of the outcomes obtained from the implementation. These results are then critically discussed, evaluating their implications and addressing potential limitations. Finally, the thesis concludes with a summary of the findings and an outlook for future research and development.

1.1 Partner company

This case study is undertaken in collaboration with OMICRON Electronics GmbH. This company has developed a keen interest in projects related to artificial intelligence (AI) as the field has witnessed significant advancements. The company develops test equipment for testing, protection, and measurement devices in the power industry. The power industry, or the energy industry, encompasses various activities related to generating, transmitting, distributing, and utilizing electrical power. It plays a vital role in meeting the energy demands of societies, powering homes, businesses, industries, and infrastructure worldwide. OMICRON's offerings consist of hardware and software products, and they are recognized for their extensive market presence, selling their products in over 160 countries across the globe.

An exemplary product in this context is the Primary Test Manager (PTM), designed as a comprehensive management tool for testing various assets, including power transformers, circuit breakers, and current transformers. It serves as a control system for automated test procedures and streamlines the testing process by providing step-by-step guidance to users through test workflows. The PTM is a software suite and can be used with various test systems, including the CPC 100, as shown in Figure 1.1. The CPC 100 is a hardware solution for diagnostic tests on multiple assets. While tests can be performed directly on the CPC 100, leveraging the PTM simplifies the testing process and provides additional features.

When using the PTM, users can generate test plans specific to their requirements. These test plans outline the steps and procedures for conducting tests, ensuring a systematic approach. Additionally, the PTM generates comprehensive reports that provide detailed insights and analysis of the test results. This helps users understand the performance and condition of the tested assets.

Furthermore, the PTM facilitates the integration of asset data into a centralized database. This allows for easy access and management of the test data, providing a comprehensive view of the assets' testing history and enabling efficient tracking of any changes or trends over time.

In summary, the PTM software suite enhances the testing process when used in conjunction with devices, such as the CPC 100 test system. It simplifies testing by providing customized test plans, generating comprehensive reports, and integrating asset data into a database for better management and analysis.

Figure 1.1: CPC 100

1.2 Problem

The energy industry is experiencing significant changes, with the growing use of renewable energy sources leading to a distributed approach to energy generation, distribution, and transmission. In recent years, there has been a surge in the construction of small substations, with more anticipated, while existing substations require maintenance and extension to meet future expectations. Furthermore, the industry faces a talent gap, with many engineers approaching retirement and difficulty attracting new talent to manage the energy transition.

Effective maintenance of individual assets in the power system requires sufficient data to plan and execute maintenance activities. In addition to operational data, monitoring data (online data) and offline data are needed. Two primary approaches to asset maintenance exist; the time-based approach, which involves carrying out tests, revisions, and repairs on assets within predefined time windows, and the event-based approach, which consists of planning tests, overhauls, and repairs based on online measured data from monitoring devices. Visual inspection carried out regularly plays a vital role in maintenance activities. It involves checking if the equipment behaves appropriately and identifying signs of corrosion, oil leakage, burn marks, or other issues. Infrared cameras are often used to identify hot spots on equipment. Visual inspection is subjective and often based on a classification task, with a key ranging from one to five or a traffic light system.

With the energy network's rapid expansion, finding and retaining talent for visual inspection is expected to become increasingly difficult. Additionally, visual inspection is inherently subjective, making obtaining consistent and objective results challenging. However, object detection technology can address these issues by providing more objective outcomes. By employing object detection techniques, a visual inspection can be enhanced, leading to improved objectivity in the results obtained. The primary goal is to support the decisions made by test engineers and mitigate wrong choices rather than fully automate the testing process. Complete automation would demand high accuracy and compliance with legal requirements, which can be complex and challenging. However, it is essential to note that human expertise and judgment are still crucial in the decision-making process, as certain factors may require subjective interpretation.

The following part of this section gives an overview of the power industry's critical components used in this study.

The power transformer is an essential asset in electrical energy distribution and transport. It has the task of transforming the voltage generated by the power plants to a higher voltage level for transport. It also transforms the voltage for the distribution and the consumers to a lower voltage level. Furthermore, the power transformer often has the highest criticality rating compared to the other assets in the electrical power grid. These other assets are mainly current transformers, voltage transformers, and circuit breakers. Figure 1.2 shows the main components of a power transformer. Here, a three-phase two-winding transformer is shown, which has a tap changer (6) in detail. Three-phase means that the transformer has a three-leg core (1). Two-winding in this context means the transformer has a primary (3) and a secondary (2) winding. The low voltage side has the low voltage bushings (4), and the high voltage side has the high voltage bushings (5). In addition, this transformer has an expansion tank for the oil. Finally, this transformer has a cooling system (8), in this example, even with radiators [1]. There are several cooling methods for

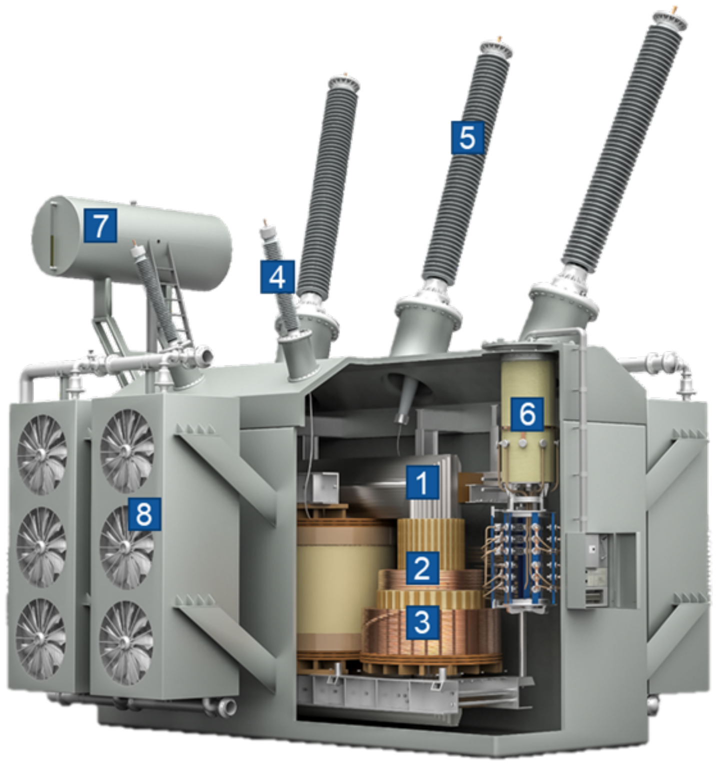


Figure 1.2: Three-phase Two-winding Power Transformer. (Source: [1])

liquid-immersed power transformers available. A four-letter code describes the cooling class in more detail [2].

First letter: Internal cooling medium:

- O: mineral oil or synthetic insulating liquid with fire point below 300 °C
- K: insulating liquid with fire point > 300 °C
- L: insulating liquid with no measurable fire point

Second letter: Circulation mechanism for internal cooling medium:

- N: natural thermosiphon flow through cooling equipment and in windings
- F: forced circulation through cooling equipment, thermosiphon flow in windings
- D: forced circulation through cooling equipment, directed from the cooling equipment into at least the main windings

Third letter: External cooling medium:

- A: air
- W: water

Fourth letter: Circulation mechanism for external cooling medium:

- N: natural convection
- F: forced circulation (fans, pumps)

1.3 Delimitation

The main aim of this study is to conduct an image-based visual inspection focusing on power transformers, which are critical components in the energy industry for power transmission and distribution. The study aims to identify various features of power transformers, including bushings, surge arresters, oil expansions, and cooling systems within photographs. Additionally, it aims to detect two specific potential flaws: oil leaks and corrosion. The objective is to demonstrate the feasibility of object detection for visual inspection tests without initially requiring high precision. Any specific restrictions or requirements regarding picture conditions, such as weather, time of day, or angles, have not been imposed, as the available photos were not acquired specifically for this project but were collected from various sources. However, if this study reveals that such conditions significantly impact the results, they may be considered requirements for future development.

Upon discussing the desired outcome with the partner company, it has been determined that a traffic light system will suffice as an initial evaluation for the visual inspection tests. This system categorizes the inspection results into green, yellow, and red levels, representing good condition, potential issues, and clear failures. By employing this simplified and intuitive approach, quick decisions can be made based on the categorized levels while acknowledging that further analysis may still be required in some instances.

1.4 Business Objective

The business objective of this project is to develop a web service that offers image-based visual inspection classification as a service. The service is intended to receive images from clients, resulting in documentation of the classified data and a test report. The primary purpose of this service is to achieve objectivity, standardization, and documentation of the visual inspection task, ultimately providing clients with reliable and consistent results.

2 State of the Art

This chapter presents a comprehensive overview of the current state of the art in machine learning applied to visual inspection tests. It is divided into several subchapters, each focusing on a key aspect relevant to visual inspection. The subchapters cover various topics, including an exploration of different types of machine learning approaches, an introduction to the importance of data labeling and data augmentation, and the significance of monitoring. Next, the chapter delves into neural networks, followed by recurrent neural networks and convolutional neural networks, widely used for object detection. Lastly, the popular YOLO (You Only Look Once) algorithm and its evolution from the first to the fifth version are introduced.

By examining these different areas, this chapter establishes a strong foundation and understanding of the latest advancements and methodologies in machine learning for visual inspection tests. This chapter serves as a vital foundation for the subsequent chapters, as successfully implementing the visual inspection tests relies on the knowledge presented here. Understanding state of the art in machine learning techniques, data labeling, data augmentation, monitoring, neural networks (including recurrent neural networks and convolutional neural networks), and the YOLO algorithm is essential for conducting effective visual inspection tests. The necessary knowledge to delve into the practical application of visual inspection in the following chapters is provided by comprehending the concepts and methodologies discussed in this chapter.

2.1 Machine Learning Types

Machine learning can be broadly classified into two main types: supervised and unsupervised. In supervised learning, algorithms learn to predict outputs by observing labeled data, making it valuable for classification, regression, and prediction tasks. On the other hand, unsupervised learning algorithms operate on unlabeled data, which contains features utilized to discover patterns, relationships, and structures within the data [3, p. 104-105]. In this case study, there is access to labeled data specifically for object detection, which makes supervised learning the primary focus of this thesis.

2.2 Data Labeling

For the implementation of this case study, labeled data is available. Therefore, this chapter gives a comprehensive introduction to the data labeling process.

Data labeling is an essential pre-processing step for training a supervised machine learning model. This process involves annotating the available data for training and validation by adding bounding boxes or other annotations to provide the necessary context for the models. The most common formats for annotating data are JSON, Pascal VOC, which stores the annotations in an XML file, and YOLO, which stores annotations in a text file. Although data labeling may seem straightforward, training a high-quality machine learning model requires significant effort, particularly for large datasets. Different approaches for companies to perform data labeling with different benefits are presented below.

- **Internal Labeling -** Internal experts label the data, which provides high-quality labels but requires a lot of time and resources.
- **Synthetic Labeling -** New labeled data is generated from existing datasets which is time efficient but requires extensive computing power.
- **Programmatic Labeling -** Scripts are used to label data which reduces the required time but must be verified by quality insurance to detect technical errors.
- **Outsourcing -** A common way is to outsource the process to freelancers, which still can take time but saves internal resources.
- **Crowd-sourcing -** This is distributed web-based and is a quick approach to label data which is also cost-effective but can lack in quality of the labels. Recaptcha is the most famous example of crowd-sourcing, where users identify images with a certain asset to prove that they are not a bot.

The most significant trade-off in data labeling is between label quality, which affects model accuracy, and the associated costs and time required for implementation [4]. The data for this case study has been labeled by domain experts from the partner company.

2.3 Data Augmentation

Data augmentation techniques were applied to enhance the performance of the models employed in this case study. Consequently, this chapter presents an introduction to data augmentation.

Insufficient training data can lead to overfitting in machine learning models. More precisely, overfitting occurs when a machine learning model excessively tailors itself to the training data, reducing generalization performance on unseen data. Data augmentation is a method that artificially increases the amount of available data by applying random transformations to the existing data. By introducing such modifications, each image is essentially altered and rendered unique, reducing the likelihood of overfitting and improving the model's generalization ability.

Frequently used data augmentation methods in machine learning include rotating, flipping, cropping, scaling, adjusting brightness, saturation, and contrast, manipulating color schemes, and introducing noise. Rotating images at different angles helps capture object orientation variations, and flipping enhances the model's ability to generalize across different object orientations. Cropping allows for diverse viewpoints and focuses on specific object features. Scaling helps the model handle objects of various sizes, and adjusting brightness, saturation, and contrast enables better recognition under different lighting conditions. Manipulating color schemes and introducing noise improve the model's ability to handle variations in color, texture, and background noise encountered in real-world scenarios. These techniques augment the dataset, enhance model performance, and increase robustness. It is worth noting that data augmentation offers a wide range of possibilities beyond these techniques [5, p. 138-142].

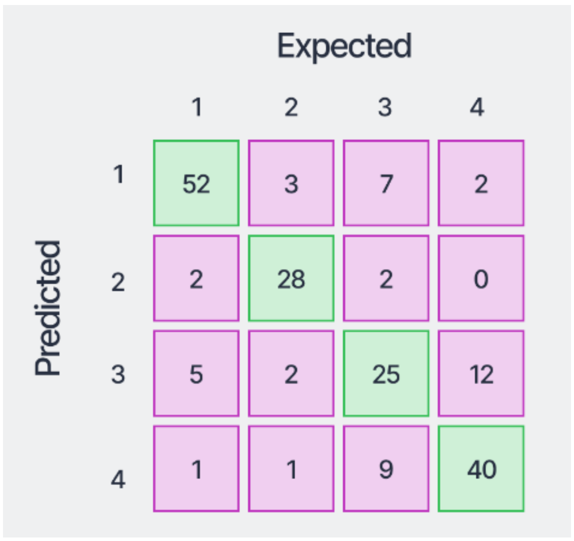
2.4 Monitoring

Monitoring the machine learning training process is a critical aspect of model development. It involves closely observing and analyzing the model's performance and behavior during training. Callback functions provide the required information to receive feedback on the model's learning state. This feedback helps to improve the model's training, as many things can not be predicted from the start.

One frequently used callback function to optimize training time is "early stopping." This function interrupts the training process if the validation loss fails to improve for a specified number of epochs and stores the epoch with the best training result. Another valuable callback is the "logging of training and validation metrics," which captures the metrics achieved in each epoch. This information can be utilized to visualize the training progress through various graphs, providing valuable insights into the training process [6, p. 249-256]. Géron [7, p. 90-97] provides a comprehensive overview of various performance measures; two common metrics used in this case study are described below.

- **Confusion Matrix -** Counts the number of times instances of class A are classified as class B, showing the number of times the classifier confused between two classes. The predicted classes are listed on the vertical axis, and the expected classes are on the horizontal axis. The fields in the diagonal from top left to bottom right show the number of true positive values, illustrated in Figure 2.1

- **Precision and Recall -** Precision measures the proportion of correctly predicted instances out of the total predicted instances. In contrast, recall measures the proposition of correctly predicted instances out of the total actual positive instances. The precision and recall curve visualizes the trade-off between the two values. Figure 2.2 shows how this curve can look. The two metrics are often combined into the F1 score, the harmonic mean of precision and recall.



		Expected			
		1	2	3	4
Predicted	1	52	3	7	2
	2	2	28	2	0
	3	5	2	25	12
	4	1	1	9	40

Figure 2.1: Example of Confusion Matrix representing four Classes. (Source: [8])

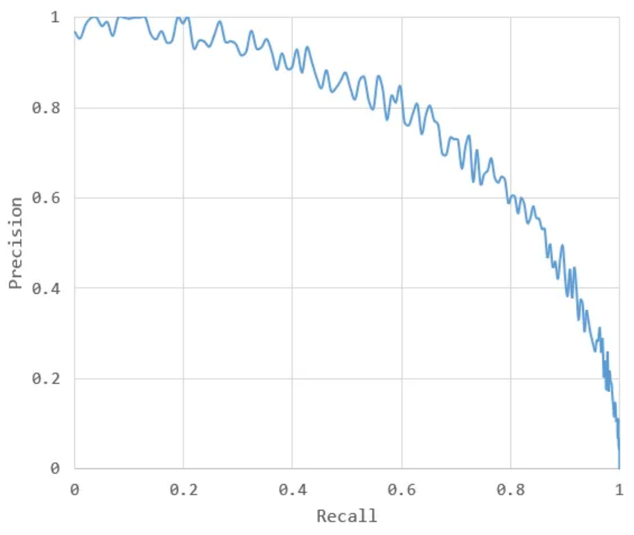


Figure 2.2: Example of Precision and Recall Curve. (Source: [9])

2.5 Neural Networks

This chapter delves into neural networks, comprehensively exploring their history, biological foundations, and artificial implementations. The chapter is divided into several subchapters, each focusing on a key aspect of neural networks. It begins with an overview of the historical development of neural networks, tracing their evolution from early inspirations to their modern-day applications. Subsequently, the subchapter on biological neurons delves into the fundamental workings of neurons in the human brain, elucidating their role as the inspiration for artificial neural networks. The following section explores the concept of the artificial neuron, explaining its structure and functionality in the context of neural network models. Additionally, the subchapter on the Threshold Logic Unit delves into this specific type of artificial neuron, highlighting its role in early neural network models. It further discusses the activation function, loss function, and gradient descent, which are vital components for training neural networks. Finally, the chapter delves into the Multilayer Perceptron (MLP) and backpropagation; these are crucial components of modern neural networks. By dissecting these subtopics, this chapter provides a comprehensive understanding of the historical foundations, biological inspiration, and key components that form the building blocks of neural networks.

