FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Smart Manufacturing: Integrating AI-Based Defect Detection in Manufacturing Execution Systems

Beatriz Lopes dos Santos



Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Luís Filipe Teixeira

July 26, 2024

Smart Manufacturing: Integrating AI-Based Defect Detection in Manufacturing Execution Systems

Beatriz Lopes dos Santos

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Prof. Rui Rodrigues

Referee: Prof. Vitor Filipe

Referee: Prof. Luís Filipe Teixeira

July 26, 2024

Abstract

Optimizing production efficiency significantly relies on effectively identifying defects within a production line. This process, inherently complex and prone to repetition, is susceptible to human errors. Automating this procedure with speed and accuracy substantially benefits manufacturers, enhancing production efficiency and product quality.

This research aims to conceptualize and develop a proof-of-concept for a generic Artificial Intelligence training system, designed to generate viable defect identification models for various processes and products, using a set of known images. The primary focus is on utilizing deep learning-based models, specifically supervised methods, to detect anomalies in product manufacturing.

Experiments were conducted using various deep learning models, with a particular emphasis on the YOLO (You Only Look Once) family of models and Faster R-CNN. The research involved training these models with labelled datasets, where images describing possible defects served as the training data. The performance of different models, including YOLOv5, YOLOv8, and YOLOv9, was evaluated to determine their accuracy in identifying defects.

Furthermore, the investigation analyses the feasibility of non-technical users training machine learning models using defective images. The integration of this AI-based defect detection system into a Commercial Off-The-Shelf Manufacturing Execution System (MES) is explored, with a specific focus on the Critical Manufacturing MES. This integration aims to provide manufacturers with an advanced and efficient means of detecting and addressing product defects.

Results from this study indicate that the YOLOv9 model showed particularly promising results, achieving high accuracy in defect detection on the Northeastern University Detection Dataset and the GDXray Castings Dataset. The developed system received positive feedback from users, confirming its ease of integration into MES solutions. Users reported high levels of satisfaction with the system's usability, performance, and effectiveness, highlighting its potential to improve operational efficiency and defect management in manufacturing environments.

Resumo

A otimização da eficiência da produção depende significativamente da identificação eficaz de defeitos numa linha de produção. Este processo, inerentemente complexo e propenso a ser repetitivo, é suscetível a erros humanos. Automatizar este procedimento com rapidez e precisão beneficia substancialmente os fabricantes, aumentando a eficiência da produção e a qualidade do produto.

Esta investigação tem como objetivo concetualizar e desenvolver uma prova de conceito para um sistema genérico de treino de inteligência artificial (IA), concebido para gerar modelos viáveis de identificação de defeitos, para vários processos e produtos, usando um conjunto de imagens conhecidas. O foco principal é a utilização de modelos baseados em *deep learning*, especificamente métodos supervisionados, para detetar defeitos no fabrico de produtos.

Foram realizadas experiências utilizando vários modelos de *deep learning*, com especial ênfase na família de modelos YOLO (*You Only Look Once*) e no Faster R-CNN. A investigação envolveu o treino destes modelos com conjuntos de dados anotados, em que as imagens que descrevem possíveis defeitos servem como dados de treino. O desempenho de diferentes modelos, incluindo YOLOv5, YOLOv8 e YOLOv9, foi avaliado para determinar a sua precisão na identificação de defeitos.

Além disso, a investigação analisa a viabilidade de utilizadores não técnicos treinarem modelos de *machine learning* utilizando imagens defeituosas. É explorada a integração deste sistema de deteção de defeitos baseado em IA num Sistema de Execução de Fabrico Comercial pronto a utilizar, com especial incidência no Sistema de Execução de Fabrico da Critical Manufacturing. Esta integração tem como objetivo fornecer aos fabricantes um meio avançado e eficiente de detetar e tratar os defeitos dos produtos.

Os resultados deste estudo indicam que o modelo YOLOv9 apresentou resultados particularmente promissores, alcançando uma elevada precisão na deteção de defeitos no conjunto de dados de deteção da Northeastern University e no conjunto de dados de peças fundidas GDXray. O sistema desenvolvido recebeu feedback positivo dos utilizadores, confirmando a sua facilidade de integração em soluções de Sistema de Execução de Fabrico. Os utilizadores relataram elevados níveis de satisfação com a facilidade de utilização, o desempenho e a eficácia do sistema, destacando o seu potencial para melhorar a eficiência operacional e a gestão de defeitos em ambientes de fabrico.

Acknowledgements

I would like to thank Professor Luís Teixeira for the support and guidance provided during the development of this dissertation.

To Luís Ponte for giving me this opportunity and believing in my potential since the beginning.

To Tiago who helped me every week, even when I was stressed, he reassured me that I would make it until the end.

To my team, Simão, Rita, Daniel and Diogo for taking the time to listen to my existential problems and helping me when I needed.

To José, Rodrigo and Gonçalo that, despite not being from the team, were also a valuable help this semester.

To my family, my father who was the biggest fan of my academic path and always wanted to see what I was up to, and for being the best advisor. My mother for all the headaches but always wanted the best for my future and to bring the best in me. To my sister and brother for annoying me like siblings do but for always being there for me. To Conceição for being there since I was born. And thank you all for handling my bad mood.

To my college friends and a special thanks to Chicos for the last five years of happiness, tears, anger and fun. For always making sure I was okay when I was not.

To my girls Finas for all of these years together and for the unconditional emotional support.

To the friends I made at NIAEFEUP and JuniFEUP for putting me in uncomfortable situations that made me grow to be who I am today, and thank you for everything I learned, belonging to them as a member and a leader.

To my family in Slovenia and, finally, thank you to Balázs for the past year and for being another supporter of my success.

As my father always says: *Concentração, estupidez natural e Zás Pimba*

Beatriz

"Yesterday I was clever, so I wanted to change the world.
Today I am wise, so I am changing myself."

Rumi

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 1
- 1.3 Objectives . . . 2
- 1.4 Document Structure . . . 2

2 Fundamental Concepts and Related Work **4**
- 2.1 Introduction . . . 4
- 2.2 Machine Learning . . . 4
  - 2.2.1 Supervised Learning . . . 5
- 2.3 Deep Learning . . . 5
  - 2.3.1 Artificial Neural Networks . . . 5
  - 2.3.2 Autoencoders . . . 7
  - 2.3.3 Computer Vision . . . 7
  - 2.3.4 Convolutional Neural Networks . . . 8
- 2.4 Defect Identification . . . 10
- 2.5 Anomaly Identification . . . 11
  - 2.5.1 Anomaly Detection Using Autoencoders . . . 11
  - 2.5.2 Difference Between Defect Identification and Anomaly Detection . . . 11
- 2.6 Advanced Object Detection Algorithms . . . 12
  - 2.6.1 Faster R-CNN . . . 12
  - 2.6.2 YOLO . . . 13
- 2.7 Manufacturing Execution System . . . 14
  - 2.7.1 Electronic Failure Catalogue . . . 14
- 2.8 Huawei: AI Quality Inspection at Foxconn . . . 14
- 2.9 Siemens: Data Visualization and AI Quality Control . . . 15
- 2.10 Toyota's AI-driven Quality Control System . . . 15
- 2.11 Summary . . . 15

3 Requirements and Functionalities **17**
- 3.1 Introduction . . . 17
- 3.2 Current Process . . . 17
- 3.3 Proposed Solution . . . 17
  - 3.3.1 Generic AI Training System . . . 17
- 3.4 Use Cases . . . 18
  - 3.4.1 Actors of the System . . . 19
  - 3.4.2 Packages of the System . . . 19
- 3.5 Functional Requirements . . . 22

3.6 Non-Functional Requirements . . . 23
3.7 Summary . . . 25

4 Architecture **26**
4.1 Introduction . . . 26
4.2 MES GUI . . . 26
4.3 Python API for Defect Detection . . . 30
4.4 Backend Communication . . . 30
4.5 Architecture Diagram . . . 30
4.6 Summary . . . 31

5 Implementation **32**
5.1 Introduction . . . 32
5.2 Datasets . . . 32
5.2.1 NEU-DET Dataset . . . 32
5.2.2 GC10-DET Dataset . . . 33
5.2.3 Data Preprocessing . . . 34
5.2.4 Data Augmentation . . . 35
5.3 Models . . . 35
5.3.1 YOLO Model Training . . . 35
5.3.2 Faster R-CNN Model Training . . . 36
5.4 Hyperparameter Tuning . . . 37
5.4.1 Grid Search . . . 37
5.4.2 Random Search . . . 37
5.4.3 Bayesian Optimization . . . 37
5.4.4 Ultralytics `tune` Function . . . 38
5.4.5 Computational Resources and Time . . . 38
5.5 Adam Optimizer . . . 38
5.6 System Integration . . . 39
5.6.1 Python API Development . . . 39
5.6.2 Updating MES Interface . . . 39
5.7 Deployment . . . 40
5.8 Summary . . . 40

6 Results Analysis **41**
6.1 Introduction . . . 41
6.1.1 Used Metrics . . . 41
6.2 Results Analysis . . . 46
6.2.1 One-Stage Object Detection Performance - YOLO . . . 50
6.2.2 Two-Stage Object Detection Performance - Faster R-CNN . . . 57
6.2.3 Performance Metrics . . . 58
6.3 User Feedback . . . 59
6.3.1 Usability . . . 59
6.3.2 Performance . . . 60
6.3.3 Effectiveness . . . 61
6.3.4 Satisfaction . . . 61
6.3.5 Technical Issues . . . 62
6.3.6 Additional Feedback . . . 63
6.3.7 User feedback overview . . . 64

6.4 Discussion . . . 64
6.5 Summary . . . 64

7 Conclusions **66**
7.1 Summary of Findings . . . 66
7.2 Key Findings . . . 66
7.3 Implications and Impact . . . 67
7.4 Future Work . . . 67
7.4.1 Proposed Enhancements . . . 67
7.4.2 Requirements and Advantages . . . 68

References **70**

A Annex **73**
A.1 NEU-DET Model Results . . . 73
A.1.1 YOLOv5m6u . . . 73
A.1.2 YOLOv8m . . . 74
A.1.3 YOLOv8x . . . 75
A.1.4 YOLOv9e . . . 76
A.1.5 Faster R-CNN . . . 77
A.2 GC10-DET Model Results . . . 78
A.2.1 YOLOv5m6u . . . 78
A.2.2 YOLOv8m . . . 79
A.2.3 YOLOv8x . . . 80
A.2.4 YOLOv9e . . . 81
A.2.5 Faster R-CNN . . . 82
A.3 Questionnaire Guide . . . 83
A.3.1 Evaluation Process Overview . . . 83
A.3.2 Getting Started . . . 83
A.3.3 Using the System . . . 83
A.3.4 Observing and Recording . . . 85
A.3.5 Providing Feedback . . . 85
A.4 Questionnaire Form . . . 85
A.4.1 Usability . . . 86
A.4.2 Performance . . . 86
A.4.3 Effectiveness . . . 87
A.4.4 Satisfaction . . . 88
A.4.5 Technical . . . 88
A.4.6 Additional feedback . . . 89

List of Figures

2.1 Artificial Neural Network (ANN) Architecture [24] . . . 7
2.2 Convolutional Neural Network (CNN) Architecture [34] . . . 9
2.3 Illustration of Max Pooling and Average Pooling [38] . . . 10
2.4 The captured images of the metallic surface show challenges in defect detection: (a) Defects with various shapes and sizes, (b1) defects with ambiguous edges and low contrast, (b2) defects with different backgrounds, (b3) fibre, (b4) dust, (b5,b6) scratches. [36] . . . 11
2.5 Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network [29]. . . . 13
2.6 YOLO versions comparison [16] . . . 14

3.1 Transition from the current manual defect detection process to the automated solution. The diagram shows the existing steps of manual defect detection and MES input and the proposed automated process using model training and automatic MES input. . . . 18

4.1 Defect model listing page in the MES GUI . . . 27
4.2 View for testing the defect model . . . 28
4.3 Defect page with the highlighted button for automatic defect creation . . . 28
4.4 Image showing multiple detected defects . . . 29
4.5 Information display for each created defect . . . 29
4.6 Architecture Scheme . . . 31

5.1 Examples of defects of the NEU-DET dataset . . . 33
(a) Crazing . . . 33
(b) NEU-DET Inclusion . . . 33
(c) Patches . . . 33
(d) Pitted Surface . . . 33
(e) Rolled in Scale . . . 33
(f) Scratches . . . 33
5.2 Examples of defects of the GC10-DET dataset . . . 34
(a) Crease . . . 34
(b) Crescent Gap . . . 34
(c) GC10-DET Inclusion . . . 34
(d) Oil Spot . . . 34
(e) Punching Hole . . . 34
(f) Rolled Pit . . . 34
(g) Silk Spot . . . 34
(h) Waist Folding . . . 34

(i) Welding Line . . . 34
(j) Water Spot . . . 34

6.1 Illustration of TP, FP, and FN in defect detection. Box 1 indicates a false positive because it was classified incorrectly as *scratch*. It also represents a false negative because the actual defect, an *inclusion*, was not identified correctly. Box 2 is a false negative as it was not identified. The other boxes are true positives. . . . 43
6.2 Visual analysis of defect instances, bounding box distribution, and bounding box sizes of the NEU-DET dataset. . . . 47
6.3 Heatmaps showing the distribution of x, y coordinates, width, and height of bounding boxes of the NEU-DET dataset. . . . 48
6.4 Visual analysis of defect instances, bounding box distribution, and bounding box sizes. . . . 49
6.5 Heatmaps showing the x, y coordinates, width, and height distribution of bounding boxes. . . . 50
6.6 Precision-Recall curves for YOLOv9, YOLOv5m6u, YOLOv8m, and YOLOv8x models on the NEU-DET dataset. . . . 52
(a) YOLOv9 PR Curve . . . 52
(b) YOLOv5m6u PR Curve . . . 52
(c) YOLOv8m PR Curve . . . 52
(d) YOLOv8x PR Curve . . . 52
6.7 Confusion matrix for YOLOv9 on NEU-DET dataset. . . . 53
6.8 Normalized confusion matrix for YOLOv9 on NEU-DET dataset. . . . 53
6.9 Precision-Recall curves for YOLOv9, YOLOv8x, YOLOv8m, and YOLOv5m6u models on the GC10-DET dataset. . . . 55
(a) YOLOv9 PR Curve . . . 55
(b) YOLOv8x PR Curve . . . 55
(c) YOLOv8m PR Curve . . . 55
(d) YOLOv5m6u PR Curve . . . 55
6.10 Confusion Matrix for the YOLOv9 model on the GC10-DET dataset. . . . 56
6.11 Normalized Confusion Matrix for the YOLOv9 model on the GC10-DET dataset. 56
6.12 Precision-Recall curves for Faster R-CNN on the NEU-DET dataset. . . . 57
6.13 Precision-Recall curves for Faster R-CNN on the GC10-DET dataset. . . . 58
6.14 Usability Feedback . . . 60
6.15 Performance Feedback . . . 60
6.16 Effectiveness Feedback . . . 61
6.17 Satisfaction Feedback . . . 62
6.18 Technical Issues . . . 62

7.1 Simplified defect detection process flow. This diagram illustrates the steps involved in the proposed optimization of the defect detection system. . . . 69

A.1 YOLOv5m6u Model Results for NEU-DET . . . 73
(a) F1 Score Curve . . . 73
(b) Precision Curve . . . 73
(c) Recall Curve . . . 73
A.2 Several Metrics Results for NEU-DET on YOLOv5m6u . . . 74
A.3 YOLOv8m Model Results for NEU-DET . . . 74
(a) F1 Score Curve . . . 74

(b) Precision Curve . . . 74
(c) Recall Curve . . . 74
A.4 Several Metrics Results for NEU-DET on YOLOv8m . . . 75
A.5 YOLOv8x Model Results for NEU-DET . . . 75
(a) F1 Score Curve . . . 75
(b) Precision Curve . . . 75
(c) Recall Curve . . . 75
A.6 Several Metrics Results for NEU-DET on YOLOv8x . . . 76
A.7 YOLOv9e Model Results for NEU-DET . . . 76
(a) F1 Score Curve . . . 76
(b) Precision Curve . . . 76
(c) Recall Curve . . . 76
A.8 Several Metrics Results for NEU-DET on YOLOv9e . . . 77
A.9 Faster R-CNN Model Results for NEU-DET . . . 77
(a) F1 Score Curve . . . 77
(b) Precision Curve . . . 77
(c) Recall Curve . . . 77
A.10 YOLOv5m6u Model Results for GC10-DET . . . 78
(a) F1 Score Curve . . . 78
(b) Precision Curve . . . 78
(c) Recall Curve . . . 78
A.11 Several Metrics Results for GC10-DET on YOLOv5m6u . . . 79
A.12 YOLOv8m Model Results for GC10-DET . . . 79
(a) F1 Score Curve . . . 79
(b) Precision Curve . . . 79
(c) Recall Curve . . . 79
A.13 Several Metrics Results for GC10-DET on YOLOv8m . . . 80
A.14 YOLOv8x Model Results for GC10-DET . . . 80
(a) F1 Score Curve . . . 80
(b) Precision Curve . . . 80
(c) Recall Curve . . . 80
A.15 Several Metrics Results for GC10-DET on YOLOv8x . . . 81
A.16 YOLOv9e Model Results for GC10-DET . . . 81
(a) F1 Score Curve . . . 81
(b) Precision Curve . . . 81
(c) Recall Curve . . . 81
A.17 Several Metrics Results for GC10-DET on YOLOv9e . . . 82
A.18 Faster R-CNN Model Results for GC10-DET . . . 82
(a) F1 Score Curve . . . 82
(b) Precision Curve . . . 82
(c) Recall Curve . . . 82

List of Tables

3.1 Actors of the System 19
3.2 Packages of the System 19
3.3 UC.001 - Upload Training Dataset 19
3.4 UC.002 - Train Model 20
3.5 UC.003 - Integrate with MES 20
3.6 UC.004 - Detect Defects in Product Images 21
3.7 UC.005 - View Defect Detection Results 21
3.8 REQ.001 - Model Training Interface 22
3.9 REQ.002 - Real-Time Defect Detection 22
3.10 REQ.003 - Integration with MES 23
3.11 REQ.004 - Defect Detection Visualization 23
3.12 NFR.001 - Performance 23
3.13 NFR.002 - Usability 24
3.14 NFR.003 - Scalability 24
3.15 NFR.004 - Reliability 24
3.16 NFR.005 - Maintainability 25

6.1 Confusion matrix typical representation 44
6.2 Performance metrics on the NEU-DET dataset 58
6.3 Performance metrics on the GC10-DET dataset 59

Abbreviations

ADAM	Adaptive Moment Estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
COTS	Commercial Off-The-Shelf
CNN	Convolutional Neural Network
CM	Critical Manufacturing
DIP	Dependency Inversion Principle
EFC	Electronic Failure Catalogue
FP	False Positive
FN	False Negative
GUI	Graphical User Interface
IoU	Intersection over Union
ISP	Interface Segregation Principle
LSP	Liskov Substitution Principle
mAP	Mean Average Precision
MES	Manufacturing Execution System
ML	Machine Learning
NEU-DET	Northeastern University Detection Dataset
NFR	Non-Functional Requirement
OCP	Open-Closed Principle
OEE	Overall Equipment Effectiveness
PR	Precision-Recall
REQ	Requirement
ReLu	Rectified Linear Activation Function
RPN	Region Proposal Network
R-CNN	Region-based Convolutional Neural Network
ROI	Regions of Interest
SOLID	Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Dependency Inversion Principles
SRP	Single Responsibility Principle
SSD	Single Shot MultiBox Detector
TP	True Positive
UC	Use Case
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
VGG16	Visual Geometry Group Network (16 layers)
YOLO	You Only Look Once
GC10-DET	GDXray Castings Dataset

Chapter 1

Introduction

1.1 Context

The manufacturing industry is continuously evolving, with a constant drive to enhance operational efficiency and product quality. One of the most critical aspects of maintaining high production standards is the identification and rectification of defects within the production line. Traditionally, this task has been performed manually, making it susceptible to human error due to its complexity and repetitive nature. The need for a more efficient and accurate solution has led to the exploration of automated methods for defect detection.

Manufacturing Execution Systems (MES) are sophisticated computerized systems that play a pivotal role in tracking and documenting the transformation of raw materials into finished products. These systems provide real-time data that is crucial for making informed decisions on the production floor, thereby optimizing manufacturing processes.

Within the Critical Manufacturing (CM) MES, the Electronic Failure Catalogue (EFC) is a feature that allows users to define and link images to defects identified during production. However, the current system still relies on manual input, which can be time-consuming and prone to errors. This research aims to take the EFC to the next level by leveraging machine learning (ML) techniques to enable real-time, automated defect identification from product images, thereby improving production efficiency and accuracy.

1.2 Motivation

The primary motivation for this research originates from the critical need to improve the speed and accuracy of defect identification in manufacturing. Inaccurate defect identification not only affects production efficiency but also compromises product quality, leading to increased costs and potential customer dissatisfaction. The integration of ML into the defect identification process presents a transformative opportunity to address these challenges.

ML technologies have shown great potential in various applications, yet their integration into Commercial Off-The-Shelf (COTS) MES poses significant challenges due to different industries'

diverse settings and requirements. This research seeks to bridge this gap by developing a user-friendly artificial intelligence (AI) training system that allows non-technical users to train models using images of defects. This system aims to be adaptable across different industries and product types, making advanced AI technologies accessible to a broader range of users.

The current MES workflow involves operators identifying defects in materials, uploading photos of these defects, marking their locations, and classifying them. This process has resulted in a vast dataset of products with identified defects, which is ideal for training supervised learning models that are particularly suitable for this task as they provide high accuracy in identifying specific types of defects.

This research will explore various supervised learning techniques to determine the most effective manufacturing defect identification approach to automate the described flow.

1.3 Objectives

The primary goals of this research are as follows:

- **Development of a Generic AI Training System** - Conceptualize and design a proof-of-concept for an AI training system capable of generating defect detection models.
- **Generation of Defect Detection Models** - Utilize a set of known images illustrating various defects as training data. Given the current workflow in MES, where operators manually photograph, identify and classify defects, this study will primarily focus on supervised learning methods. These methods involve training models with labelled datasets, which are particularly suitable for this task as they provide high accuracy in identifying specific types of defects.
- **Evaluation of the Amount of Training Data for High Performance** - Determine the volume of training data required to achieve a high defect identification performance.
- **Evaluation of the Applicability Across Industries and Products** - Investigate the reliability of the developed defect identification model across diverse industries and a range of products.
- **Integration for Advanced Defect Detection** - Evaluate the technology's readiness for integration into commercially available MES solutions.

These objectives form a comprehensive roadmap for this research, aiming to develop a robust and adaptable solution for automated defect identification in manufacturing.

1.4 Document Structure

This document is structured into eight chapters, each addressing a key aspect of the research. Chapter 2 provides a review of fundamental concepts related to machine learning, deep learning,

and MES. Chapter 3 discusses related work in the field of defect identification. Chapter 4 outlines the requirements and functionalities of the proposed solution. Chapter 5 details the system architecture, while Chapter 6 describes the implementation process. Chapter 7 presents the results and analysis of the research, and Chapter 8 concludes with the final conclusions of this study and a discussion of future work and potential improvements.

Chapter 2

Fundamental Concepts and Related Work

2.1 Introduction

This chapter explores the foundational concepts that underpin this research, focusing on machine learning, deep learning, computer vision, and their applications in defect identification within Manufacturing Execution Systems. Understanding these concepts is crucial for appreciating the technical challenges and solutions presented in this research.

2.2 Machine Learning

Machine learning is a subset of artificial intelligence that enables computers to learn from and make decisions based on data. It involves the use of algorithms to parse data, learn from it, and make predictions or decisions without being explicitly programmed for specific tasks.

Arthur Lee Samuel, a trailblazer in the field, characterized machine learning as the area of study that empowers computers to learn from data and improve their performance over time without being explicitly programmed [32]. This ability is achieved through two main phases: training and testing. During the training phase, the algorithm processes a large dataset to identify patterns and adjust its internal parameters. The testing phase evaluates the model's performance on new, unseen data to ensure it generalizes well to real-world scenarios.

Machine learning can be categorized into several types based on the nature of the learning process:

- **Supervised Learning**: The algorithm is trained on a labelled dataset, where each input is paired with the correct output. The goal is to learn a mapping from inputs to outputs that can be applied to new data.
- **Unsupervised Learning**: The algorithm is trained on an unlabelled dataset, where it must find patterns and relationships in the data without explicit guidance.

- **Semi-Supervised Learning**: Combines labelled and unlabelled data to improve learning accuracy.
- **Reinforcement Learning**: The algorithm learns by interacting with an environment and receiving feedback in the form of rewards or penalties.

This research focuses on supervised learning due to its applicability in defect detection, where labelled examples of defects are used to train the model.

2.2.1 Supervised Learning

As already mentioned, supervised learning involves training a model on a labelled dataset. The model learns to map inputs to outputs by identifying patterns in the data. The goal is to make accurate predictions or classifications when presented with new, unseen data.

In the context of defect detection, supervised learning involves training a model using images of defective products whose classification of these defects is known. The model learns to distinguish between different types of defects based on the features extracted from these images. This approach ensures high accuracy in identifying defects, as the model is guided by labelled examples during the training process.

2.3 Deep Learning

Deep learning, a subset of machine learning, uses artificial neural networks to solve complicated tasks. The related viewpoint regarding deep learning models is that they are engineered systems inspired by the structure and functioning of the biological brain [10] and they are particularly well-suited for tasks such as image and speech recognition, natural language processing, and more.

Deep learning automatically extracts valuable insights from data, revealing hidden connections and complexities that humans might miss. The term "deep" refers to the multiple layers (three or more), or depth, in these neural networks.

According to Bhatt (2021, 2), this technique "has proved to be exceptionally successful in object detection and classification, facial detection, pattern recognition, fault diagnosis, target tracking, and a wide variety of other image-based applications" [4]. The ability of deep learning models to extract features from raw data automatically makes them highly effective for defect detection in manufacturing.

2.3.1 Artificial Neural Networks

Artificial Neural Networks (ANNs), drawing inspiration from the complex workings of the human brain, serve as a network of interconnected neurons, mimicking the neural connections in biological systems. The fundamental objective of an ANN is to generate an output pattern in response to a given input pattern [30].

2.3.1.1 Architecture

The fundamental architecture of an ANN, illustrated in Figure 2.1, consists of the following key components:

- **Input Layer** - These layers receive the data or features. Each node represents an input feature, and the layer serves as the entry point for information into the network. According to Baheti (2021), "it is the only visible layer in the complete Neural Network architecture that passes the complete information from the outside world without any computation" [3].
- **Hidden Layers** - Located between the input and output layers, one or more of these hidden layers process the input data. According to Baheti (2021), "they are intermediate layers that do all the computations and extract the features from the data" [3].
- **Weights and Connections** - Each connection between neurons has an associated weight, signifying the strength of the connection. Training entails dynamically optimizing the connections within the network through weight updates, leading to improved performance capabilities.
- **Activation Function** - Neurons perform a non-linear transformation on the weighted sum of their inputs using an activation function. This adds complexity to the model, enabling it to capture more intricate patterns and features compared to linear models. According to Baheti (2021), "without this, the output would be a linear combination of input values and would not be able to introduce non-linearity in the network" [3].
- **Output Layer** - Aggregates the processed information from the hidden layers and generates the final result or prediction. Its architecture, specifically the number of nodes, adapts to the type of task, such as classification with multiple categories or regression for continuous values.
- **Bias** - Neurons have a special setting (bias), letting them adjust their sensitivity to information and learn even trickier connections between things. According to Baheti (2021), "the role of bias is to shift the value produced by the activation function" [3].

