Arab Academy for Science, Technology and Maritime Transport

College of Engineering and Technology

Department of Electronics and Communication

B. Sc. Final Year Project

PCB Defect Detection and Sorting System

Using YOLOv8 algorithm

Presented By:

Mennatallah Yousri
Norhan Magdy
Omar Ahmed
Mohamed Magdy
Mohamed Salah

Supervised By:

Dr. Hesham Hamdy Ali
Dr. Mohamed Atef Al Khoraibee

J u n e – 2 0 2 4

DECLARATION

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Bachelor of Science in Electronics and Communication Engineering is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____

Registration No.: _____

Date: / /

ACKNOWLEDGMENT

First and foremost, all perfect praise be to Allah, The Almighty and The Most Merciful, for the blessings bestowed upon us during the research and development process of this project. May his blessing and wisdom continue to guide us through further academic achievements.

We would like to express our heartfelt gratitude to everyone who gave us the opportunity to complete this project, especially our supervisors Dr. Hesham Hamdy and Dr. Mohamed Atef, for their tremendous patience, support and assistance in this project. The completion of this project would not have been possible without their continuous guidance.

We would also like to take this opportunity to thank Dr. Mohamed Abaza, Head of Electronics and Communications Engineering Department, for his leadership and support for our academic endeavors, ensuring that each student has the opportunities to gain the required knowledge for starting successful careers. In addition, we would like to extend our thanks to Dr. Mona Fouad, Dean of the College of Engineering and Technology, for her commitment in fostering an intellectual curiosity encouraging environment, and Dr. Ismail Abd El Ghafar President of Arab Academy for Science, Technology, and Maritime Transport for his commitment and vision in creating a world-class academic institution that prioritizes research and innovation.

ABSTRACT

Most industries nowadays depend mainly on printed circuit boards (PCB) as the key component in electronic devices. With the realization of the significance of PCBs and their extensive utilization across various electronic products, the production of these components has been entrusted to specialized factories for efficient mass manufacturing. However, it has been discovered that defects in PCBs may remain undetected during manual inspection, resulting in product malfunction. We propose a specialized station in the production line where the PCBs undergo thorough examination for various types of defects. Subsequently, the defective PCBs are automatically sorted for potential repair or reuse. To accomplish our goal, we have developed a highly accurate defect detection model based on the state-of-the-art YOLOv8 architecture. Our model has undergone meticulous training using the TDD-Net dataset, which encompasses a wide range of PCB layouts including 6 types of surface defects. Through rigorous testing, employing different versions of the YOLO framework and experimenting with different trade-offs, we have thoroughly evaluated the performance of each model and achieved an accuracy of 84.38% and 98.4% mAP with our YOLOv8 model which was then deployed on a Raspberry Pi to perform real-time detection of PCBs on the production line through continuous livestream.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

AI	Artificial Intelligence
AOI	Automated Optical Inspection
CAD	Computer Aided Design
CNN	Convolutional Neural Network
CPU	Central Processing Unit
FN	False Negative
FP	False Positive
GPU	Graphics Processing Unit
IR	Infrared
mAP	Mean Average Precision
P	Precision
PCB	Printed Circuit Board
R	Recall
RPi	Raspberry Pi
TN	True Negative
TP	True Positive
VENV	Virtual Environment
VM	Virtual Machine
YOLO	You Only Loon Once

LIST OF UTILIZED STANDARDS

The design and development of this project was conducted with accordance to a comprehensive set of industry and engineering standards to ensure compliance with safety regulations, compatibility with existing industrial infrastructure, and adherence to best practices in engineering design and manufacturing. The following standards were utilized throughout the project:

1. **Safety Standards:**
   - IEC 61010-1:2020 Safety requirements for electrical equipment for measurement, control, and laboratory use - General requirements.

     A standard that covers safety regulations including protection against electric shocks, fire prevention, mechanical hazard control, compliance verification and others.

   - ANSI/RIA R15.06-2018 Safety requirements for industrial robots and robotic systems.

     A standard that encompasses all aspects of robotic safety including safeguarding, documentation, and labelling.

2. **Image Processing and Deep Learning Standards:**
   - IEEE 830-2015 Standard for Representation of Electronic Diagrams

     It establishes a standardized format for representing and exchanging electronic diagrams to ensure machine readability, enhanced documentation, improved design quality and automated design tools integration.

   - PASCAL VOC (Pattern Analysis, Statistical Modeling, and Computational Learning in Vision) VOC2012

     This standard provides a common dataset and evaluation framework for object detection and image classification tasks. It also includes guidelines for labeling bounding boxes and object classes in images.

3. **Industry 4.0 Standards:**
   - ISO IEC 62852:2019 Telecommunications and information exchange between systems – Local and metropolitan area networks – Part 2: Technical characteristics of industrial automation networks (TSN)

     It is an international standard that defines the technical characteristics of industrial automation networks. It specifies a set of protocols and mechanisms that enable deterministic and low-latency communication in industrial environments. The key features of this standard are low latency, clock synchronization, and deterministic data delivery.