2.5.1 History

Neural networks, also called ANN (Artificial Neural Networks), were introduced in 1943 by neurophysiologist Warren McCulloch and mathematician Walter Pitts [10]. The Idea behind ANNs is to transform how biological neurons in animal brains might work into a simplified computational model. After the introduction of ANNs, there was a strong belief that the technology would lead to intelligent machines. However, in the 1960s, this belief faltered, and ANNs disappeared from the scene. In the 1980s, ANNs reappeared with new architectures, but the interest in ANNs stopped again in the 1990s. This was due to the development of powerful machine learning technologies, such as support vector machines. Nowadays, the interest in ANNs is again increasing rapidly for the following reasons. A vast quantity of data has been collected over the last few years, which can be used to train neural networks for complex problems. The computing power since 1990 has increased significantly, as has the performance of GPU cards, which can be used for training. As a result of these advancements, training large neural networks within a reasonable time frame has become achievable. Over time, the algorithms used for training have constantly improved by minor changes, which had a tremendous positive impact. Lastly, the notable success of recent projects using ANNs has generated significant attention and funding, which increases the progress for further advancements [7, p. 279-281].

2.5.2 Biological Neuron

The functionality of artificial neurons is based on biological neurons of animal brains, as shown in Figure 2.3. Therefore, their functionality is briefly explained. A biological neuron consists of a cell body containing the nucleus and most complex components of the cell, many dendrites, the branching extensions, and the axon, the longest extension. At the end of the axon, it splits into multiple branches called telodendria, and these branches end in minuscule structures called synaptic terminals, also called synapses. The synapses are connected to the dendrites of other cell bodies. Signals between the cell bodies are transmitted over the axons by impulses called action potentials (APs). When a neuron receives a signal, it releases its electrical impulse. One individual neuron seems simple, but the brain connects billions into a complex network where one neuron is connected with thousands of other neurons. Due to this complex network, highly complex computations can be performed [7, p. 281-283].

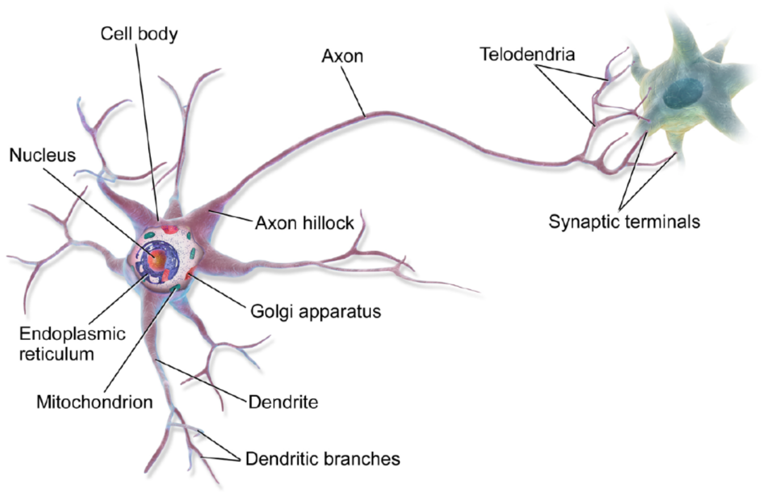


Figure 2.3: Biological Neuron. (Source: [7])

2.5.3 Artificial Neurons

An illustrative example is provided to improve the understanding of how artificial neurons function by demonstrating networks that compute logical operations. The artificial neurons in this example have one or many binary inputs and one binary output. The output is activated if a certain number of the inputs is active. Connecting these simple units of neurons makes it possible to construct networks that can compute any logical operation. Four small networks of artificial neurons, depicted in Figure 2.4, can compute any logical proposition. The neurons in this example are activated when at least two of their input are active. In the first network, the identity function is implemented, whereby the activation of neuron A leads to the activation of neuron C. The second network represents a logical AND operation, where the activation of both neurons A and B results in the activation of neuron C. The third network embodies a logical OR operation, with the activation of either neuron A or neuron B leading to the activation of neuron C. Lastly, the fourth network functions as a logical NOT operation, where the activation of neuron A and the deactivation of neuron B cause the activation of neuron C. These basic networks can create more complex networks that compute complex logical expressions [7, p. 283-284].

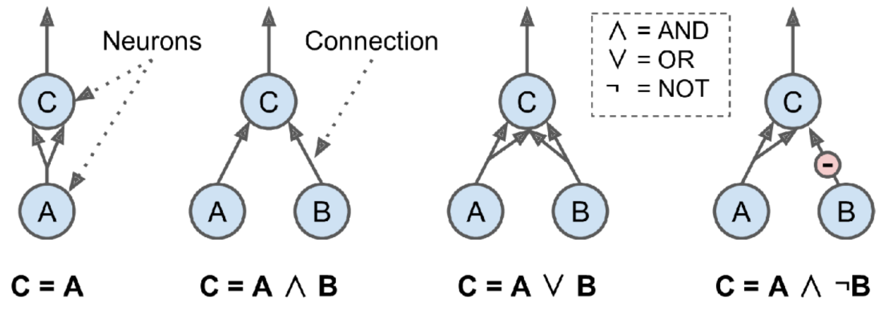


Figure 2.4: Logical computations through Artificial Neural Networks. (Source: [7])

2.5.4 Threshold Logic Unit

Threshold logic units (TLUs) seen in Figure 2.5 are another form of artificial neurons used in ANN Architectures. In contrast to artificial neurons, the inputs and outputs of TLUs are numbers, and each input includes a weight. They evaluate the sum of inputs, including their corresponding weights, and subsequently generate an output by applying a step function. A single TLU can perform a simple binary classification by computing a linear combination of inputs, generating positive or negative output based on a specific threshold [7, p. 284-285].

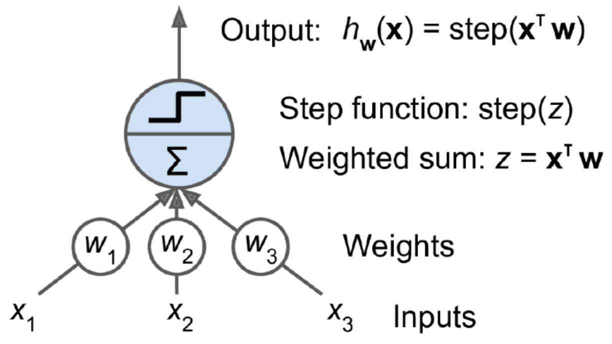

Output: $h_{\mathbf{w}}(\mathbf{x}) = \text{step}(\mathbf{x}^T \mathbf{w})$
Step function: step(z)
Weighted sum: $z = \mathbf{x}^T \mathbf{w}$
Σ
w_1
w_2
w_3
Weights
x_1
x_2
x_3
Inputs

Figure 2.5: Threshold Logic Unit. (Source: [7])

2.5.5 Activation Function

An activation function is a mathematical function applied to the inputs of a neuron, determining its output. It plays a vital role in neural networks by introducing non-linearity, enabling the modeling of complex data patterns. Selecting the appropriate activation function is critical when designing neural networks, as different functions are suited for other target variables. Figure 2.6 shows pictorial representations of various activation functions. The most basic function is the identity function, which provides no non-linearity. This function is mainly used when the output's target value is real. During the early stages of neural network development, the sign, sigmoid, and hyperbolic tangent (Tanh) functions were commonly employed as the traditional activation functions.

The sign activation function assigns a negative one for values below zero and one for values above zero, making it useful for binary outputs. The sigmoid activation function produces values between zero and one, allowing them to be interpreted as probabilities. The tanh activation function is similar to sigmoid but extends to negative values between minus one and one. However, the functions rectified linear unit (ReLU) and hard tanh have largely replaced sigmoid and tanh activation functions over time. These replacements offer computational efficiency, making them more suitable for training large-scale neural networks [11, p. 31-33]. Due to the ongoing advancements in this field, new activation functions are regularly being introduced. In this case study, the YOLO algorithm is employed, which will be described at the end of this chapter. The algorithm's standard implementation incorporates an updated version of the ReLU, the leaky ReLU, as the activation function. Leaky ReLU introduces a slight slope for negative values instead of being limited to zero, as illustrated in Figure 2.7. This updated function addresses a limitation of the ReLU and prevents the neurons from saturating [11, p. 153-154].

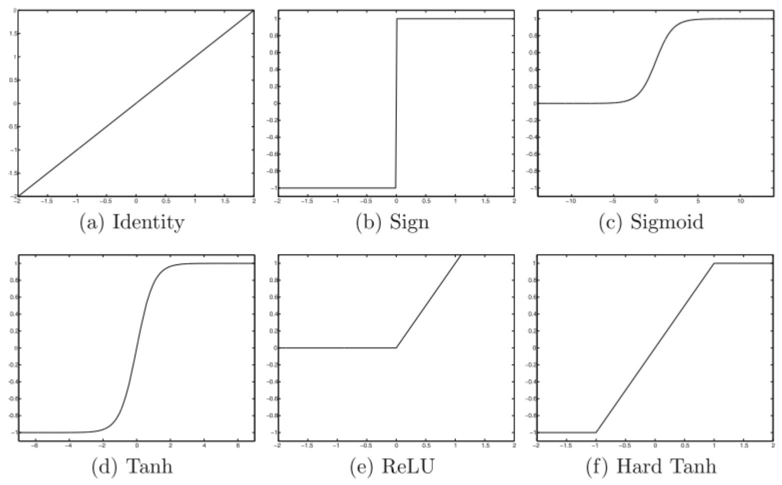


Figure 2.6: Various Activation Functions. (Source: [11])

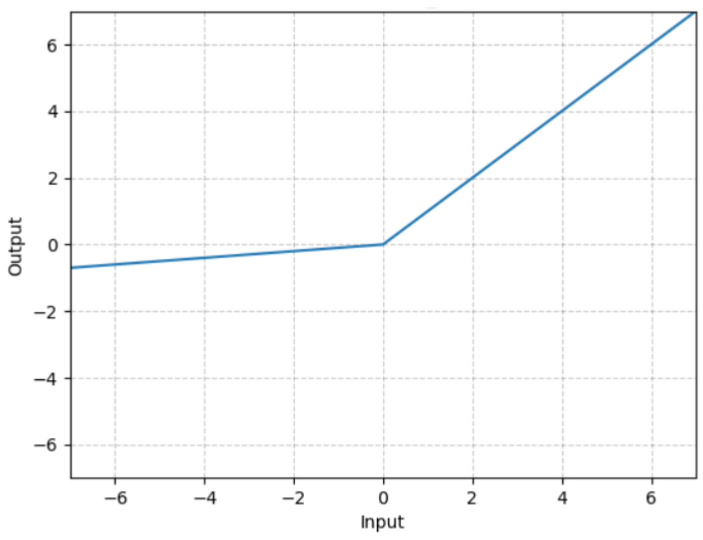


Figure 2.7: Leaky ReLU. (Source: [12])

2.5.6 Loss Function

The loss function, also called the cost function, is a mathematical tool to quantify the discrepancy between the algorithm's current output and the desired output. It plays a crucial role in evaluating the algorithm's data modeling capabilities. The loss function can be classified into two groups based on the nature of the task. The first group contains the loss functions for classification tasks involving discrete values. The second group includes loss functions for regression tasks involving continuous values. These categories allow for the appropriate selection of the loss function that aligns with the specific modeling objectives and the characteristics of the data being analyzed. Cross-entropy is a commonly used loss function for classification tasks, which uses the negative logarithm to amplify the difference between the predicted and true probability. When the prediction gets closer to the true value, the loss decreases. The mean square error (MSE) is a commonly used loss function for regression tasks. The MSE calculates the difference between predicted and actual values and squares the results. Then it calculates the mean of all squared differences to provide an overall measure of the model's performance [13]. For this case study, the YOLOv5 implementation by Jocher [14] is employed, which is described in more detail later in this chapter. This implementation utilizes the cross-entropy loss function to calculate the classification loss.

2.5.7 Gradient Descent

Gradient Descent is a generic optimization algorithm that aims to find the best solutions for various problems. Its core principle involves iteratively adjusting parameters to minimize a loss function. To illustrate this concept, Geron [7] provides a relatable analogy where someone finds themselves lost in a dense fog on mountainous terrain, relying solely on sensing the slope beneath their feet. In such a situation, the most effective strategy to reach the valley quickly would be to descend along the steepest slope. This parallels the behavior of Gradient Descent, which calculates the local gradient of the error function concerning the parameter vector and moves in the direction of the descending gradient. The size of the steps during the optimization process is determined by a hyper-parameter called the "learning rate." The process continues until the gradient reaches zero, indicating the minimum has been reached. Initially, the parameter values are randomly initialized, then gradually refined through incremental adjustments to decrease the loss function. This iterative refinement continues until the algorithm converges to a minimum [7, p. 118-121]. Figure 2.8 visually represents the optimization process.

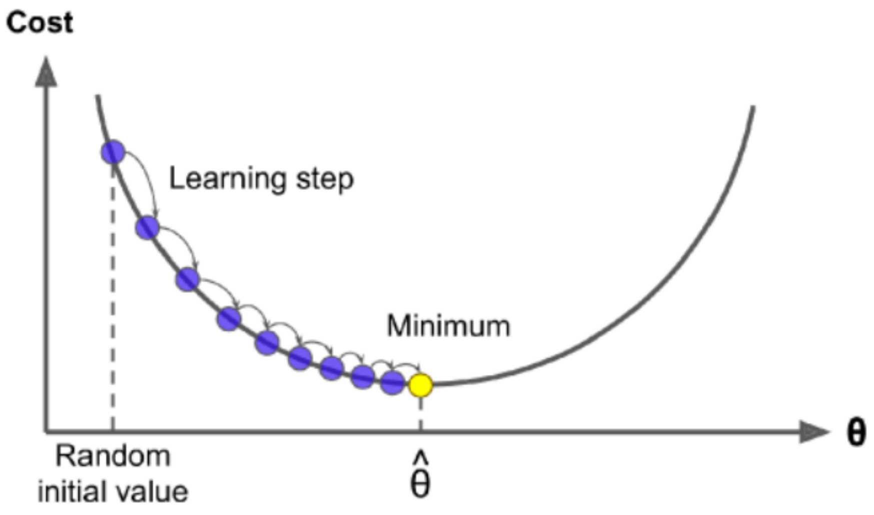


Figure 2.8: Depiction of Gradient Descent. (Source: [7])

2.5.8 Multilayer Perceptron and Backpropagation

A Multilayer Perceptron (MLP), as seen in Figure 2.9, is an ANN architecture that consists of multiple layers. It consists of one input layer containing input neurons and a bias neuron, one or many hidden layers containing TLUs and a bias neuron, and one output layer composed of TLUs. The layers close to the input are often referred to as lower layers, and the ones close to the output as upper layers.

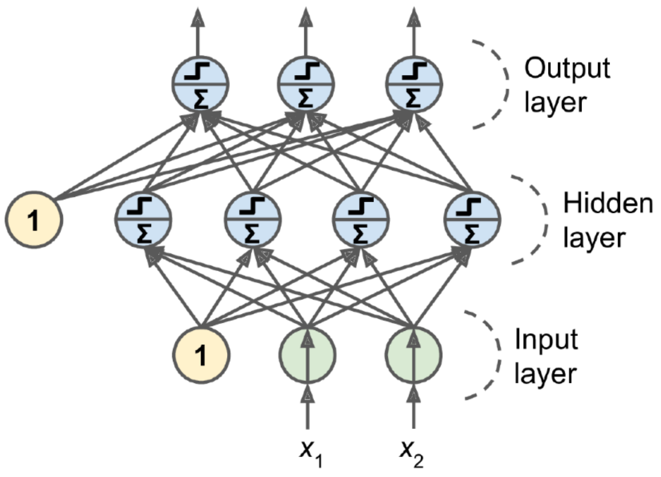


Figure 2.9: Multilayer Perceptron. (Source: [7])

To train an MLP, the backpropagation training algorithm is used, published in 1986 by David Rumelhart, Geoffrey Hinton, and Ronald Williams. The main concept of the algorithm is that it computes the gradients passing the network two times (forward and backward) and optimizes each weight and bias to reduce the error.

The following is a brief description of how the algorithm works. In the beginning, all weights in the hidden layers are randomly initialized. Then the network is passed forward by a certain number of instances called mini-batch multiple times, where every neuron computes the output for each instance. All the intermediate results are stored, and a loss function calculates the network's output error. A loss function is required to evaluate the error by comparing the network's actual output with the desired output. Subsequently, the network is passed backward, computing the influence on the error of each output and the error gradient across all connections. Finally, a Gradient Descent is performed on the error gradient to optimize each weight in the network. This passing is repeated multiple times, and a single passing is called an epoch. Additionally, the step function inside the TLUs is replaced with an activation function, most commonly the logistic (sigmoid) function. This is a significant change as the step function contains flat segments on which the Gradient Descent can not move, as the logistic function has a non-zero derivative throughout the gradient, which allows the Gradient Descent to make some progress at every step [7, p. 289-291].

2.6 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are designed to handle sequential data, where the order of the data matters. They work by saving the output of a layer and feeding it back to the input to predict the output of the layer. Figure 2.10 illustrates a Recurrent Neural Network (RNN), presented on the left side, and the same RNN displayed as an unfolded computational graph on the right side. The input values are denoted by "x" and the output by "o." The "L" value represents the loss function that evaluates the difference between the output and the corresponding target value "y." The weights are represented by "W." Common applications of RNNs are language modeling, speech recognition, and machine translation.[3, p.378-381].