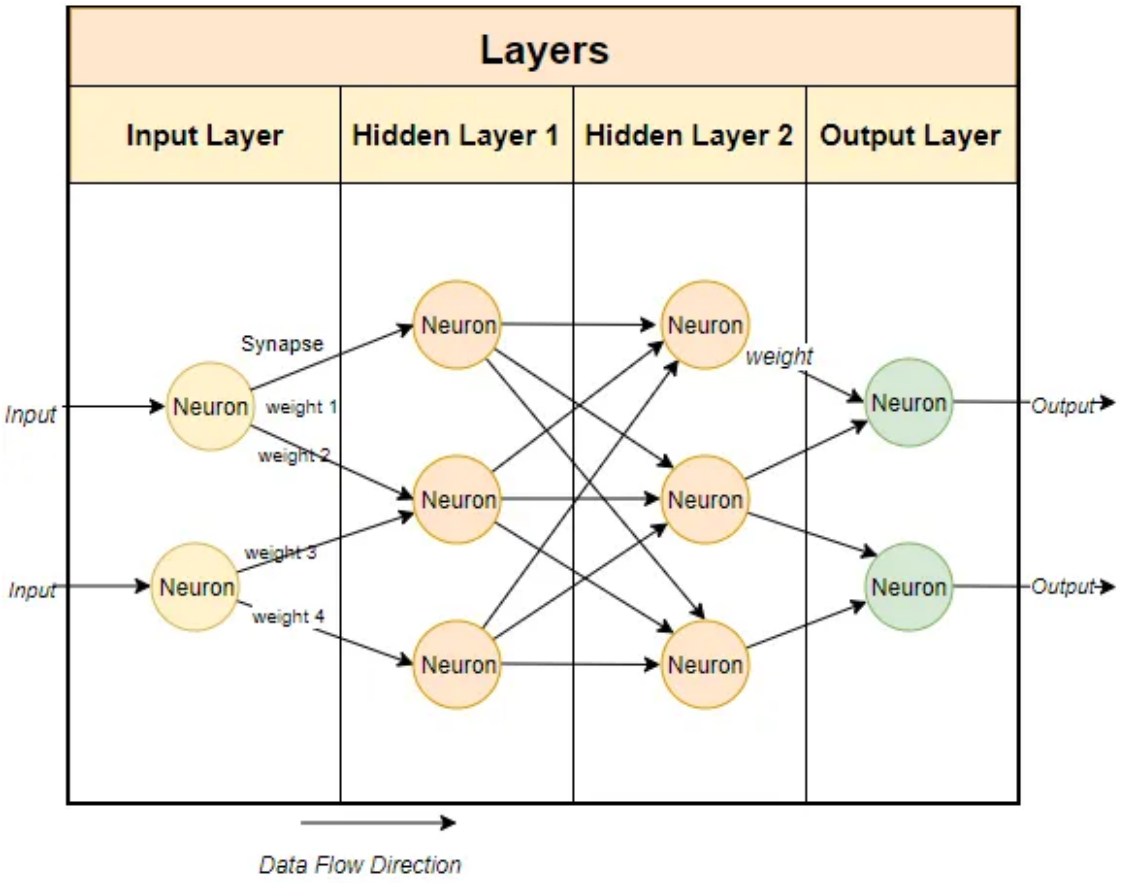


Figure 2.1: Artificial Neural Network (ANN) Architecture [24]

2.3.2 Autoencoders

Autoencoders are a type of ANNs used to learn efficient input data codings, typically for dimensionality reduction or anomaly detection.

2.3.2.1 Structure of Autoencoders

An autoencoder comprises an encoder and a decoder. The encoder converts the input data into a compressed form in a latent space, while the decoder reconstructs the input data based on this representation [7]. The objective is to minimize the difference between the input and reconstructed data.

2.3.3 Computer Vision

Computer vision is a field of study that focuses on enabling machines to interpret and make decisions based on visual data. It involves various techniques for processing and analyzing images and videos to extract meaningful information.

In the context of deep learning, computer vision systems leverage advanced neural network architectures to achieve high precision and accuracy in tasks such as object detection, image classification, and defect identification. These systems can analyze visual data at a fine level, detecting irregularities that may be missed by human inspectors.

2.3.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have appeared as a powerful and specialized architecture within the domain of deep learning, particularly tailored for tasks related to computer vision.

A CNN features a convolutional layer that employs the convolution technique. As Sharma (2023) proposed, this is a mathematical operation applied to two functions, producing a third function. This third function illustrates how the shape of one function is influenced and modified by the other [34].

In the context of computer vision, a convolutional kernel or filter is applied to an input image, generating activation maps that highlight pertinent features within the provided input image. According to Sharma (2023), "the deeper one moves into the convolutional neural network, the more the layers start detecting various higher-level features such as objects, faces, etc [34].

Following the described layer, a pooling layer takes place. The objective of this layer is to lower the spatial dimensions of the Convolved Feature, and there are two paths to achieve this: max polling (section 2.3.4.1) or average polling (section 2.3.4.2). By doing this, the computational complexity of the network is reduced.

Toward the end of the network, the fully connected layers take place. It is responsible for classification, utilizing features extracted by previous layers and their diverse filters. Figure 2.2 depicts the architecture of a typical CNN.

Convolutional and pooling layers typically employ ReLu functions (Rectified Linear Activation Function - a linear function that will return the input directly if it is positive; if not, the output is zero). Fully-connected layers commonly use a softmax activation function for accurate classification, generating probabilities from 0 to 1.

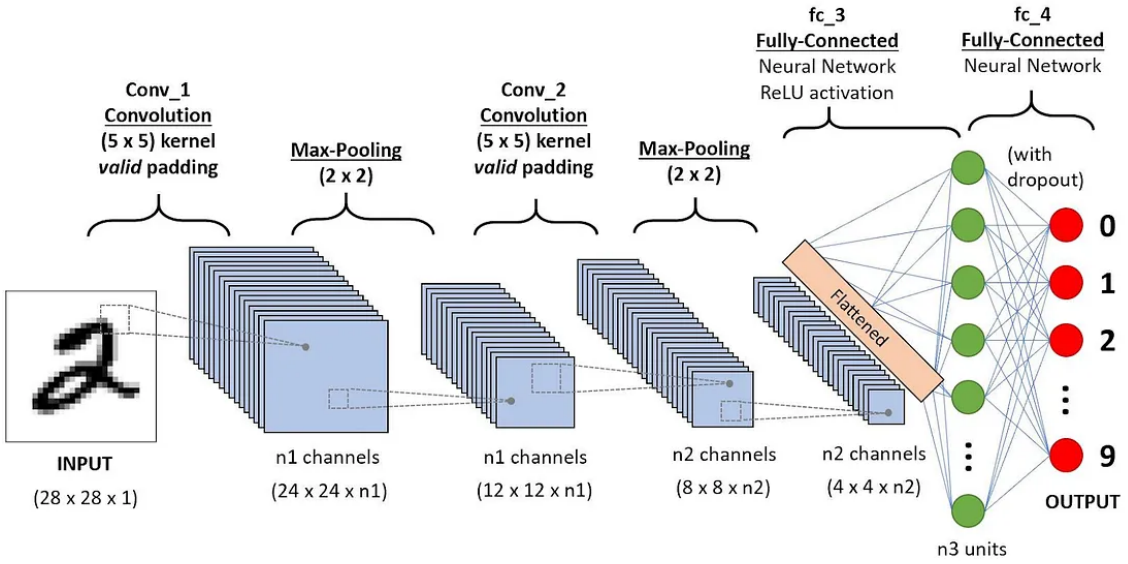


Figure 2.2: Convolutional Neural Network (CNN) Architecture [34]

2.3.4.1 Max Polling

This method involves taking the maximum value from a group of neighbouring pixels. The maximum value is retained in a specified region, and the rest are discarded. The illustration in Figure 2.3 demonstrates max pooling with a 2x2 pool size. In the max pooling example, the largest value from each 2x2 block is retained, resulting in a reduced output matrix. Specifically, the 2x2 windows in the input matrix contain values, and the maximum values selected from each window form the smaller output matrix. This process accentuates the most prominent features and helps reduce dimensionality.

2.3.4.2 Average Polling

Average pooling calculates the average value from a group of neighbouring pixels. As shown in Figure 2.3, with a 2x2 pool size, average pooling computes the average of all pixel values within each 2x2 block. The average values from these blocks form the smaller output matrix. This method summarizes the information within each block, helping to reduce dimensionality while preserving the overall structure and patterns within the feature maps.

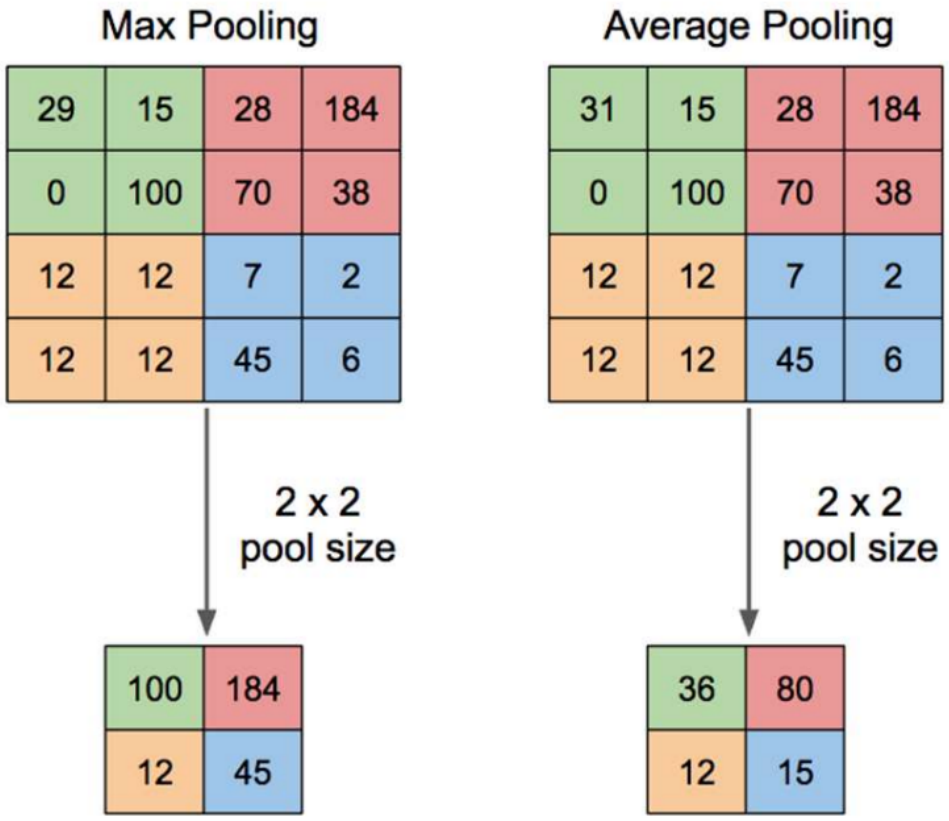


Figure 2.3: Illustration of Max Pooling and Average Pooling [38]

2.4 Defect Identification

Defect identification in manufacturing is a critical process for maintaining product quality and operational efficiency. It involves detecting and classifying imperfections or irregularities in products during the production cycle. Early detection of defects helps to minimize costs associated with defective products, such as rework, scrap, and warranty claims, and enhances customer satisfaction by ensuring high-quality products.

Minimizing defects also contributes to environmental sustainability. Producing fewer defective items reduces waste and boosts resource efficiency.

Defect identification systems usually employ image processing techniques such as edge detection, grayscale thresholding, and image segmentation. These techniques allow the system to highlight and analyze regions of interest in an image, boosting the detection of defects. However, the challenge lies not only in detecting defects but also in accurately identifying the type and severity of each defect. Figure 2.4 illustrates the various challenges in defect detection on a metallic surface, showing defects of different shapes and sizes, ambiguous edges and low contrast, and varying backgrounds. Specific examples include glue marks, scratches, dust, fibres, and

damaged spots, highlighting the complexity and diversity of defects that need to be identified in a manufacturing setting.

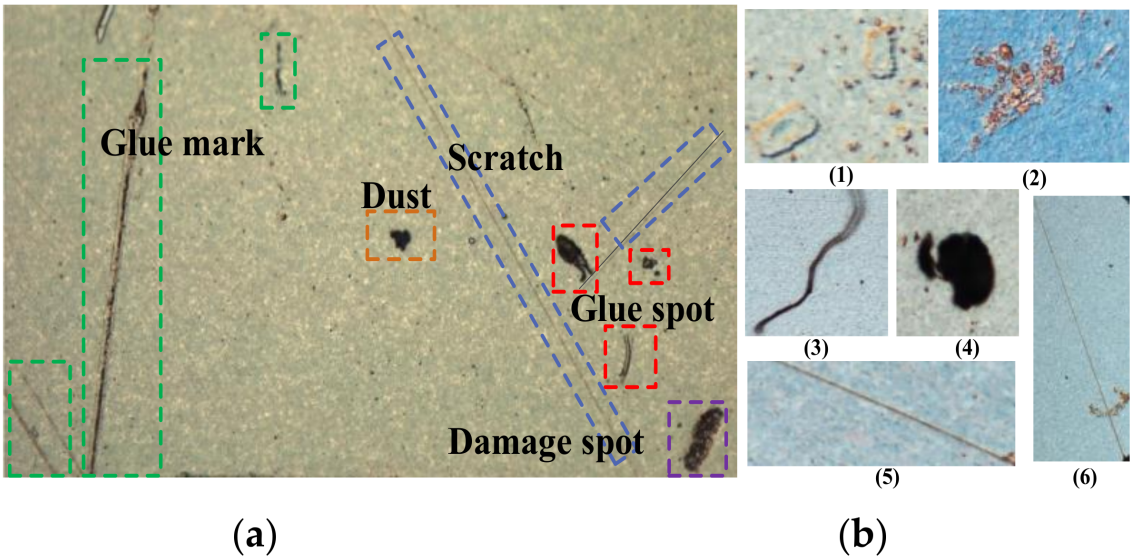


Figure 2.4: The captured images of the metallic surface show challenges in defect detection: (a) Defects with various shapes and sizes, (b1) defects with ambiguous edges and low contrast, (b2) defects with different backgrounds, (b3) fibre, (b4) dust, (b5,b6) scratches. [36]

2.5 Anomaly Identification

Anomaly identification, or anomaly detection, involves identifying patterns in data that do not conform to expected behaviour [23]. Anomalies are rare events or observations that raise doubts by varying significantly from the majority of the data. Although anomaly detection is not explored in this study, it is proposed as a future solution, and it is important to understand its principles and significance.

2.5.1 Anomaly Detection Using Autoencoders

Autoencoders can be used for anomaly detection by training them on normal data. During this process, they learn to reconstruct normal patterns accurately. When an irregular data point, in this context, an unknown defect, is fed into the autoencoder, it fails to reconstruct it accurately, resulting in a high reconstruction error. This high error can be used to flag the data point as an anomaly.

2.5.2 Difference Between Defect Identification and Anomaly Detection

While defect identification focuses on detecting specific, known defects in products, anomaly detection is about identifying unusual patterns that may signify unknown or unexpected defects.

Defect identification relies on labelled data to train models to recognize specific defects, whereas anomaly detection frequently works with unlabeled data to find deviations from normal patterns.

2.6 Advanced Object Detection Algorithms

In the realm of defect detection within manufacturing, advanced object detection algorithms play a crucial role. These algorithms are designed to identify and classify objects within images, making them highly suitable for detecting various types of defects in manufacturing processes. This section will explore two prominent object detection algorithms, Faster R-CNN and YOLO, which have been adopted for their accuracy and efficiency in real-time applications.

2.6.1 Faster R-CNN

Faster R-CNN (Region-based Convolutional Neural Network) is a state-of-the-art object detection algorithm that has gained significant traction in defect identification applications. It combines the precision of region-based methods with the speed of convolutional neural networks to achieve high accuracy in localizing and classifying defects.

The Faster R-CNN architecture includes a Region Proposal Network (RPN) that generates candidate regions of interest (ROIs) within an image. These ROIs are then passed to the region classification network (R-CNN) [28], which accurately classifies each ROI as either containing a defect or not. This makes this solution a Two-Stage Object Detector. Its precise localization enables the system to pinpoint the exact location of defects, facilitating efficient removal and rework processes.

As illustrated in Figure 2.5, Faster R-CNN is a unified network where the RPN module serves as the ‘attention’ mechanism, improving the overall object detection capability [29].

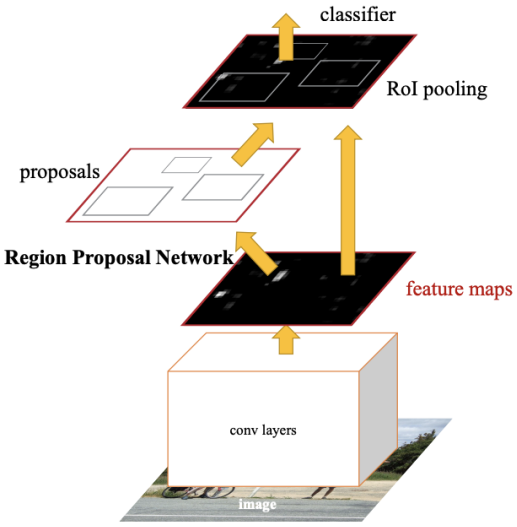


Figure 2.5: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network [29].

2.6.2 YOLO

YOLO (You Only Look Once) [5] is a well-known and effective object detection model, known for its real-time object detection performance. YOLO is a Single-Stage Object Detector [6], which is a class of object detection architectures that are one-stage, contrary to faster R-CNN. Unlike traditional region-based methods that process an image in multiple stages, YOLO applies a single neural network to the entire image, dividing it into a grid overlay on the image. Each grid cell simultaneously predicts the location (bounding box) and the type (class probability) of any objects present within that cell. According to Kundu (2023), "one of the main advantages of YOLO is its fast inference speed, which allows it to process images in real time" [19]. This makes it particularly suitable for applications where quick decision-making is critical, such as defect detection in manufacturing.

YOLO's modular architecture allows easy customization and integration with other machine learning and image processing techniques, providing flexibility in expanding the capabilities of the defect identification system. Therefore, using this model, "you only look once" at an image to predict what objects are present and where they are.

The most recent versions of YOLO, including YOLOv8 [16] and experimental YOLOv9 [37], mark significant advancements in object detection technology, offering improved accuracy, performance, and general applicability. Figure 2.6 shows two graphs comparing the latency and parameters of different YOLO object detection models. Compared to the previous versions, the performance of YOLOv8 pre-trained on the COCO dataset (Common Objects in Context - a large-scale dataset for object detection, segmentation, and captioning tasks in computer vision) is the

fastest and most efficient. While YOLOv9 is still in the experimental stages, it shows promise for even better performance [16].

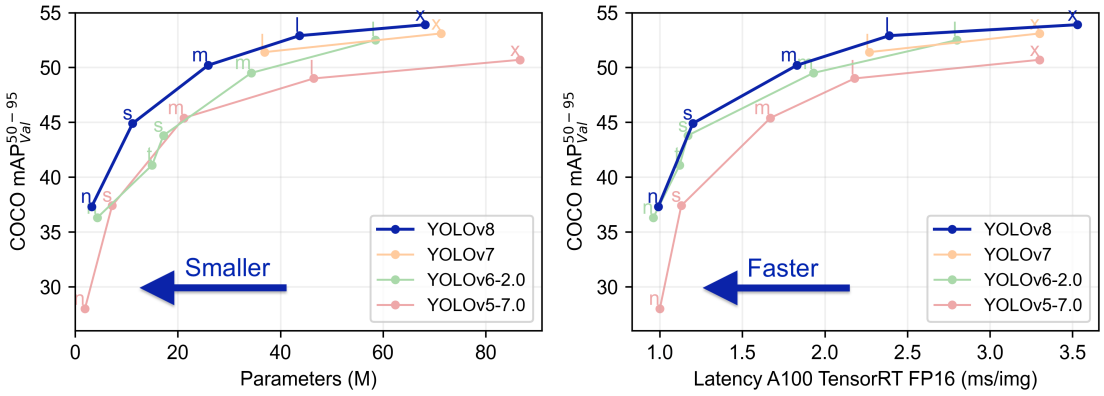


Figure 2.6: YOLO versions comparison [16]

2.7 Manufacturing Execution System

A Manufacturing Execution System (MES) is a complete software solution designed to manage and control manufacturing operations on the shop floor. MES systems track and document the transformation of raw materials into finished products, providing real-time data to optimize production processes [31].

Integrating machine learning-based defect detection into an MES can significantly improve product quality and operational efficiency. By automating the detection and classification of defects, manufacturers can reduce the need for manual inspections, lower production costs, and prevent the distribution of faulty products. This integration ensures compliance with quality standards, accelerates time-to-market and improves overall equipment effectiveness (OEE).

2.7.1 Electronic Failure Catalogue

Within the Critical Manufacturing MES, the Electronic Failure Catalogue (EFC) allows operators to define and link images to defects identified during production. Operators upload photos of defects, mark their locations, and classify them, creating a vast dataset of labelled defect images. This dataset is ideal for training supervised learning models, allowing the development of automated defect detection systems.

2.8 Huawei: AI Quality Inspection at Foxconn

Huawei's AI Quality Inspection system, deployed at Foxconn, demonstrates AI's potential to revolutionise manufacturing quality control. The system uses advanced machine learning algorithms

to perform real-time inspections of products on the production line. This approach minimizes human error and boosts inspection efficiency. The AI system can detect several defects by analysing vast amounts of data, ensuring that only high-quality products proceed through manufacturing. It leverages deep learning techniques to continuously improve its detection capabilities, adapting to new defect patterns over time. This adaptability is crucial for maintaining high standards in fast-paced manufacturing environments. Moreover, the system's ability to provide actionable insights helps in preemptively addressing potential quality issues, further streamlining the production process [13].

2.9 Siemens: Data Visualization and AI Quality Control

Siemens, in collaboration with Cybord, has developed an AI-based quality control system that combines data visualization with machine learning algorithms. This system captures images from Surface Mount Technology (SMT) assembly machines to ensure full traceability and real-time detection of defective or counterfeit parts. The integration of AI and data visualization enables manufacturers to achieve higher reliability and quality in their production lines.

The Siemens system focuses on identifying defects at the component level, which is critical in preventing defective parts from progressing through the assembly process. By using AI to analyze visual data, the system can detect anomalies, reducing the risk of costly recalls and ensuring that only components meeting strict quality standards are used.

In addition to defect detection, the Siemens system provides visualizations that help engineers and quality control professionals understand the nature and root causes of defects [35].

2.10 Toyota's AI-driven Quality Control System

Predictive analytics, a key component of AI in quality control, enables the anticipation of potential quality issues before they occur. This proactive approach helps maintain consistent product quality and minimize rework costs. Real-time monitoring and automated inspections further streamline the production process, enhancing overall efficiency. Case studies highlighted by Redress Compliance include Toyota's AI-driven quality control system, which achieved a 98% accuracy rate in defect detection, and Samsung's computer vision technology, which detected surface defects with a 95% accuracy rate [8].

2.11 Summary

This chapter covered the foundational concepts essential for this research, focusing on machine learning, deep learning, computer vision, and their applications in defect identification within MES.

It began with an introduction to machine learning, discussing its types: supervised, unsupervised, semi-supervised, and reinforcement learning. Supervised learning, particularly relevant for

defect detection, was emphasized. Deep learning was then explored, highlighting its use of artificial neural networks for complex tasks. The architecture of ANNs was detailed, and autoencoders for dimensionality reduction and anomaly detection were introduced.

Two prominent object detection algorithms were reviewed - Faster R-CNN and YOLO - each offering unique advantages for defect identification in industrial applications. Faster R-CNN provides precise localization through its two-stage detection process. YOLO excels in real-time detection due to its single-stage architecture, making it suitable for applications requiring quick decision-making. This comparative analysis sets the foundation for selecting and implementing the most appropriate model for specific defect detection requirements in manufacturing.

Furthermore, computer vision was discussed as a critical field for interpreting visual data, with CNNs being particularly effective for defect identification. Techniques such as edge detection, grayscale thresholding, and image segmentation were mentioned as key methods for identifying and classifying defects.

The challenges in defect detection were highlighted using Figure 2.4, which shows various types of defects on a metallic surface. Anomaly identification was introduced as a complementary approach to defect detection, focusing on identifying unusual patterns in data. The differences between defect identification and anomaly detection were clarified, emphasizing the use of labelled data for the former and unlabeled data for the latter.

Finally, MES was discussed as a comprehensive solution for managing manufacturing operations, with a focus on integrating machine learning for defect detection. The CM MES and its Electronic Failure Catalogue were provided as examples of effective integration.

Lastly, key implementations of AI-based defect detection systems in the manufacturing industry were explored. These examples illustrate how advanced AI technologies can significantly enhance quality control processes, boost efficiency, and ensure higher product standards in manufacturing environments.

This chapter set the stage for further exploration of these technologies and their applications in the following chapters.

Chapter 3

Requirements and Functionalities

3.1 Introduction

This chapter outlines the requirements and functionalities essential for developing a comprehensive defect detection system. It covers the current manual processes, the proposed automated solutions leveraging ML techniques, and detailed use cases to illustrate system interactions. Additionally, it defines the functional and non-functional requirements necessary to ensure the system meets operational needs.

3.2 Current Process

The current defect detection process involves manual steps where an operator captures images, manually draws defect boxes and identifies defects, and inputs the defect box and reason into the MES.

3.3 Proposed Solution

The proposed solution aspires to address the challenges outlined in the context section 1.1 by developing a complete and adaptable defect identification system. This system will leverage ML techniques to automate defect detection processes within manufacturing environments. The transition from the current manual defect detection process to the automated solution is illustrated in Figure 3.1. The diagram shows the existing steps of manual defect detection and MES input and the proposed automated process using model training and automatic MES input.

3.3.1 Generic AI Training System

The main principle of the proposed solution is the development of a generic AI training system. This system will provide an interface for non-technical users to train machine learning models for defect identification and classification. This component aims to establish access to advanced AI technologies within manufacturing settings by simplifying the model training process.

The system will generate defect identification models by utilizing a diverse dataset of known images depicting various defects. These models will be trained using supervised learning techniques, with labeled training sets to distinguish between various types of defects and classify them according to the corresponding reasons.

Leveraging the existing electronic failure and defect registration capabilities within MES platforms, the solution will provide manufacturers with an advanced and efficient means of detecting and addressing product defects. This integration will improve production processes' overall quality and efficiency, contributing to improved operational performance.