LIST OF REALISTIC CONSTRAINTS

1. **Resource Constraints**
   This project is subject to a tight budget and must be completed within an eight-month timeframe.

2. **Technological Constraints**
   The software must integrate with existing factory systems and compatible with installed devices running on Linux-based operating systems.

3. **External Constraints**
   The project must comply with the safety requirements specified by industrial factories, and data transfer must comply with General Data Protection Regulation (GDPR) and ensure that user data is handled securely and in accordance with data protection laws.

4. **Operational Constraints**
   The software must support handling livestreams from multiple cameras at once.
   The processing node should have edge capabilities.
   The electronic components consumed current must be limited to low levels.

5. **Performance Constraints**
   The manufactured parts size must be limited to the size of laser cutting machines available.
   Electronic components should be multithreaded to run simultaneously on the single-core processor used.

Chapter One

1 INTRODUCTION

1.1 BACKGROUND

Printed circuit boards (PCBs) are the backbone of all electronic devices on which several components are placed and connected. Selecting the appropriate production process of PCBs is affected by several factors including the complexity of the design, budget of the company and the degree of dependency of a company on the PCB. However, the attempts of some managements to cut costs by outsourcing designs to lax manufacturers often lead to the manufacturing of several faulted PCBs. This can be attributed to several factors such as negligence or inattentiveness of employees responsible for testing the PCBs during manufacturing, poor quality tests including using outdated methods or cheap low-quality equipment, and/or using low quality materials in production.

1.2 MOTIVATION

As a part of the technological community and inhabitants of this planet, we believe that any action in one area impacts the outcomes of others and thus the development of the industrial and technological aspects of the country is a crucial step in achieving the sustainability and economic growth. This aligns with SDG's goal 9, related to industry, innovation, and infrastructure [1] which expresses that technological progress and promoting sustainable industries are keys in finding lasting solutions for economic and environmental challenges. In addition, the initiative "Egypt Makes Electronics," led by the Presidency and overseen by the Ministry of Communications and Information Technology, seeks to establish Egypt as a regional and global hub for the design and production of advanced electronics for the African, Arab, and European markets by 2030 [2]. So, we believe that it is our mission to provide a sustainable automated solution that integrates AI into one of the most integral steps of electronics manufacturing, PCBs inspection.

1.3 PROBLEM DEFINITION

The defects of PCBs are varied in nature, some can be detected in bare circuit boards and others only occur after component assembly. These defects can be organized, according to the PCB feature they impact, into 4 primary classes: trace, component, solder, and via/pad. The defects can be categorized into subclasses according to their similarities. Trace defects include 4 subclasses. Additive trace defects are represented in spurs, which are excess conductor along traces, spurious copper, that are excess conductor patches in unintended locations, and short circuits. Subtractive defects are

open circuits and mouse bites, which is a term for depressions along traces. Damage defects are pin holes, flux residues and scratches, while design errors include missing traces, less-than-minimum optimal space between traces and excessive shorts [3]. While most of these defects can be avoided by using high quality materials and investing in more expensive equipment, human error can still cause issues in the final product during handling and inspection. In high-profile companies, legacy methods using manual data manipulation for various stages in producing and inspecting PCBs can also be error-prone and time-consuming. Dependency on them can result in hindering the production process and the inability to keep up with product complexities, shortened project schedules, reduced resources, and slashed project budgets [4].

1.4 REPORT ORGANIZATION

This document provides a comprehensive overview of the project's development process. Our project aims to address a prevalent problem and its documentation includes 5 main sections addressing the following:

1. **Chapter 2: Literature Review** explores the previous research and existing works that covers the same objective as ours in both software and hardware aspects.

2. In **Chapter 3: Proposed Project**, we explain our approach for providing a solution to the problem discussed.

3. **Chapter 4: Materials and Methods** details methods used in this project including algorithms, tools and implementation procedures as well as all materials used to build the project.
4. Through C**hapter 5: Results and Discussion**, we go through the results attained by our project and analyze the impact of these results.
5. Finally, **Chapter 6: Conclusion and Recommendation** wraps up the outcome of the project and possible further steps.

Chapter Two

2 LITERATURE REVIEW

As printed circuit boards (PCBs) get smaller and more complex, rigorous quality control procedures must be followed both during the manufacturing process and during testing. Conventional inspection techniques, which frequently depend on human labour, find it difficult to keep up with these developments. This inefficiency raises the risk of errors, delays in production, and higher expenses. The industry has implemented a number of inspection methods to overcome these constraints, Automated Optical Inspection (AOI) being one of the most popular. Even though AOI has many benefits, it still has drawbacks, such as inaccurate assessments brought on by changes in illumination and irregularities in the surface. These flaws frequently require manual reinspection, which brings back the chance of human error. Deep learning-based defect detection techniques are becoming a viable substitute in response to these difficulties. The capacity of deep learning to automatically extract features from data has enormous promise for increasing the precision and effectiveness of PCB inspection.