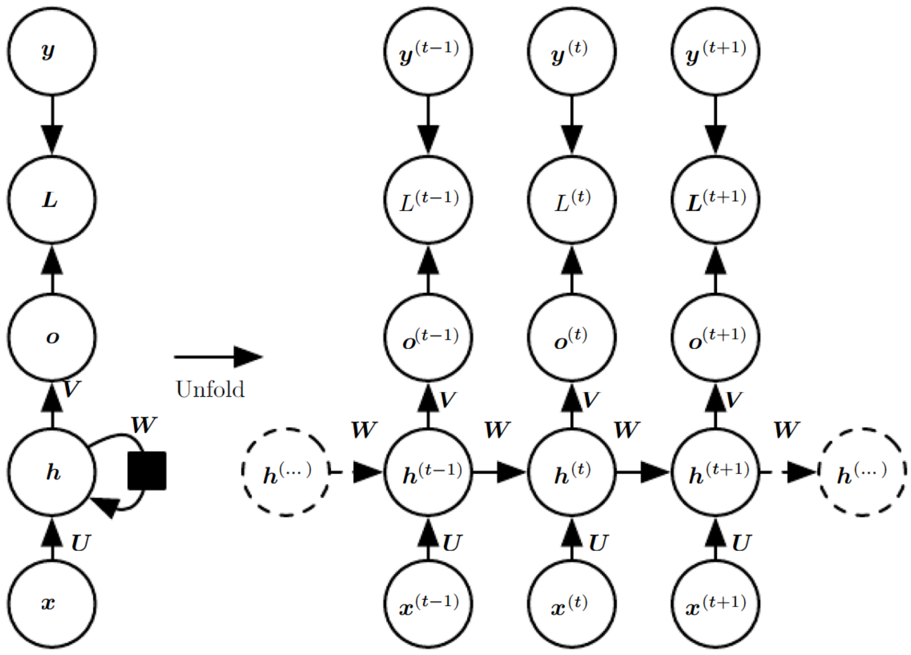


Figure 2.10: Recurrent Neural Network Architecture. (Source: [3])

2.7 Convolutional Neural Networks

Convolutional and pooling layers are included to transition from a neural network to a convolutional neural network (CNN). CNNs are capable of processing data with a grid-like topology. For example, time-series data can be interpreted as a one-dimensional grid, and image data can be interpreted as a two-dimensional grid of pixels. The network's name is based on the convolution operation performed, which is a mathematical operation.

CNNs play a crucial role in Computer Vision (CV). CV is a field of study and technology that focuses on enabling computers to extract and understand visual information from digital images or videos, mimicking human visual perception. By employing advanced algorithms and machine learning techniques, computer vision systems can analyze and interpret visual data, recognize objects, detect patterns, and even understand complex scenes. In the last two decades, the development of computer vision research has been rapid. At the same time, mathematical techniques to recover three-dimensional shapes and appearances in images have also developed. While significant progress has been made, current technologies still need to be able to describe a photograph with the same level of detail as a human. However, computer vision continues to find widespread applications in various real-world scenarios [15, p. 3-10].

The main components of the standard architecture of a CNN, seen in figure 2.11, are convolutional layers, pooling layers, and a fully connected layer, and by stacking convolutional and pooling layers, the network gets deeper [3, p. 330-331].

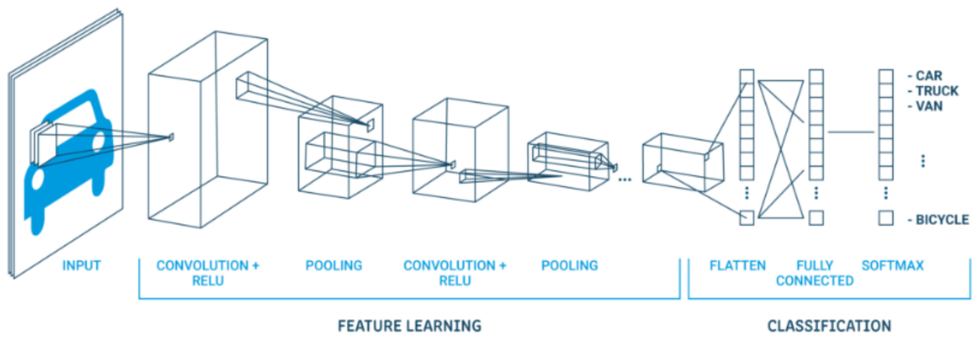


Figure 2.11: Standard Architecture of a CNN. (Source: [16])

2.7.1 Convolutional Layer

The convolutional layer performs a convolution, a mathematical operation, on the input. This operation uses a filter, which is also known as a kernel. A common approach used to preserve the spatial dimensions of the input volume is padding. When padding is applied, additional border pixels are added around the input before the convolution operation is performed, ensuring that the output feature map is the same size as the input. The kernel has a defined width and height and is initialized with values. A multi-dimensional filter called kernel map is applied if the input has more than one dimension, such as an RGB input with three dimensions. The filter is applied to the input grid starting from the top left corner, and it calculates the dot product between the filter and the corresponding input cells. Then the result is stored in a new two-dimensional grid called a feature map. Subsequently, an activation function is applied to ensure non-linearity. The most common activation function is the rectified linear unit, also called ReLU, which will be further described later in this chapter. As illustrated in Figure 2.12, this process is repeated by shifting the filter by one cell and performing the dot product calculation until the entire input grid is covered and the feature map is complete. The process occurs within the first and third cubes labeled "convolution + ReLU" in Figure 2.11 and is applied multiple times with different kernels or kernel maps. Notably, the depth of the outputs from the convolutional layers increases because various filters are used, and the resulting feature maps are then concatenated to form a three-dimensional output [3, p. 331-334].

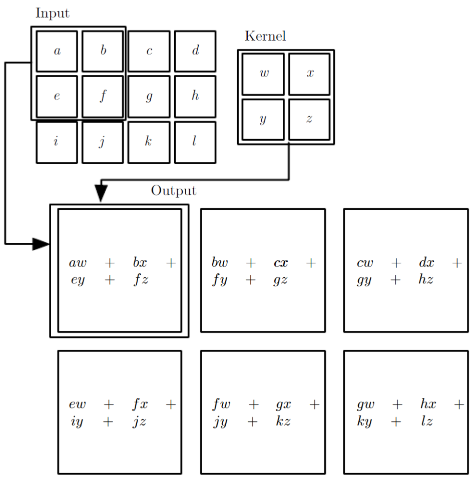


Figure 2.12: 2-D Convolution. (Source: [3])

2.7.2 Pooling Layer

The pooling layer of a convolutional neural network (CNN) modifies the feature map created in the previous convolutional layer by reducing its spatial dimensions. This operation aims to extract the essential information from the feature map while discarding some of the less important details. This process is depicted in Figure 2.11, through the second and fourth cubes labeled as "pooling." It helps to prevent overfitting, as the number of parameters in the network is reduced, and to introduce some translation invariance. Translation invariance means that the network will recognize the same feature regardless of where it is located in the input image. A potential downside of the layer is that it can cause a loss of information, particularly in cases where the window is large.

To reduce the spatial dimension, small rectangular regions, also called windows, of the feature map are taken, reduced to one value, and stored for the output. Standard sizes for the window are 2x2 or 3x3, and they are moved along the feature map with a defined step size called "stride." Two common types of pooling are max pooling and average pooling. In max pooling, illustrated in Figure 2.13, the maximum value within the window is selected as the output value, while in average pooling, the average value is used [3, p. 339-345].

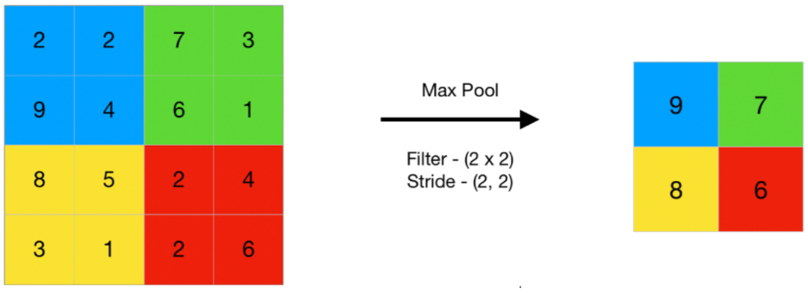


Figure 2.13: Illustration of Max Pooling. (Source: [17])

2.7.3 Fully Connected Layer

The previous layer's output is flattened to a single vector in the fully connected layer, called the dense layer. The single vector is then fully connected to a hidden layer to increase the attributes for the prediction. The hidden layer is then connected to neurons, each representing a class to be predicted. A probability distribution is calculated to predict the likelihood of an input belonging to a particular class by applying a softmax function to the output [18]. This process is illustrated in the right section of Figure 2.11.

2.8 YOLO

Joseph Redmon introduced YOLO, an object detection architecture with exceptional speed and accuracy, in a 2015 paper. YOLO stands for "you only look once," as it performs object detection of various objects by predicting the objects in a single algorithm run. Over the years, the algorithm has improved, resulting in multiple versions [7, p. 489]. The most recent version of the algorithm is YOLOv8. Due to its speed, it can detect objects in real-time on videos.

2.8.1 Unified Detection

In YOLO, object detection components are combined into a single neural network. By utilizing the entire image's features, the network predicts bounding boxes and simultaneously indicates them for all classes in the image. This global reasoning allows the network to consider the entire image and its objects. YOLO's design, seen in Figure 2.14, enables end-to-end training, real-time processing, and high average precision. The system divides the input image into a square grid with a specified size, where each grid cell is responsible for detecting an object if its center falls into that cell. Each grid cell predicts bounding boxes and confidence scores. These confidence scores indicate the model's certainty about an object's presence and its predicted box's accuracy. The confidence score is defined by multiplying the probability for an object with the IOU (Intersection Over Union) truth prediction, with a value of zero if no object exists in the cell [19].

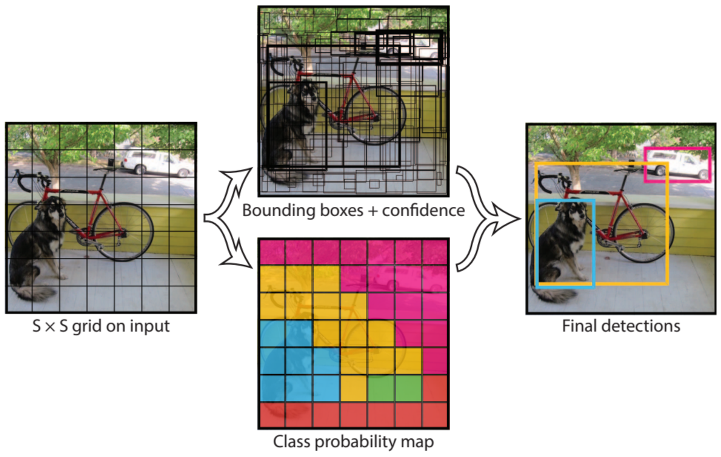
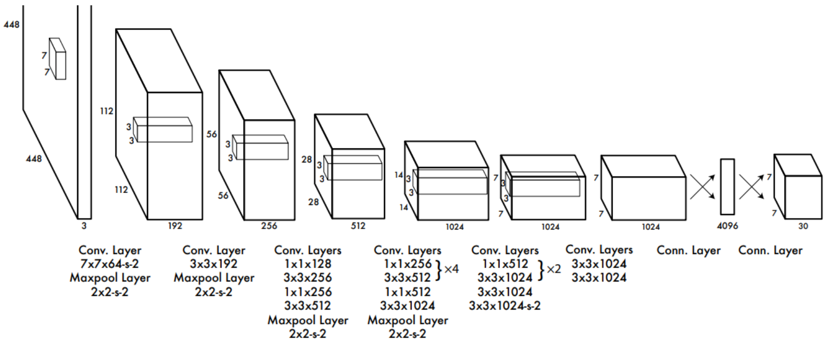


Figure 2.14: YOLO Model. (Source: [19])

For each bounding box, YOLO provides five predictions: x, y, w, h, and confidence. The (x, y) coordinates define the box's center relative to the grid cell bounds, while the width and height are predicted relative to the entire image. The confidence prediction represents the IOU between the predicted box and any ground truth box. In addition, each grid cell predicts conditional class probabilities, which are conditioned on the presence of an object within the cell. Despite the number of boxes, only one set of class probabilities is predicted per grid cell. During testing, the algorithm computes class-specific confidence scores for each box by multiplying the conditional class probabilities, individual box confidence predictions, and the IOU truth prediction. These scores reflect the likelihood of a specific class appearing in the box and the accuracy of the predicted box's fit to the object [19].

2.8.2 Network Design

For the network design, a convolutional neural network is utilized. The initial convolutional layers of the network extract features, and the fully connected layers predict the output's probabilities and coordinates. The network consists of 24 convolutional layers and two fully connected layers inspired by the GoogLeNet image classification model. GoogLeNet is a neural network architecture introduced in the paper "Going deeper with convolutions" [20], by Google in 2014. A fundamental building block of the architecture is the inception module. The inception module uses parallel convolutional layers of different filter sizes (1x1,3x3,5x5) to capture features at different spatial scales. The outputs of the inception model are concatenated and fed into the subsequent layer. Instead of using inception modules like GoogLeNet, YOLO employs 1x1 reduction layers, followed by 3x3 convolutional layers [19]. The network architecture is illustrated in Figure 2.15.



448
448
7
7
3
112
112
3
3
192
56
56
3
3
256
28
28
3
3
512
14
14
3
3
1024
7
7
3
3
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 2.15: YOLO Architecture. (Source: [19])

Additionally, a faster version of YOLO (You Only Look Once) for object detection aims to achieve faster processing speeds. This version utilizes a neural network with nine convolutional layers, reducing the number of filters in those layers compared to the original YOLO. The training and testing parameters remain the same for YOLO and Fast YOLO. The network output is a 7x7x30 tensor containing predictions for object probabilities and coordinates [19]. According to Goodfellow [3], a tensor is defined as an array of numbers arranged on a regular grid with a variable number of axes.

2.9 Evolutionary Progression of YOLO

In this case study, the fifth version of YOLO was implemented utilizing the code provided by Jocher [14]. Therefore, this chapter gives an overview of the algorithm's improvements in each iteration up to the fifth version.

2.9.1 YOLOv2

YOLOv2, also called YOLO9000 due to the reason that it could detect 9000 categories, was published in the paper "YOLO9000: Better, Faster, Stronger" [21] by Joseph Redmon and Ali Farhadi in the year 2016. The second version came with various improvements to improve the algorithm, keep the speed, and make it stronger. The improvements were the following:

- **Batch Normalization -** This was performed on all convolutional layers and reduced overfitting.

- **High-resolution Classifier -** In YOLOv1, the model was pre-trained with ImageNet [22], a large database with more than 14 million images at a resolution of 224 x 224. The improvement was to finetune the model over ten epochs with a resolution of 448 x 448, which improved the performance on higher-resolution inputs.

- **Fully Convolutional -** The architecture shifted from dense layers to a fully convolutional architecture.

- **Anchor Boxes -** These are predefined shapes that match prototypical object shapes and are used in object detection. Each grid cell has multiple anchor boxes, and the system predicts the coordinates and class for each anchor box, with the network output size being proportional to the number of anchor boxes per grid cell.

- **Dimension Clusters -** The authors improved bounding box prediction accuracy by utilizing k-means clustering on training bounding boxes to select five prior boxes, balancing recall and model complexity.
- **Finner-grained Features -** Compared to YOLOv1, one pooling layer was removed to obtain a feature map of 13 x 13 for the input image with a resolution of 416 x 416. Additionally, a passthrough layer that stacks adjacent features into different channels is utilized with finner-grained features.
- **Multi-scale training -** No fully connected layers are used. Therefore, different input sizes could be used, and the model could be trained more robustly using different input sizes.

These improvements increased the average precision (AP) by 15.2 percent on the Pascal VOC2007 dataset [23].

2.9.2 YOLOv3

YOLOv3 was published in the paper "YOLOv3: An Incremental Improvement" [24] by Joseph Redmon and Ali Farhadi in the year 2018. The third version included a bigger architecture while keeping the real-time performance. The changes compared to YOLOv2 are described in the following:

- **Bounding Box Prediction -** The bounding box prediction was improved by predicting an objectness score for each bounding box via logistic regression. This notes the anchor box with the highest overlap to the ground truth as one and the other anchor boxes as zero. This score assigns only one bounding box to each ground truth, resulting in a faster network.
- **Class Prediction -** For the classification, binary cross-entropy was used, allowing the assignment of multiple labels to one box. Therefore, the same object can be classified with different labels.
- **New Backbone -** The feature extractor's size was increased to 53 convolutional layers, resulting in higher accuracy.
- **Spatial Pyramid Pooling (SPP) -** In addition to the new backbone, a modified SPP block was added to the architecture. This block concatenates multiple max pooling outputs with different kernel sizes (1,5,9 and 13), allowing each neuron to refer to a larger effective area of the input image.

The benchmark dataset was changed from Pascal VOC2007 to the MS COCO dataset [25], resulting in an average precision of 50 percent (AP50) of 60.6 percent and an AP of 36.2 percent.

2.9.3 YOLOv4

YOLOv4 was published in the paper "YOLOv4: Optimal Speed and Accuracy of Object Detection" [26] by Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao in the year 2020. Although the implementation was published by authors other than the previous ones, the philosophy of YOLO was maintained. As this paper provided satisfactory improvements, it was accepted by the community as the official fourth version of the algorithm. The improvements were the following:

- **Enhanced Architecture with Bag-of-Specials (Bos) Integration -** After experiments with different backbone architectures, the best-performing one was a modified Darknet-53, which is a combination of the network used by YOLOv2 and Darknet-19 [27] by Redmon with cross-stage partial connections (CSPNet) [28] and Mish activation function [29]. The neck incorporated modified versions of spatial pyramid pooling (SPP) from YOLOv3-spp, path aggregation network (PANet) [30], and spatial attention module (SAM) [31], while the detection head used anchors similar to YOLOv3, resulting in the model named CSPDarknet53-PANet-SPP. These modifications helped reduce computation while maintaining accuracy, and the SPP block increased the effective area of the input image without impacting inference speed.