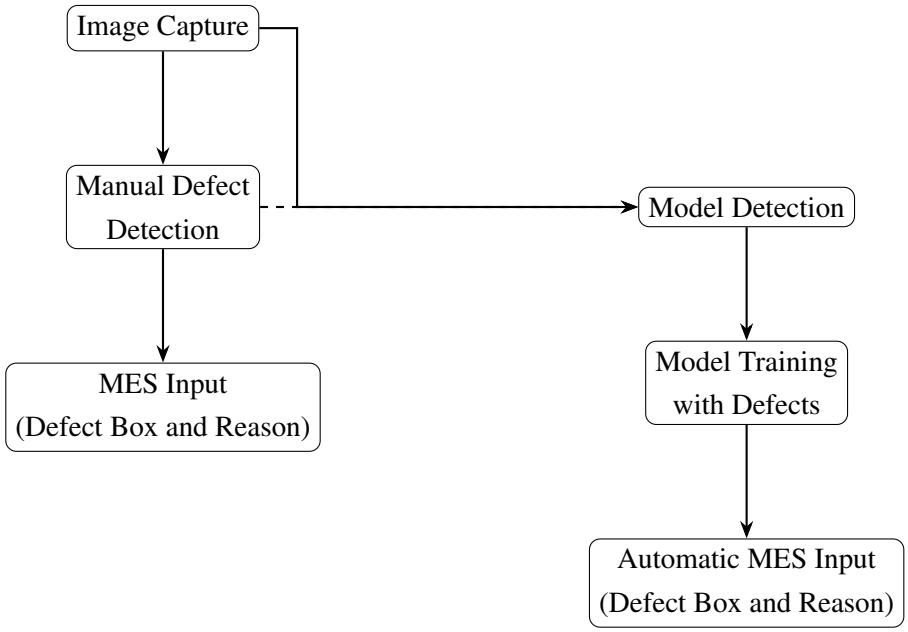


Figure 3.1: Transition from the current manual defect detection process to the automated solution. The diagram shows the existing steps of manual defect detection and MES input and the proposed automated process using model training and automatic MES input.

It will be designed with scalability and adaptability in mind, allowing for coherent integration into diverse manufacturing environments. Modular architecture and flexible deployment options will enable the system to accommodate varying industry requirements and evolving production landscapes.

3.4 Use Cases

The use cases are structured to describe the interactions between the actors (users) and the system to achieve specific goals. Each use case includes an ID, Name, Objective, Description, Preconditions, and Normal Flow. The primary actor is the Operator, who interacts with the MES to manage defect detection processes.

3.4.1 Actors of the System

The actors of the system represent the individuals or entities that interact with the defect detection system. Table 3.1 outlines the main actor involved:

Table 3.1: Actors of the System

Actor	**Description**
Operator	Individuals using the system, typically operators performing tasks that require simultaneous handling of multiple processes within the MES.

3.4.2 Packages of the System

The system is divided into several packages, each responsible for different functionalities. Table 3.2 describes these packages:

Table 3.2: Packages of the System

Package	**Description**
Model Training	Processes associated with training the defect detection models.
MES Integration	Interfacing the defect detection system with the MES platform.
Defect Detection	Involves all processes related to detecting defects in products.

3.4.2.1 Package: Model Training

This package involves cases studied to train the defect detection models using uploaded datasets. Table 3.3 describes the use case for uploading training datasets:

Table 3.3: UC.001 - Upload Training Dataset

Use Case ID	UC.001
Name	Upload Training Dataset
Objective	Allow operators to upload datasets for training the YOLO model.
Description	Operators upload labelled images of products, specifying defect types.
Pre-conditions	The system must have the capability to accept and process dataset uploads.

Normal Flow	1. The operator uploads a dataset of labelled images. 2. The system validates the dataset format. 3. The system stores the dataset for training purposes.

Table 3.4 outlines the use case for training the model using the uploaded datasets:

Table 3.4: UC.002 - Train Model

Use Case ID	UC.002
Name	Train Model
Objective	Train the model using the uploaded datasets.
Description	The system uses the uploaded dataset to train the model for defect detection.
Pre-conditions	A valid dataset must be uploaded to the system in the required model format.
Normal Flow	1. The operator initiates the training process. 2. The system processes the dataset and trains the model. 3. The system saves the trained model for future use.

3.4.2.2 Package: MES Integration

This package includes use cases for integrating the defect detection system with the MES platform. Table 3.5 describes the use case for integrating the system with the MES:

Table 3.5: UC.003 - Integrate with MES

Use Case ID	UC.003
Name	Integrate with MES
Objective	Seamlessly integrate the defect detection system with the MES platform.
Description	The system interfaces with the MES to provide real-time defect detection capabilities.
Pre-conditions	The MES platform must support integration with external systems.

Normal Flow	1. The system establishes a connection with the MES platform. 2. The system sends defect detection results to the MES. 3. The MES displays the defect detection results in relevant workflows.

3.4.2.3 Package: Defect Detection

This package includes use cases for detecting defects in products using the trained model. Table 3.6 describes the use case for detecting defects in product images:

Table 3.6: UC.004 - Detect Defects in Product Images

Use Case ID	UC.004
Name	Detect Defects in Product Images
Objective	Use the model to identify defects in product images.
Description	When an operator uploads a product image, the system detects defects and highlights them.
Pre-conditions	The model must be trained and available for use.
Normal Flow	1. The operator uploads a product image to the system. 2. The system processes the image using the YOLO model associated with the product. 3. The system highlights detected defects and displays the results to the operator.

Operators need to view the results of defect detection for effective decision-making. Table 3.7 outlines the use case for viewing defect detection results:

Table 3.7: UC.005 - View Defect Detection Results

Use Case ID	UC.005
Name	View Defect Detection Results
Objective	Allow operators to view the results of defect detection.

Description	The system displays the detected defects on the product image along with details such as defect type and confidence level.
Pre-conditions	Defects must be detected in the product image.
Normal Flow	1. The operator navigates to the results page. 2. The system displays the product image with highlighted defects. 3. The operator can view details about each detected defect.

3.5 Functional Requirements

The functional requirements define the essential behaviours and capabilities of the system. Each requirement is specified with an ID, Name, Priority, Description, and Motivation. Tables 3.8 to 3.11 describe in detail these requirements.

Table 3.8: REQ.001 - Model Training Interface

Requirement ID	REQ.001
Name	Model Training Interface
Priority	Essential
Description	Provide an interface for operators to upload datasets and train defect detection models.
Motivation	To enable non-technical users to create and update defect detection models.

Table 3.9: REQ.002 - Real-Time Defect Detection

Requirement ID	REQ.002
Name	Real-Time Defect Detection
Priority	Essential
Description	The system must detect defects in real time during the manufacturing process.
Motivation	To ensure immediate identification and rectification of defects in the production line.

Table 3.10: REQ.003 - Integration with MES

Requirement ID	REQ.003
Name	Integration with MES
Priority	Essential
Description	Ensure integration with the MES platform to display defect detection results.
Motivation	To provide operators with real-time defect information within their existing workflows.

Table 3.11: REQ.004 - Defect Detection Visualization

Requirement ID	REQ.004
Name	Defect Detection Visualization
Priority	Essential
Description	Visualize detected defects on product images, highlighting defect areas with details like the type of defect.
Motivation	To help operators easily identify and understand detected defects.

3.6 Non-Functional Requirements

Non-functional requirements specify the criteria used to judge the operation of a system, such as performance, usability, reliability, scalability and maintainability. Tables 3.12 to 3.16 describe in detail these requirements.

Table 3.12: NFR.001 - Performance

Requirement ID	NFR.001
Name	Performance
Priority	High
Description	The system must perform defect detection and model training efficiently without significant delays.
Motivation	To ensure the system does not delay manufacturing operations.

Table 3.13: NFR.002 - Usability

Requirement ID	NFR.002
Name	Usability
Priority	High
Description	The system should have a user-friendly interface that is easy to navigate and use.
Motivation	To ensure operators can efficiently use the system without extensive training.

Table 3.14: NFR.003 - Scalability

Requirement ID	NFR.003
Name	Scalability
Priority	Medium
Description	The system should be scalable to handle increased data volumes and more complex models.
Motivation	To accommodate growth and changes in manufacturing processes.

Table 3.15: NFR.004 - Reliability

Requirement ID	NFR.004
Name	Reliability
Priority	High
Description	The system should be reliable, providing consistent performance with minimal downtime.
Motivation	To ensure continuous defect detection capabilities in the manufacturing process.

Table 3.16: NFR.005 - Maintainability

Requirement ID	NFR.005
Name	Maintainability
Priority	Medium
Description	The system should be easy to maintain and update, with clear documentation and support.
Motivation	To ensure long-term usability and adaptability of the system.

3.7 Summary

In this chapter, the requirements and functionalities essential for developing a comprehensive defect detection system were outlined. Firstly, by examining the current manual processes, followed by proposing an automated solution leveraging ML techniques. Detailed use cases were provided to illustrate system interactions and the roles of various actors. Additionally, functional and non-functional requirements were defined to ensure the system's effective operation. This structured approach ensures clarity and efficiency in understanding and implementing the defect detection system, paving the way for improved accuracy and productivity in manufacturing environments.

Chapter 4

Architecture

4.1 Introduction

This chapter details the architecture of the proposed system for integrating AI-based defect detection in MES. The architecture is designed to ease the interaction between the MES, a Python API using the YOLO model for image analysis, and a user-friendly Graphical User Interface (GUI).

The architecture consists of three main components: the MES GUI, the Python Application Programming Interface (API) for defect detection and data processing, and the backend communication. These components interact to form an integrated system that supports automated defect detection in manufacturing processes, eliminating users' need for manual bounding box drawing.

4.2 MES GUI

The MES GUI serves as the front-end component of the system, evolving the MES and CORE components already in place at Critical Manufacturing. A new customization sub-module was developed to integrate AI-based defect detection capabilities. This sub-module uses Angular, adhering to established Angular Architecture Patterns and Best Practices.

The Angular application is structured into three main layers: Presentation Layer, Abstraction Layer, and Core Layer. The Presentation Layer contains all Angular components that represent the user interface and delegates user actions to the Core Layer through the Abstraction Layer. The Abstraction Layer serves as a bridge, decoupling the Presentation Layer from the Core Layer to maintain a clean separation of concerns. The Core Layer is responsible for data manipulation and communication with backend services [27].

The design follows SOLID principles to ensure flexibility and maintainability:

- **SRP (Single Responsibility Principle)**: Each Angular component and service has a single responsibility, ensuring that changes in the software have one reason to occur [1].
- **OCP (Open-Closed Principle)**: The software is designed to allow its behaviour to be extended without modifying existing code, facilitating easier updates and feature additions [1].

- **LSP (Liskov Substitution Principle)**: Components and services are designed so that derived classes can be substituted for their base classes without altering the correctness of the program [1].
- **ISP (Interface Segregation Principle)**: Interfaces are designed to be specific to each behaviour, resulting in smaller, more focused interfaces [1].
- **DIP (Dependency Inversion Principle)**: High-level modules do not depend on low-level modules; both depend on abstractions, ensuring that changes in lower-level details do not impact high-level policies [1].

In the MES GUI, there is a page that lists all the defect models created. Users have the ability to create new models, train them using a formatted dataset in YOLO format, and test the prediction by sending an image. A product can only be associated with one defect model. The visual representation of the defect model listing can be seen in Figure 4.1.

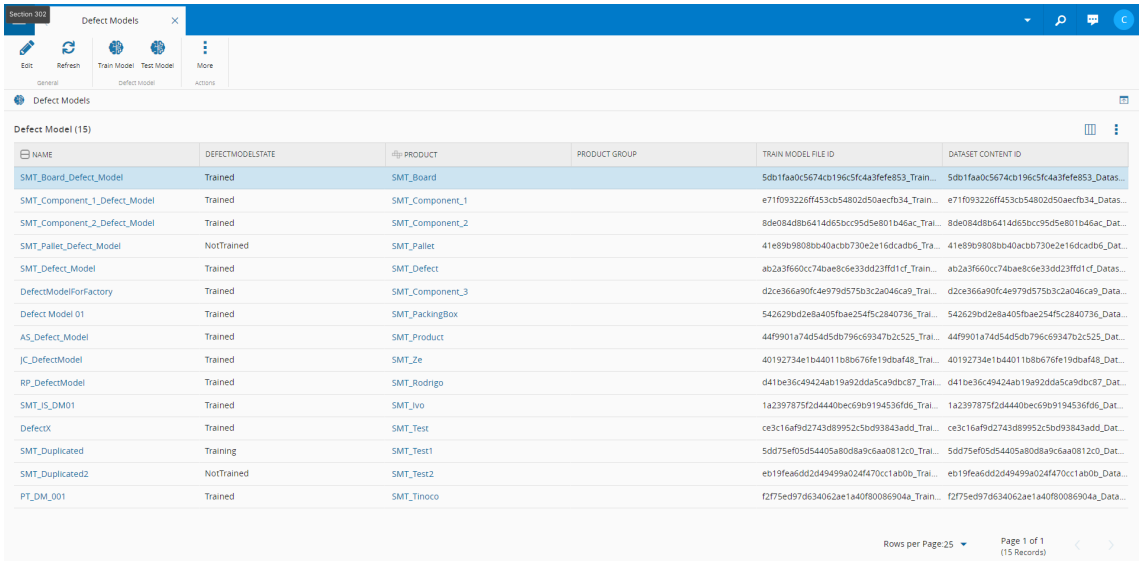


Figure 4.1: Defect model listing page in the MES GUI

Figure 4.2 illustrates an example of one of the views encountered by users when testing the defect model. In this view, users can input multiple images and review the results.

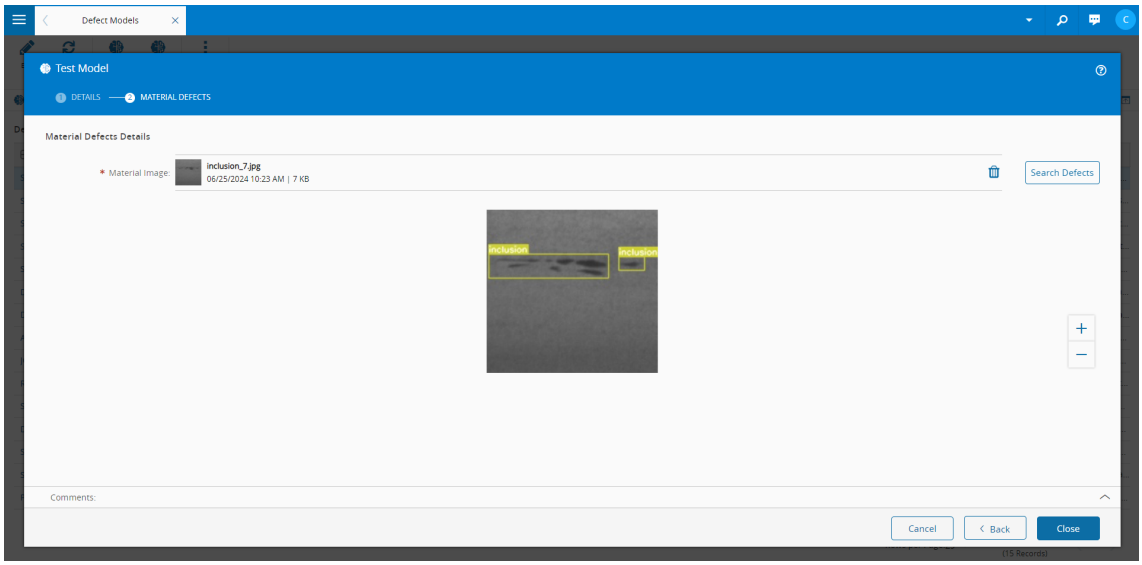


Figure 4.2: View for testing the defect model

In the CM MES, an instance of a product is called a material, so each defect is created for a specific material.

In addition, a new method of recording defects has been added to the existing defect pages for materials for which a product model has been created, and that said model has already finished training. Figure 4.3 depicts the highlighted button, which allows for the automatic creation of defects.

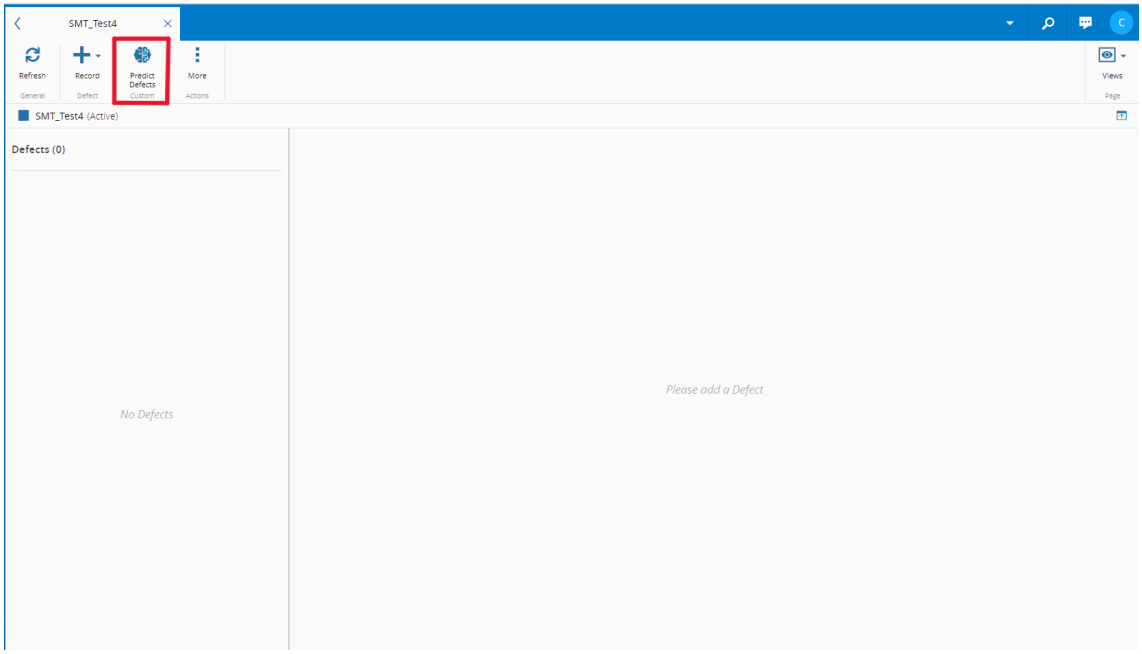


Figure 4.3: Defect page with the highlighted button for automatic defect creation

When selecting this button, it is possible to create the defects. Figure 4.4 represents an example of an image in which more than one type of defect was detected.

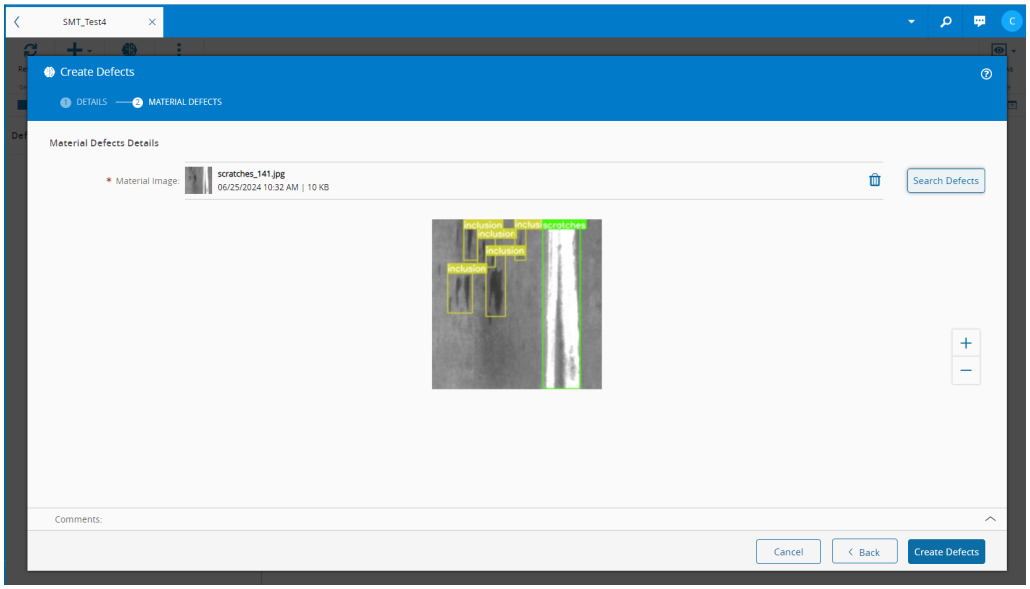


Figure 4.4: Image showing multiple detected defects

Once the *Create Defects* button is selected, they are stored and associated with the corresponding reason, being the reason already created in the MES. Figure 4.5 shows an example of the information displayed for each created defect.

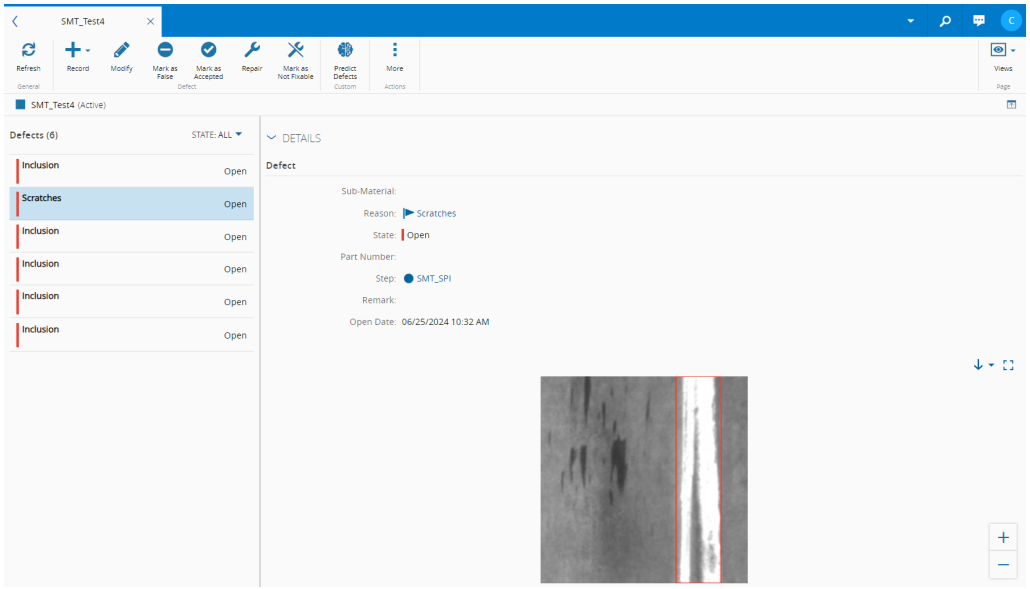


Figure 4.5: Information display for each created defect

4.3 Python API for Defect Detection

The Python API is the core processing unit of the system, responsible for handling tasks related to training and predicting defects using the YOLO model. The API interacts with the MES host to receive the dataset and model information. Upon receiving the data, the API saves the files and divides the dataset into training and testing subsets. The user, through the MES GUI, can initiate the training process.

The training process involves feeding the training dataset into the YOLO model to learn the characteristics of different types of defects. Once the model is trained, it can predict defects in new images. The user can upload new images via the MES GUI, which are then sent to the Python API for defect prediction. The API analyzes the images using the trained model and identifies any defects, sending the results back to the MES GUI for display.

All data, including images, defect annotations, and detection results, are stored in a centralized database. This centralized storage ensures easy access and analysis of the data. Furthermore, data processing pipelines are established to manage data flow from input to output. They are designed to handle large volumes of data efficiently and include data validation, transformation, and enrichment mechanisms, ensuring that the data is accurate and useful for defect detection. The pipelines also support parallel processing, using threads for I/O-bound tasks and multiprocessing for CPU-bound tasks. This approach allows multiple data streams to be processed simultaneously, which leads to an improvement in the system's performance.

The Python API guarantees secure and efficient data handling, keeping the integrity and privacy of manufacturing data. In addition to defect detection, the API includes model evaluation and performance monitoring functionalities. This allows for continuous model improvement and optimises it under different conditions. The API also supports logging and error handling, making diagnosing and resolving issues easier.

4.4 Backend Communication

The backend communication module is responsible for managing the data flow between the MES GUI and the Python API. This module ensures that all components of the system work together efficiently.

The backend, implemented in .NET, handles storing information about the defect models, such as the trained model file ID and the dataset ID for each specific defect model. The design follows established .NET design patterns and practices, such as the Repository Pattern for data access and the Unit of Work Pattern for managing changes.

4.5 Architecture Diagram

Figure 4.6 is a diagram that illustrates the interaction between the components of the system.

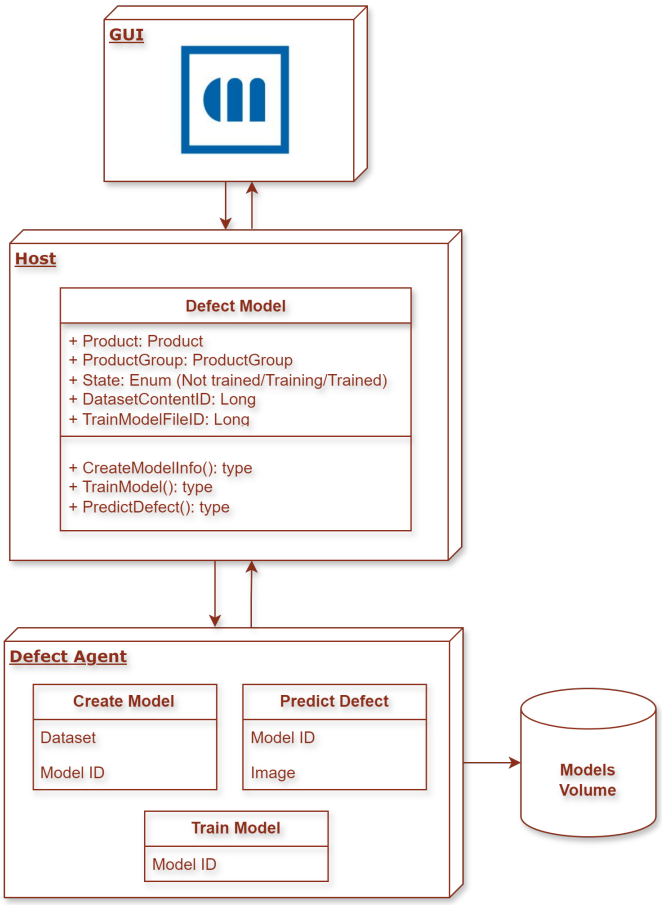


Figure 4.6: Architecture Scheme

The diagram highlights the MES GUI, which acts as the user interface for interaction with the system. The Python API manages model training, prediction, and storage of the dataset and trained models, while the backend securely stores all relevant information to make this flow work.

4.6 Summary

This chapter provided an in-depth overview of the architecture for integrating AI-based defect detection in MES. It detailed the MES GUI, which offers a user-friendly interface for managing defect models; the Python API, which handles the core processing tasks of training and predicting defects; and the backend communication module, which ensures seamless data flow between the GUI and the API. The architecture is designed with scalability, maintainability, and efficiency in mind, leveraging SOLID principles and established design patterns to create a robust and adaptable system.