In essence, the task of surface defect detection involves utilizing Deep Learning algorithms to identify defects in images. As such, our primary focus is on reviewing Deep Learning and various types of algorithms that have been employed in the past when exploring PCB defect detection mechanisms.

2.1 DEEP LEARNING

Deep learning [5] plays a crucial role in automated PCB defect detection due to its ability to learn complex patterns from data. This technology utilizes multi-layered neural networks that mimic the human brain's learning process. These networks analyze vast amounts of training data, automatically extracting relevant features and internal representations. This capability makes deep learning highly effective in various fields, including object detection, natural language processing, and image processing.

Within the realm of target detection, deep learning algorithms can be categorized as one-stage or two-stage. One-stage algorithms, such as SSD and YOLO, prioritize speed over accuracy. They achieve rapid processing by directly predicting bounding boxes and class probabilities in a single network pass. This makes them well-suited for real-time applications like PCB defect detection in industrial settings, where production efficiency hinges on fast inspection times.

On the other hand, two-stage algorithms (e.g., R-CNN, Fast R-CNN, Faster R-CNN) emphasize higher accuracy by generating region proposals followed by classification within those regions. However, this added step comes at the cost of slower processing speeds.

Considering the importance of speed in industrial PCB inspection, one-stage algorithms emerge as the preferred approach. Among these, YOLO stands out as a popular choice in most recent research due to its efficient detection capabilities.

2.2 YOLO

One of the most prominent deep learning approaches for PCB defect detection is the You Only Look Once (YOLO) algorithm. YOLO prioritizes speed and efficiency, making it well-suited for real-time industrial applications [5]. It achieves this by processing the entire image at once, dividing each pixel into multiple bounding boxes of different sizes to detect potential objects. Each bounding box is assigned a confidence level indicating the likelihood of an object being present.

A key strength of YOLO is its end-to-end nature. Unlike traditional methods that require separate pre-processing steps, YOLO utilizes a convolutional neural network to directly process the raw image data. This eliminates the need for pre-processing and reduces computational overhead during training and testing, leading to faster model training and deployment. Additionally, YOLO employs a fully connected layer to predict both bounding boxes and object categories in a single pass. This simultaneous prediction fosters a holistic understanding of the image and minimizes the risk of false positives.

2.2.1 YOLOv5

YOLOv5 builds upon the core principles of YOLO and offers a step-by-step object detection methodology [6]. First, the input layer preprocesses the data using techniques like mosaic data augmentation and adaptive anchoring, enhancing the training data and improving model robustness. Following this, the backbone network, comprised of Focus and CSP structures, extracts key features from the image that are relevant for object detection. Next, the neck network, featuring an FPN+PAN architecture, gathers and integrates these extracted features, providing a comprehensive representation for the detection stage. Finally, the detection stage

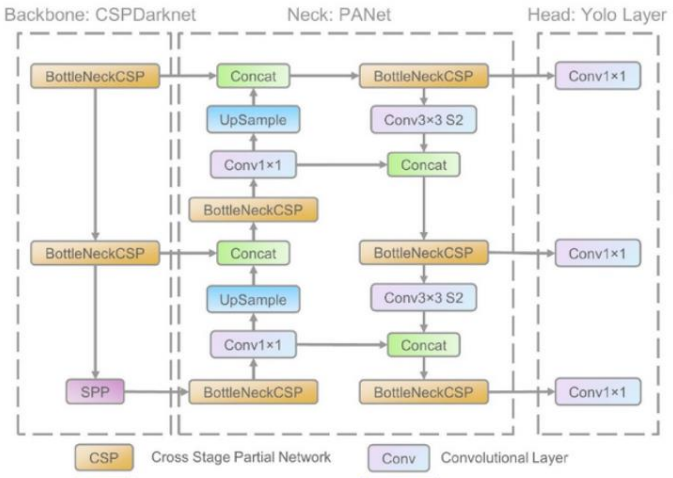


Figure 2.1 YOLOv5 architecture

predicts the bounding boxes and classifies the objects within them, calculating the associated loss during training. The complete architecture is shown in Figure 2.1. While the original YOLOv5 excels at detecting large objects in datasets like COCO, it may struggle with smaller components on circuit boards. This necessitates further development to enhance its ability to detect these smaller targets.

2.2.2 YOLOv8

YOLOv8 is the most recent iteration of the YOLO series by the time of writing this report, incorporating a state-of-the-art model and a novel network backbone structure [5]. It provides various model sizes (N/S/M/L/X) to cater to diverse industry needs. YOLOv8, shown in Figure 2.2, builds upon the strengths of previous generations by leveraging the CSP concept in its backbone network. It replaces the C3 module from YOLOv5 with a C2f module, facilitating a richer gradient flow. Additionally, it removes the convolutional structure from the PAN-FPN up-sampling stage and directly feeds the features extracted by different backbone stages into the up-sampling operation. This approach, combined with the use of different channel numbers for various scale models, offers greater structural adaptability.