- **Bag-of-Freebies (BoF) Integration -** In addition to the regular data augmentations such as contrast, scaling, flipping, and rotating, mosaic augmentation was implemented. This augmentation combines four images into a single one. This allows the detection of objects in a different context and reduces the need for large batch sizes.

- **Hyperparameter Optimization with Genetic Algorithms -** This feature enables users to find optimized hyperparameters using a genetic algorithm over multiple evolutions. Therefore, the effort to develop a high-performing model is reduced.

These improvements increased the AP50 on the MS COCO dataset by 5.1 percent and the AP by 7.3 percent.

2.9.4 YOLOv5

YOLOv5 [14] was released in 2020 by the founder of Ultralytics, Glen Jocher. In contrast to the previous version implemented with darknet, it was developed with PyTorch [32], an optimized tensor library. Besides the earlier versions, Jocher did not publish a paper accompanying his release of YOLOv5. Still, the community mainly accepted the algorithm, which has been forked over 14 thousand times.

The main improvement achieved was implementing PyTorch and using a Ultralytics algorithm called AutoAnchor. This algorithm checks and adjusts anchor boxes to the dataset and training settings. Then it applies a k-means function to the dataset's labels, initializing them for a Genetic Evolution (GE) algorithm. The GE algorithm then evaluates the best possible recall within a set number of generations. The evaluation uses a fitness function, where the desired metrics to be maximized can be defined.

The evaluation of the architecture resulted in an AP of 50.7 percent on the MS COCO dataset, which is an increase of 7.2 percent.

3 Implementation and Experimental Setting

This chapter describes the implementation and experimental setting of visual inspection tests. The process begins by selecting the YOLO version and the available hardware for model training. Subsequently, the Azure DevOps pipeline used for continuous integration into the team foundation server is explained. Moving forward, the open-source platform ClearML is introduced as the logging tool utilized. The available datasets are then described, followed by the training processes for detecting power transformers, sub-components, rust, and oil leaks. Finally, the merging of the results into an inspection report is detailed. This chapter provides comprehensive insight regarding the steps taken to conduct visual inspection tests and the tools and processes involved.

3.1 YOLOv5

The YOLO algorithm implementation chosen for deployment was version 5, provided by Jocher [14]. While benchmarks from Stereolabs [33] indicate that versions 7 and 8 have lower latency than version 5, as seen in 3.1, this criterion is not particularly relevant to evaluate the feasibility of visual inspection tests via object detection. This is because the detection is only performed on images, and the time of the complete inspection test is not a relevant criterion, as the focus is on evaluating the feasibility and not on providing a releasable product. Additionally, the benchmark illustrates that the newer versions demonstrate higher average precision. While precision is a crucial aspect for the case study, there is no significant reason for exclusively opting for the latest version, as this disparity tends to decrease when deploying large models. Choosing version five over the newest version offers the advantage of broader adoption and more comprehensive documentation.

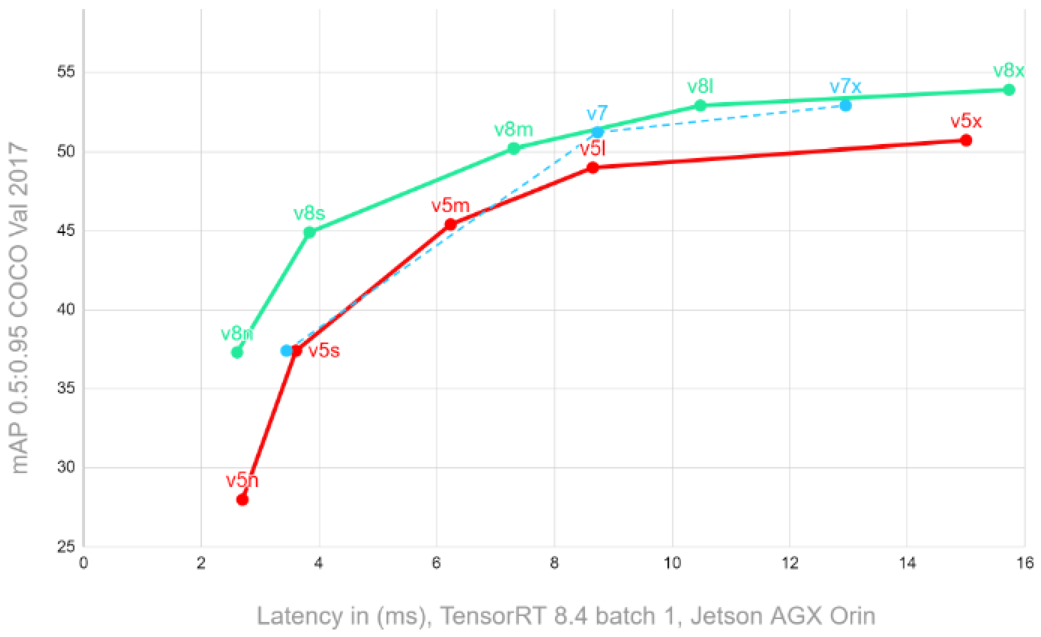


Figure 3.1: Performance of YOLO v5, v7 and v8. (Source: [33])

3.2 Hardware

The partner company provided a Linux-based workstation to train the models for this thesis. The workstation has an NVIDIA Quadro RTX4000 GPU that provides 8GB GDDR6 memory, a 256-bit memory bus, and a bandwidth of 416 GB/s [34]. This graphic card can utilize CUDA [35], a parallel computing platform and programming model developed by NVIDIA for their GPUs. CUDA enables faster training of machine learning models for tasks such as object detection by leveraging the power of the GPU's parallel processing capabilities.

3.3 Azure DevOps CI Pipeline

The code base was maintained on a TFS (Team Foundation Server) by Azure DevOps to adapt to the workflow of the partner company. Additionally, this allowed viewing the code base of previous experiments and rollback if necessary. To improve the workflow, a continuous integration (CI) pipeline [36] was established, which triggers when code changes are pushed to the server. The pipeline executes the code on the remote Linux Workstation, which functions as a self-hosted agent [37]. The pipeline first deletes the old docker image to save storage on the agent. Once the deletion process is completed, a fresh docker image is constructed, and then a docker container is launched using the updated image. The required docker file was added to the YOLOv5 implementation provided by Jocher [14]. The docker file contains all requirements to build the docker image and the command to run the training script with specified parameters. After the docker container is activated, the specified command is executed, and the training is run on the agent.

3.4 Logging

ClearML [38], an open-source platform, was utilized to log and monitor the outcomes of each training session. The platform encompasses the capabilities of managing, tracking, and visualizing machine learning experiments. It provides a centralized hub for experiment management that enables users to efficiently log and monitor various aspects of their machine-learning workflow, including console outputs, hyperparameters, and results. As the platform automatically logs these aspects and visualizes them in several diagrams, it simplifies the process of reproducing experiments and analysis.

The YOLOv5 implementation, created by Jocher [14], comes pre-integrated with ClearML. To make use of the platform, an account is required. The free version of ClearML allows up to three users and provides 100GB of storage for artifacts, which is ample for this case study. Users can generate credentials within the account and incorporate them into the training machine. Once the connection is established, the web application can conveniently access the training results.

3.5 Datasets

The implementation focused on utilizing three datasets. The partner company internally collected and labeled the data for power transformer detection using the "labelimg" tool [39]. The labels generated were saved in text files, which were required for the YOLO implementation. However, generic open-source datasets were utilized for rust and oil leak detection. This approach was primarily chosen due to time limitations within the project, as gathering and labeling data specific to each case would have exceeded the project's time frame.

The datasets used initially and their corresponding training-validation ratios are presented in Table 3.1. For rust detection, the open-source dataset [40] was utilized, while dataset [41] was employed for oil leak detection. After conducting initial experiments on these datasets, it was concluded that additional data is required to achieve satisfactory performance.

Amount of Data			
Dataset	Training	Validation	Ratio
Power Transformer	126	41	75%- 25%
Rust Detection	838	120	85%- 15%
Oil Leak Detection	124	16	87%- 13%

Table 3.1: Initial Datasets

Consequently, the dataset for power transformer detection was internally expanded by collecting additional data and labeling it accordingly. Multiple generic open-source datasets were collected and merged to train the rust and oil leak detection models. For rust detection, the datasets [40], [42]–[44] were utilized, while for oil leak detection, the datasets [41], [45]–[48] were employed. An overview of the extended datasets, including the total data and the split between training and validation data, can be found in Table 3.2. The training-validation data ratio for rust and oil leak detection results from combining these open-source datasets.

Amount of Data			
Dataset	Training	Validation	Ratio
Power Transformer	342	86	75%- 25%
Rust Detection	3350	796	76%- 24%
Oil Leak Detection	1938	246	87%- 13%

Table 3.2: Extended Datasets

3.6 Training Models

This sub-chapter delves into the steps undertaken in the training process for detecting power transformers and their sub-components, rust, and oil leaks. It provides an insight into the methodology employed for training models tailored to each specific task. The process encompasses crucial stages such as model selection, occurred difficulties, data augmentations, and hyperparameter tuning.

3.6.1 Detection of Power Transformer and Sub-components

The primary focus of the initial model training was the detection of power transformers and their sub-components, as they hold significant importance in inspection tests. The model training process involved training a single model encompassing both power transformers and their sub-components. First, a model was trained using the initial dataset consisting of 167 images with a split ratio of 75%-25%, listed in Table 3.1. Different image sizes and batch sizes were employed during the training process. For the initial experiments, the chosen model was "YOLOv5s," the smaller version offering faster training than larger models. Concurrently, the dataset was expanded to enhance the model's performance. Table 3.2 presents the final dataset of 428 images with a split ratio of 75%-25%.

Due to the GPU's memory limitations, the flexibility to vary image and batch sizes was restricted. Consequently, the maximum image size was set to 640, and the maximum batch size was limited to five. An option suggested by Jocher [14] is to use "auto" as the batch size, which automatically selects the maximum feasible batch size based on the available GPU memory. The outcomes of the initial experiments are summarized in Table 3.3, providing key details such as batch size, which refers to the number of training examples processed together in a single iteration and impacts training efficiency and resource utilization.

Additionally, the table includes information about image size, denoting the dimensions of the input images used for training or testing, which influences the level of detail captured by the model. The table also presents the mean Average Precision (mAP) scores at different IoU (Intersection over Union) thresholds, which measure the average precision of object detection across multiple IoU thresholds, offering an overall assessment of the model's accuracy. Furthermore, the table includes precision, quantifying the proportion of correctly identified positive predictions out of the total predicted positives, indicating the model's ability to avoid false positives. Finally, the table provides the recall metric, representing the proportion of correctly identified positive predictions out of the total actual positives, indicating the model's ability to capture all relevant instances.

This table provides a concise overview of the model's performance and ability to detect and classify objects accurately. The highest-performing configuration, which achieved the best result, was observed with an image size of 640 and an auto batch size. This configuration demonstrated an mAP of 0.397 across IoU thresholds ranging from 50 to 95 percent, a separate mAP of 0.733 at an IoU threshold of 50 percent, a precision of 0.74, and a recall of 0.64. This particular configuration is highlighted in green.

First experiments with Model YOLOv5s					
Batch Size	Image Size	mAP@0.5:0.95	mAP@0.5	Precision	Recall
32	640	0.393	0.71	0.7	0.59
16	640	0.396	0.713	0.69	0.62
8	640	0.4	0.73	0.72	0.6
4	640	0.384	0.72	0.73	0.63
auto	640	0.397	0.733	0.74	0.64
4	1280	0.364	0.68	0.72	0.62
8	1280	-	-	-	-
16	1280	-	-	-	-
32	1280	-	-	-	-
auto	1280	0.342	0.66	0.71	0.58

Table 3.3: First experiments utilizing Model YOLOv5s. ("-" marks that the experiment required more memory than available)

Regarding the first evaluations, the configuration with an image size of 640, auto batch size, and the model "YOLOv5s" was used for further experiments. A data augmentation pipeline was implemented using Albumentations [49] for these experiments. Several experiments were conducted to determine the pipeline's optimal configuration. Therefore, the best-resulting setup for the corresponding percentages for each step and the complete pipeline was evaluated. The improvements of the single augmentation steps and the entire pipeline with different probabilities are shown in Table 3.4. The best result, highlighted in green, was attained by applying the whole data augmentation pipeline with a likelihood of 20% for each step. It resulted in an mAP of 0.427 across IoU thresholds ranging from 50 to 95 percent, a separate mAP of 0.755 at an IoU threshold of 50 percent, a precision of 0.8, and a recall of 0.75.

Experiments Data Augmentation				
Augmentation	mAP@0.5:0.95	mAP@0.5	Precision	Recall
No Augmentation	0.397	0.733	0.74	0.64
Horizontal Flip 50%	0.418	0.744	0.77	0.66
Vertical Flip 50%	0.417	0.743	0.77	0.66
Conditions 50%	0.423	0.746	0.76	0.64
Color 50%	0.417	0.753	0.78	0.67
Quality 50%	0.422	0.751	0.78	0.67
Crop 50%	0.423	0.752	0.79	0.7
Every step 20%	0.427	0.755	0.8	0.75
Every step 30%	0.417	0.724	0.78	0.73
Every step 40%	0.423	0.728	0.75	0.71
Every step 50%	0.422	0.719	0.74	0.71

Table 3.4: Power Transformer Detection Experiments - Data Augmentation

The following description outlines the final pipeline utilized after conducting the experiment.

- Random Crop - 20%
- Conditions - 20% (Used to make the model more robust for photographs taken with different weather conditions. It contains five augmentations and applies one of them: random snow, random fog, random rain, random sun flare, and random shadow.)
- Colors - 20% (Used to reduce the impact of the color. It contains three augmentations and applies one of them; the augmentations are to grayscale - 60%, channel shuffle - 30%, and invert - 10%.)
- Quality - 20% (Used to increase the variety of the data and simulate bad photograph conditions. It contains three augmentations and applies one: Gaussian noise, random brightness, contrast, and random gamma.)
- Vertical Flip - 20%
- Horizontal Flip - 20%

After evaluating the data augmentation pipeline setup, experiments utilizing different network models were conducted to evaluate the best-performing model size. The experiments showed that the performance increases with the increase of the models' size. Therefore, the largest model, "YOLOv5x" achieved the best results, with the batch size set to "auto" and an image size of 640. It resulted in an mAP of 0.47 across IoU thresholds ranging from 50 to 95 percent, a separate mAP of 0.80 at an IoU threshold of 50 percent, a precision of 0.86, and a recall of 0.76. The outcome of the experiments is shown in Table 3.5.

Experiments Model Size				
Model	mAP@0.5:0.95	mAP@0.5	Precision	Recall
YOLOv5n	0.415	0.73	0.81	0.69
YOLOv5s	0.427	0.755	0.80	0.75
YOLOv5m	0.457	0.77	0.85	0.77
YOLOv5l	0.47	0.79	0.86	0.75
YOLOv5x	0.47	0.80	0.86	0.76

Table 3.5: Power Transformer Detection Experiments - Model Size

Finally, a hyperparameter tuning was conducted using the most promising configuration obtained from the previous experiments to enhance the performance. The used YOLOv5 implementation by Jocher [14] contains a method to perform a hyperparameter evolution. This method uses a genetic algorithm, tries different hyperparameters over several evolutions, and returns the hyperparameters that led to the best result. A genetic algorithm is a heuristic search algorithm that utilizes random search along with historical data to direct the search into the region of better performance [50]. The procedure starts with the hyperparameters declared in the used YAML file for hyperparameters. The distribution to choose which value should be maximized can be adjusted in the fitness method. The metrics that can be weighted are precision, recall, mAP@0.5, and mAP@0.5:0.95. For the power transformer, the metric mAP@0.5:0.95 was weighted the highest to maximize the accuracy of detecting the sub-components. This metric calculates the average precision over all classes with an IOU (Intersection over Union) threshold of 50 to 95 percent. The default number of evolution iterations for the method is 300. However, in the case of power transformer detection, this number was raised to 500 to enhance the likelihood of discovering improved hyperparameters.

The final model for power transformer detection was trained over 1000 epochs using an image size of 640, an auto batch size, the above-mentioned data augmentation pipeline, and the resulting hyperparameters of the hyperparameter evolution. This final training resulted in an mAP of 0.49 across IoU thresholds ranging from 50 to 95 percent, a separate mAP of 0.84 at an IoU threshold of 50 percent, a precision of 0.89, and a recall of 0.8.

3.6.2 Rust Detection

An image size of 640 and an auto batch size were chosen for the initial experiments for training the model for rust detection. The first experiments were performed to evaluate the impact of the model size. The dataset, listed in Table 3.1, consisting of 359 images with a split ratio of 85%-15%, was used for these experiments. The results of the first experiments are shown in Table 3.6, with the best result highlighted in green. The best result was achieved by the model size "YOLOv5x" with an mAP of 0.307 across IoU thresholds ranging from 50 to 95 percent, a separate mAP of 0.634 at an IoU threshold of 50 percent, a precision of 0.7, and a recall of 0.615.