Chapter 5

Implementation

5.1 Introduction

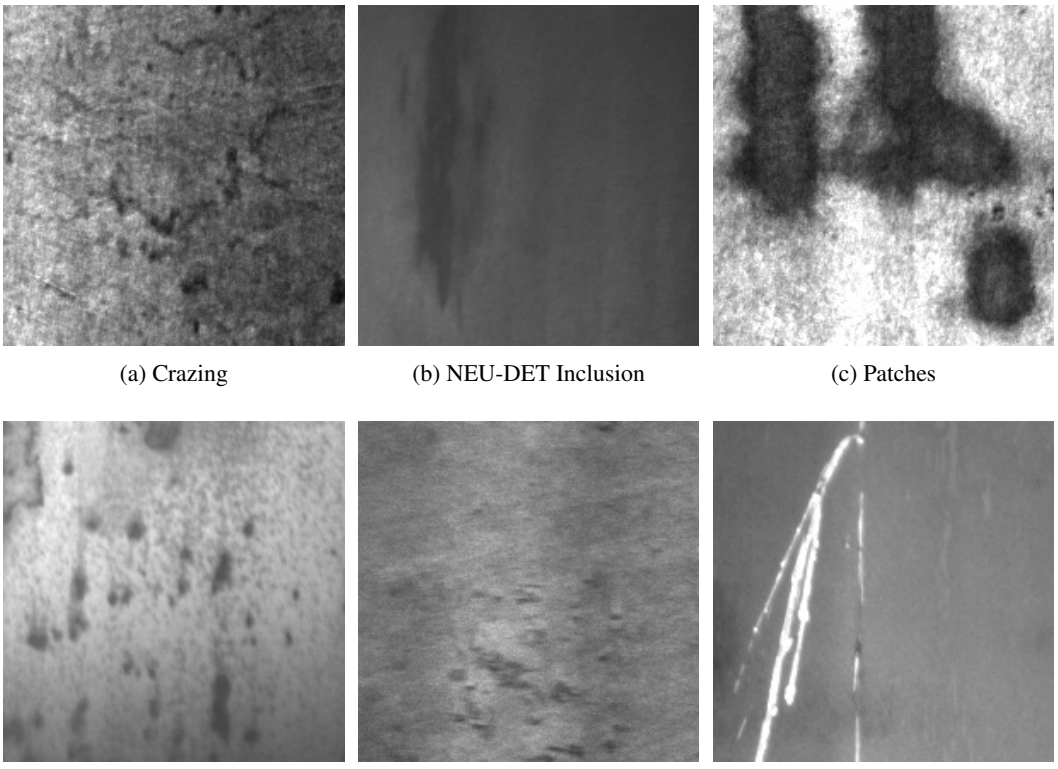
This chapter describes the implementation process of the AI-based defect detection system integrated into the Manufacturing Execution System. It covers dataset preparation, model training, system integration, hyperparameter tuning, and deployment, providing a detailed roadmap of the implementation process.

5.2 Datasets

The Northeastern University Detection Dataset (NEU-DET) [12] and the GDXray Castings Dataset (GC10-DET) [22] were used for training and evaluating the defect detection models. These datasets provide an extensive collection of images with labelled defects, which are a good example of possible datasets used for manufacturers.

5.2.1 NEU-DET Dataset

The NEU-DET dataset contains images of surface defects on hot-rolled steel strips. This data set originates from Northeastern University (NEU) and is commonly used for benchmarking defect detection systems. It includes six types of defects: crazing, inclusion, patches, pitted surface, rolled-in scale, and scratches. Each image is labelled with the type of defect and includes bounding box annotations that are essential for training machine learning models to not only recognize but also accurately pinpoint defects in real-world manufacturing settings. Figure 5.1 shows an example of each type. These high-resolution images capture the defects in great detail, facilitating the development of image analysis algorithms. However, it is important to note that in practical manufacturing environments, the conditions for capturing such high-quality images are not always ideal. Defect detection systems may encounter additional challenges due to this factor [12].



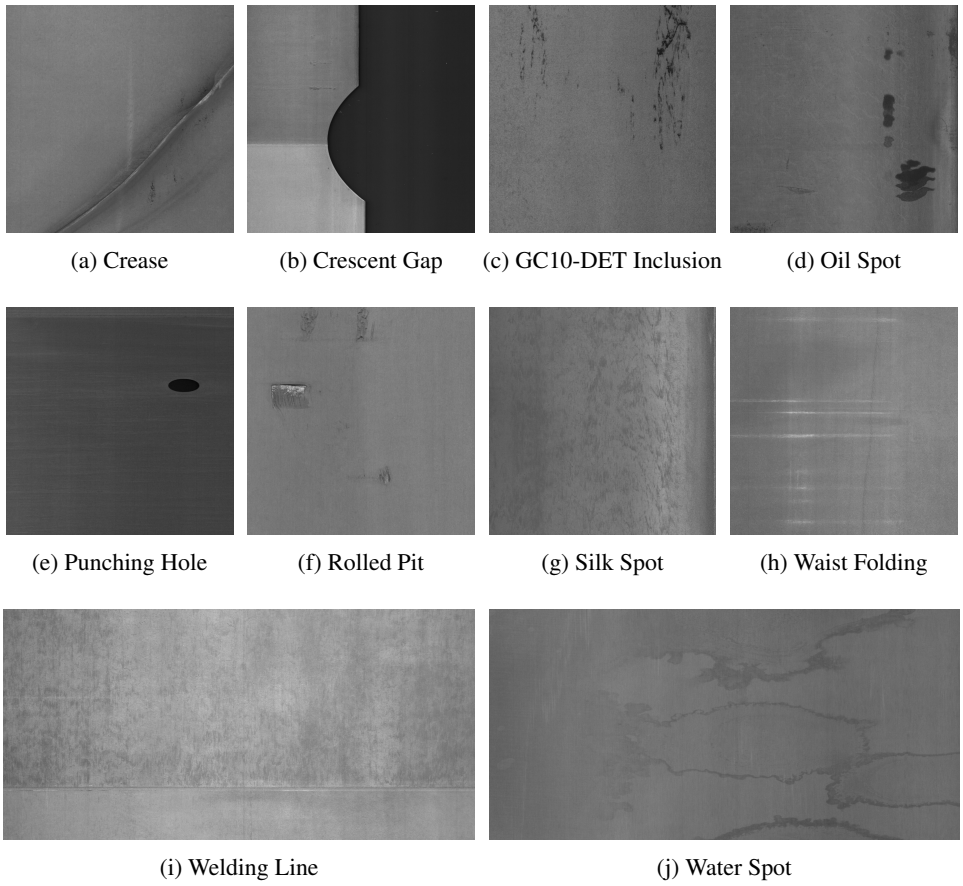
(a) Crazing (b) NEU-DET Inclusion (c) Patches

(d) Pitted Surface (e) Rolled in Scale (f) Scratches

Figure 5.1: Examples of defects of the NEU-DET dataset

5.2.2 GC10-DET Dataset

The GC10-DET dataset, created by Tianjin University in China, includes 2300 high-resolution images depicting ten types of defects in steel manufacturing, such as punching holes, crescent gaps, inclusion, oil spots, rolled pits, silk spots, creases, waist folding, welding line and water spots [22]. This dataset also provides bounding box annotations for each defect, facilitating the training and evaluation of object detection models. Figure 5.2 shows an example of each class.



(a) Crease (b) Crescent Gap (c) GC10-DET Inclusion (d) Oil Spot

(e) Punching Hole (f) Rolled Pit (g) Silk Spot (h) Waist Folding

(i) Welding Line (j) Water Spot

Figure 5.2: Examples of defects of the GC10-DET dataset

5.2.3 Data Preprocessing

The datasets were preprocessed to ensure they were suitable for training the models. This included the following steps:

- **Resizing**: Images were resized to a consistent dimension to match the input size requirements of the models.
- **Normalization**: Pixel values were normalized to a range of [0, 1] to standardize the input data.
- **Annotation Conversion**: Annotations were converted to the required format for each model.

Since the images in both NEU-DET and GC10-DET datasets were already in grayscale, conversion to grayscale was not necessary. However, this step is generally beneficial for models dealing with colour images to reduce computational complexity and focus on structural details. It is a step present in the implemented solution.

Given that colour information can be crucial for detecting certain types of defects, the preprocessing pipeline has been designed to be flexible. For defects where colour variation is a significant

indicator, the images are processed in their original colour format to preserve this information. This ensures that the model can leverage colour details when necessary while still allowing for grayscale conversion in cases where it enhances the detection of structural defects.

In practice, the preprocessing step includes a check to determine whether to convert an image to grayscale or retain its colour based on the type of defect being analyzed. This conditional approach ensures that the preprocessing is optimized for the specific requirements of the defect detection task.

5.2.4 Data Augmentation

Data augmentation techniques were employed to enhance the diversity and robustness of the training dataset. These techniques included:

- **Random Rotations**: Rotating images by random angles to simulate different viewing perspectives.
- **Flipping**: Applying horizontal and vertical flips to increase data variability.
- **Brightness and Contrast Adjustments**: Modifying the brightness and contrast to improve the model's robustness to varying lighting conditions.
- **Scaling**: Changing the size of the objects in the images to help the model learn to detect defects at different scales.

These augmentations help make the model more robust by exposing it to various image transformations, improving its generalization capabilities. Additionally, classes with less representation of images received the most emphasis during augmentation to balance the dataset and ensure the model could effectively learn to detect less common defect types [21].

5.3 Models

The models used in this study were various versions of YOLO provided by Ultralytics and Faster R-CNN. The YOLO versions were provided by Ultralytics, a company that offers a range of software tools and pre-trained models that are extensively utilized in applications such as object detection and classification. The YOLO models were initialized with pre-trained weights supplied by Ultralytics. Faster R-CNN was initialized with a pre-trained model using the ResNet-50 backbone.

5.3.1 YOLO Model Training

The training process for the YOLO models involved several key steps and included multiple variations:

- **Initializing with Pre-trained Weights**: The YOLO models were initialized with pre-trained weights from Ultralytics. Specifically, several versions were explored, including YOLOv5m6u, YOLOv8m, YOLOv8x, and YOLOv9e. Each version has distinct characteristics:

  - **YOLOv5m6u**: This medium-sized version of YOLOv5 is a specialized variant that is designed for ultra-real-time applications with enhanced performance on diverse datasets. The "6u" denotes that this model is optimized for use cases where ultra-speed is critical, maintaining a balance between speed and accuracy. YOLOv5m6u leverages advancements in YOLOv5 architecture, such as efficient backbone and head networks, optimized anchor boxes, and improved training strategies, making it suitable for real-time applications with good precision. This model includes additional optimizations and custom layers tailored for ultra-fast inference, making it particularly useful in manufacturing environments where real-time defect detection is paramount [17].
  - **YOLOv8m**: The medium-sized version of YOLOv8 improves on YOLOv5 by incorporating newer techniques and optimizations for better performance. Key differentiators include enhanced feature extraction capabilities, refined anchor-free detection mechanisms, and better handling of overlapping objects. YOLOv8 models generally offer higher accuracy and efficiency compared to their YOLOv5 counterparts [16].
  - **YOLOv8x**: The extra-large version of YOLOv8 provides the highest accuracy among YOLOv8 models due to its increased capacity and more complex architecture. It includes more layers and parameters, enabling it to capture finer details and patterns in the data. However, it is more computationally demanding and is best suited for applications where maximum accuracy is critical [16].
  - **YOLOv9e**: An experimental version of YOLO designed to explore new advancements in object detection. YOLOv9e incorporates novel attention mechanisms, advanced augmentation techniques, and more efficient training procedures. It aims to improve performance and efficiency compared to YOLOv8, but it is still in the experimental stage and may undergo further development [37].

- **Training with Optimal Parameters**: The models were trained on the NEU-DET and GC10-DET datasets. This training process involved setting parameters such as the number of epochs, batch size, optimizer type, and learning rate based on the optimal values identified during hyperparameter tuning. The Adam optimizer, explained in Section 5.5, was used due to its efficiency in handling sparse gradients and adaptive learning rate capabilities.

5.3.2 Faster R-CNN Model Training

The training process for Faster R-CNN involved using a pre-trained Faster R-CNN model with ResNet-50 as the backbone. To adapt the model to the defect detection task, the final layers were modified to output the required number of classes. The training pipeline was set up with appropriate data loaders, transforms, and loss functions, and data augmentation and preprocessing steps similar to those used for the YOLO models were applied to ensure consistency. Additionally, the hyperparameters identified by the `tune` function, explained in Section 5.4.4, from Ultralytics for the YOLO models were used for the Faster R-CNN model to maintain a consistent comparison

environment. This approach allowed for a fair comparison between the YOLO and Faster R-CNN models by evaluating them under similar conditions and hyperparameters settings.

5.4 Hyperparameter Tuning

Hyperparameter tuning is a critical and time-consuming process that significantly impacts the performance of machine learning models. Several approaches were utilized to fine-tune the hyperparameters of the defect detection models, including Grid Search, Random Search, and Bayesian Optimization.

5.4.1 Grid Search

Grid Search systematically explores a predefined subset of the hyperparameter space by evaluating model performance for each combination of parameters. This exhaustive approach ensures that the optimal combination is found within the specified range. However, it can be computationally intensive due to the large number of evaluations required. In this experiment, the sheer volume of possible combinations made Grid Search impractical, as it would have required excessive computational resources and time [20].

5.4.2 Random Search

Random Search addresses the inefficiency of Grid Search by randomly sampling combinations of hyperparameters from a specified distribution. This approach is often more efficient and can identify good hyperparameter settings more quickly than Grid Search, as it does not evaluate all possible combinations but instead focuses on a representative sample. However, the randomness of this method led to sub-optimal coverage of the hyperparameter space, resulting in less consistent and reliable outcomes [25].

5.4.3 Bayesian Optimization

Bayesian Optimization employs a probabilistic model to guide the search for optimal hyperparameters. It balances exploration and exploitation by using a surrogate model to predict the performance of different hyperparameter combinations and selecting the most promising ones for evaluation. This method is particularly effective for optimizing complex functions with fewer evaluations compared to Grid Search and Random Search. Despite its advantages, Bayesian Optimization still faced challenges in this experiment due to the high dimensionality and the need for frequent updates to the surrogate model, which increased the computational burden [9].

5.4.4 Ultralytics `tune` Function

The best performance was achieved using the `tune` function from Ultralytics. This advanced tuning function leverages sophisticated optimization techniques to efficiently search the hyperparameter space and identify the optimal settings.

The `tune` function systematically searches through the hyperparameter space using advanced optimization algorithms, reducing the need for manual intervention. It focuses on the most promising areas of the hyperparameter space, thereby reducing the time and computational resources required for tuning. By evaluating various combinations of hyperparameters, the `tune` function identifies the settings that maximize model performance, leading to improved accuracy and robustness [15].

Key hyperparameters tuned included learning rate, batch size, number of epochs, optimizer type, and data augmentation parameters, being the last available for the YOLO models included in the training process.

5.4.5 Computational Resources and Time

The hyperparameter tuning process is inherently time-consuming. In this study, with the available processor, an *NVIDIA GeForce RTX 2080 Ti*, the tuning process took approximately one day to complete an iterations set of 50 tries for approximately 1500 images of the NEU-DET dataset, and five days to complete the same numbers of iterations for 3000 images from the GC10-DET dataset. Despite the long duration, hyperparameter tuning is an essential step to ensure that the models achieve the highest possible performance and generalization capabilities. The investment in time and computational resources is justified by the significant improvements in defect detection accuracy and robustness achieved through optimized hyperparameters. For the implementation of this solution, the user can use the processor of their choice, but a faster and more robust one is recommended to ensure quicker processing.

5.5 Adam Optimizer

The Adam optimizer (short for Adaptive Moment Estimation) is an optimization algorithm utilised for training deep learning models. It combines the advantages of AdaGrad and RMSProp, other extensions of stochastic gradient descent [18]. Here's what makes the Adam optimizer particularly effective:

- **Adaptive Learning Rate**: Adam adjusts the learning rate for each hyperparameter dynamically, using estimates of the first and second moments of the gradients. This allows the optimizer to adapt the learning rate throughout the training process, making it suitable for problems with sparse gradients or noisy data.
- **Moment Estimation**: Adam uses exponential moving averages of the gradient (first moment) and the squared gradient (second moment) [26]. The first moment helps smooth the

gradients, while the second moment helps control the learning rate. These estimates are used to update the model hyperparameters efficiently.

- **Bias Correction**: During the initial stages of training, the moving averages are biased towards zero. Adam includes bias correction terms to counteract these biases, providing more accurate estimates of the first and second moments.
- **Efficiency**: Adam is computationally efficient, requiring little memory and performing well with large datasets and high-dimensional parameter spaces. It is robust to different data characteristics, making it widely used across various deep-learning applications.

Using the Adam optimizer benefits the training process with faster convergence and improved performance, particularly in complex deep-learning tasks such as defect detection in industrial images.

5.6 System Integration

The trained models were integrated into the MES to enable real-time defect detection during the manufacturing process. This involved the following steps:

5.6.1 Python API Development

A Python API was developed to handle model inference and communicate with the MES. The API performs tasks such as:

- Receiving images from the MES.
- Running inference using the trained Yolov5m6u, YOLOv8, YOLOv9 and Faster R-CNN models.
- Performing hyperparameter tuning to set the best hyperparameters for the models.
- Saving the best-trained model after hyperparameter tuning.
- Sending defect detection results back to the MES.

5.6.2 Updating MES Interface

The MES interface was updated to create the models and display defect detection results in real-time. This included adding functionalities for:

- Create the model and upload the dataset.
- Uploading images for testing the created model.
- Uploading images for defect detection in a material being manufactured.

- Viewing detected defects with bounding boxes associated with the corresponding reason for the defect.
- Creating defects automatically in the existing MES defect system.

5.7 Deployment

The integrated system was deployed in a manufacturing environment. The deployment process involved:

- Setting up a server to host the Python API and all defect detection models.
- Configuring network settings for communication between the MES and the server.
- Conducting initial testing to ensure the integrated system functions correctly.

5.8 Summary

This chapter outlines the implementation of an AI-based defect detection system within a Manufacturing Execution System. It details the use of the NEU-DET and GC10-DET datasets for training and evaluating models, including data preprocessing and augmentation techniques to enhance robustness. Various YOLO versions and Faster R-CNN were trained with optimized hyperparameters identified using the Ultralytics tune function. The Adam optimizer was employed for its efficiency and adaptive learning rate capabilities. System integration involved developing a Python API for model inference and updating the MES interface for real-time defect detection. The deployment process included server setup, network configuration, and initial testing to ensure functionality, resulting in an effective defect detection solution that improves manufacturing efficiency and product quality.

Chapter 6

Results Analysis

6.1 Introduction

This chapter presents a comprehensive evaluation of the performance and reliability of the integrated defect detection system. The testing phase involved extensive experiments to assess various aspects of the system's accuracy and robustness. The primary focus was on evaluating the performance of defect detection models using several metrics, which provide insights into the system's predictive capabilities. These evaluations were conducted on two datasets: NEU-DET and GC10-DET.

6.1.1 Used Metrics

In the domain of defect detection and classification with images, evaluating the performance of models is crucial to ensure their reliability and accuracy. Several metrics are commonly used to assess various aspects of model performance. These metrics provide insights into different facets of the model's predictive capabilities, from precision and recall to more complex evaluations like mean average precision and confusion matrices. Below are the key metrics used in this study:

- **Intersection over Union (IoU)** is a metric used to evaluate the accuracy of an object detector on a particular dataset. It is defined as the area of overlap between the predicted bounding box and the ground truth bounding box divided by the area of their union:

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{6.1}$$

  An IoU value of 1 indicates a perfect overlap, while an IoU value of 0 indicates no overlap at all. IoU is critical in determining how well the predicted bounding boxes match the ground truth boxes. Higher IoU thresholds (e.g., 0.75 or 0.95) indicate stricter criteria for a positive detection, while lower thresholds (e.g., 0.5) are more tolerant [33].

- **Precision** is the ratio of correctly predicted positive observations to the total predicted positives. It is calculated as:

$$P = \frac{TP}{TP+FP} \tag{6.2}$$

where TP (true positives) is the number of correctly identified and classified images with defects, and FP (false positives) is the number of images incorrectly identified as having defects or images with defects that were misclassified. High precision indicates that the model has a low false positive rate, meaning most positive predictions are correct. Conversely, low precision indicates that the model has a high false positive rate, meaning a significant number of positive predictions are incorrect [33].

In the context of defect detection, true positives occur when the model correctly identifies and classifies an image that has defects. False positives happen when the model incorrectly identifies an image as defective or when it misclassifies an image with defects. Additionally, false negatives FN refer to the number of defective images that the model fails to identify [33].

A threshold of 0.5 for the IoU was applied to the detection confidence score to determine whether a predicted bounding box is considered a positive detection. This threshold means that a predicted bounding box must overlap with the ground truth bounding box by at least 50% for it to be counted as a true positive. This value was chosen because it provided a good balance between precision and recall for the defect detection task. Predictions with an IoU above 0.5 were counted as true positives, while those below this threshold were considered false positives or false negatives, ensuring that the model's performance was optimized for the specific requirements of this application.

It is important to note that if an image is classified into the wrong defect class, it counts as a false positive for the incorrectly assigned class and as a false negative for the actual class. This dual misclassification affects both the misidentified class's precision and the correct class's recall.

Figure 6.1 illustrates these concepts:

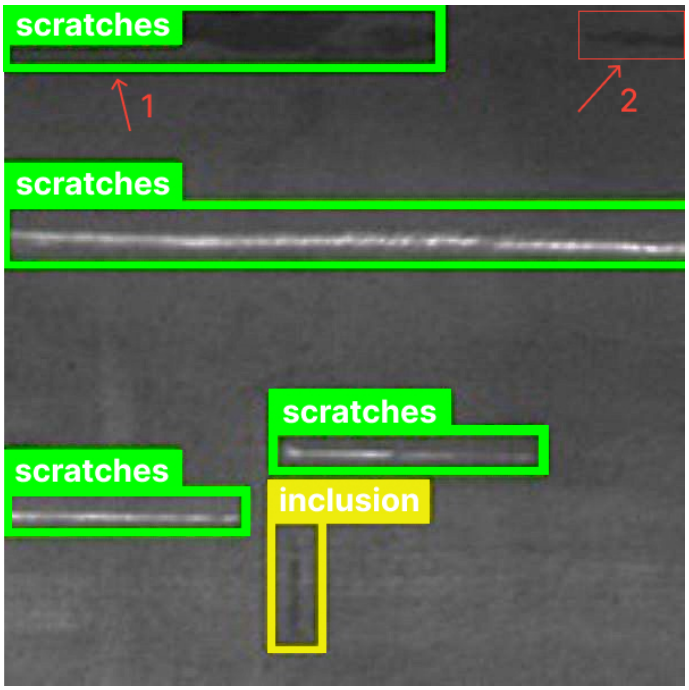


Figure 6.1: Illustration of TP, FP, and FN in defect detection. Box 1 indicates a false positive because it was classified incorrectly as *scratch*. It also represents a false negative because the actual defect, an *inclusion*, was not identified correctly. Box 2 is a false negative as it was not identified. The other boxes are true positives.

- **Recall** is the ratio of correctly predicted positive bounding boxes to all bounding boxes in the actual class. The formula is as follows:

$$R = \frac{TP}{TP+FN} \tag{6.3}$$

  where FN is the number of images with defects that were incorrectly identified as not having defects.

  High recall indicates that the model correctly identifies most actual positive instances, resulting in a low false negative rate. On the other hand, low recall indicates that the model misses a significant number of actual positive instances, resulting in a high false negative rate [33].

- **Average Precision (AP)** summarizes the precision-recall curve, which plots precision (P) against recall (R) at various threshold settings. AP is computed as the area under the precision-recall curve:

$$AP = \int_0^1 P(R)\,dR \tag{6.4}$$

  In practice, this is often approximated using a finite number of points on the precision-recall curve.

A high AP indicates that the model maintains high precision across a wide range of recall values, demonstrating a good balance between precision and recall. Conversely, a low AP suggests that the model struggles to maintain precision as recall increases, indicating a significant trade-off where increasing recall substantially decreases precision.

- **Mean Average Precision (mAP)** is the mean of the average precision scores for each class. It is a common metric for evaluating the accuracy of object detectors. The formula for *mAP* is:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{6.5}$$

where N is the number of classes and AP_i is the average precision for class i.

A high mAP signifies that the model excels across all classes, consistently achieving high precision and recall for each category. On the other hand, a low mAP indicates that the model has difficulty accurately detecting objects in one or more classes, reflecting a lower overall performance [33].

- **mAP@0.5 (mAP50)**: Mean Average Precision at IoU threshold of 0.5. This metric measures the average precision across all classes when the IoU threshold is set to 0.5. It is a common metric for evaluating object detectors, as it indicates how well the model detects objects with a moderate overlap criterion.

- **mAP@0.5:0.95 (mAP50-95)**: Mean Average Precision averaged over multiple IoU thresholds from 0.5 to 0.95 with a step size of 0.05. This metric provides a more complete evaluation of the model's performance, as it considers the precision and recall at different levels of overlap between the predicted and ground truth bounding boxes. It is a stricter and more challenging metric compared to mAP50, as it requires the model to perform well across a range of IoU thresholds.

- **Confusion Matrix**: Shows the performance of the classification model by displaying the actual versus predicted classifications. It helps identify the correct and incorrect classifications made by the model.

  It is typically represented as follows:

Table 6.1: Confusion matrix typical representation

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

- **Normalized Confusion Matrix**: Provides a normalized view of the confusion matrix to show the proportion of correct and incorrect classifications, offering a clearer view of the model's performance across different classes.

- **F1-Confidence Curve**: Illustrates the F1 score against different confidence thresholds, giving insights into the balance between precision and recall at various confidence levels. The F1 score is the harmonic mean of precision and recall [11], calculated as:

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{6.6}$$

  A typical F1-confidence curve might show that the F1 score is low at both very low and very high confidence thresholds. At very low thresholds, the model might predict too many positives, leading to a high number of false positives (low precision). At very high thresholds, the model might miss many true positives, leading to a high number of false negatives (low recall). The highest point of the curve signifies the confidence threshold at which the model reaches the optimal equilibrium between precision and recall.

  By analyzing the F1-confidence curve, we can select the confidence threshold that aligns with specific performance requirements, ensuring that the model operates optimally for the intended use case.

- **Label Distribution and Correlogram**: Displays the distribution of different defect labels and their spatial distribution. This helps in understanding the prevalence and spatial characteristics of each defect type.

- **Precision-Confidence Curve**: Plots precision against confidence thresholds, helping to understand how the model's precision changes with different confidence levels.

  If precision remains high at lower confidence thresholds, the model is robust and accurate even with less certainty. Conversely, a sharp decline in precision as the confidence threshold decreases indicates that the model makes more false positive errors at lower confidence levels.

- **Precision-Recall (PR) Curve**: Shows the trade-off between precision and recall for different thresholds. This curve is useful for evaluating the model's performance in scenarios with imbalanced datasets [2].

  A larger area under the precision-recall curve indicates better overall performance of the model [14]. Also, it illustrates the trade-off between precision and recall. A model with high precision but low recall might miss many positive instances (high false negatives), while a model with high recall but low precision might have many false positives.

- **Recall-Confidence Curve**: Displays recall against confidence thresholds, showing how recall changes with different confidence levels.

  If recall remains high at lower confidence thresholds, the model is capable of identifying most positive instances even with less certainty. As the confidence threshold increases, a decline in recall indicates that the model becomes more conservative, potentially missing more positive instances (higher false negatives) at higher confidence levels.

- **Training and Validation Loss**: Tracks the loss values for both training and validation sets over the epochs, indicating the model's learning progress. Lower loss values generally indicate better model performance.

  If the training loss decreases, the model is effectively learning from the training data. If the validation loss decreases, the model is generalizing well to unseen data. However, if the training loss continues to decrease after a certain epoch while the validation loss begins to increase, it indicates overfitting. To prevent this, training can be stopped, or regularization techniques can be applied.

6.2 Results Analysis

Several defect detection experiments were performed on the NEU-DET and GC10-DET datasets. The performance of different models with different data manipulations was made on these datasets to see how they best behave on them.

The analysis of the visual data provides critical insights into defect patterns and characteristics. The following descriptions concern about the NEU-DET dataset:

- **Defect Type Distribution:** The bar chart in the top left of Figure 6.2 reveals that *inclusion* defects have the highest occurrence, followed by *patches*. This indicates a higher frequency of these defect types in the dataset, which can impact the model's performance and the focus of defect detection strategies.

- **Bounding Box Aspect Ratios:** The visualization in the top right of Figure 6.2 shows the variations in the aspect ratios of the bounding boxes. This helps understand the diversity of shapes and sizes the model needs to accommodate, indicating that defects come in different proportions.

- **Bounding Box Coordinates:** The heatmap in the bottom left of Figure 6.2 shows the distribution of the x and y coordinates of the bounding boxes. The higher density in the central region indicates that defects are more frequently located towards the center of the images.