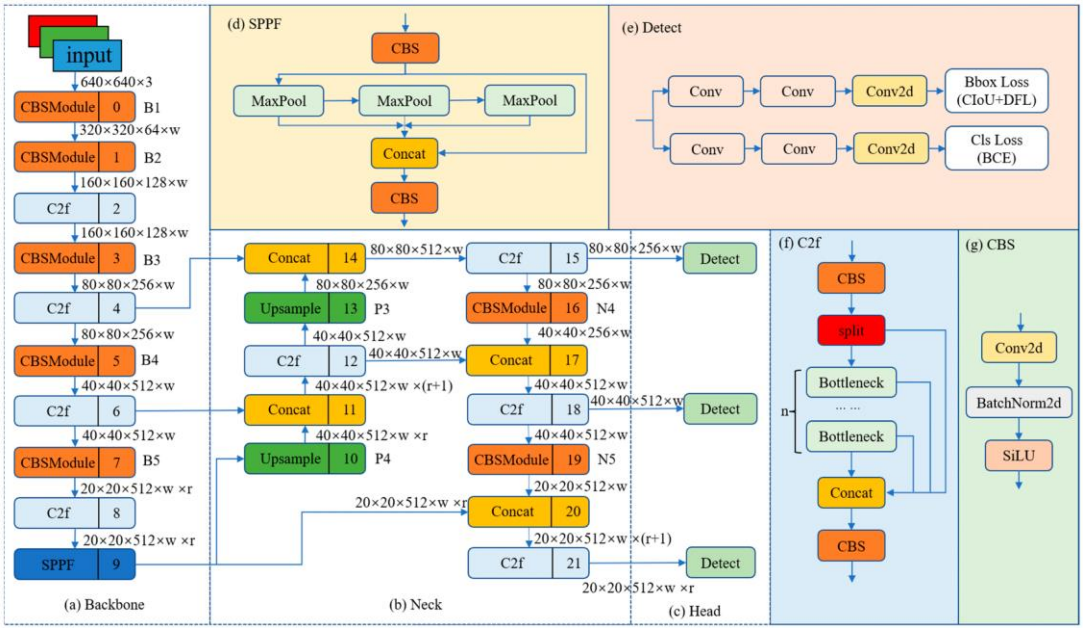


Figure 2.2 YOLOv8 architecture

Furthermore, YOLOv8 introduces a Decoupled Head, separating the classification and detection heads. It also transitions from an anchor-based approach in YOLOv5 to an anchor-free approach, integrating various deep learning techniques to elevate detection accuracy. The SPPF module, another innovation in YOLOv8, offers significant speed improvements compared to the traditional SPP module while maintaining comparable performance.

2.3 RELATED WORKS

In order to eliminate human error or reduce it to a new minimum, research approaches have been redirected to a new approach of utilizing deep learning models in identifying PCB defects. From the previous research conducted in this field, multiple approaches have been proposed such as *J. Lim et al* [7] contribution of a YOLOv5-based model for PCB defect detection with anomalous alarming system which introduces a modified FPN and enhanced regression accuracy of CIoU loss. This approach yielded impressive outcomes including 99.17% mAP and real-time inference at 90 frames per second on TDD-Net dataset of 10,688 images including six types of defects. Other approaches based on YOLOv5 is proposed by *J. Tang et al* [8] where they create a modified YOLOv5 algorithms through implementing K-means++ algorithm for setting more suitable anchors and utilizing EIoU loss function for optimizing the regression process of prediction and detection frames, enhancing the localization of small targets and achieving a satisfactory balance between performance and consumption, reaching 95.97% mAP at 92.5 FPS on Peking University Intelligent Robot Development Laboratory dataset of 1386 images of 6 defect types. Another proposed model by *Y. Yang and H. Kang* [9] introduces an improved YOLOv7 architecture for detecting PCB defects by adding a SwinV2_TDD module which serves as one more convolutional layer to extract the local features of the PCB images, resulting in a remarkable mAP of 98.74% on Peking University's opensource dataset of 693 images which had undergone augmentation to reach 9920 images of the same 6 defect types. More recent studies proposed YOLOv8-based models such as the W-YOLOv8, proposed by *P. Chen and F. Xie* [5], which is an improvement on the YOLOv8 algorithm. It utilizes a dynamic non-monotonic focusing mechanism with gradient gain allocation and wise IOU to optimize model performance. W-YOLOv8 achieves superior results in terms of precision, recall, and mAP compared to the original YOLOv8. Notably, it achieves an mAP50 of 97.3%, surpassing YOLOv8 by 1.35%, and an mAP50-95 of 55.4%, exceeding YOLOv8 by 3.94%. This demonstrates W-YOLOv8's effectiveness in real-time detection of six common PCB surface defects without requiring additional training data. The experiments were conducted on an open-source PCB defect dataset from Peking University containing 1386 images of various defect types. Another model for surface defect detection is NHD-YOLOv8, proposed by *J. Chen et al* [10], which incorporates three independent improvements: SFPN, ADH, and SSOCP. SFPN enhances feature extraction by fusing features from multiple backbone levels. ADH leverages the enhanced features from SFPN for improved classification and regression. Finally, SSOCP optimizes the training process to boost overall detection accuracy. NHD-YOLOv8 was evaluated on the PKU-Market-PCB dataset containing 693 images with six defect categories. Compared to YOLOv8, NHD-YOLOv8 achieved a significant improvement in Average Precision (AP) by 1.4 points, demonstrating its effectiveness in enhancing detection accuracy, particularly for smaller objects. Finally, a very comprehensive study conducted by *J. Lu and S. H. Lee* [11] proposes a novel network architecture for industrial surface defect detection. It utilizes a lightweight residual