Rust Detection Experiments - Model Size - Initial Dataset				
Model	mAP@0.5:0.95	mAP@0.5	Precision	Recall
YOLOv5s	0.284	0.623	0.693	0.535
YOLOv5m	0.298	0.627	0.7	0.593
YOLOv5l	0.288	0.646	0.703	0.6
YOLOv5x	0.307	0.634	0.7	0.615

Table 3.6: Rust Detection Experiments - Model Size - Initial Dataset

Because the results were not as sufficient as expected, the dataset was extended by adding different open-source datasets to the existing one. The extended dataset, shown in Table 3.2, consists of 4.146 images with a split ratio of 76%-24%. Subsequently, further experiments were performed on the extended dataset. Table 3.7 displays the experiments conducted on the extended dataset and their respective performance results, with the best result highlighted in green. The best experiment resulted in an mAP of 0.22 across IoU thresholds ranging from 50 to 95 percent, a separate mAP of 0.486 at an IoU threshold of 50 percent, a precision of 0.55, and a recall of 0.49. Upon comparing the results of the experiments conducted on the extended dataset with the initial experiments on the first dataset, it was observed that the performance with the extended dataset decreased. Consequently, the decision was made to revert to the initial dataset for rust detection.

For the rust detection task, the final performed experiments involved the application of a modified version of the data augmentation pipeline used for power transformer and sub-components detection. The experiments were conducted on the best result: the experiment featuring the "YOLOv5x" model with the initial dataset. The adjustments to the data augmentation pipeline consisted of removing the color-altering and weather condition-related augmentation steps.

Rust Detection Experiments - Model Size - Extended Dataset				
Model	mAP@0.5:0.95	mAP@0.5	Precision	Recall
YOLOv5s	0.214	0.453	0.55	0.45
YOLOv5m	0.219	0.479	0.535	0.479
YOLOv5l	0.221	0.478	0.582	0.46
YOLOv5x	0.222	0.486	0.55	0.49

Table 3.7: Rust Detection Experiments - Model Size - Extended Dataset

This decision was based on the understanding that rust's most significant characteristic is its color, which remains relatively consistent across different weather conditions. The modified pipeline, which excluded these augmentations, is outlined below.

- Random Crop - 20%
- Conditions - 0%
- Colors - 0%
- Quality - 20%
- Vertical Flip - 20%
- Horizontal Flip - 20%

However, despite these changes, the experiment did not improve the performance. As a result of the experiments, the final model for rust detection was trained on the initial dataset with an image size of 640, an auto batch size, without applying the data augmentation pipeline, and using the largest version of the model, "YOLOv5x". This resulted in an mAP of 0.22 across IoU thresholds ranging from 50 to 95 percent, a separate mAP of 0.486 at an IoU threshold of 50 percent, a precision of 0.55, and a recall of 0.49.

3.6.3 Oil Leak Detection

The training process for the oil leak detection model followed a similar approach to that of the rust detection model. An image size of 640 and an auto batch size were utilized. To reduce training time, initial experiments were performed using the small model "YOLOv5s" and the large model "YOLOv5x" to evaluate the impact of model size on the performance. These experiments were performed with the initial dataset, seen in Table 3.1, consisting of 140 images with an 87%-13% split ratio.

Table 3.8 displays the results of these initial experiments, with the row indicating that the "YOLOv5x" model achieved better performance, highlighted in green. The model "YOLOv5x" resulted in an mAP of 0.397 across IoU thresholds ranging from 50 to 95 percent, a separate mAP of 0.738 at an IoU threshold of 50 percent, a precision of 0.706, and a recall of 0.663. However, since the achieved performance was still insufficient, additional data was collected to improve the results. Table 3.2 provides an overview of the extended dataset, consisting of 2184 images with a split ratio of 87%-13%.

Oil Leak Detection Experiments - Model Size - Extended Dataset				
Model	mAP@0.5:0.95	mAP@0.5	Precision	Recall
YOLOv5s	0.386	0.725	0.691	0.638
YOLOv5x	0.397	0.738	0.706	0.663

Table 3.8: Oil Leak Detection Experiments - Model Size - Initial Dataset

To evaluate the impact of the extended dataset, the experiment with the large model "YOLOv5x", which performed best in the previous experiments, was repeated with the extended dataset. The experiment resulted in a significant performance increase with an mAP of 0.686 across IoU thresholds ranging from 50 to 95 percent, a separate mAP of 0.886 at an IoU threshold of 50 percent, a precision of 0.908, and a recall of 0.824.

For the final oil leak detection experiment, an adjusted version of the data augmentation pipeline used for power transformer and sub-components detection was utilized. Since the detection circumstances for oil leaks are similar to those for rust detection, the pipeline described in the previous chapter for rust detection was utilized. This decision was based on the similarity of conditions required to detect both oil leaks and rust. The experiment achieved a slight increase of about one percent in mAP. Therefore the pipeline was subsequently utilized.

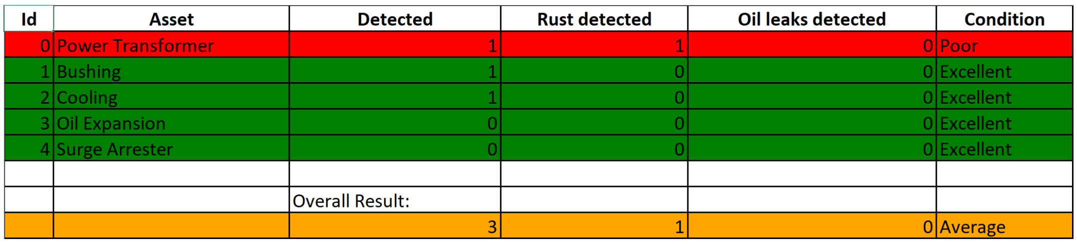
The final model was trained for 1000 epochs, employing an image size of 640 and an auto batch size, along with the adjusted data augmentation pipeline. The model achieved a slight increase in performance with an mAP of 0.693 across IoU thresholds ranging from 50 to 95 percent, a separate mAP of 0.894 at an IoU threshold of 50 percent, a precision of 0.911, and a recall of 0.834. Therefore, this model was used for further development.

3.7 Merge Results to Inspection Report

The trained models were combined in a Python script to create a report for the inspection tests. The script performs the detection for each type in different subprocesses. To minimize the risk of overlooking flaws, the detections are conducted using a low confidence threshold of 15 percent. Consequently, all detections exceeding this confidence threshold are considered for further evaluation. First, the power transformer and sub-component detection model are applied to an input image. For each detected class, a sub-directory is created, storing each detection as an image. In the next step, there are two possibilities. The first one is that power transformers or sub-components have been detected. In that case, rust and oil leak detection is performed on each detected instance. These detections are performed within a thread pool to parallelize the computation of the detections and increase the detection speed. Additionally, the detections are performed on the input image to show the detections on the whole input. In the second case, no power transformer or sub-component was found in the input image. Then the detections for rust and oil leaks are only performed on the input image. This function is implemented to enable the inspection of close-up photos for certain cases.

Since all detections have been stored, the complete inspection can be traced in later reviews. In addition to the stored detections, an Excel file is created to give an overview of the inspection test. When the inspection test is performed regarding power transformers and sub-components, the Excel file contains five columns: asset, detected, rust detected, oil leaks detected, and condition. The Excel sheet is completed by iterating the prior created directories and counting the results. In the case that no power transformers or sub-components are found, the Excel file contains three columns: "rust detected", "oil leaks detected", and "condition". This sheet is completed by incrementing the columns rust detected and oil leaks detected regarding the detections.

After the detections are performed, all results are registered in the Excel sheet, which is subsequently evaluated using a traffic light system. The evaluation is again performed differently for the two cases mentioned above. The numbers of detected assets, rust, and oil leaks are summed up in the first case. Then the detected rust and oil leaks are compared to the total number of detections. If more than 50 percent contain rust or oil leaks, the condition is evaluated as "poor", and the column is highlighted in red. If the number of rust and oil leaks is zero, the condition is assessed as "excellent", and the column is highlighted in green. In the cases between zero and 50 percent, the condition is assessed as "average", and the column is highlighted in orange. Figure 3.2 visually represents this case to illustrate the traffic light system. Further information on the result is given in the following chapter. For the second case, the concept is more straightforward. If both columns "rust detected" and "oil leaks detected" contain zero, the condition is evaluated as "excellent". If either column has a one, it is assessed as "average". If both include one, it is considered "poor". Figure 3.3 visually describes the process.



Id	Asset	Detected	Rust detected	Oil leaks detected	Condition
0	Power Transformer	1	1	0	Poor
1	Bushing	1	0	0	Excellent
2	Cooling	1	0	0	Excellent
3	Oil Expansion	0	0	0	Excellent
4	Surge Arrester	0	0	0	Excellent
		Overall Result:			
		3	1	0	Average

Figure 3.2: Example of Result with Traffic Light System

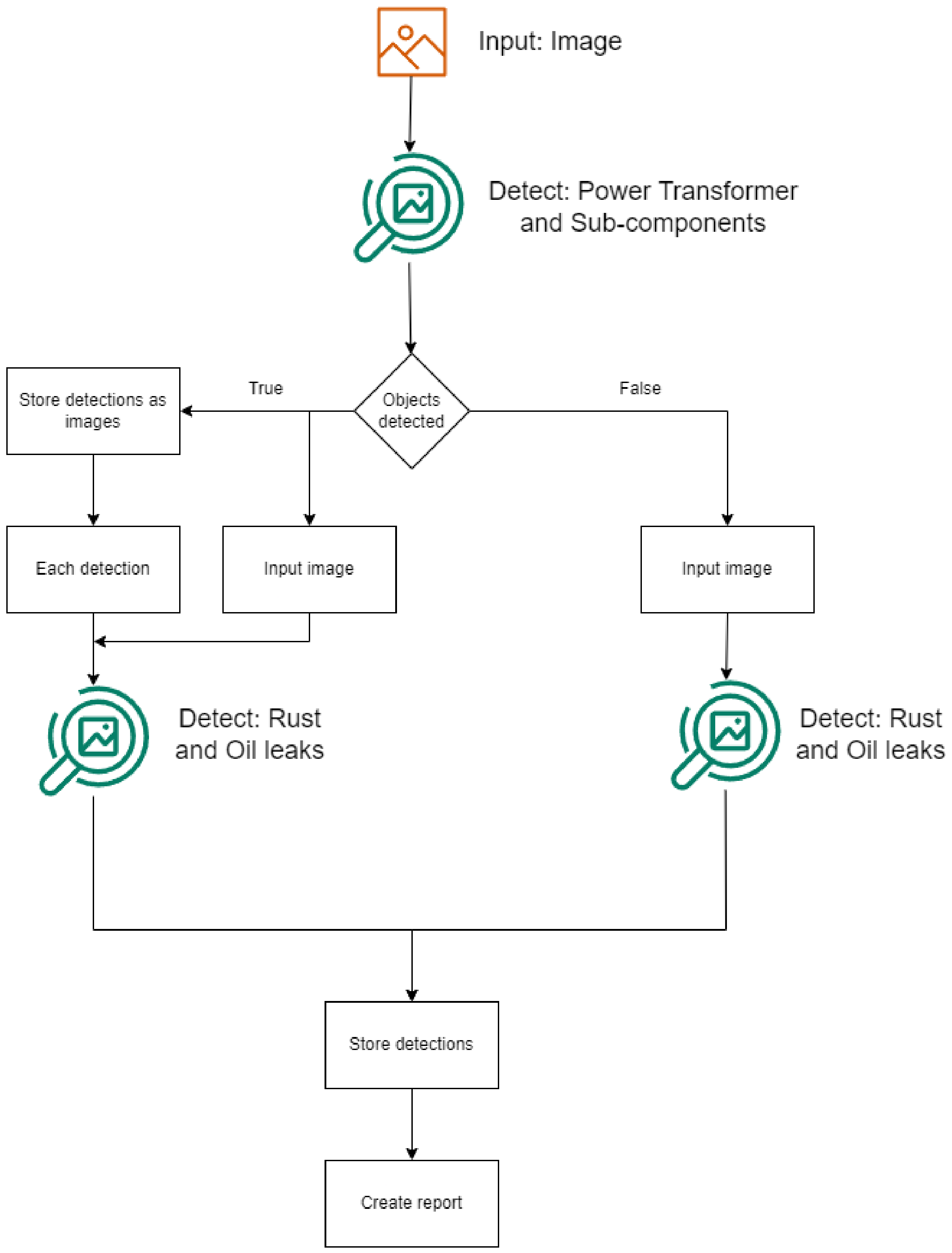

Input: Image
Detect: Power Transformer and Sub-components
Objects detected
True
False
Store detections as images
Each detection
Input image
Input image
Detect: Rust and Oil leaks
Detect: Rust and Oil leaks
Store detections
Create report

Figure 3.3: Visualized Inspection Test Process

4 Experimental Results

This chapter presents the training results for power transformer and sub-component detection, rust detection, and oil leak detection. The evaluation process involved the utilization of a confusion matrix, emphasizing the true positive rate, to assess the models' performance. An F1 Score-Confidence curve was employed to illustrate the F1 score corresponding to varying confidence levels. Furthermore, a comprehensive overview of metrics, including loss, precision, and recall, was provided to comprehensively understand the models' performance. These evaluation techniques contribute to a comprehensive analysis of the detection capabilities of the models, enabling insights for further improvements in visual inspection tests. Finally, the chapter describes the outcomes of merging the trained models into an inspection test and subsequently generating the results, providing insights into both the outcomes' positive and negative aspects.

4.1 Training Results

The following sections present the results obtained from training the models to detect power transformers and sub-components, rust, and oil leaks.

4.1.1 Power Transformer and Sub-components

The confusion matrix presented in Figure 4.1 offers valuable insights into the model's performance regarding detecting power transformers and their sub-components. The x-axis represents the actual classes, while the y-axis represents the predictions. The color range on the right side of the graph, ranging from white to dark blue, visually represents the percentage values used within the matrix. While there is no officially defined threshold to categorize a true positive rate (TPR) as satisfactory, for detecting power transformers and their sub-components, a targeted TPR of 75 percent was agreed upon with the domain expert of the partner company. The power transformer demonstrates an impressive true positive rate of 96%, indicating high accuracy in its identification. The model showcases effective detection capabilities among the sub-components, surpassing the targeted TPR. It achieves a TPR of 84% for bushings, 80% for cooling components, and 76% for oil expansions. Nevertheless, surge arresters fall short of the targeted True Positive Rate (TPR) with a TPR of 66%. This is likely due to their resemblance to bushings and the dataset's imbalanced distribution of labeled instances. Figure 4.2 illustrates the class distribution in the training data. The x-axis represents the classes, while the y-axis represents the number of instances within the dataset. Furthermore, it is noteworthy that approximately seven percent of the detected surge arresters were identified as false positives, mistakenly classified as a bushing.

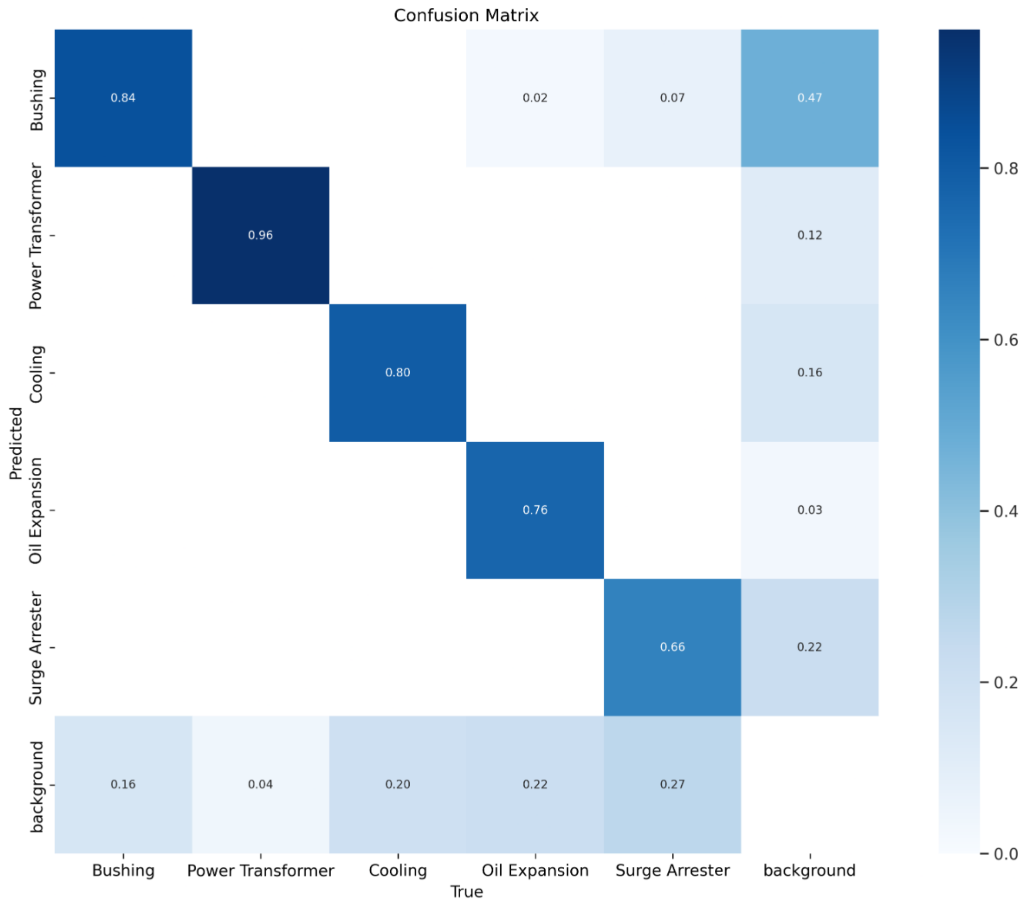


Figure 4.1: Confusion Matrix - Power Transformer and Sub-components

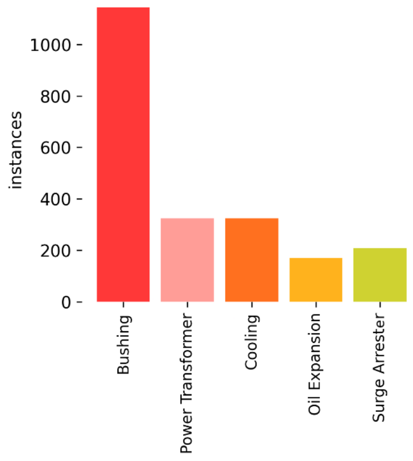


Figure 4.2: Class Distribution in Power Transformer and Sub-components Training Data

In Figure 4.3, the F1-confidence curve of the model is displayed. The x-axis represents the confidence, while the y-axis represents the F1 score. The F1 score peaked at 0.82, at a confidence level of 33 percent. The curve shows a consistent trend within the confidence range of 10 percent to 65 percent, where the F1 score remains at 0.76. However, a notable decline in the F1 score becomes evident beyond this range, indicating a significant decrease in model performance. According to the F1-Confidence curve, for reliable detection results, choosing a confidence threshold between 0.1 and 0.65 for this model is recommended.