- **Bounding Box Sizes:** The heatmap in the bottom right of Figure 6.2 shows the distribution of the width and height of the bounding boxes. The concentration of points in the lower left area of the heatmap suggests that most defects have smaller bounding boxes, with width and height values clustering towards the lower end of the scale.

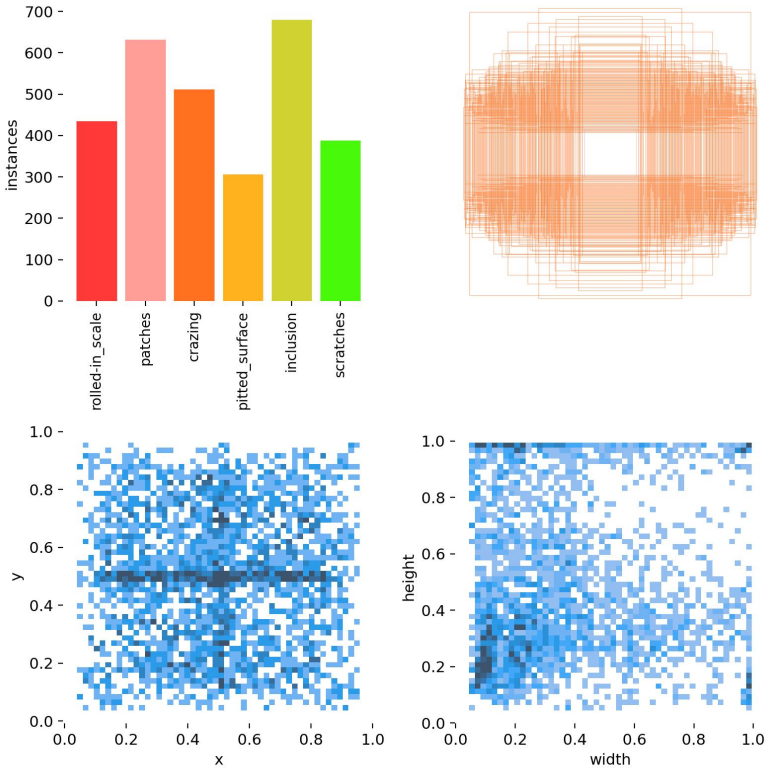


Figure 6.2: Visual analysis of defect instances, bounding box distribution, and bounding box sizes of the NEU-DET dataset.

The analysis of Figure 6.3 reveals more insights into the bounding box distributions within the NEU-DET dataset:

Positional Correlations: The scatter plots and histograms indicate how the x and y coordinates of bounding boxes are distributed. There is a higher density of bounding boxes around the center, reflecting a central concentration of defects.

Size Correlations: The relationships between width and height show a noticeable correlation, suggesting that as the width of the bounding box increases, the height also tends to increase. This correlation is critical for understanding the proportionality of defect sizes.

Distribution Insights: The histograms along the diagonal provide insight into the individual distributions of x, y, width, and height. Most bounding boxes have smaller dimensions, as indicated by the peak at the lower end of the width and height histograms.

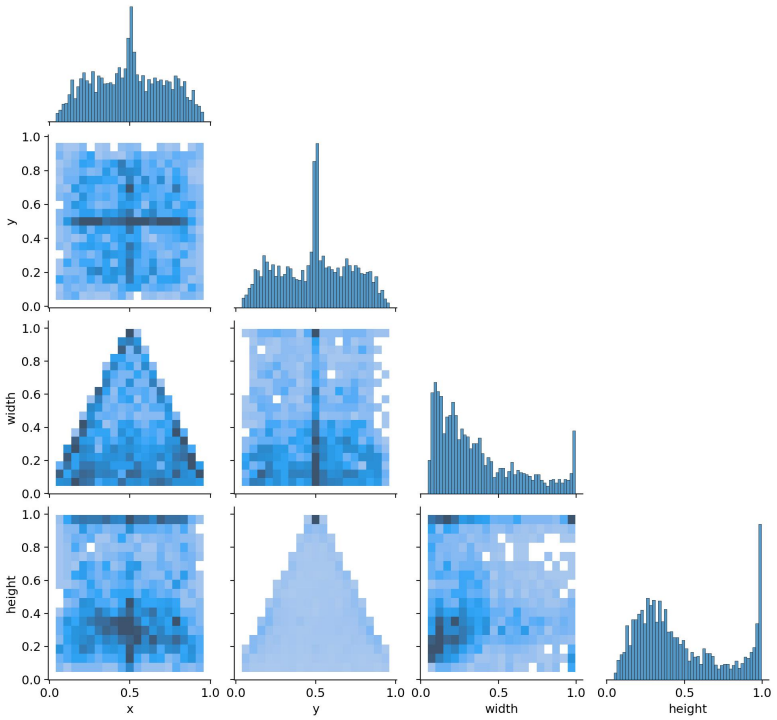


Figure 6.3: Heatmaps showing the distribution of x, y coordinates, width, and height of bounding boxes of the NEU-DET dataset.

The following descriptions concern the GC10-DET dataset:

- **Defect Type Distribution:** The bar chart in the top left of Figure 6.4 shows that *siban* defects have the highest occurrence, followed by *youban*.
- **Bounding Box Coordinates:** The central region of the bottom left heatmap of Figure 6.4 shows a higher density, indicating that defects tend to be located towards the middle of the images.
- **Bounding Box Sizes:** The bottom right heatmap in Figure 6.4 suggests that the clustering of points in the lower left suggests that most defects are smaller, with width and height values concentrated at the lower end of the spectrum.

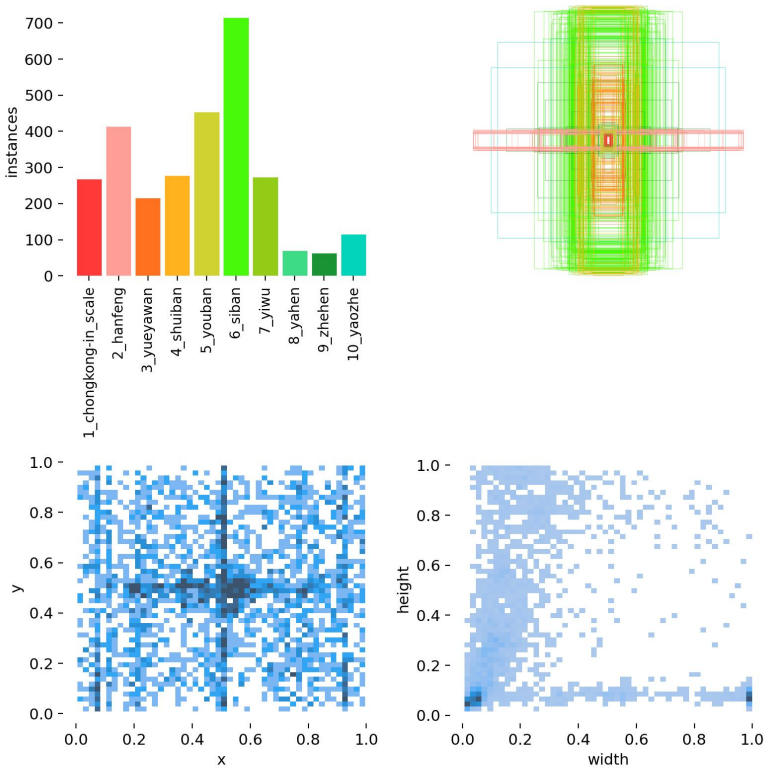


Figure 6.4: Visual analysis of defect instances, bounding box distribution, and bounding box sizes.

Figure 6.5 provides other insights into the bounding box distributions within the GC10-DET dataset:

Positional Correlations: A noticeable concentration of bounding boxes near the center indicates a central clustering of defects.

Distribution Insights: Many bounding boxes are small in dimension, as shown by the peaks in the lower range of the width and height histograms.

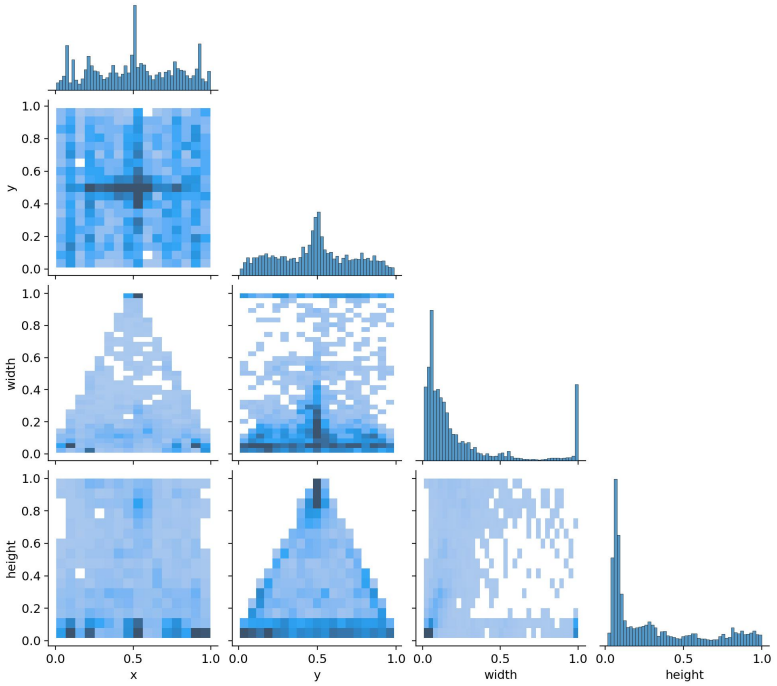


Figure 6.5: Heatmaps showing the x, y coordinates, width, and height distribution of bounding boxes.

These observations are crucial for understanding defect patterns and guiding the development and application of detection models to improve accuracy and reliability.

6.2.1 One-Stage Object Detection Performance - YOLO

In this section, we analyze the performance of different YOLO models on the NEU-DET and GC10-DET datasets, focusing on precision, recall, and mean average precision (mAP). The models evaluated include YOLOv5m6u, YOLOv8m, YOLOv8x and YOLOv9e. Data augmentation techniques were applied to assess their impact on model performance.

6.2.1.1 Performance on NEU-DET Dataset

After performing hyperparameter tuning over 100 epochs, the optimal parameters were determined as follows:

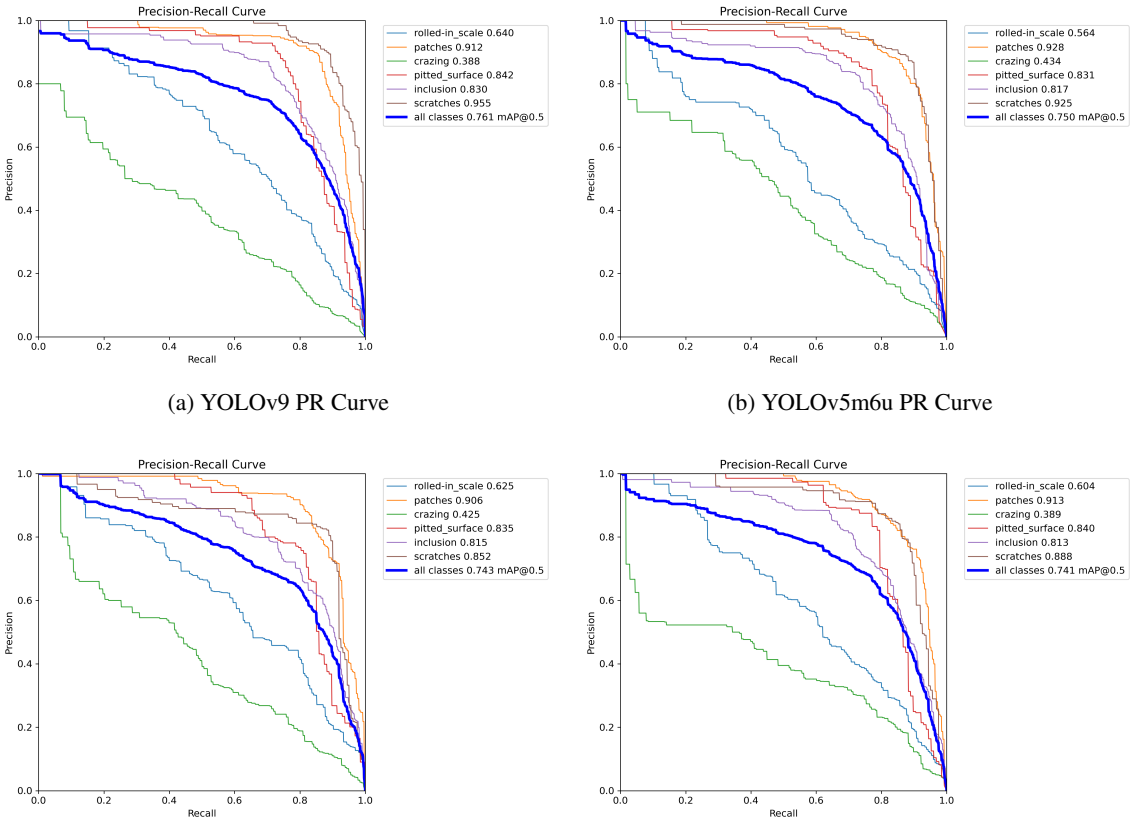
- **Initial Learning Rate:** 0.0001
- **Final Learning Rate:** 0.2
- **Momentum:** 0.9
- **Weight Decay:** 0.0005

- **Warmup Epochs:** 5
- **Warmup Momentum:** 0.9
- **Warmup Bias Learning Rate:** 0.1
- **Nominal Batch Size:** 64

The batch size is dependent on the GPU memory capacity, as some models can handle larger or smaller batch sizes without exhausting the GPU's memory. This flexibility ensures efficient utilization of available resources and optimal model performance.

- **YOLOv9**: The YOLOv9 model achieved the highest performance among the tested models on the NEU-DET dataset. It demonstrated superior precision and recall, resulting in the highest mAP scores. Interestingly, the model performed better without data augmentation, achieving a mAP@0.5 of 0.761 compared to 0.744 with augmentation. This discrepancy might be due to the specific characteristics of the YOLOv9 architecture and training dynamics, despite using the same data augmentation logic as YOLOv8.
- **YOLOv5m6u**: The YOLOv5m6u model surprisingly outperformed the YOLOv8m model, achieving a mAP@0.5 of 0.75. This indicates that the architectural improvements and training methodologies of YOLOv5m6u were particularly effective for the NEU-DET dataset.
- **YOLOv8m** The YOLOv8m model performed well but was slightly behind YOLOv5m6u in terms of mAP@0.5, scoring 0.743. The precision-recall curve and other performance metrics indicate that while YOLOv8m is robust, it did not match the peak performance of YOLOv5m6u.
- **YOLOv8x** The YOLOv8x model achieved a mAP@0.5 of 0.741, performing slightly below the YOLOv8m model. While it maintains robustness and reliability, it did not surpass the performance of YOLOv5m6u and YOLOv9.

The Figure 6.6 shows the precision-recall curves for each of the YOLO models tested on the NEU-DET dataset. These curves provide a visual representation of the models' performance in terms of precision and recall across different defect classes.



(a) YOLOv9 PR Curve

(b) YOLOv5m6u PR Curve

(c) YOLOv8m PR Curve

(d) YOLOv8x PR Curve

Figure 6.6: Precision-Recall curves for YOLOv9, YOLOv5m6u, YOLOv8m, and YOLOv8x models on the NEU-DET dataset.

From the precision-recall curves, it is evident that the models demonstrate varying degrees of performance across different defect classes. The models generally perform well on classes such as *patches* and *scratches*, achieving high precision and recall scores. This consistency across models indicates that these defect types are easier to detect with the current dataset and model configurations. Conversely, classes like *crazing* and *rolled-in scale* exhibit lower precision and recall scores, suggesting that these defect types are more challenging to detect.

The confusion matrix visually represents each defect class's true positive, false positive, and false negative rates. Each model showed very similar outcomes. However, let us analyse the model YOLOv9e since it was the one with the best performance. Figure 6.7 shows the absolute number of predictions made by the model for each class in this model.

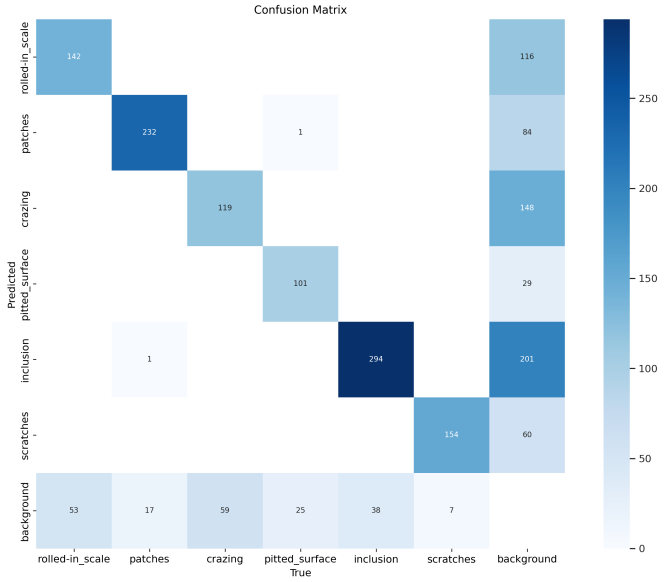


Figure 6.7: Confusion matrix for YOLOv9 on NEU-DET dataset.

The confusion matrix shows that the model performs exceptionally well in identifying *patches* and *inclusion* defects, with true positive counts of 232 and 294, respectively. However, there are notable misclassifications, such as 116 instances of *rolled-in scale* and 148 of *crazing* being incorrectly classified as *background*.

Figure 6.8 shows the normalized values, which help in understanding the proportion of correctly predicted instances relative to the total number of instances for each class in the YOLOv9 model.

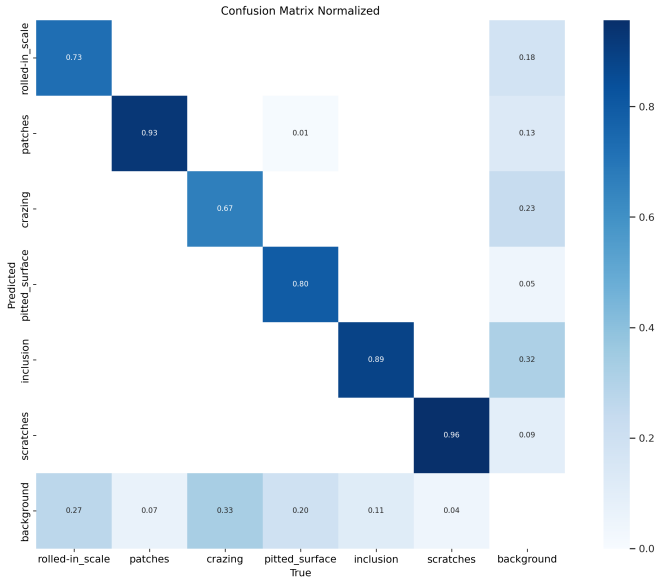


Figure 6.8: Normalized confusion matrix for YOLOv9 on NEU-DET dataset.

The normalized confusion matrix reveals that the model has a high true positive rate for *patches*

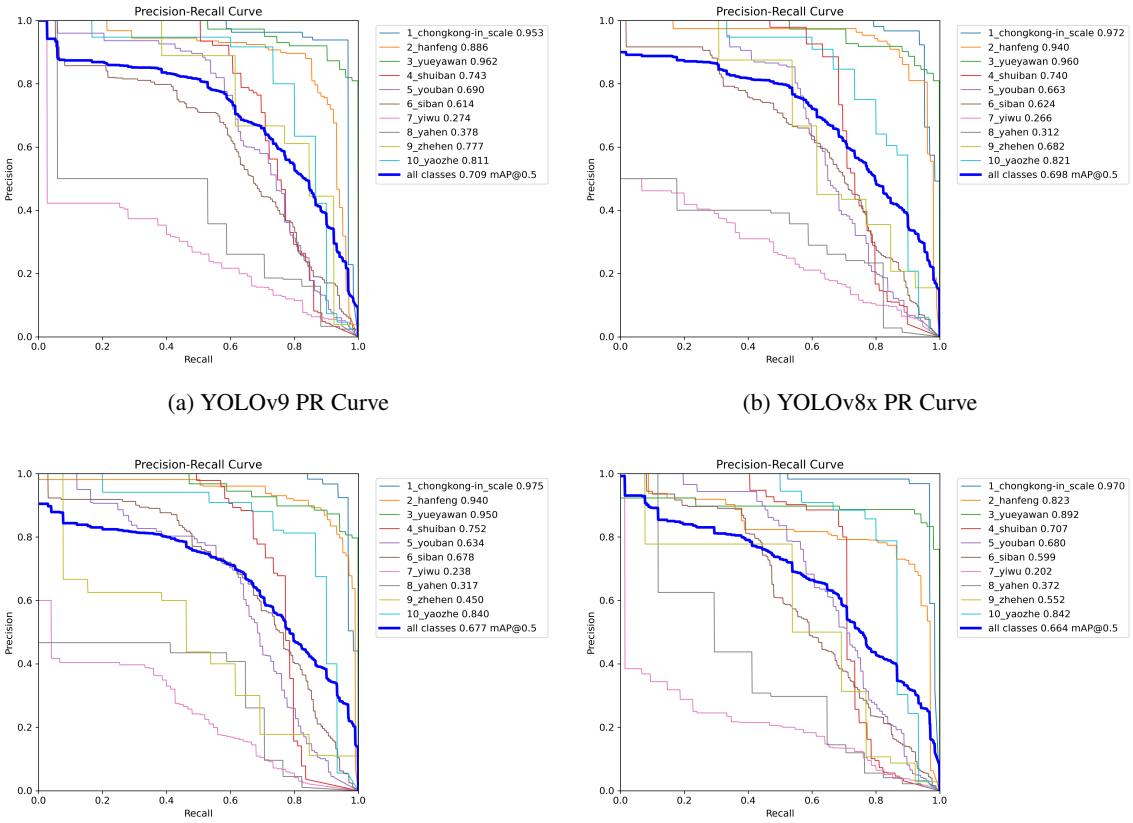
(0.93) and *scratches* (0.96), indicating robust performance in these categories. Conversely, the model struggles more with *crazing* and *rolled in scale*, where the true positive rates are 0.67 and 0.73, respectively.

6.2.1.2 Performance on GC10-DET Dataset

The parameters used for the NEU-DET dataset also worked well on the GC10-DET dataset. The results for the models on the GC10-DET dataset were as follows:

- **YOLOv9:** The YOLOv9 model achieved a mAP@0.5 of 0.709. This model demonstrated the highest performance among the tested models on the GC10-DET dataset, maintaining its position as the best-performing model.
- **YOLOv8x:** The YOLOv8x model achieved a mAP@0.5 of 0.698. While it performed well, it was slightly behind the YOLOv9 model.
- **YOLOv8m:** The YOLOv8m model achieved a mAP@0.5 of 0.677. Although it performed robustly, it did not surpass the YOLOv8x or YOLOv9 models.
- **YOLOv5m6u:** The YOLOv5m6u model achieved a mAP@0.5 of 0.664. This indicates that while the model is effective, it was outperformed by the other models on the GC10-DET dataset.

The order of the best models changed compared to the NEU-DET dataset; however, the YOLOv9 model remained the top performer. Figure 6.9 shows the models' precision-recall (PR) curves.



(a) YOLOv9 PR Curve

(b) YOLOv8x PR Curve

(c) YOLOv8m PR Curve

(d) YOLOv5m6u PR Curve

Figure 6.9: Precision-Recall curves for YOLOv9, YOLOv8x, YOLOv8m, and YOLOv5m6u models on the GC10-DET dataset.

The PR curves indicate that while some defect classes, such as *chongkong-in scale* and *yueyawan*, achieve high precision and recall scores, other classes like *siban* and *yiwu* significantly lower the overall mAP.

The confusion matrices in Figures 6.10 and 6.11 provide further insights into the performance of the YOLOv9e model on the GC10-DET dataset. As already explained for the NEU-DET dataset, the confusion matrix (Figure 6.10) shows the raw counts of predicted versus true labels, and the normalized confusion matrix (Figure 6.11) presents these counts as proportions.

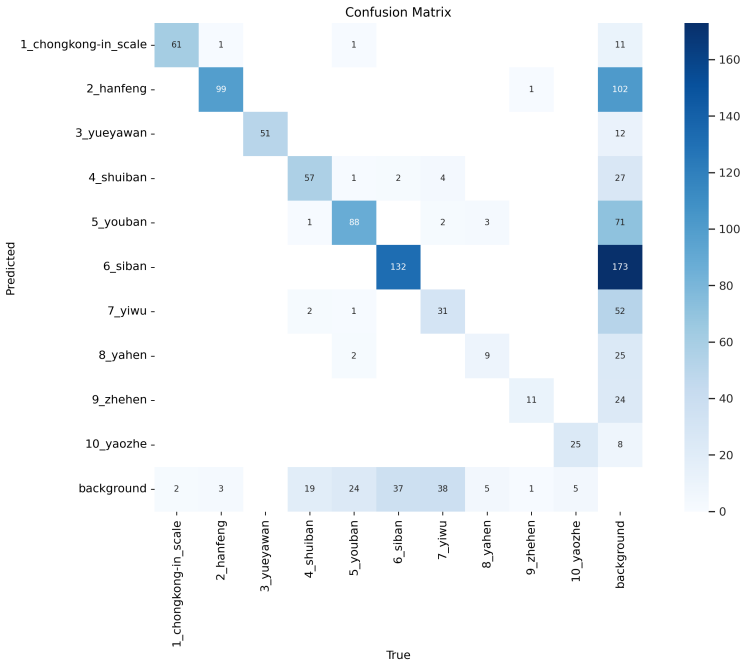


Figure 6.10: Confusion Matrix for the YOLOv9 model on the GC10-DET dataset.

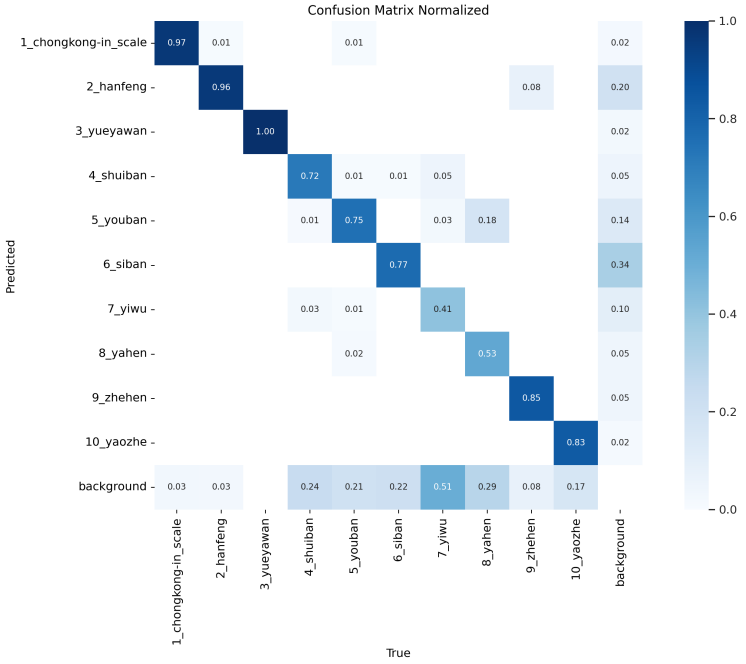


Figure 6.11: Normalized Confusion Matrix for the YOLOv9 model on the GC10-DET dataset.

It is clear that certain classes, such as **siban** and **yiwu**, have higher misclassification rates, which contributes to the lower overall mAP.