structure with an attention mechanism to strengthen feature extraction within residual blocks. This approach allows the network to capture stronger features. Additionally, a feature pyramid network is employed to aggregate multiscale and multi-depth feature maps, providing the model with richer information and enabling it to focus on relevant features. The network outperforms previous models on various datasets, including PCB dataset, NEU_DET dataset, Tianchi fabric dataset, Tianchi ceramic tile dataset. The results on the PCB dataset achieved high mAP results for YOLOv8 models (YOLOv8s: 98.2%, YOLOv8n: 98.0%) on GPUs and maintained good performance (YOLOv8n: 98.1%) even when deployed on a Raspberry Pi, showcasing its potential for real-world applications. Other models were also proposed for defect detection using other architectures like *B. Hu and J. Wang* model using the Faster-RCNN architecture focuses on improving network structure to detect large targets with efficiency. The study reports a mean average precision of 94.21% and a detection speed of 0.08 seconds per image, indicating an improvement of 9%. However, it is essential to note that performance declined when tested with other datasets [12]. In addition, one notable project we encountered while exploring related works prioritizing cost-efficiency while maintaining high performance is a project investigating various deep learning models for PCB defect detection, including MobileNetV2 SSD FPN-Lite 320x320, FOMO, and YOLOv5, with different parameter configurations to optimize performance [13]. The FOMO (Faster Objects, More Objects) model emerged as the most effective choice, achieving an accuracy exceeding 80% for defect detection. Edge Impulse served as the platform for image annotation within the dataset and subsequent model training. The deployment process for the model was on a Raspberry Pi and an 8-megapixel V2 camera was used. On the Raspberry Pi 4 the model had a latency of approximately 1500ms (equivalent to 1 frame per second). This author suggests that a Raspberry Pi 4 might not be the optimal hardware choice for industrial settings requiring inspection of a high volume of PCBs but could be utilized to execute small projects.

Chapter Three

3 PROPOSED PROJECT

3.1 PROJECT HIGHLIGHTS

Aiming at the problem of automating PCB surface defect detection in industrial factories, combined with the research work of relevant literature, the detection of small defects on a moving production line is studied and an all-inclusive system for detecting, classifying and sorting defects is modelled and implemented in this project. The highlight of this work are as follows.

1. A comparative analysis of the performance of different YOLO versions and weights on custom PCB dataset was performed under fixed training parameters to determine the most suitable model taking with respect to a trade-off between performance, speed and size.
2. Deployment techniques of YOLOv8 on low-end devices was explored combined with the ability to detect defects in real-time from a continuous livestream of the production line.
3. Emulating the production line role in the PCB manufacturing process, a sorting mechanism was added for the defective PCBs in order to allow for further operations such as fixing the defects.

3.2 PROCESS FLOW

The provided flowchart, shown in Figure 3.1, visually depicts the workflow along the conveyor belt. Here's how it operates: first, the PCB reaches the camera connected to the Raspberry Pi 5. The deployed model on the Raspberry Pi analyses the PCB and detects any defects. If a defect is identified, the model transmits the defect's index (stored in an array) to the ATmega328 microcontroller. If no defect is detected, nothing is sent. The ATmega328p then stores the received index in a variable named "K." Next, it checks whether the IR sensor corresponding to the value of "K" is emitting a low signal (indicating the PCB's presence). If the IR sensor is low, signifying a PCB is present, the conveyor belt stops. Simultaneously, the servo motor arm rotates 90 degrees. The belt then restarts, carrying the PCB until it reaches the arm. Once in position, the belt stops again, and the servo motor arm rotates back to 0 degrees, effectively dropping the PCB into its designated container based on the identified defect (represented by the index value).