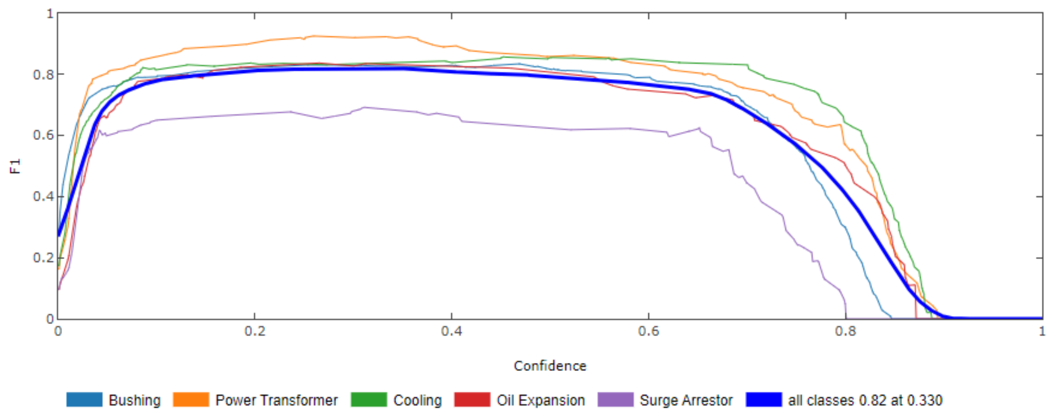


Figure 4.3: F1 Score - Confidence Curve - Power Transformer and Sub-components

Figure 4.4 illustrates the metrics obtained during the initial 200 training epochs. The x-axis represents the number of epochs ranging from zero to 200, while the y-axis represents the corresponding metric values. Notably, all metrics significantly improved within the first 40 epochs. The first three plots demonstrate the trend of the loss, which exhibited a rapid decrease in the initial 40 epochs followed by a slight decline until epoch 200. The last two plots showcase the metrics precision and recall, which experienced a notable increase within the first 40 epochs and then stabilized throughout the subsequent epochs. The training achieved its best result in epoch 176.

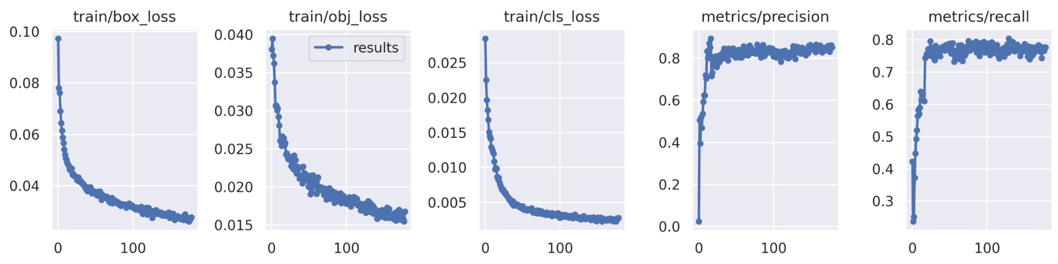


Figure 4.4: Training Metrics - Power Transformer and Sub-components

4.1.2 Rust

The confusion matrix presented in Figure 4.5 provides an insight into the model's performance in detecting rust. The x-axis represents the actual classes, while the y-axis represents the predictions. The color range on the right side of the graph, ranging from white to dark blue, visually represents the percentage values used within the matrix. Same as for power transformers, a true positive rate of 75 percent was targeted. According to the matrix, the true positive rate for rust identification stands at 66%. However, this current level of accuracy falls short of the desired outcome, indicating the need for improvement. To perform reliable inspection tests, the accuracy of detecting instances of rust needs to be increased. Therefore, it becomes necessary to focus on enhancing the true positive rate.

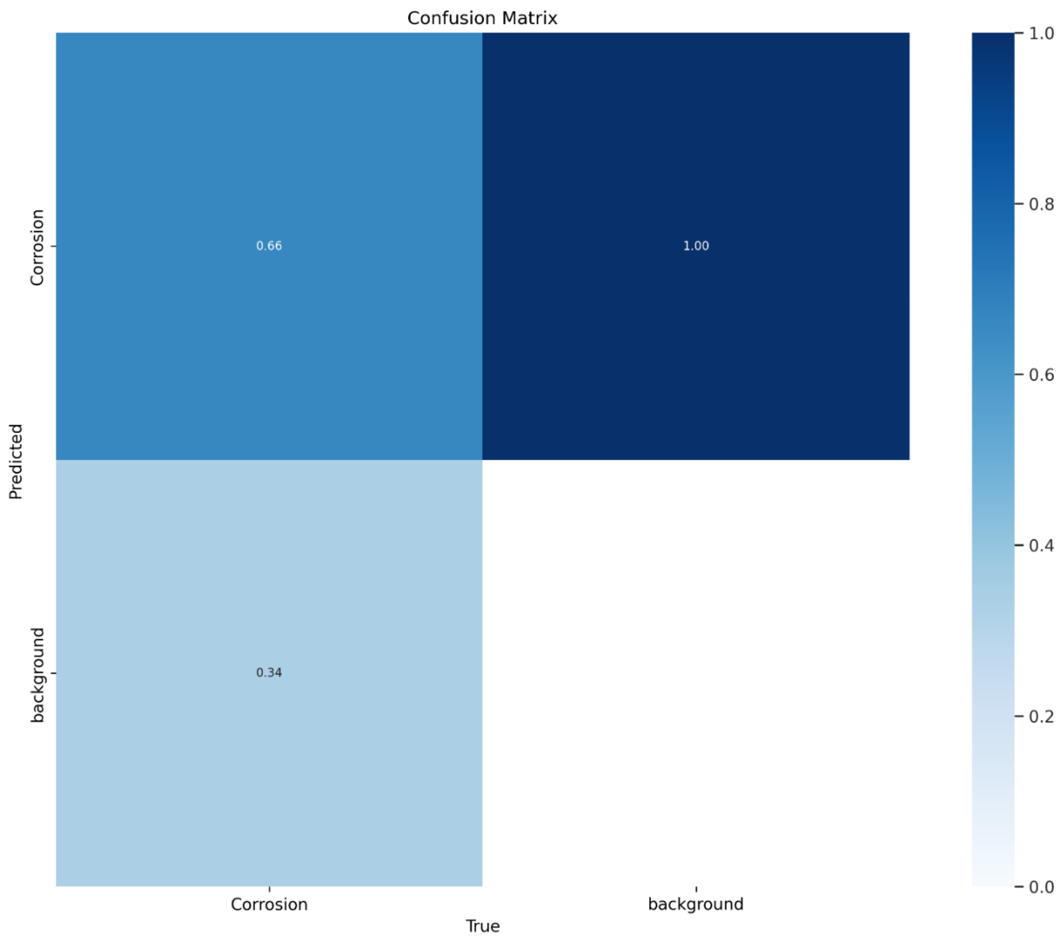


Figure 4.5: Confusion Matrix - Rust

In Figure 4.6 the F1-confidence curve of the model is visualized. The x-axis represents the confidence, while the y-axis represents the F1 score. The F1 score peaked at 0.65 at a confidence level of 35 percent. The curve shows a consistent trend within the 10 percent to 60 percent confidence range, where the F1 score remains above 0.6. Beyond this range, the curve demonstrates a gradual decline, indicating a significant degradation in the model's performance. According to the F1-Confidence curve, for reliable detection results, choosing a confidence threshold between 0.1 and 0.6 for this model is recommended.

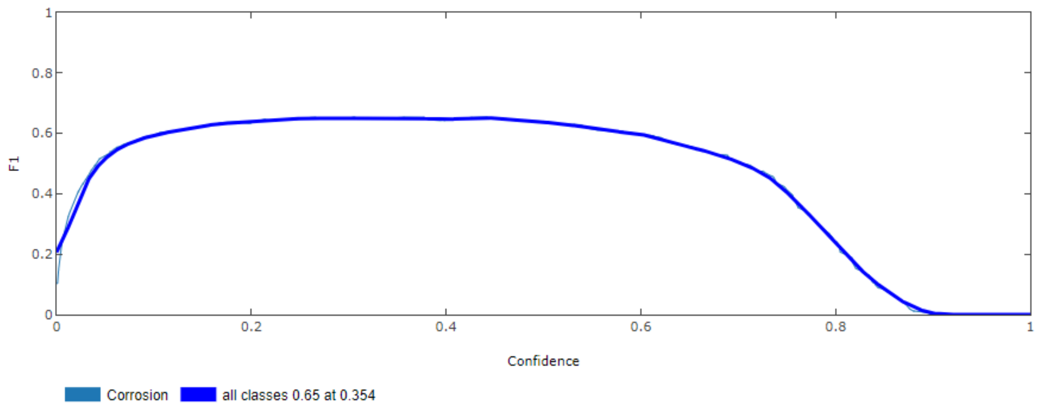


Figure 4.6: F1 Score - Confidence Curve - Rust

Figure 4.7 illustrates the metrics obtained during the initial 200 training epochs. The x-axis represents the number of epochs ranging from zero to 200, while the y-axis represents the corresponding metric values. Same as the metrics for power transformers, all metrics demonstrated their most significant improvement within the first 20 epochs. The first two plots demonstrate the trend of the loss, which exhibited a rapid decrease in the initial 20 epochs followed by a slight decline until epoch 200. Since the model for rust detection is trained on a single class dataset, the class loss seen in the third plot is zero. The last two plots showcase the metrics precision and recall, which experienced a notable increase within the first 20 epochs and then continued with a slight increase within the subsequent epochs. The training achieved its best result in epoch 132.

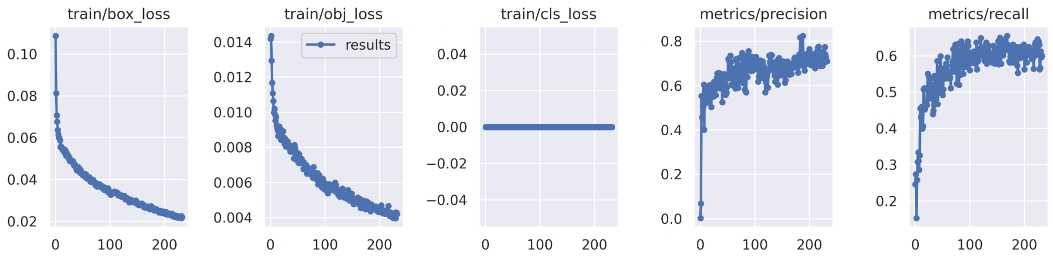


Figure 4.7: Training Metrics - Rust

4.1.3 Oil leaks

The confusion matrix in Figure 4.8 reveals the model's performance in detecting oil leaks. The x-axis represents the actual classes, while the y-axis represents the predictions. The color range on the right side of the graph, ranging from white to dark blue, visually represents the percentage values used within the matrix. Same as for power transformers, a true positive rate of 75 percent was targeted. The true positive rate for identifying oil leaks stands at 86%. This indicates that the model successfully and accurately identifies a significant portion of oil leak instances of the validation set, surpassing the targeted TPR. The high true positive rate signifies a promising result of the model in detecting oil leaks and highlights its potential to be used for visual inspection tests.

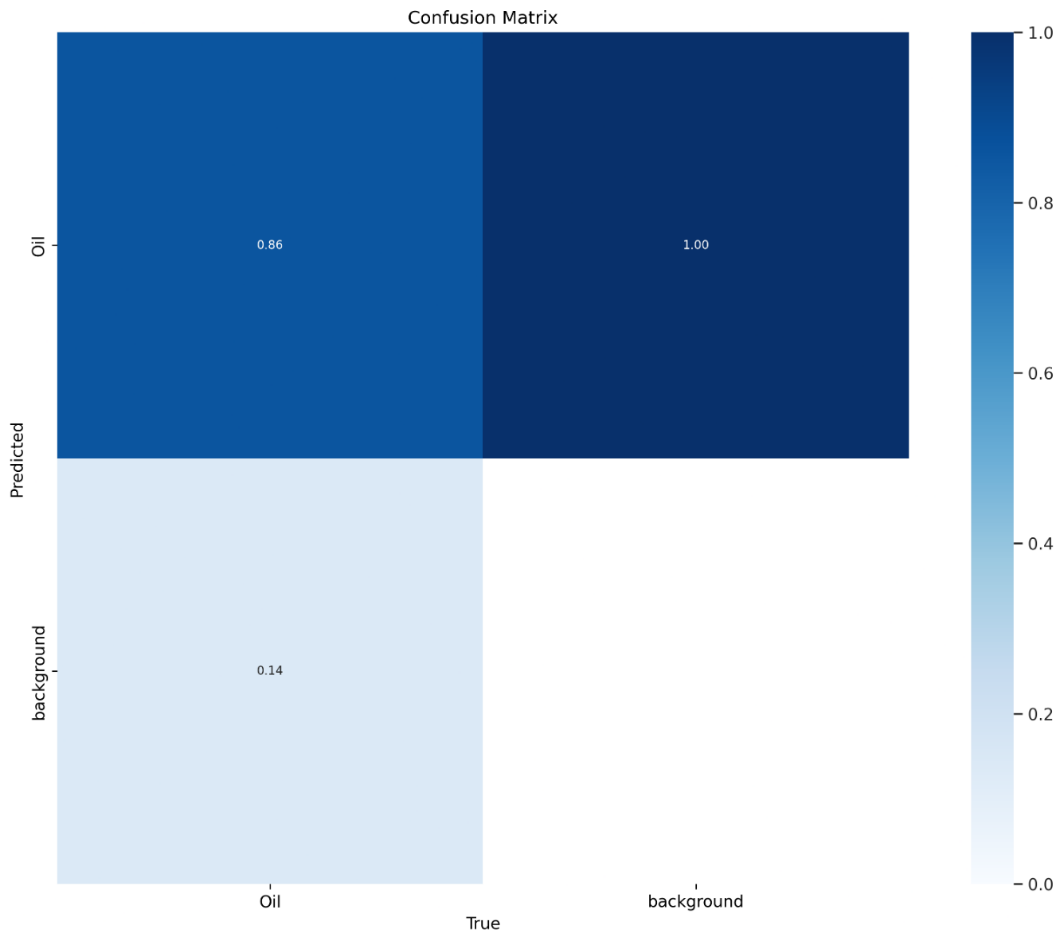


Figure 4.8: Confusion Matrix - Oil Leaks

In Figure 4.9 the F1-confidence curve of the model is visualized. The x-axis represents the confidence, while the y-axis represents the F1 score. The F1 score peaked at 0.87 at a confidence level of 42 percent. The curve shows a consistent trend within the confidence range of 10 percent to 78 percent, where the F1 score remains above 0.8. Beyond this range, the curve demonstrates a gradual decline, indicating a significant degradation in the model's performance. According to the F1-Confidence curve, for reliable detection results, choosing a confidence threshold between 0.1 and 0.78 for this model is recommended.

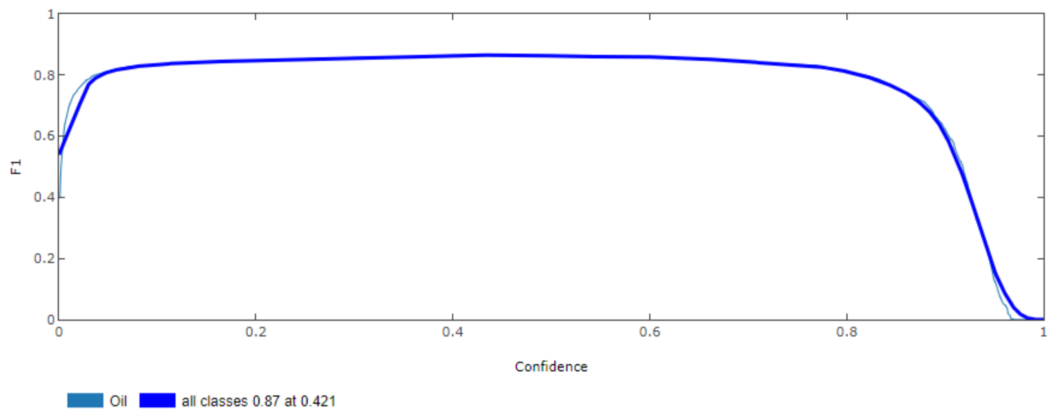


Figure 4.9: F1 Score - Confidence Curve - Oil Leaks

Figure 4.10 illustrates the metrics obtained during the initial 400 training epochs. The x-axis represents the number of epochs ranging from zero to 200, while the y-axis represents the corresponding metric values. Same as the metrics for power transformers, all metrics demonstrated their most significant improvement within the first 40 epochs. The first two plots demonstrate the trend of the loss, which exhibited a rapid decrease in the initial 40 epochs followed by a slight decline until epoch 400. Same as the model for rust detection, this model is trained on a single class dataset. Therefore, the class loss seen in the third plot is zero. The last two plots showcase the metrics precision and recall, which experienced a notable increase within the first 40 epochs and then continued with a slight increase within the subsequent epochs. The training achieved its best result in epoch 132.