6.2.2 Two-Stage Object Detection Performance - Faster R-CNN

In this section, we evaluate the performance of the Faster R-CNN model, a two-stage object detection framework, on two different datasets: NEU-DET and GC10-DET. Faster R-CNN is known for its robustness and high accuracy in detecting objects by combining RPN with Fast R-CNN. We analyze how well this model performs in identifying various defect types within these datasets.

The hyperparameters were modified as follows: weight decay was set to 0.0001, and the learning rate was adjusted to 0.01.

6.2.2.1 Faster R-CNN on NEU-DET

The PR curves for the Faster R-CNN model on the NEU-DET dataset are illustrated in Figure 6.12. The Faster R-CNN model shows a generally good level of recall, indicating good performance in detecting defects in the NEU-DET dataset. The model achieves a precision of 0.30 and a recall of 0.79. The mAP at a 0.5 IoU threshold is 0.553. The pattern repeats as the defect *patches* is the one with the lowest mAP.

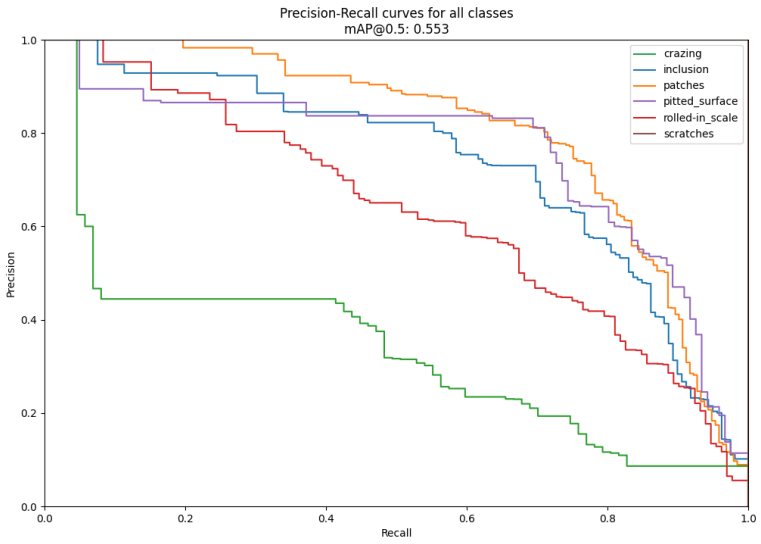


Figure 6.12: Precision-Recall curves for Faster R-CNN on the NEU-DET dataset.

The lower precision compared to recall suggests that while the model is good at finding most of the defects, it also has a higher rate of false positives. This discrepancy could be due to the nature of the NEU-DET dataset, which might contain defects that are visually similar to non-defective areas, making it challenging for the model to distinguish between true defects and background noise.

6.2.2.2 Faster R-CNN on GC10-DET

Figure 6.13 presents the PR curves for the Faster R-CNN model on the GC10-DET dataset. The model achieves a precision of 0.36 and a recall of 0.63 on the GC10-DET dataset, with an mAP@0.5 of 0.549.

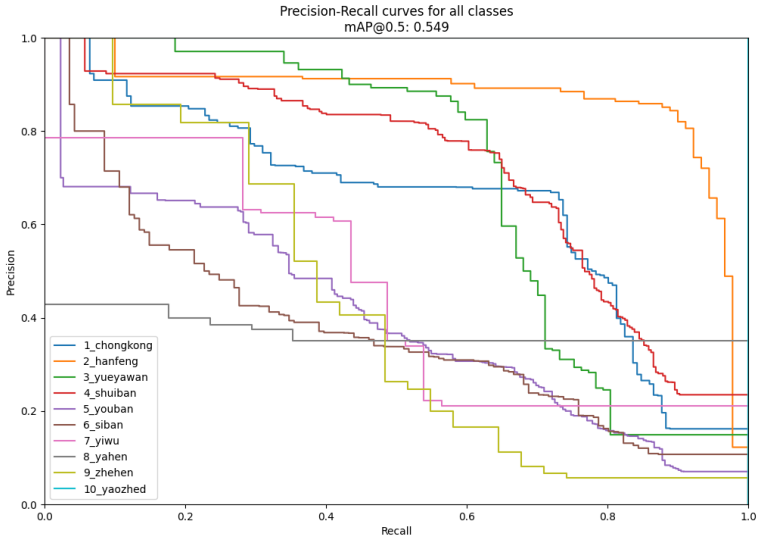


Figure 6.13: Precision-Recall curves for Faster R-CNN on the GC10-DET dataset.

The lower recall suggests that the model fails to identify a significant number of actual defects in this dataset. As we can observe, the results exhibit considerable variability and lack consistency, indicating room for improvement in the model's performance.

6.2.3 Performance Metrics

Table 6.2 and 6.3 summarize the performance metrics for the models on both datasets:

Table 6.2: Performance metrics on the NEU-DET dataset

Model	Data Augmentation	Precision	Recall	mAP@0.5
YOLOv9e	No	0.744	0.75	0.761
YOLOv9e	Yes	0.746	0.717	0.744
YOLOv5m6u	Yes	0.714	0.7076	0.75
YOLOv8m	Yes	0.7236	0.7223	0.743
YOLOv8x	Yes	0.7289	0.7367	0.741
Faster R-CNN	Yes	0.30	0.79	0.553

Table 6.3: Performance metrics on the GC10-DET dataset

Model	Data Augmentation	Precision	Recall	mAP@0.5
YOLOv9	Yes	0.7061	0.73129	0.709
YOLOv5m6u	Yes	0.70956	0.68345	0.664
YOLOv8m	Yes	0.6478	0.6945	0.677
YOLOv8x	Yes	0.71239	0.70049	0.698
Faster R-CNN	Yes	0.36	0.63	0.549

When comparing the performance of the YOLO model, such as its variants (YOLOv5m6u, YOLOv8m, YOLOv8x, and YOLOv9e), to Faster R-CNN, the YOLO models generally achieve higher precision values compared to Faster R-CNN. For instance, YOLOv9e achieves a precision of 0.744 without data augmentation on the NEU-DET dataset, significantly higher than Faster R-CNN's precision of 0.30. Faster R-CNN's recall is relatively higher on the NEU-DET dataset (0.79), compared to YOLO models, which have recall values around 0.71 to 0.75. YOLO models also exhibit higher mAP@0.5 values. YOLOv9e achieves the highest mAP@0.5 of 0.761 on the NEU-DET dataset, while Faster R-CNN achieves 0.553 on the same dataset. Additionally, the YOLO models train faster, making them more efficient for practical deployment in real-time applications.

In summary, while Faster R-CNN showed strong recall capabilities on NEU-DET dataset, it falls short in precision and overall mAP when compared to the YOLO models. The YOLO models, particularly YOLOv9e, provide a better balance of precision and recall, leading to higher overall performance metrics. This indicates that for applications requiring better overall performance, YOLO models might be more suitable,

Additional figures illustrating various metrics for each model, such as F1 score curves, precision curves, recall curves, and the metrics shown in the above image (including box loss, class loss, precision, recall, mAP50, and mAP50-95), are presented in the Appendix A.

6.3 User Feedback

To further evaluate the defect detection system, a survey was conducted among eight workers to gather feedback on usability, performance, effectiveness, satisfaction, and any technical issues encountered. The survey results are presented below with accompanying graphics and explanatory text.

6.3.1 Usability

The usability of the defect detection system was evaluated based on three main criteria: ease of use, intuitiveness of the user interface, and the learning curve for new users. Figure 6.14 outlines these results.

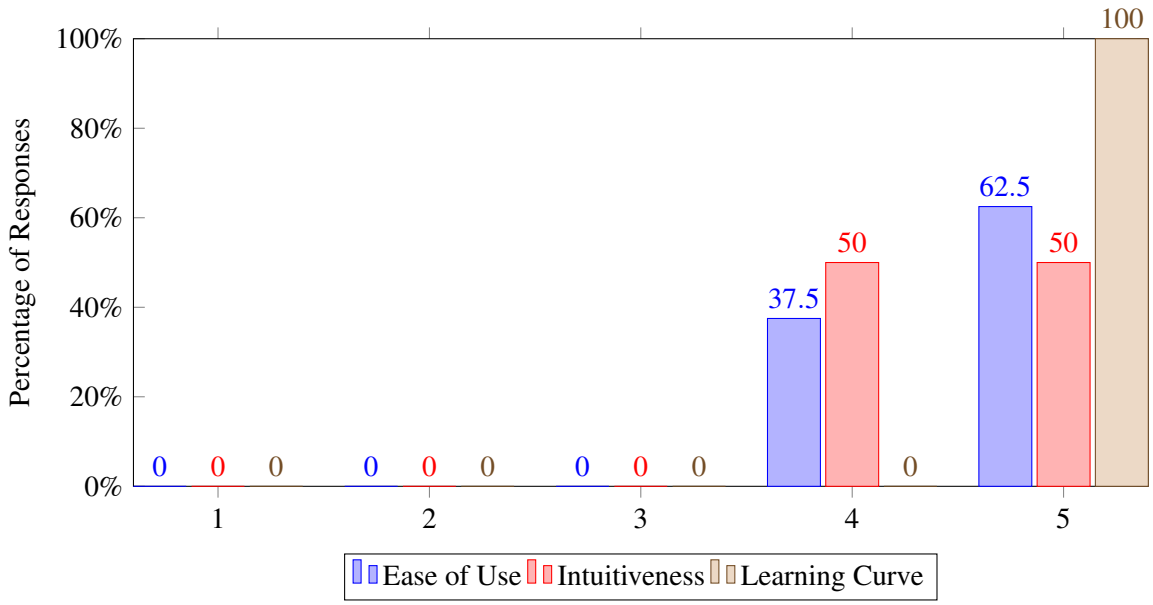


Figure 6.14: Usability Feedback

The high ratings in ease of use and intuitiveness indicate that the system is user-friendly and easy to navigate, making it accessible to operators with varying levels of technical expertise. The learning curve data shows that all workers were able to learn the system very quickly.

6.3.2 Performance

The performance of the system was assessed based on accuracy, speed, and any performance issues experienced. For a visual representation, see Figure 6.15.

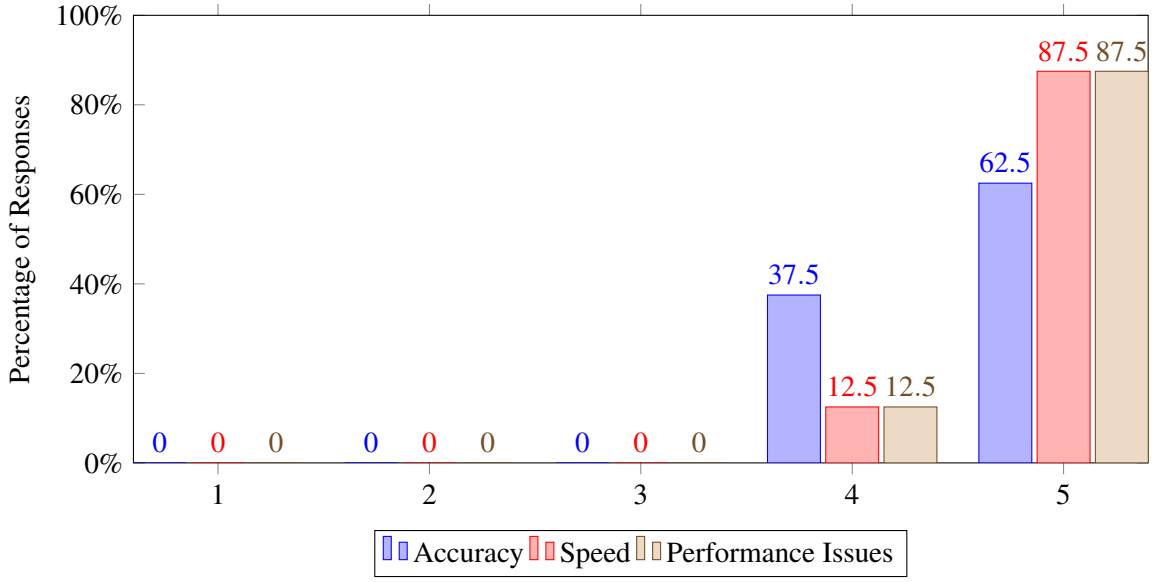


Figure 6.15: Performance Feedback

The high accuracy and speed ratings demonstrate that the system reliably identifies defects quickly, ensuring high-quality control in the manufacturing process. The minimal reports of performance issues suggest that the system is stable and operates smoothly under typical working conditions.

6.3.3 Effectiveness

The effectiveness of the system was evaluated in terms of improvement over previous methods, reliability of detection results, and time efficiency. The outcomes are shown in Figure 6.16.

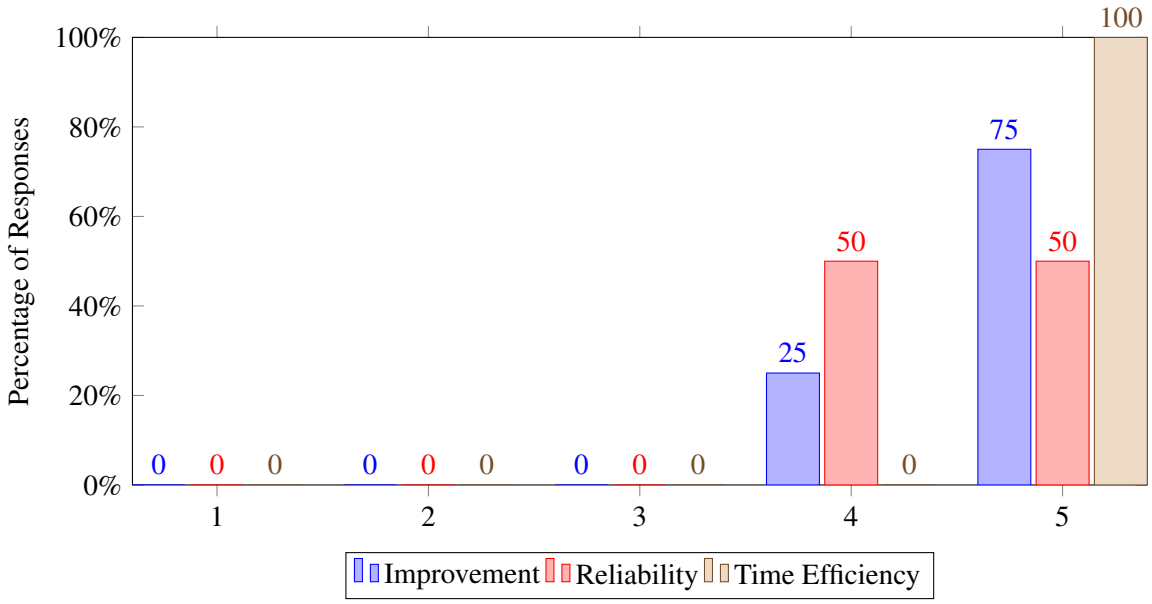


Figure 6.16: Effectiveness Feedback

The system significantly improves the defect detection process, making it more efficient and reliable than older methods. Unanimous feedback on time efficiency highlights the system's effectiveness in streamlining defect detection.

6.3.4 Satisfaction

The overall satisfaction with the defect detection system and preference for it over previous methods were evaluated. The visual results are in Figure 6.17.

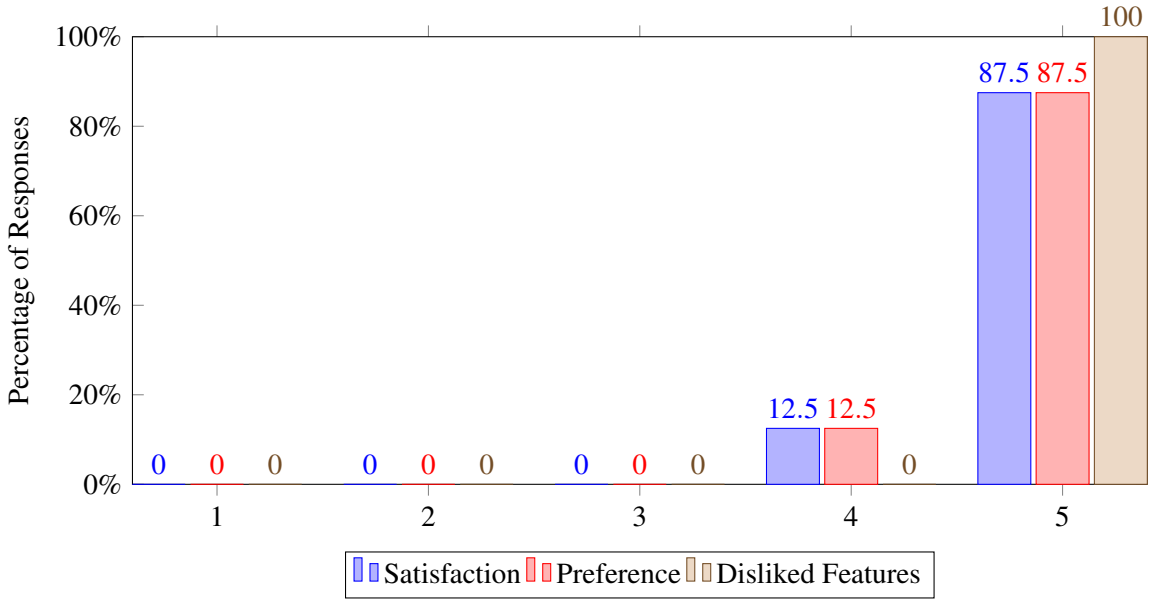


Figure 6.17: Satisfaction Feedback

The high satisfaction ratings reflect the system's positive impact on the participants' workflow and overall experience. The preference for the new system indicates that it offers significant advantages over traditional methods, making it a preferred choice for defect detection. The feedback also suggests that the system's features are well-designed and meet workers' needs effectively.

6.3.5 Technical Issues

None of the participants reported encountering any technical issues or bugs while using the system. Figure 6.18 presents these results.

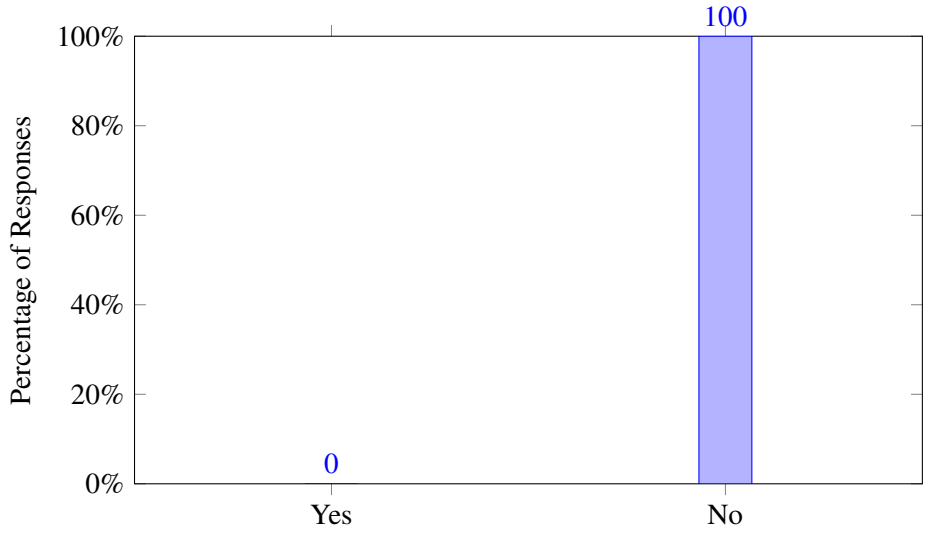


Figure 6.18: Technical Issues

The system's robustness and reliability are evident as there are no reported technical issues, resulting in minimal disruptions to the workflow.

6.3.6 Additional Feedback

Some suggestions were offered for enhancing the system and adding more features.

6.3.6.1 Suggestions for Improvement

The participants made several suggestions for improving the user experience, such as:

> "Make the *Upload* button default, no need to press it."
>
> "Enhance the visibility of the *Search Defects* button in the wizard."
>
> "Add notifications when the model finishes training."

These suggestions offer valuable insights into areas where the system can be further refined to enhance user experience.

6.3.6.2 Additional Features

Some participants suggested features like restricting the marking of false positives to certain roles, automatically placing material on hold after a certain threshold of detected defects, and allowing manual addition of defects not detected by the system.

> "Restrict marking defects as false positives to certain roles, such as *Quality Engineers*."
>
> "Automatically place material on hold or mark it as a loss after a certain threshold of defects."
>
> "Allow manual addition of defects not detected by the system."

Incorporating these additional features could enhance the system's functionality and better meet the specific needs of the participants.

6.3.6.3 Other Comments

The participants provided positive feedback, noting that the system is intuitive and effective, with some minor User Interface/User Experience (UI/UX) suggestions for further improvement.

> "The solution is very intuitive and easier to use than the previous defect handling workflow."
>
> "The defect detection mechanism seems very good at detecting many kinds of defects."

The positive comments reinforce overall satisfaction with the system and offer valuable suggestions for minor improvements.

6.3.7 User feedback overview

Overall, the feedback from the participants indicates a high level of satisfaction with the defect detection system. The system is considered user-friendly, effective, and efficient in improving the defect detection process. The suggestions for improvement and additional features provided by the participants will be valuable in further refining the system to meet their needs better. The new system's reliability and effectiveness are underscored by the absence of technical issues and a clear preference over the previous methods.

6.4 Discussion

In this section, we delve into the implications and significance of the results obtained from the models used for defect detection. The primary focus is on evaluating the model's performance, its integration into the Manufacturing Execution System (MES), and its potential impact on the manufacturing process.

- **Model Selection and Performance**: After evaluating multiple models, including Faster R-CNN, YOLOv5m6u, YOLOv8m, and YOLOv8x, the YOLOv9e model was selected for its superior performance. The decision was based on comprehensive testing and comparison of precision-recall metrics, where YOLOv9e consistently demonstrated higher accuracy and reliability in detecting defects.
- **Integration with MES**: The integration of YOLOv9e into the MES was seamless, significantly enhancing the defect detection capabilities of the system. The Python API developed for this purpose ensured efficient communication between the MES GUI and the backend, allowing for real-time defect identification and response. This integration not only automated the defect detection process but also reduced the need for manual inspections, thereby improving operational efficiency.
- **User Feedback and Usability**: Feedback from users highlighted the system's ease of use and its intuitive interface. The MES GUI, enhanced with Angular, allowed non-technical users to train models, upload datasets, and visualize results effectively. Users appreciated the automated defect detection, which significantly reduced the time spent on manual inspections. However, some suggestions for improvement were left to be taken into account.

6.5 Summary

This chapter detailed the integrated defect detection system's testing and performance evaluation using various metrics. The analysis covered model performance on the NEU-DET and GC10-DET datasets, highlighting the strengths and weaknesses of different models. The YOLOv9e model emerged as the top performer, particularly excelling in precision, recall, and mAP metrics.

The feedback from users was highly favourable, highlighting the system's ease of use, performance, and overall effectiveness. Integrating the system into the MES went smoothly, leading to a marked improvement in operational efficiency. Additionally, the lack of technical issues and strong user satisfaction levels attest to the system's reliability and robustness.

The insights gained from this testing phase provide a solid foundation for further refinements and potential enhancements to the defect detection system, ensuring it meets the high standards required for practical deployment in manufacturing environments.

Chapter 7

Conclusions

7.1 Summary of Findings

This chapter presents the key findings and conclusions of the research on integrating AI-based defect detection within Manufacturing Execution Systems.

This research addresses the need to improve operational efficiency in manufacturing by automating defect detection. It highlights the benefits of integrating ML techniques into manufacturing processes, focusing on overcoming the challenges of adapting these technologies to diverse industrial settings. The key motivations included improving production quality and speed by reducing human errors in defect identification. The primary goal was to develop a generic AI training system capable of generating defect identification models using known images, with supervised learning being the main focus due to how the current system is made.

The research builds on fundamental concepts of machine learning. It delves into deep learning techniques, emphasizing ANNs and CNNs for image-based defect detection. The importance of MES in managing manufacturing operations and integrating defect detection capabilities is also discussed.

Existing object detection algorithms, such as Faster R-CNN and YOLO, were reviewed for their suitability in defect identification. The proposed solution involved developing a user-friendly AI training system for defect detection and integrating it into an MES to leverage its capabilities.

The system architecture includes a user interface for interaction, a Python API for model training and prediction, and a backend for data management. Implementation details covered model development, GUI creation, deployment, and testing.

The performance analysis of the YOLO and Faster R-CNN models was conducted to evaluate their effectiveness in defect detection and MES integration.

7.2 Key Findings

Model Performance

The YOLOv9e model consistently outperformed other models in terms of precision, recall, and mean average precision. Its ability to accurately identify and classify defects makes it a positively reliable tool for automated defect detection in manufacturing.

System Integration

The integration of the YOLOv9e model into the MES was achieved seamlessly through the development of a Python API. This integration automated the defect detection process, reducing the need for manual inspections and significantly enhancing operational efficiency.

User Feedback

Feedback from users was very positive, highlighting the system's ease of use, intuitive interface, and substantial time savings. Users appreciated the automated defect detection feature, which minimized the time spent on manual inspections and improved the overall workflow.

7.3 Implications and Impact

The successful integration of AI-based defect detection within MES has several important implications. The ability to automatically and accurately detect defects ensures higher product quality and reduces the likelihood of defective products reaching customers. By minimizing manual inspections and reducing defect-related rework, the system helps lower production costs. The user-friendly nature of the system allows it to be easily adapted and scaled across different industries and production environments.

With that being said, the research confirms that AI-based defect detection is both feasible and effective.

7.4 Future Work

Several improvements can be integrated into the current system to optimise the defect detection process. The existing system primarily maintains a photographic record of defective products.

Installing cameras at strategic points in the production line permits a continuous dataset of the produced material to be created, allowing for real-time defect detection and analysis. To utilize the generated images effectively, it is important to understand certain concepts.

7.4.1 Proposed Enhancements

1. **Camera Installation:** A camera should be installed to take photos of the produced material at a consistent angle. This ensures uniformity in the dataset, which is crucial for training accurate models.

2. **Dataset Accumulation:** Over time, the system can begin integrating the models for defect detection as the dataset grows.

3. **YOLO Training:** The YOLO model should be trained with known defects and their respective reasons.

4. **Autoencoder Training:** An autoencoder should be trained using only non-defective images of the material. This encode-decode process enables the autoencoder to recognize anomalies that deviate from the normal patterns. When a defective product is presented to the autoencoder, it will likely produce a high reconstruction error due to its inability to recreate the unknown defect patterns. This high error can be a reliable indicator of anomalies, enabling early detection of new and unforeseen defects.

5. **Model Integration:** Once the models are trained, they can be integrated into the production line. The process involves:

   (a) The image is first processed by the trained YOLO model to detect known defects and associate them with the respective reason, as seen possible in this study.

   (b) Subsequently, the image passes through the autoencoder, which flags any patterns it does not recognize from the non-defective training images.

   (c) If both models return defect detections, the bounding boxes generated are compared. If there is no overlap, it indicates an unknown defect detected by the autoencoder.

6. **Defect Classification:** Known defects are associated with their respective reasons, which the MES already recognizes. A flag is raised for unknown defects, and the operator is prompted to review and classify these defects.

7. **Continuous Improvement:** The system continuously saves images for ongoing training, ensuring that the models improve over time and become more reliable.

With the proposed solution, the system can detect not only known defects but also new, previously unseen defects. Combining YOLO for known defects and autoencoders for anomalies ensures a comprehensive defect detection mechanism and, as the system continues to learn and adapt, it becomes more robust and capable of handling a wider range of defects, improving over time.