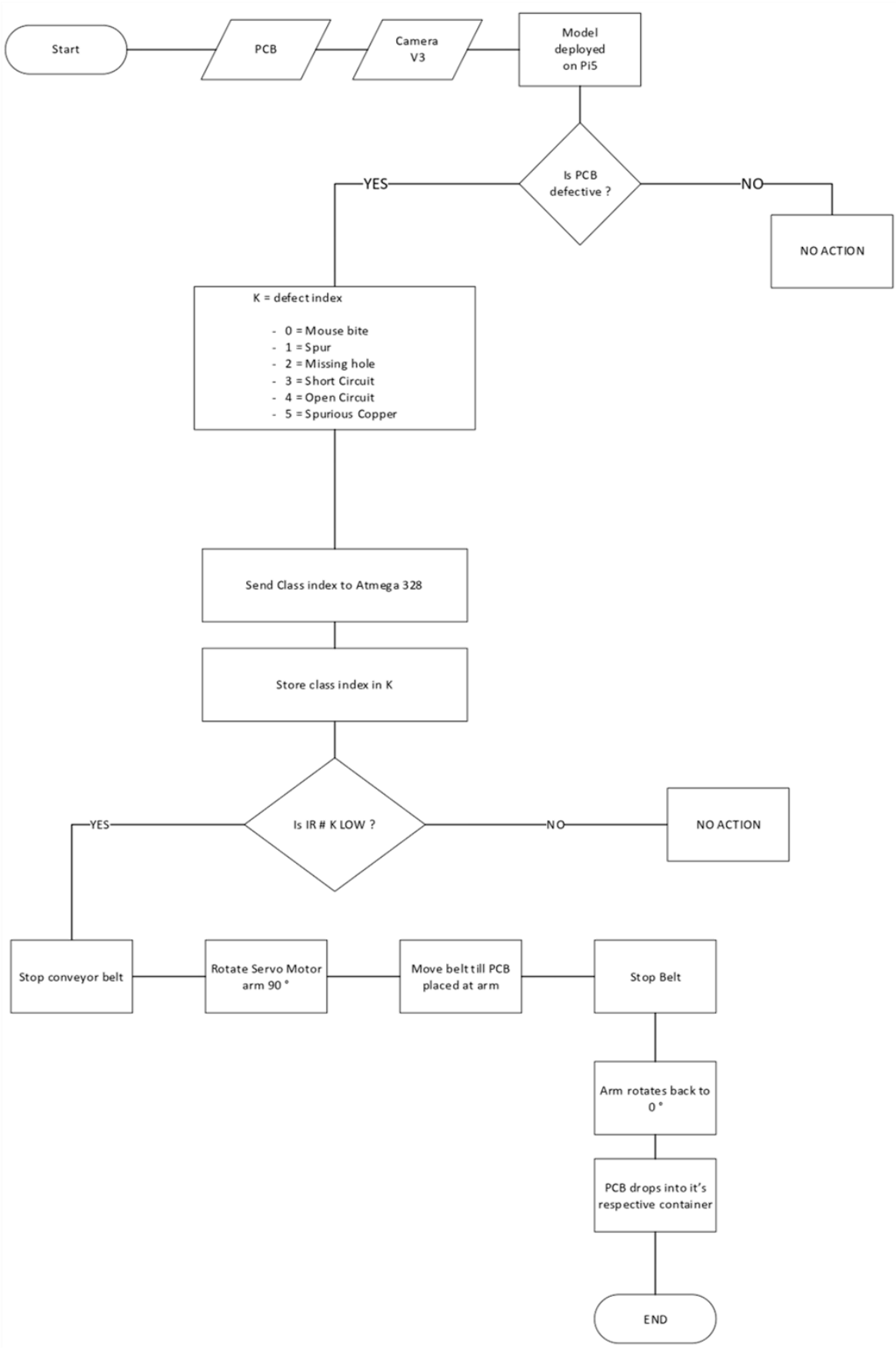


Figure 3.1 Proposed idea flowchart

Chapter Four

4 MATERIALS AND METHODS

In this chapter, all procedures utilized in planning, designing and implementing this project are detailed.

4.1 MATERIALS

4.1.1 For model training

The programming language used in this project is Python, which is an interpreted high-level open-source language that has been widely used in machine and deep learning recently in both industry and academia. In the early days, knowledge of C++ and CUDA were required for creating deep learning models, although the development of Theano and TensorFlow frameworks for Python greatly simplified the implementation of new models [14] as well as innovative libraries that are used with machine and deep learning such as PyTorch, Keras and Caffe. Training deep learning models requires intensive resources that are currently unattainable within the allocated budget for this project, so we opted for cloud-based approaches. Colab is a cloud service developed by Google Research that hosts Jupyter notebooks, saves them to user's drive and makes them shareable in the same manner as Google Docs. Colab offers three runtime types for free, CPU, T4 GPU and TPU. A free standard runtime can last up to 12 hours depending on availability of resources and user's usage patterns [15]. The python library psutil [16] is used on a Colab notebook to retrieve VM specs of Colab T4 GPU runtime workspace, which retrieved the following: CPU: dual Intel® Xeon® CPU with 2 vCPUs @ 2.30GHz, GPU: NVIDIA Tesla K80 T4 with 12GB of VRAM, RAM: 13GB. It is also worth noting that Colab Internet infrastructure is fast, especially when accessing resources from other Google services, such as Drive [17]. On the flip side, some of the limitations of this cloud service are as follows.