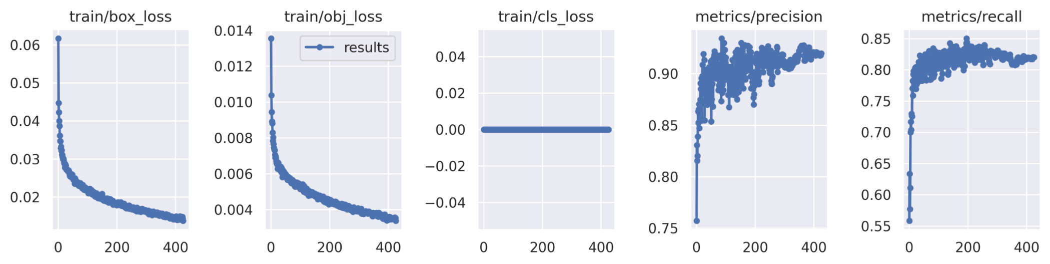
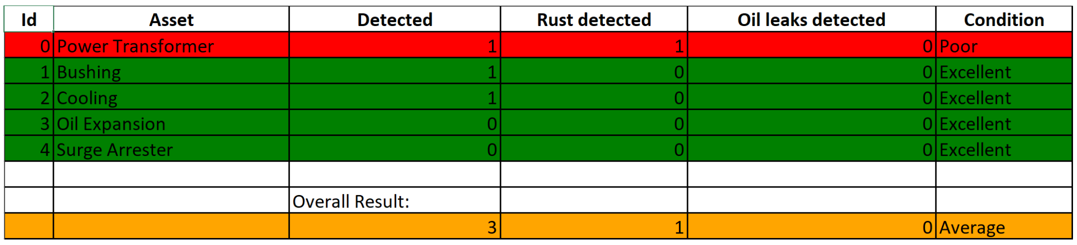


Figure 4.10: Training Metrics - Oil Leaks

4.2 Merged Inspection Test

New data from various cases was gathered to assess the inspection test using the final object detection models. This dataset encompasses images of power transformers, close-up shots highlighting rust and oil leaks, and photographs of power transformers captured from different angles. Figure 4.11 displays an inspection test report generated for an input featuring a power transformer. The report highlights the successful detection of the power transformer, a bushing and cooling system, and the identification of rust within the power transformer. As the detection yielded one flaw out of 3 detections, which is below 50%, the condition is assessed as "average."



Id	Asset	Detected	Rust detected	Oil leaks detected	Condition
0	Power Transformer	1	1	0	Poor
1	Bushing	1	0	0	Excellent
2	Cooling	1	0	0	Excellent
3	Oil Expansion	0	0	0	Excellent
4	Surge Arrester	0	0	0	Excellent
		Overall Result:			
		3	1	0	Average

Figure 4.11: Test Report - Results with Power Transformer

Every image in the gathered data underwent the visual inspection test. Subsequently, the yielded results were analyzed. The most significant findings from this analysis are summarized in Table 4.1, and further details are provided in the subsequent paragraphs.

Key Findings - Visual Inspection Test	
Model	Findings
Power Transformer and Sub-components	• Successful detection of most components • Successful detection of unknown construction types • Outcome is influenced by the angle • Surge arresters misinterpreted as bushings
Rust detection	• Successful detection of rust spots • Best results with close-up photographs • Not all occurrences are detected • Detection is not flawless
Oil leak detection	• Successful detection of oil leaks • Best results with close-up photographs • Certain misclassifications • Detection is not flawless

Table 4.1: Key Findings

The inspection tests successfully detected most of the power transformers and their sub-components with a confidence ranging from 45 percent to 85 percent. Furthermore, it showcased the capability to identify power transformers of construction types that were not included in the training data. An example of this case is shown in Figure 4.12, where the power transformer, the cooling system, and the oil expansion were successfully detected. Nevertheless, the analysis of the results indicated that the detection is influenced by the angle at which the photograph was taken. Certain angles may result in the inability to detect components, especially the cooling system and the oil expansion. Figure 4.13

provides an illustrative example of the results obtained from two angles. The left section of the figure demonstrates that the power transformer, three bushings, and the oil expansion are not detected, primarily due to the angle of the photograph. Conversely, the cooling system and one bushing are not detected in the right section. Another common issue observed is the misinterpretation of surge arresters as bushings and vice versa. As mentioned in the previous chapter, this is mainly due to the imbalanced distribution of the training data.

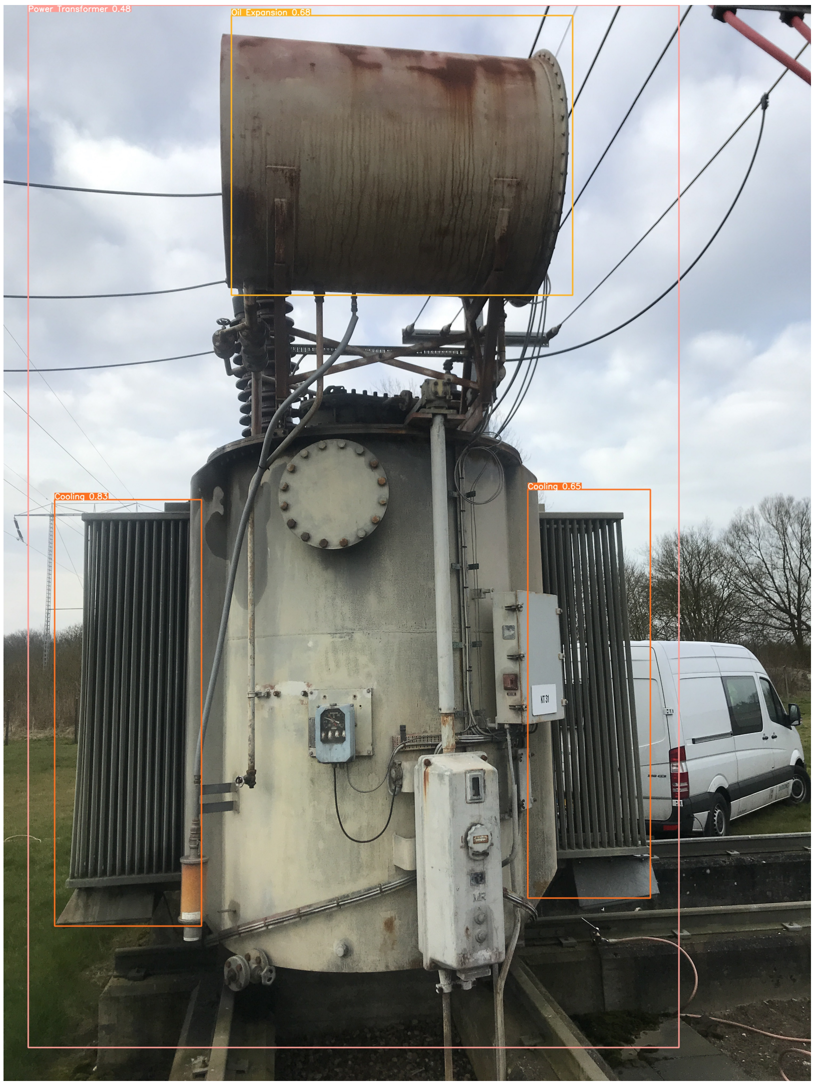


Figure 4.12: Power Transformer Inspection on Unknown Type

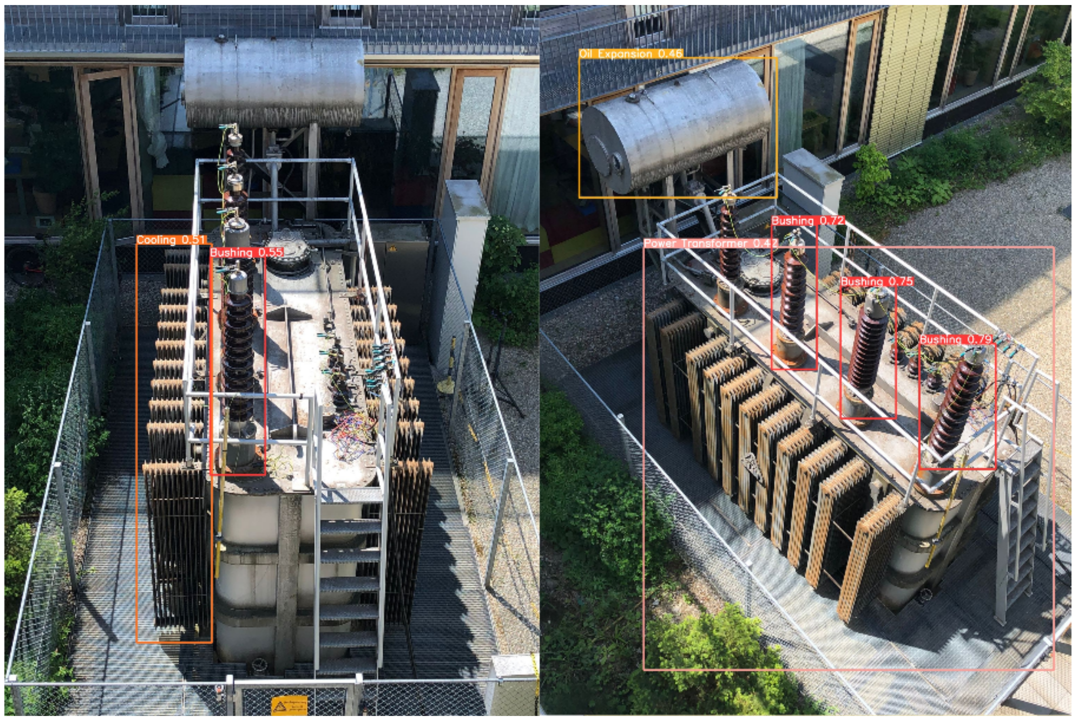


Figure 4.13: Power Transformer Inspection from Various Angles

The results showed that the model for rust detection can recognize some rust spots but not all of them. The confidence mostly ranged from 25 percent to 85 percent. While the accuracy is not as high as desired, the outcomes still provide a good indication of whether the image contains rust or not. Figure 4.14 shows an example of rust detection on a power transformer. The model accurately identifies areas with rust marked by red rectangles. However, there is a misinterpretation on the far right, where a red-colored cable is mistakenly classified as rust. Also, not every occurrence of rust is detected. It's important to note that close-up photographs produce better detection results because the dataset used only had close-up images. Although the model for rust detection doesn't capture every instance, the outcomes generally match the characteristics observed in the input image in most cases.

The results revealed that the oil leak detection model can detect oil leaks, with a confidence ranging from 45 percent to 55 percent. It performs best on close-up photographs, and its accuracy decreases when analyzing images of power transformers, where it sometimes misses existing oil leaks. Figure 4.15 shows an example of oil leak detection on a close-up photograph. A red rectangle highlights the identified oil leak. The example shows that an oil leak

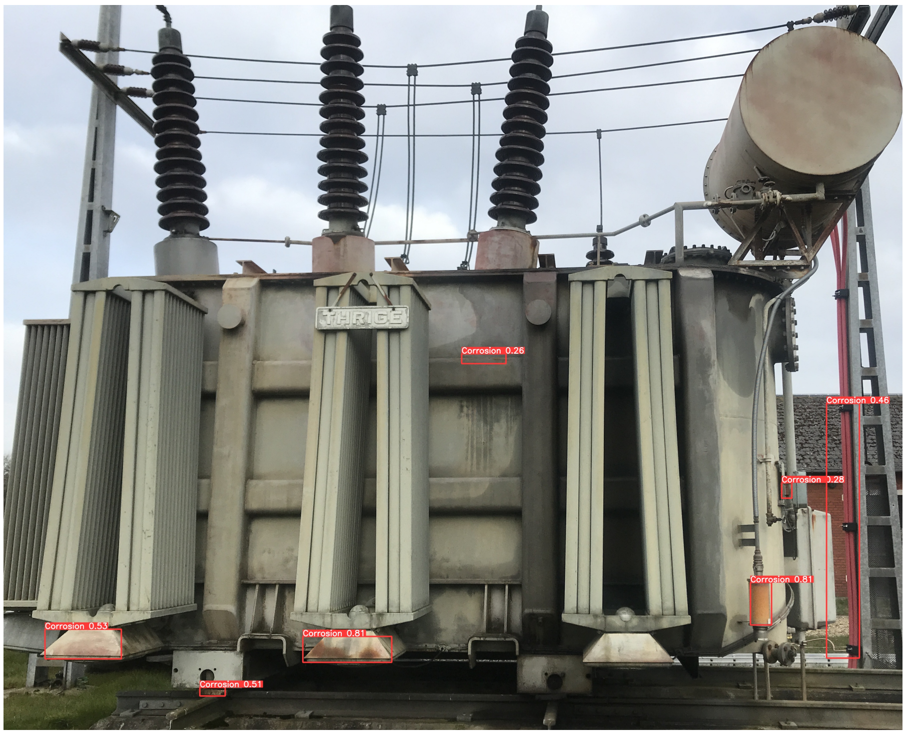


Figure 4.14: Rust Detection on Power Transformer

has been detected, but there are still oily parts within the image that have not been detected. Despite not detecting all occurrences, the outcome still matches the characteristics observed in the input image. A common issue detected is the model's tendency to misclassify tree tops as oil leaks. Like the model for rust detection, the model for oil leak detection is not flawless, but it generally matches the distinctive features in the input image.

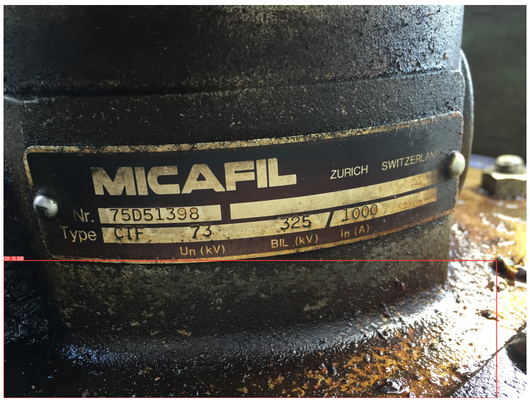


Figure 4.15: Oil Detection on Close-up Photograph

5 Discussion of the Results

In this chapter, the results of the case study are discussed. The sub-chapters encompass the implemented Azure DevOps Pipeline, the Hardware facilitated for model training, the detection of power transformers and their sub-components, rust detection, oil leak detection, and the resulting inspection report. Through an in-depth exploration of these areas, the outcomes obtained from the study are analyzed and interpreted while also providing insights for potential future enhancements in visual inspection testing.

5.1 Azure DevOps Pipeline

The partner company manages its software development projects on an Azure DevOps Server [51], previously known as Team Foundation Server (TFS), using Git version control [52], allowing seamless management and version control. By integrating the YOLOv5 implementation by Jocher [14] into the existing structure, the pipeline has brought a significant workflow improvement. This combination of methodologies holds considerable value for the partner company, as it enables object detection in the current project and lays a foundation for future projects in this domain. Moreover, the pipeline can be easily adapted to accommodate different versions of YOLO, enhancing its versatility. The integration of Git version control for storing the historical codebase has brought significant advantages and proven highly beneficial. It allows for seamless rollbacks to previous experiment configurations, such as when the rust detection dataset experienced a decline in performance upon its expansion. This capability provides the flexibility to efficiently manage and address issues that may arise during the development and optimization of the model.

5.2 Hardware

For further developments, acquiring a GPU with increased memory capacity is essential to mitigate the occurred limitations in the training process. This case study employs the NVIDIA Quadro RTX 4000 GPU, which features 8GB of GDDR6 memory. According to Puget Systems [53], 8GB is considered the minimum size, while 12 to 24 GB is commonly recommended. Hence, upgrading to a GPU within this memory range is advised. This upgrade will grant more outstanding training capabilities and enable the utilization of larger input images. Since the detection process focuses on sub-components, there are instances where the resolution of these components is notably low, primarily due to the initial image size. By leveraging the computational power of a GPU capable of handling larger images, the accuracy of sub-component detection can be enhanced, addressing the limitations imposed by their low resolution.

5.3 Detection

This section discusses the training outcomes for power transformer and sub-component detection, rust detection, and oil leak detection. It builds upon the results obtained in the previous chapter and explores potential development steps for future advancements in visual inspection tests.

5.3.1 Power Transformer and Sub-components

The current power transformer and sub-component dataset has already yielded a well-performing model for detection. Particularly the detection of power transformers yielded a true positive rate (TPR) of 96 percent and confidence peaking at 90 percent. An issue of the model was the detection of bushings and surge arresters, especially surge arresters with a TPR of 66 percent and maximum confidence of 65 percent. This is due to their resemblance and the dataset's imbalanced distribution of labeled instances. Therefore, there remains unused potential to further enhance its performance by expanding the dataset to balance the distribution and improving the quality of the labels. In particular, increased availability of labeled surge arresters is essential to increase the model's performance in accurately identifying these components. Applying these changes allows an opportunity to enhance and elevate the model's overall performance. Goodfellow [3] describes determining whether more data must be gathered. This can be evaluated by comparing the detection results performed on validation data and data from the training set. If the detection performed on the training data has high accuracy and decreases on the validation data, this is a sign that more data is required. For power transformers and their sub-components, the detection on the training set has a constant confidence of around 80 percent, whereas the detection performed on the validation data ranges from 50 to 80 percent. This indicates that gathering more data can increase the model's performance.

The results indicate that the photograph's perspective significantly affects the detection process. Mainly, cooling is predominantly detected in front-view photos. To address this issue in the future, inspection tests could be performed on a video, where the asset is recorded from various angles. This could provide a solution as the video captures the asset from multiple angles. Therefore, the angle would not affect the inspection test anymore. Another potential solution would involve giving explicit instructions regarding the required angles for capturing photographs. These instructions would lead to photos from the required angles, solving the problem. However, it is essential to acknowledge that this approach carries a higher risk of errors, as instructions may be disregarded or misinterpreted.