7.4.2 Requirements and Advantages

The requirements for implementing this solution are minimal, demanding only cameras and a strong processor with GPU capable of training the models. By integrating these enhancements, the accuracy and efficiency of the defect detection process can be significantly improved, making it more robust and adaptable to various production environments.

Figure 7.1 illustrates the proposed improved defect detection process.

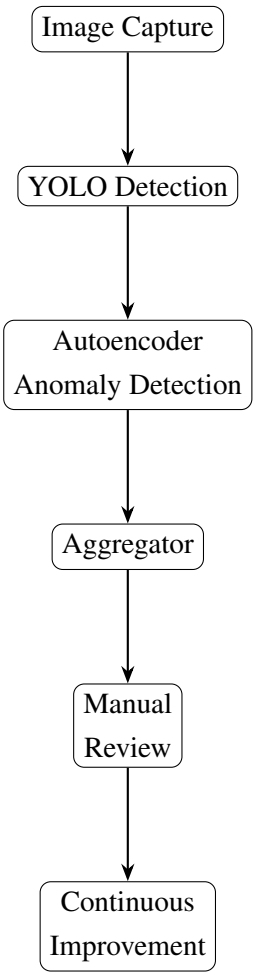


Figure 7.1: Simplified defect detection process flow. This diagram illustrates the steps involved in the proposed optimization of the defect detection system.

References

[1] G Kumar Arora. *SOLID Principles Succinctly*. CreateSpace Independent Publishing Platform, 2017.

[2] Pavlos Avgoustinakis, Giorgos Kordopatis-Zilos, Symeon Papadopoulos, Andreas L Symeonidis, and Ioannis Kompatsiaris. Audio-based Near-Duplicate Video Retrieval with Audio Similarity Learning. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 5828–5835. IEEE, 2021.

[3] Pragati Baheti. The Essential Guide to Neural Network Architectures. `https://www.v7labs.com/blog/neural-network-architectures-guide`, Jul 2021. V7.

[4] Prahar Bhatt, Rishi Malhan, Pradeep Rajendran, Brual Shah, Shantanu Thakar, Yeo Jung Yoon, and Satyandra Gupta. Image-Based Surface Defect Detection Using Deep Learning: A Review. *Journal of Computing and Information Science in Engineering*, 21:1–23, 2021.

[5] Yuh-Wen CHEN and Jing Mau Shiu. An Implementation of Yolo-family Algorithms in Classifying the Product Quality for the ABS Metallization. *PREPRINT (Version 1) available at Research Square*, Sep 2021.

[6] Tausif Diwan, G Anirudh, and Jitendra V Tembhurne. Object detection using YOLO: Challenges, architectural successors, datasets and applications. *multimedia Tools and Applications*, 82(6):9243–9275, 2023.

[7] FasterCapital. Autoencoders: Unveiling Data Patterns with DLOM-based Autoencoders. `https://fastercapital.com/content/Autoencoders--Unveiling-Data-Patterns-with-DLOM-based-Autoencoders.html#Types-of-Autoencoders.html`, June 2024. FasterCapital.

[8] Fredrik Filipsson. AI in Quality Control: Revolutionizing Defect Detection. `https://redresscompliance.com/ai-quality-control/`, June 2024. Redress Compliance.

[9] Peter I Frazier. A Tutorial on Bayesian Optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[11] Haihong Guo, Xu Na, Li Hou, and Jiao Li. Classifying Chinese Questions Related to Health Care Posted by Consumers Via the Internet. *Journal of Medical Internet Research*, 19(6):e220, 2017.

[12] Yu He, Kechen Song, Qinggang Meng, and Yunhui Yan. An end-to-end steel surface defect detection approach via fusing multiple hierarchical features. *IEEE Transactions on Instrumentation and Measurement*, 69(4):1493–1504, 2020.

[13] Huawei. Intelligent Manufacturing: Keeping a Sharper Eye on Industrial Quality Inspection with Smart Technologies. `https://e.huawei.com/en/case-studies/industries/manufacturing/ai-quality-inspection-2023`, 2023. Huawei.

[14] Lu Jin, Zechao Li, and Jinhui Tang. Deep Semantic Multimodal Hashing Network for Scalable Image-Text and Video-Text Retrievals. *IEEE Transactions on Neural Networks and Learning Systems*, 34(4):1838–1851, 2020.

[15] Glenn Jocher. Ultralytics YOLO Hyperparameter Tuning Guide. `https://docs.ultralytics.com/guides/hyperparameter-tuning/`, 2023. Ultralytics.

[16] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLOv8. `https://github.com/ultralytics/ultralytics`, 2023. Ultralytics.

[17] Glenn Jocher, Lakshantha Dissanayake, Burhan, and Sergiu Waxmann. Comprehensive Guide to Ultralytics YOLOv5 . `https://docs.ultralytics.com/yolov5/`, 2023. Ultralytics.

[18] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Rohit Kundu. YOLO: Algorithm for Object Detection Explained. `https://www.v7labs.com/blog/yolo-object-detection`, 2023. V7.

[20] Petre Lameski, Eftim Zdravevski, Riste Mingov, and Andrea Kulakov. SVM Parameter Tuning with Grid Search and Its Impact on Reduction of Model Over-fitting. In *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing: 15th International Conference, RSFDGrC 2015, Tianjin, China, November 20-23, 2015, Proceedings*, pages 464–474. Springer, 2015.

[21] Wei Li, Chen Chen, Mengmeng Zhang, Hengchao Li, and Qian Du. Data augmentation for hyperspectral image classification with deep CNN. *IEEE Geoscience and Remote Sensing Letters*, 16(4):593–597, 2018.

[22] Xiaoming Lv, Fajie Duan, Jia-jia Jiang, Xiao Fu, and Lin Gan. Deep Metallic Surface Defect Detection: The New Benchmark and Detection Network. *Sensors*, 20(6), 2020.

[23] Kartik Mahajan. Fog Computing: A Systematic Review of Architecture, IoT Integration, Algorithms and Research Challenges with Insights into Cloud Computing Integration. *Authorea Preprints*, 2023.

[24] Farhad Malik. Neural Networks Bias And Weights. `https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da`, 2019. Medium.

[25] Rafael G Mantovani, André LD Rossi, Joaquin Vanschoren, Bernd Bischl, and André CPLF De Carvalho. Effectiveness of Random Search in SVM hyper-parameter tuning. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[26] Ricardo Perera, Javier Montes, Alejandra Gómez, Cristina Barris, and Marta Baena. Unsupervised autoencoders with features in the electromechanical impedance domain for early damage assessment in frp-strengthened concrete elements. *Engineering Structures*, 315:118458, 2024.

[27] Bartosz Pietrucha. Angular Architecture Patterns and Best Practices (that help to scale). `https://dev-academy.com/angular-architecture-best-practices/`, 2019. Dev Academy.

[28] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. pages 373–382, 2015.

[29] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2016.

[30] Sivanandam S. and Paulraj M. *Introduction to Artificial Neural Networks*. Vikas Publishing House Pvt Limited, 2009.

[31] Benoît Saenz de Ugarte, Abdelhakim Artiba, and Robert Pellerin. Manufacturing execution system - A literature review. *Production Planning and Control - PRODUCTION PLANNING CONTROL*, 20:525–539, 2009.

[32] Arthur L Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[33] Deval Shah. Mean Average Precision (mAP) Explained: Everything You Need to Know. `https://www.v7labs.com/blog/mean-average-precision`, 2023. V7.

[34] Pranshu Sharma. Basic Introduction to Convolutional Neural Network in Deep Learning. `https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/#h-background-of-cnns`, 2023. Analytics Vidhya.

[35] Siemens. Data Visualization and AI Quality Control to Detect Defective Parts. `https://webinars.sw.siemens.com/pt-BR/data-visualization-and-ai-quality-control-to-detect-defective-parts/`.

[36] Xian Tao, Dapeng Zhang, Wenzhi Ma, Xilong Liu, and De Xu. Automatic Metallic Surface Defect Detection and Recognition with Convolutional Neural Networks. *Applied Sciences*, 8(9):1575, 2018.

[37] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. *arXiv preprint arXiv:2402.13616*, 2024.

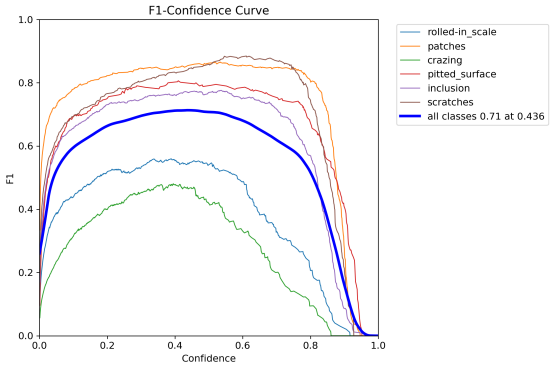
[38] Muhamad Yani, S Irawan, and Casi Setianingsih. Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. *Journal of Physics: Conference Series*, 1201:012052, 05 2019.

Appendix A

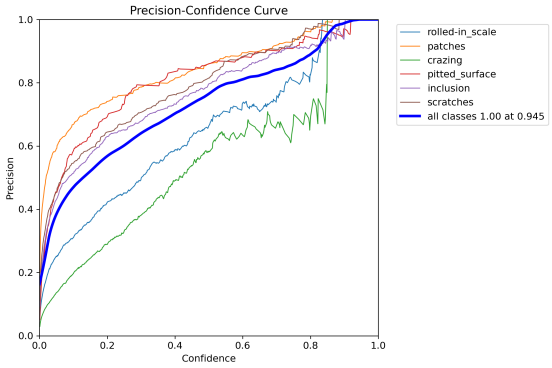
Annex

A.1 NEU-DET Model Results

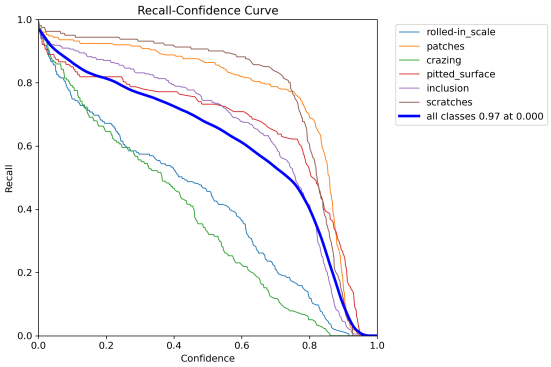
A.1.1 YOLOv5m6u



(a) F1 Score Curve



(b) Precision Curve



(c) Recall Curve

Figure A.1: YOLOv5m6u Model Results for NEU-DET

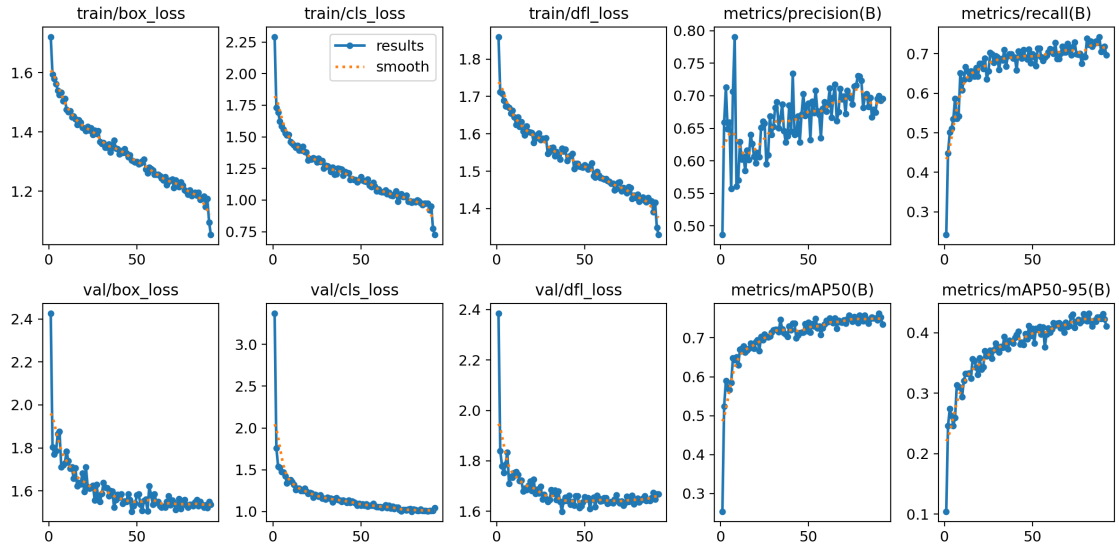
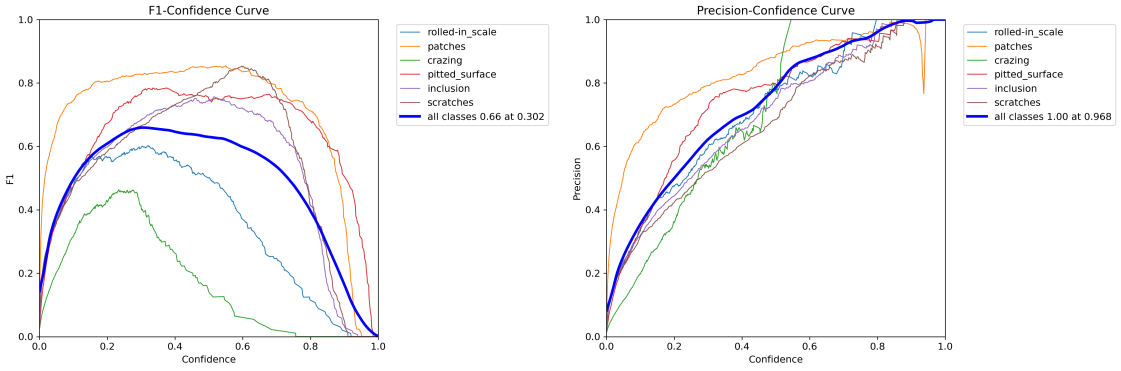


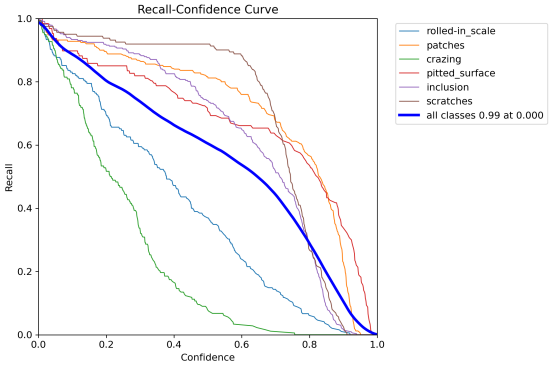
Figure A.2: Several Metrics Results for NEU-DET on YOLOv5m6u

A.1.2 YOLOv8m



(a) F1 Score Curve

(b) Precision Curve



(c) Recall Curve

Figure A.3: YOLOv8m Model Results for NEU-DET

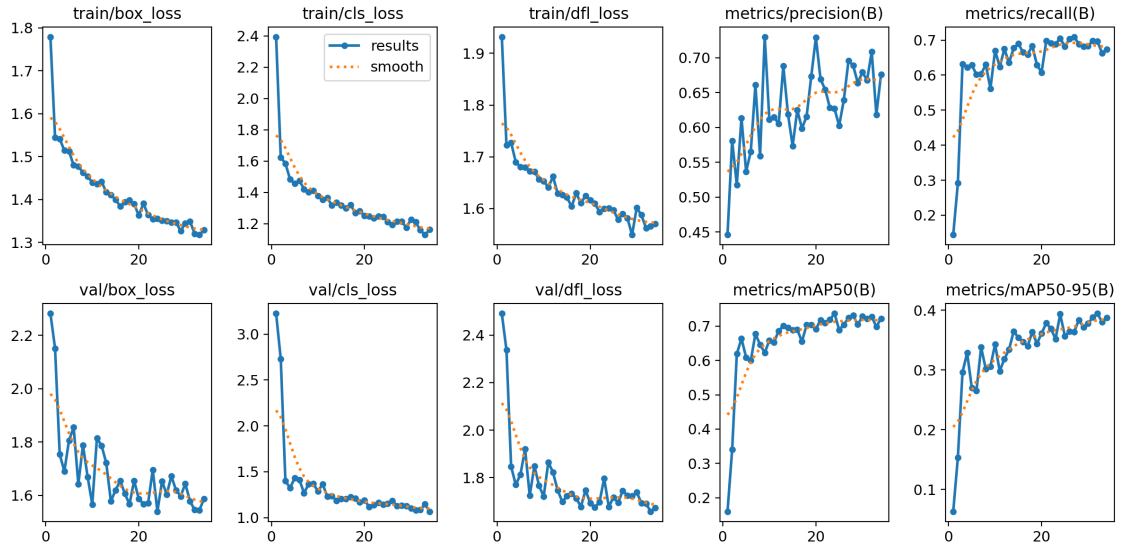
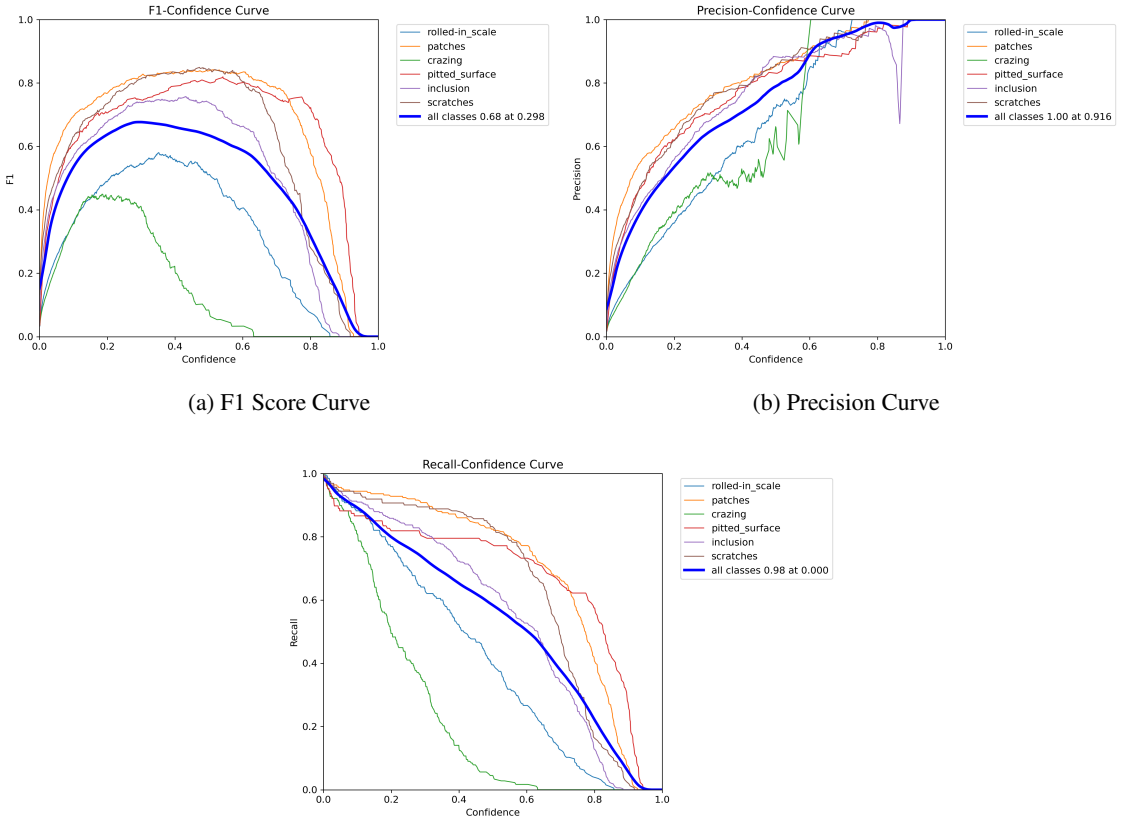


Figure A.4: Several Metrics Results for NEU-DET on YOLOv8m

A.1.3 YOLOv8x



(a) F1 Score Curve

(b) Precision Curve

(c) Recall Curve

Figure A.5: YOLOv8x Model Results for NEU-DET

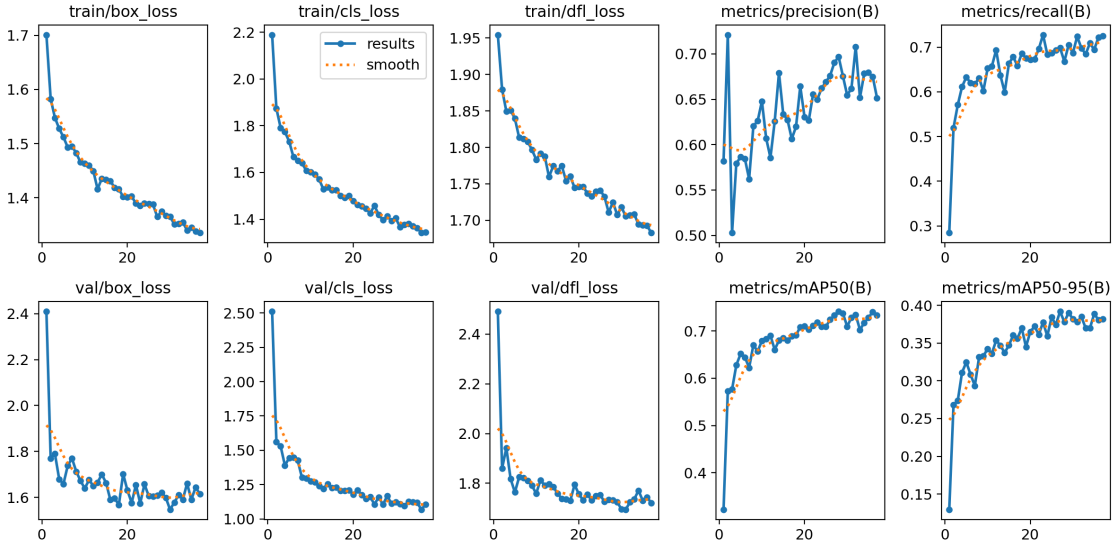
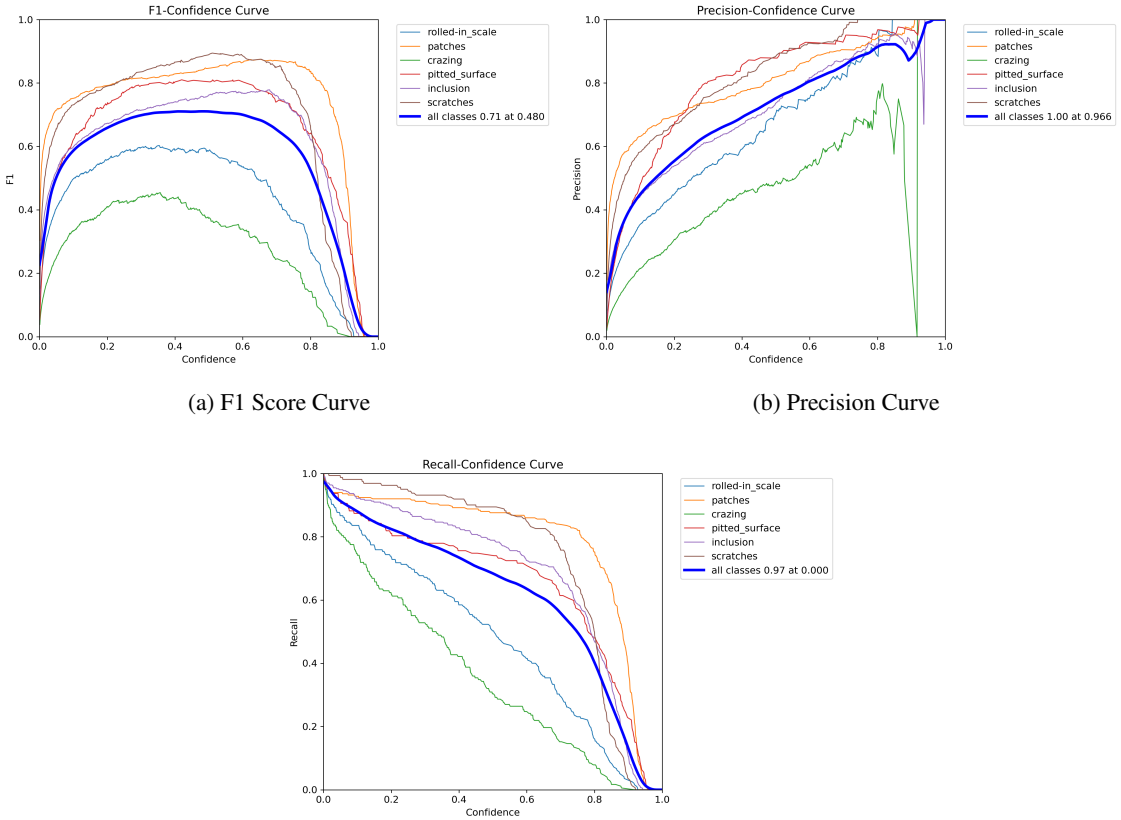


Figure A.6: Several Metrics Results for NEU-DET on YOLOv8x

A.1.4 YOLOv9e



(a) F1 Score Curve

(b) Precision Curve

(c) Recall Curve

Figure A.7: YOLOv9e Model Results for NEU-DET

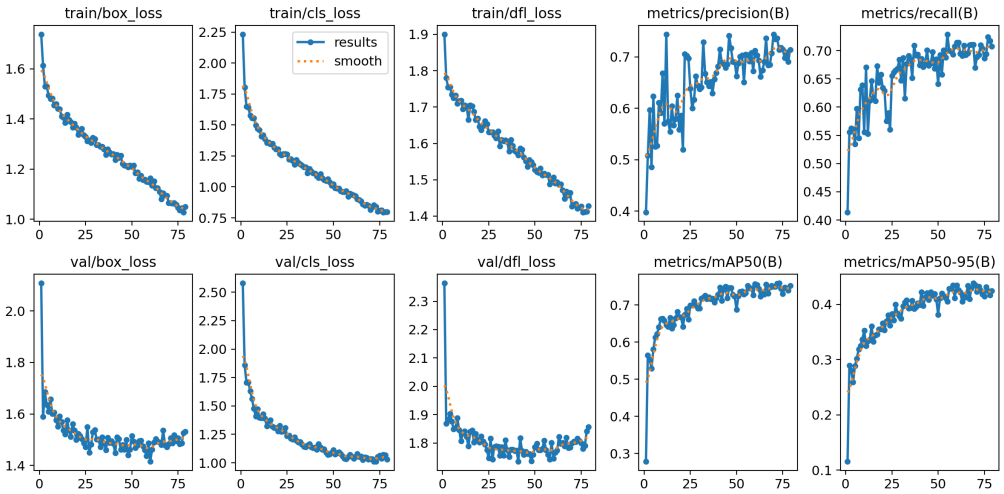
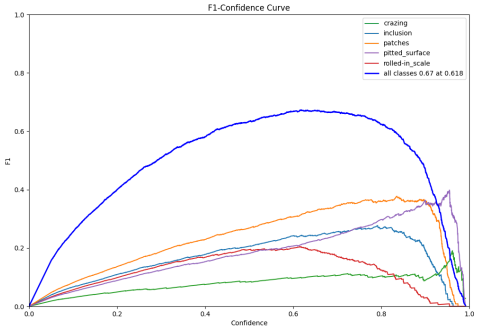
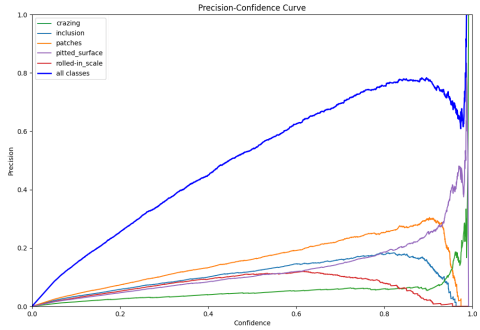