1. Programs written in compiled languages must be compiled on the user's device then uploaded for execution.
2. The VM and all files are lost after 12 hours and the user needs to restart and execute runtime from scratch if no checkpoints are saved and uploaded to an external disk, for example Drive, before runtime is interrupted.

Taking these limitations into consideration, Google Colab was the ideal platform to use in this project.

4.1.2 For model deployment

The chosen device used in deployment is Raspberry Pi, a single-board computer belonging to the flagship series, often referred to by the shorthand "Raspberry Pi", which was initially developed in the UK by Raspberry Pi Foundation with the intention of promoting the teaching of computer science in schools. The flagship series offers high-performance hardware, a full Linux operating system, and a variety of common ports in a form factor roughly the size of a credit card [18]. The model used in this project is Raspberry Pi 5, refer to Figures 4.1 and 4.2, the latest model developed by the time of writing this paper. It offers several features including PoE+ capable Gigabit Ethernet, 2.4/5GHz dual-band 802.11ac Wi-Fi 5 (300Mb/s), and Broadcom BCM2712 SoC with processor Quad-core Arm Cortex-A76 @ 2.4GHz that provides new features delivering performance uplift of 2-3x over its predecessor Raspberry Pi 4 for common CPU or I/O-intensive use cases.

Figure 4.1 Raspberry Pi 5 board [25]

Figure 4.2 Raspberry Pi 5 Case [18]

4.1.3 For production line emulation

Designing:

To design the production line prototype and emulate its working principle, we turned to Fusion 360, a cloud-based 3D modelling software developed by Autodesk in 2013. Fusion 360 employs a top-down approach, allowing users to build complex structures by assembling smaller, customizable components. This approach provides control over each element, letting you tailor the design to your specific needs.

Furthermore, Fusion 360's cloud-based nature facilitates seamless collaboration, enabling multiple co-designers to work on the same project simultaneously. This fosters efficient teamwork and real-time design iterations.

Matching the user-friendliness of most CAD software, Fusion 360 boasts a clear and intuitive interface. The top menu dynamically adapts its toolset based on your current task. For example, while sketching, tools like trim, mirror, offset, fillet, dimension, and projection appear at the top menu. As opposed to Solid tasks which has tools like extrude, revolve, hole, press pull and joint for assembling different parts together.

Fabrication:

Laser cutting is a subtractive manufacturing process that utilizes a computer-controlled, high-powered laser beam to achieve precise material removal. This technology allows for the creation of intricate two-dimensional shapes from various sheet materials. In the context of computer-aided design (CAD) software like Fusion 360, laser cutting offers a direct pathway to fabricate physical parts from virtual designs. The process begins by exporting the 3D model from Fusion 360 as a DXF file, a format widely recognized by laser cutting software. This file translates the geometric information of the design into a series of cutting paths for the laser to follow. Once the DXF file is uploaded to the laser cutter, shown in Figure 4.3, and the material is secured on the cutting bed, the laser beam executes the programmed path, generating clean and accurate cuts through the material. By leveraging the high precision of laser cutting, designers can fabricate complex wooden parts for applications such as prototyping, functional components, and decorative elements. In our project, it is specifically used to cut the conveyor belt parts for later assembly.

Figure 4.3 Laser Cutting Machine [26]

Processing:

An ATmega328p chip runs the entire production line prototype. ATmega328p is the 8-bit microcontroller used in Arduino Uno development board. Arduino is an open-source development board hosting a powerful microcontroller with a specialized environment called Arduino IDE which enables programming the board in C or C++ language. There are several board models varying in their capabilities, the one referred to during the development of this project is Arduino Uno running on the microcontroller ATmega328p @ 16MHz with auto-reset. The ATmega328p chip offers various features including low power consumption, wide availability, and low cost. The development board provides feasibility for using the microcontroller chip by preloading a bootloader on the attached chip enabling it to be programmed using the IDE through an on-board USB connection, and a set of easy-to-use GPIO pins enabling the Arduino board to directly connect to external components. Although Arduino boards provide ease of connectivity and simple interface, they are not suitable to use in real-world industrial projects due to their big sizes making them hard to fit in space-constrained applications, non-compliance with specific safety and regulatory certifications, and the cumbersome task of maintaining and scaling Arduino boards compared with project custom PCB design. In addition, the

ATmega328p chip is considered to have limited processing power when it comes to computationally intensive tasks. However, for the purpose of prototyping the production line in PCB factories to emulate the production process after the integration of deep learning visual defect detection model, the ATmega328p chip proves to be suitable to do the task within the limited budget.