5.3.2 Rust

Although the results demonstrated the model's capability to detect rust, the current dataset used for rust detection needed to meet the required level of accuracy for deployment in a visual inspection test product. The dataset's performance fell below the desired standards during the validation phase with a true positive rate (TPR) of 66 percent and maximum confidence of 65 percent. Several factors contribute to this underwhelming performance, including the lack of data quality, insufficient labeling, and the dataset's generic nature, which could be more optimal for addressing the specific problem. Goodfellow [3] mentions that machine learning algorithms perform best when the data for training is appropriate for the true complexity of the task. This is not the case for the rust detection model, as it is too generic and only consists of close-up photographs. To achieve a reliable performance suitable for visual inspection tests, investing more time in collecting data that encompasses particular cases and applying a labeling process with high-quality standards is imperative. It is essential to shift the focus towards acquiring real-world data from power transformers that specifically include instances of rust. By implementing these measures, it is feasible to enhance the effectiveness of the dataset and achieve the desired levels of accuracy and reliability for the model.

5.3.3 Oil Leaks

Although the dataset for detecting oil leaks has shown satisfactory performance on the validation set, including a high true positive rate of 86 percent and peaking confidence of 87 percent, the overall accuracy of the model for power transformers in specific cases did not meet the desired level. As for rust detection, considering the statement from Goodfellow [3], this discrepancy can be attributed to the limitations of the available open-source dataset, which primarily comprises close-up photographs that are not well-suited for addressing the specific problem. Consequently, further refinement and improvement are necessary to ensure accurate detection within the targeted domain. These improvements are crucial for deploying the visual inspection test as a product. They can be done by sorting out inappropriate images and incorporating additional images representing specific cases within power transformers. Another notable issue is that the current version of the model tends to misclassify parts of trees as oil leaks. To mitigate this, unlabeled images depicting trees and natural surroundings can be introduced to the dataset, facilitating the model's ability to distinguish between oil leaks and tree tops.

5.4 Inspection Report

The Excel sheet report provides a concise summary of the inspection test. Nevertheless, engaging in discussions regarding the desired design of the report with domain experts can enhance the report's quality and contribute to its future development. An in-depth evaluation of individual sub-components becomes possible by including an itemized listing of each detected object and the corresponding image's name and path. This enhanced level of detail within the sheet would offer a comprehensive overview of the inspection test, enabling a more comprehensive assessment of its outcomes.

Furthermore, enhancing the level of detail could be achieved by training the model to detect varying degrees of rust and oil leaks. This could involve differentiating between "minimal rust/oil leak" and "severe rust/oil leak." However, gathering and labeling specific data for each case is necessary to accomplish this.

6 Conclusion

This chapter summarizes the case study, including its key findings and outcomes. Following the summary, an outlook section unveils a forward-looking perspective, outlining potential areas for further exploration and development based on the insights gained from the case study. This chapter's structure provides a comprehensive understanding of the case study's implications and sets the stage for future research and advancements in visual inspection tests.

This case study assessed the viability of visual inspection tests targeting power transformers. To achieve this, three distinct object detection models were trained utilizing the implementation of YOLOv5 [14]. The training was conducted on a Linux-based workstation using an NVIDIA Quadro RTX 4000 GPU. It was acknowledged that the case study results could be enhanced by employing a more powerful GPU, which would expand the possibilities for training object detection models. An Azure DevOps Pipeline was utilized to execute each training to improve the workflow and seamlessly integrate the codebase into the partner company's infrastructure. The training results were closely monitored using the open-source platform ClearML [38]. The model for detecting power transformers was trained on data gathered and labeled by the partner company. In contrast, the rust and oil leak detection models were trained on generic data collected from open-source datasets.

The final model for power transformer and sub-component detection achieved satisfactory accuracy. However, the model's dependence on the angle of input images remains a limitation. One proposed solution for this limitation is to conduct the inspection test using a video recorded from multiple angles. By addressing this flaw, the power transformer and sub-component detection can be effectively incorporated into a visual inspection test product. The models trained on generic open-source datasets demonstrated the feasibility of detecting flaws such as rust and oil leaks in visual inspection tests. The case study implementation successfully detected and consolidated many rust spots or oil leaks within the input images into an inspection report. Analysis of the test results revealed common errors, including misclassifying oil leaks as rust and mistakenly detecting treetops as oil leaks. Proposed solutions for these issues involve improving precision through collecting and labeling specific case data and incorporating background images that contain treetops to mitigate the flaw in oil leak detection. Although the resulting models for rust and oil leak detection in this case study are not precise enough for use in a visual inspection test product, their precision can be enhanced through further improvements to meet the required standards.

The results of this case study have been presented to the management of the partner company, who expressed their admiration for the demonstrated outcomes. They acknowledged the potential for further development of the project. While no definitive decision has been made regarding the project's future, the critical tasks for its continued advancement are outlined in the subsequent sections of this chapter.

6.1 Improve Precision

The latest YOLO version, YOLOv8, can be employed to develop a releasable product to enhance the precision and speed of detection. This updated version of YOLO offers improved performance and is particularly advantageous if the decision is made to conduct inspection tests on videos, where fast detection is crucial. The performance increase is illustrated in Figure 3.1.

Furthermore, dedicating additional time and effort to enhance the quality of the datasets, particularly by making them more specific to the target cases, can significantly improve the precision of detection. Additionally, leveraging a more powerful GPU can expand the possibilities for training experiments, thereby enhancing the performance of the models.

6.2 Inspection Report

A comprehensive test result is crucial to provide the visual inspection test as a product. Therefore, the test result of this case study can be extended to be more detailed. As mentioned in the Chapter "Inspection Report" 5.4, this can be achieved by including an itemized listing of each detected object, along with the corresponding image name and path. This would enable detailed reviews of inspection tests. Furthermore, the level of detail could be enhanced by labeling the data with different stages. This could involve differentiating between "minimal rust/oil leak" and "severe rust/oil leak." Additionally, the design of the Excel sheet can be improved to make the outcomes of the inspection test more transparent.

6.3 Increase Speed

In the case study, object detections were conducted on the input image and each detected sub-component. This approach facilitated the merging of outcomes into a comprehensive test result. However, since this method involves processing parts of the image multiple times, it does not optimize the speed of the inspection test. As the goal is to offer visual inspection tests as a product, the speed of the service becomes crucial. To enhance the performance of the detection process, the detections can be performed only on the input image, and the inspection test result can be generated by evaluating the bounding boxes for each detection. By comparing the coordinates, it becomes possible to assess whether or not a flaw has occurred within a specific component. Moreover, if there is a requirement for individual detections to be documented, additional functionality can be implemented. This functionality would utilize the coordinates of the bounding boxes to create sub-images of the detections and store them in a subsequent step.

6.4 Additional Inspections

While this case study primarily concentrated on rust and oil leak detection, several other flaws must be addressed for comprehensive inspection tests. Examples of these flaws include fire traces from electric shocks and incomplete components. Gathering and labeling data specific to each flaw is necessary to implement these flaws for visual inspection tests. The experiments conducted for rust and oil leak detection can serve as a reference for training, which can then be adjusted to optimize the performance for each specific problem. The models for additional detection can be seamlessly integrated into the existing code to enable individual detections and merge the outcomes into a comprehensive test result.

6.5 Provide Webservice

Upon release, the product should be offered as a web service for easy accessibility by customers. Users can upload an image or video to the service and receive a detailed test result. By providing the visual inspection test as a web service, the customers can access the tool from any device with an internet connection. This increases the usability since the test engineers can use a smartphone or tablet to perform the test. Another advantage is that the customer can access the tool without having to install anything, and updates to the service can be provided seamlessly.

Bibliography

[1] V. Crastan, *Elektrische Energieversorgung 1*. Springer-Verlag, 2007.

[2] I. 60076-2, *Power transformers Part 2: Temperature rise for liquid-immersed transformers*. IES standard, 2011.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge, Massachusetts London The MIT Press, 2016.

[4] “What is data labeling? | ibm,” *www.ibm.com*, [Online]. Available: `https://www.ibm.com/topics/data-labeling`, (accessed February 27, 2023).

[5] F. Chollet, *Deep Learning with Python*. Shelter Island (New York, Estados Unidos): Manning, Cop, 2018.

[6] A. Emmanuel, *Building Machine Learning Powered Applications*. O’Reilly, 2020.

[7] A. Géron, *Hands-on machine learning with scikit-learn, keras and tensorflow: Concepts, Tools and Techniques to Build Intelligent Systems*. O’Reilly Media, Incorporated, 2019.

[8] “Confusion matrix: How to use it interpret results [examples],” *www.v7labs.com*, [Online]. Available: `https://www.v7labs.com/blog/confusion-matrix-guide`, (accessed June 12, 2023).

[9] “Precision and recall — a comprehensive guide with practical examples,” *Medium*, [Online]. Available: `https://towardsdatascience.com/precision-and-recall-a-comprehensive-guide-with-practical-examples-71d614e3fc43`, (accessed June 08, 2023).

[10] “Neural network primitives part 1 - mcculloch pitts neuron model (1943),” *MLK - Machine Learning Knowledge*, [Online]. Available: `https://machinelearningknowledge.ai/mcculloch-pitts-neuron-model/`, (accessed March 11, 2023).

[11] C. C. Aggarwal, *Neural Networks and deep learning a textbook*. Cham Springer, 2018.

[12] “Leakyrelu,” *LeakyReLU - PyTorch 2.0 documentation*, [Online]. Available: `https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html`, (accessed June 16, 2023).

[13] "What are loss functions?" *Medium*, [Online]. Available: https://towardsdatascience.com/what-is-loss-function-1e2605aeb904, (accessed June 08, 2023).

[14] "Yolov5 by ultralytics," *GitHub*, [Online]. Available: https://github.com/ultralytics/yolov5, (accessed February 20, 2023).

[15] R. Szeliski, *COMPUTER VISION: algorithms and applications.* S.L.: Springer Nature, 2020.

[16] "Image processing using cnn: A beginners guide," *Analytics Vidhya*, [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/image-processing-using-cnn-a-beginners-guide/, (accessed March 08, 2023).

[17] "Cnn | introduction to pooling layer," *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/, (accessed March 28, 2023).

[18] "Cnn and softmax - andrea perlato," *www.andreaperlato.com*, [Online]. Available: https://www.andreaperlato.com/aipost/cnn-and-softmax/#:~:text=Most%5C%20of%5C%20the%5C%20time%5C%20the, (accessed March 14, 2023).

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. DOI: 10.1109/CVPR.2016.91.

[20] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. arXiv: 1409.4842. [Online]. Available: http://arxiv.org/abs/1409.4842.

[21] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016. arXiv: 1612.08242. [Online]. Available: http://arxiv.org/abs/1612.08242.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[23] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results*, http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[24] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. arXiv: 1804.02767. [Online]. Available: http://arxiv.org/abs/1804.02767.

[25] T. Lin, M. Maire, S. J. Belongie, *et al.*, “Microsoft COCO: common objects in context,” *CoRR*, vol. abs/1405.0312, 2014. arXiv: `1405.0312`. [Online]. Available: `http://arxiv.org/abs/1405.0312`.

[26] A. Bochkovskiy, C. Wang, and H. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” *CoRR*, vol. abs/2004.10934, 2020. arXiv: `2004.10934`. [Online]. Available: `https://arxiv.org/abs/2004.10934`.

[27] J. Redmon, “Darknet: Open source neural networks in c,” [Online]. Available: `http://pjreddie.com/darknet/`, (accessed June 14, 2023).

[28] C. Wang, H. M. Liao, I. Yeh, Y. Wu, P. Chen, and J. Hsieh, “Cspnet: A new backbone that can enhance learning capability of CNN,” *CoRR*, vol. abs/1911.11929, 2019. arXiv: `1911.11929`. [Online]. Available: `http://arxiv.org/abs/1911.11929`.

[29] D. Misra, “Mish: A self regularized non-monotonic neural activation function,” *CoRR*, vol. abs/1908.08681, 2019. arXiv: `1908.08681`. [Online]. Available: `http://arxiv.org/abs/1908.08681`.

[30] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, “Path aggregation network for instance segmentation,” *CoRR*, vol. abs/1803.01534, 2018. arXiv: `1803.01534`. [Online]. Available: `http://arxiv.org/abs/1803.01534`.

[31] S. Woo, J. Park, J. Lee, and I. S. Kweon, “CBAM: convolutional block attention module,” *CoRR*, vol. abs/1807.06521, 2018. arXiv: `1807.06521`. [Online]. Available: `http://arxiv.org/abs/1807.06521`.

[32] “Pytorch documentation — pytorch master documentation,” *PyTorch*, [Online]. Available: `https://pytorch.org/docs/stable/index.html`, (accessed June 18, 2023).

[33] “Performance benchmark of yolo v5, v7 and v8,” *Stereolabs*, [Online]. Available: `https://www.stereolabs.com/blog/performance-of-yolo-v5-v7-and-v8/`, (accessed March 27, 2023).

[34] “Nvidia quadro rtx 4000 specs,” *TechPowerUp*, [Online]. Available: `https://www.techpowerup.com/gpu-specs/quadro-rtx-4000.c3336`, (accessed February 21, 2023).

[35] “About cuda,” *NDIDIA Developer*, [Online]. Available: `https://developer.nvidia.com/about-cuda`, (accessed February 19, 2023).

[36] “What is azure pipelines? - azure pipelines,” *learn.microsoft.com*, [Online]. Available: `https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops`, (accessed May 03, 2023).

[37] “Azure pipelines agents - azure pipelines,” *learn.microsoft.com*, [Online]. Available: `https://learn.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=azure-devops&tabs=browser`, (accessed May 03, 2023).

[38] “Clearml | the continuous machine learning company,” *ClearML*, [Online]. Available: `https://clear.ml/`, (accessed March 11, 2023).

[39] “Labelimg,” *GitHub*, [Online]. Available: `https://github.com/heartexlabs/labelImg`, (accessed February 27, 2023).

[40] “Corrosion detection instance segmentation dataset (v15, 2023-04-10 12:54am) by research,” *Roboflow*, [Online]. Available: `https://universe.roboflow.com/research-0ncz5/corrosion-detection-9lrgc/dataset/15`, (accessed March 17, 2023).

[41] “Oil training object detection dataset and pre-trained model by fire training,” *Roboflow*, [Online]. Available: `https://universe.roboflow.com/fire-training/oil-training`, (accessed May 02, 2023).

[42] “V2 merged coated and noncoated object detection dataset and pre-trained model by phil adriel,” *Roboflow*, [Online]. Available: `https://universe.roboflow.com/phil-adriel/v2-merged-coated-and-noncoated`, (accessed May 02, 2023).

[43] “Rust8 object detection dataset and pre-trained model by saaragh,” *Roboflow*, [Online]. Available: `https://universe.roboflow.com/saaragh/rust8`, (accessed May 02, 2023).

[44] “Rust9 object detection dataset by saaragh,” *Roboflow*, [Online]. Available: `https://universe.roboflow.com/saaragh/rust9`, (accessed May 02, 2023).

[45] “2354345632165 object detection dataset by 1234,” *Roboflow*, [Online]. Available: `https://universe.roboflow.com/1234-f0ryk/2354345632165`, (accessed May 02, 2023).

[46] “Oil detector object detection dataset by achraf baya chatti,” *Roboflow*, [Online]. Available: `https://universe.roboflow.com/achraf-baya-chatti-tf4mp/oil-detector`, (accessed May 02, 2023).

[47] “Oil spill detection object detection dataset by daisycat1008@outlook.com,” *Roboflow*, [Online]. Available: `https://universe.roboflow.com/daisycat1008-outlook-com/oil-spill-detection-q6qid`, (accessed May 02, 2023).

[48] “Spillage detection object detection dataset by spillage,” *Roboflow*, [Online]. Available: `https://universe.roboflow.com/spillage/spillage-detection`, (accessed May 02, 2023).

[49] "Albumentations," *Albumentations*, [Online]. Available: `https://albumentations.ai/`, (accessed April 02, 2023).

[50] "Genetic algorithms," *GeeksforGeeks*, [Online]. Available: `https://www.geeksforgeeks.org/genetic-algorithms/`, (accessed May 24, 2023).

[51] "Azure devops server," *Microsoft Azure*, [Online]. Available: `https://azure.microsoft.com/en-gb/products/devops/server`, (accessed June 22, 2023).

[52] "What is git version control?" *GitLab*, [Online]. Available: `https://about.gitlab.com/topics/version-control/what-is-git-version-control/`, (accessed June 22, 2023).

[53] "Hardware recommendations for machine learning / ai," *Puget Systems*, [Online]. Available: `https://www.pugetsystems.com/solutions/scientific-computing-workstations/machine-learning-ai/hardware-recommendations/`, (accessed June 22, 2023).

Statement of Affirmation

I declare that I have developed and written the enclosed work completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. This Bachelor Thesis was not used in the same or in a similar version to achieve an academic degree nor has it been published elsewhere.

Dornbirn, July 08, 2023 Frederick Zech

Signiert von:	Frederick Zech
Datum:	08.07.2023 22:37:28

TRUST

Dieses Dokument ist digital signiert!
Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument hat gemäß Art. 25 Abs. 2 der Verordnung (EU) Nr 910/2014 vom 23. Juli 2014 ("eIDAS-VO") die gleiche Rechtswirkung wie ein handschriftlich unterschriebenes Dokument.

Prüfinformation: Informationen zur Prüfung der elektronischen Signatur finden Sie unter: www.a-trust.at/pdf