Figure A.8: Several Metrics Results for NEU-DET on YOLOv9e

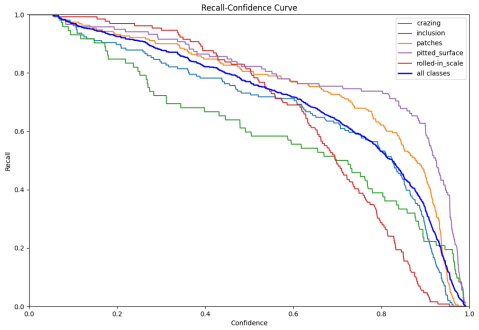
A.1.5 Faster R-CNN



(a) F1 Score Curve



(b) Precision Curve

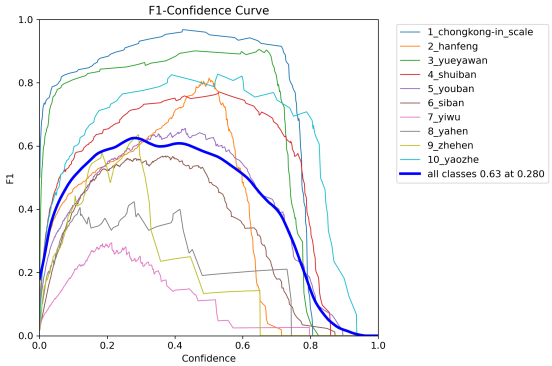


(c) Recall Curve

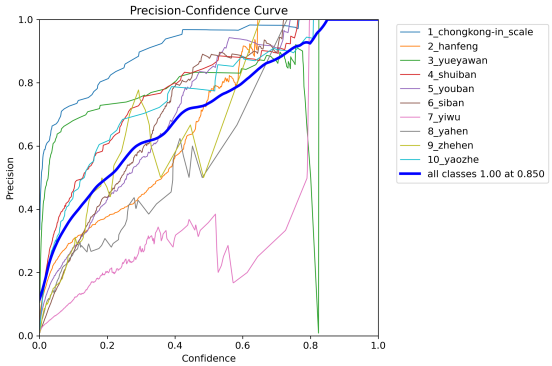
Figure A.9: Faster R-CNN Model Results for NEU-DET

A.2 GC10-DET Model Results

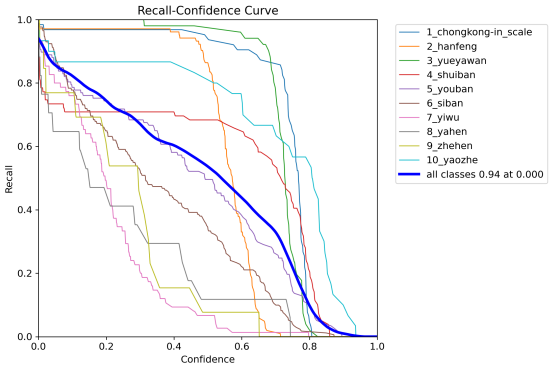
A.2.1 YOLOv5m6u



(a) F1 Score Curve



(b) Precision Curve



(c) Recall Curve

Figure A.10: YOLOv5m6u Model Results for GC10-DET

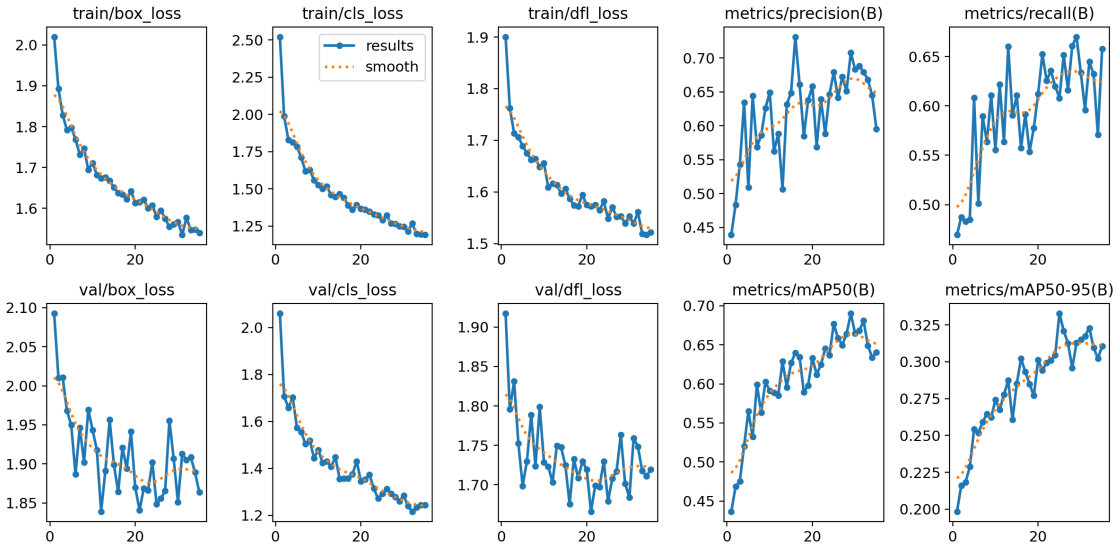
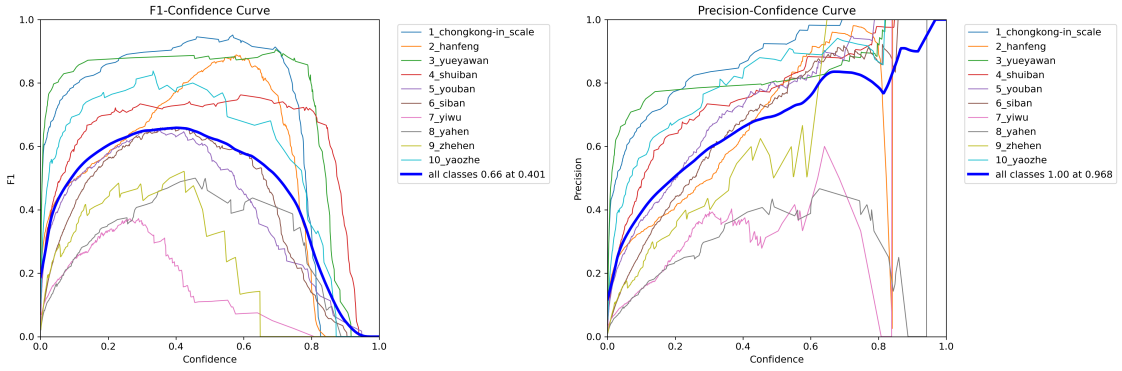


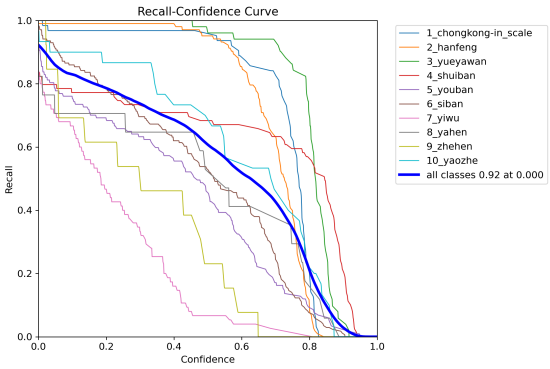
Figure A.11: Several Metrics Results for GC10-DET on YOLOv5m6u

A.2.2 YOLOv8m



(a) F1 Score Curve

(b) Precision Curve



(c) Recall Curve

Figure A.12: YOLOv8m Model Results for GC10-DET

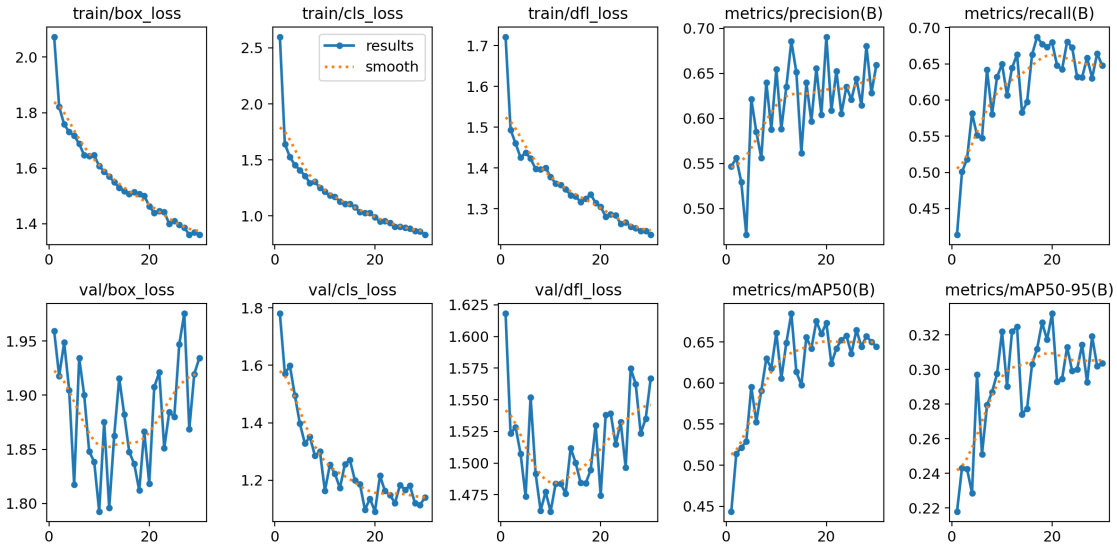
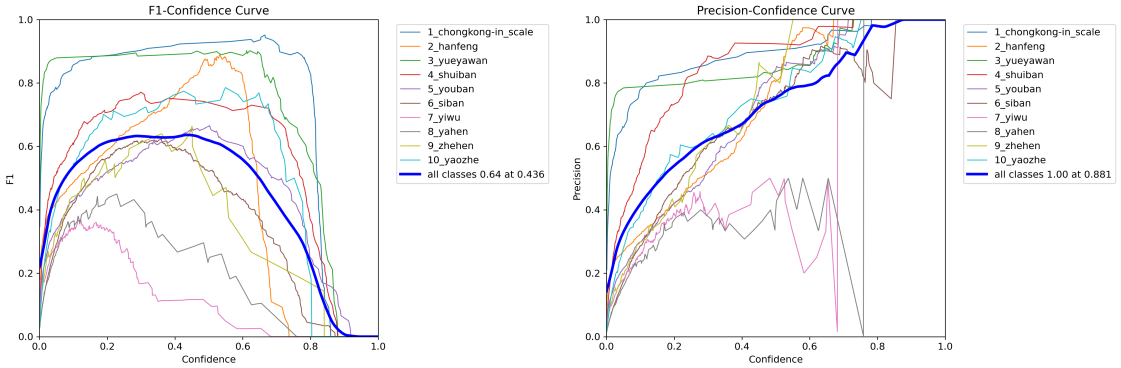


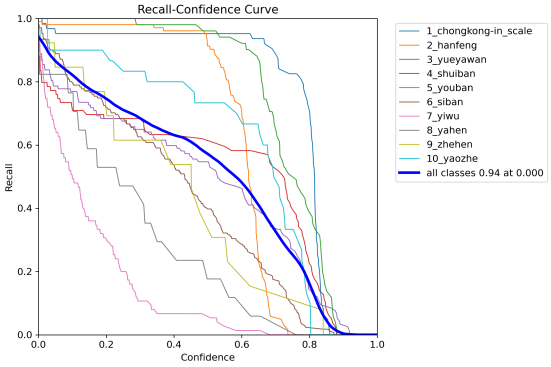
Figure A.13: Several Metrics Results for GC10-DET on YOLOv8m

A.2.3 YOLOv8x



(a) F1 Score Curve

(b) Precision Curve



(c) Recall Curve

Figure A.14: YOLOv8x Model Results for GC10-DET

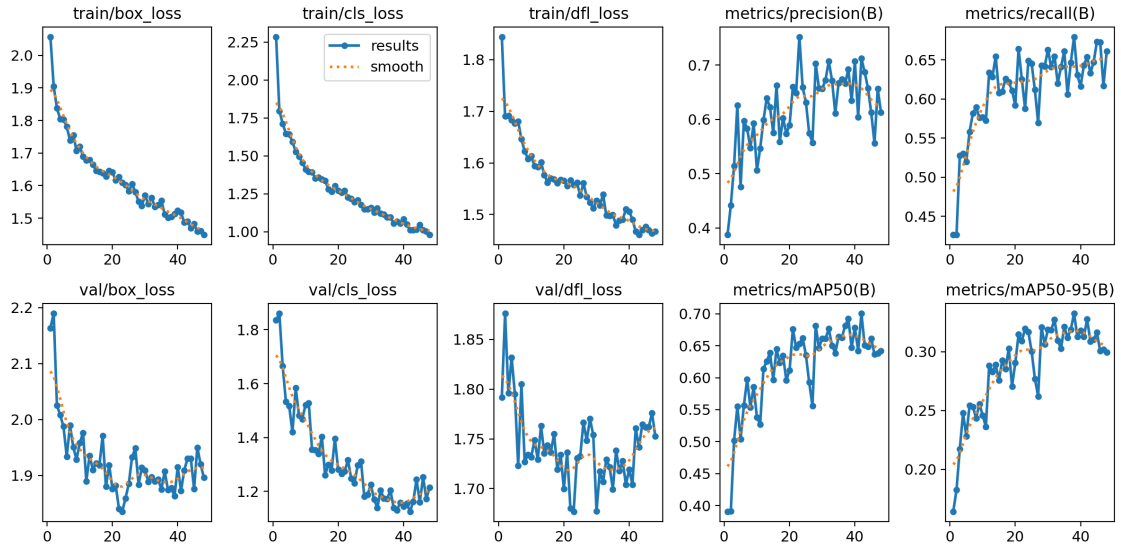
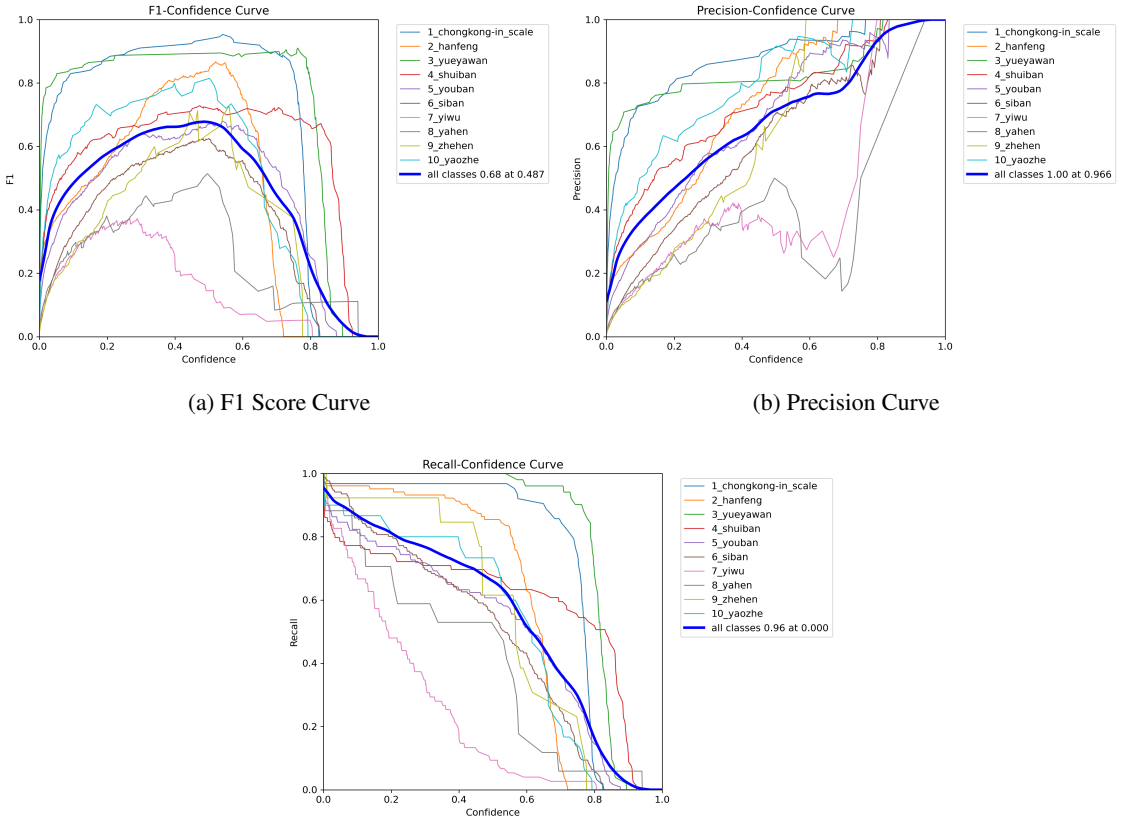


Figure A.15: Several Metrics Results for GC10-DET on YOLOv8x

A.2.4 YOLOv9e



(a) F1 Score Curve

(b) Precision Curve

(c) Recall Curve

Figure A.16: YOLOv9e Model Results for GC10-DET

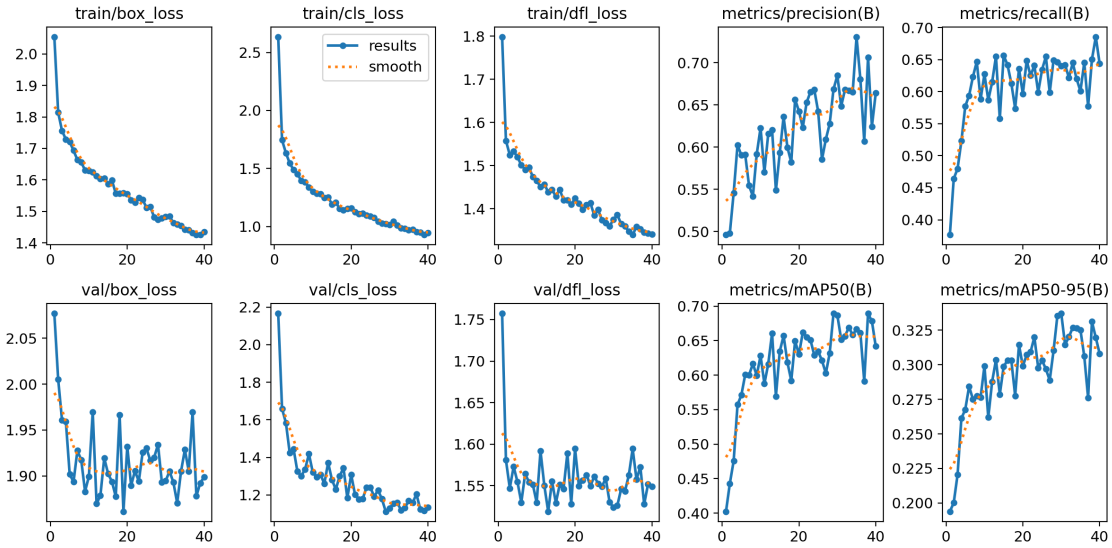
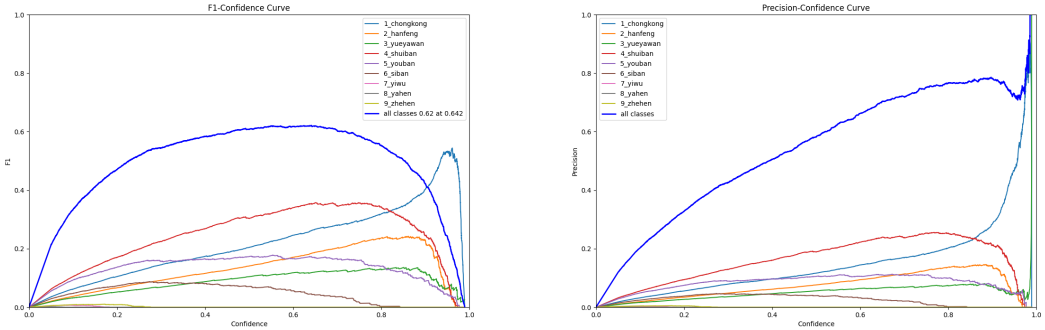


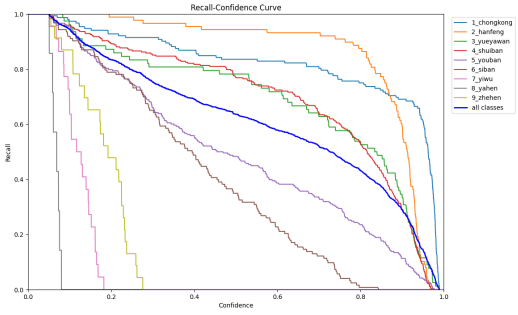
Figure A.17: Several Metrics Results for GC10-DET on YOLOv9e

A.2.5 Faster R-CNN



(a) F1 Score Curve

(b) Precision Curve



(c) Recall Curve

Figure A.18: Faster R-CNN Model Results for GC10-DET

A.3 Questionnaire Guide

Welcome to the evaluation of the AI-based defect detection system integrated into our Manufacturing Execution System. This guide is designed to help you navigate through the evaluation process, providing clear instructions on how to use the system, what to observe, and how to provide your valuable feedback. Your insights are crucial to understand the system's performance and identify areas for improvement.

A.3.1 Evaluation Process Overview

The evaluation process consists of the following steps:

1. **Getting Started:** Instructions on how to access the defect model section.
2. **Using the System:** A step-by-step guide on how to use the defect detection system for detecting defect tasks.
3. **Observing and Recording:** What to look for while using the system and how to record your observations.
4. **Providing Feedback:** Completing the evaluation questionnaire to share your experiences and suggestions.

A.3.2 Getting Started

Step 1: Accessing the System

- Ensure you have access to a computer connected to the network.
- Open the web browser and navigate to the system's URL.
- Log in using your credentials.

Step 2: System Navigation

- In the *Administration* page, you will find a button that will redirect you to the defect models page, where the information about the machine learning models is located.
- Familiarize yourself with the main dashboard and available features.

A.3.3 Using the System

Imagine the scenario where your fabric is producing hot-rolled steel strips. To guarantee the quality of the produced materials, you need to make sure that they don't contain any defects.

Assuming that you already know how to signalize a defect in the already existing defect system of the CM MES, you want to try the new feature that created the defects for you, but for that, there

are several steps you need to take into account. To make this possible you first need first to create a model with a dataset already in the required format (the one previously sent to you).

Step 1: Create a Defect Model

- Navigate to the *New* button.
- Give the information required for the model
- Create the model.

Step 2: Train the Created Defect Model

- Select the row of the model you wish to train, which must be in the *NotTrained* state.
- Navigate to the *Train Model* button. Take into account that this button will only appear when you have the defect model row selected.
- Train the model.

Like every other model, it takes time for this process to be finished, so the next step is not immediate.

After step 2, you will be redirected to a page showing you the created entity's information. Go back to the defect models dashboard page.

Step 3: Check Model Status

- Select the *Refresh* button to update the status of the model.
- If the model is trained, the state will be updated from *Training* to *Trained*. If not, you will have to come back later.

If you wish, you can test the model to check with some images if the predictions seem correct.

Step 4: Test the Model

- Select the row of the model you wish to test, which state must be *Trained*.
- Navigate to the *Test Model* button. Note that this button will only appear when you have the defect model row selected.
- Upload an image you wish to test the model on.
- Test the model.

If you're happy with the results, let's move to the next steps.

No, the scenario is that a material is in the stage where it's quality is being checked. Follow the next steps.

Step 5: Search for Defects in a Material

- Access the *Business Data* page.

- Navigate to the *Material* page.
- Select a material. **It must be associated with the same product that the defect model was created on.**
- Select the *Views* button located in the upper right corner inside the *Material* details page.
- Select the *Defects* option in the dropdown menu.
- Navigate to the *Predict Defects* button.
- Upload the image in the available section for it.
- Select the *Search Defects* button and wait for the results.

Step 6: Viewing Defect Detection Results

- After processing, an image with the results will be returned.
- If you are happy with the results, select the *Create Defects* button.
- The wizard will close, and on the left side of the page, you can observe each created defect and its information.
- Review the detected defects highlighted in the images.

A.3.4 Observing and Recording

Pay attention to the following aspects while using the system:

- Ease of navigating the user interface.
- Speed and accuracy of defect detection.
- Any technical issues or delays experienced.

Make notes of your observations to refer to when completing the questionnaire.

A.3.5 Providing Feedback

After using the system, fill out the evaluation questionnaire provided. The questionnaire consists of multiple sections covering usability, performance, effectiveness, satisfaction, and technical issues. Provide **honest** and detailed responses to each question. Your feedback is valuable for improving the system.

A.4 Questionnaire Form

A.4.1 Usability

1. **How easy is it to use the defect detection system?**

   ☐ Very difficult

   ☐ Difficult

   ☐ Neutral

   ☐ Easy

   ☐ Very easy

2. **How intuitive do you find the user interface of the system?**

   ☐ Not intuitive at all

   ☐ Slightly intuitive

   ☐ Neutral

   ☐ Intuitive

   ☐ Very intuitive

3. **How quickly were you able to learn to use the system?**

   ☐ Very slowly

   ☐ Slowly

   ☐ Neutral

   ☐ Quickly

   ☐ Very quickly

A.4.2 Performance

4. **How would you rate the accuracy of the defect detection?**

   ☐ Very inaccurate

   ☐ Inaccurate

   ☐ Neutral

   ☐ Accurate

   ☐ Very accurate

5. **How would you describe the speed of the defect detection process?**

   ☐ Very slow

   ☐ Slow

- ☐ Neutral
- ☐ Fast
- ☐ Very fast

6. **Have you experienced any delays or performance issues while using the system?**

   - ☐ Very frequently
   - ☐ Frequently
   - ☐ Neutral
   - ☐ Rarely
   - ☐ Never

A.4.3 Effectiveness

7. **How effective is the system in improving the defect detection process compared to previous methods?**

   - ☐ Very ineffective
   - ☐ Ineffective
   - ☐ Neutral
   - ☐ Effective
   - ☐ Very effective

8. **Do you find the defect detection results to be reliable?**

   - ☐ Very unreliable
   - ☐ Unreliable
   - ☐ Neutral
   - ☐ Reliable
   - ☐ Very reliable

9. **Has the system helped reduce the time needed to identify and address defects?**

   - ☐ Not at all
   - ☐ Slightly
   - ☐ Neutral
   - ☐ Significantly
   - ☐ Greatly

A.4.4 Satisfaction

10. **Overall, how satisfied are you with the defect detection system?**

    ☐ Very dissatisfied

    ☐ Dissatisfied

    ☐ Neutral

    ☐ Satisfied

    ☐ Very satisfied

11. **Would you prefer using this system over previous defect detection methods?**

    ☐ Definitely not

    ☐ Probably not

    ☐ Neutral

    ☐ Probably

    ☐ Definitely

12. **Are there any features you dislike or find unnecessary?**

    ☐ Yes

    ☐ No

13. **If you answered yes to the previous question, which features do you find unnecessary?**

A.4.5 Technical

14. **Have you encountered any technical issues or bugs while using the system?**

    ☐ Yes

    ☐ No

15. **Please describe the issues and their impact on your workflow.**

16. **How critical are these issues in terms of impact on your workflow?**

    ☐ Not critical at all

    ☐ Slightly critical

    ☐ Neutral

    ☐ Critical

    ☐ Very critical

A.4.6 Additional feedback

17. **Do you have any suggestions for improving the defect detection system?**
18. **Are there any additional features you would like to see in the system?**
19. **Any other comments or feedback?**