To utilize the ATmega328p chip independently while keeping the feasibility of Arduino IDE, the chip is burned with a bootloader code. This code is loaded when the processor is powered up and it sets some chip parameters such as the internal clock frequency and number of internal registers, it also gets the ATmega328p to accept programs from the Arduino IDE on its serial RX and TX. For equivalent efficiency to that of the development board, the ATmega328p chip must be connected to a 16MHz crystal oscillator, two 22 pf capacitors, one 10k resistor and a decoupling 10uf capacitor on the supply terminals. The connection is shown in Figure 4.4.

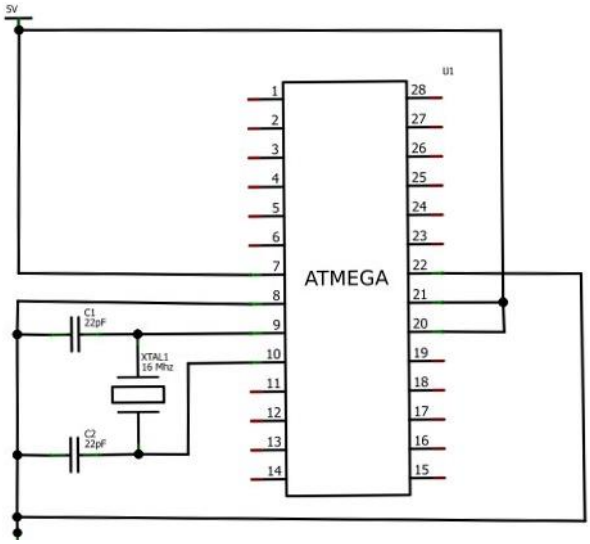


Figure 4.4 ATmega328p build hookup

4.2 BILL OF MATERIAL

All materials used in this project are listed below.

Component name	Quantity	Price/quantity(EGP)	Total price(EGP)
Arduino uno	1	394	394
Bearing mount	2	108	216
Belt	1	395	395
Cap	2	50	100
Coupler 8mm to 5mm	1	65	65
IR sensor	3	52	156
Laser cutting	1	160	160
Leather	1	43	43
Leather appliances	1	15	15
Micro HDMI to HDMI cable	1	526	526
Motor mount	2	162	324
Power supply	1	300	300
Raspberry pi 27W USB	1	767	767

Raspberry pi 5 8GB	1	4,162	4,162
Raspberry pi camera 3	1	1,486	1,486
Raspberry pi camera cable standard mini	1	158	158
Raspberry pi case for pi 5	1	688	688
Roller	2	385	770
Servo motors	2	80	160
Stepper motor	1	438	438
Value added tax		1758	1758
Wood paper	1	10	10
Total cost	-	-	13684

Table 1 Bill of Materials

4.3 METHODS

4.3.1 Design of system architecture

The system architecture can be categorized into three main systems, shown in Figure 4.6, image capturing system, detection system, and conveyor system. The first stage

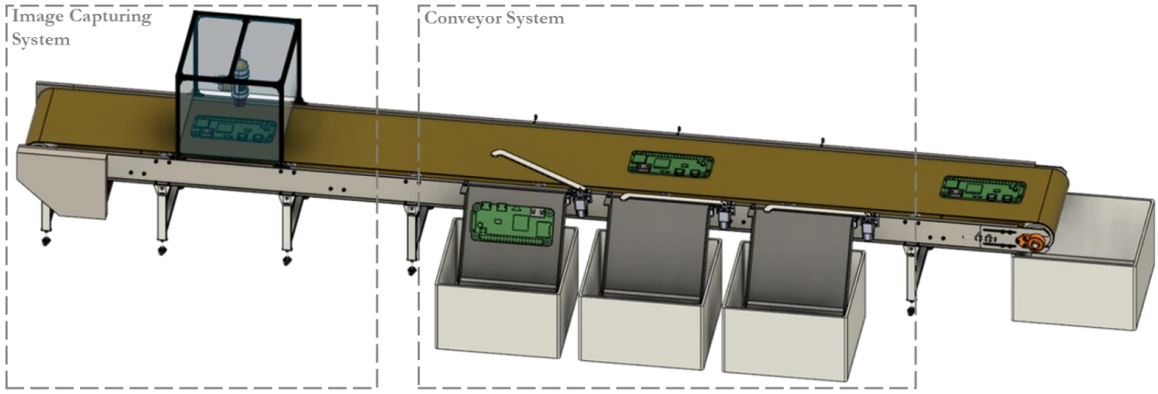


Figure 4.6 Proposed system 3D modelling

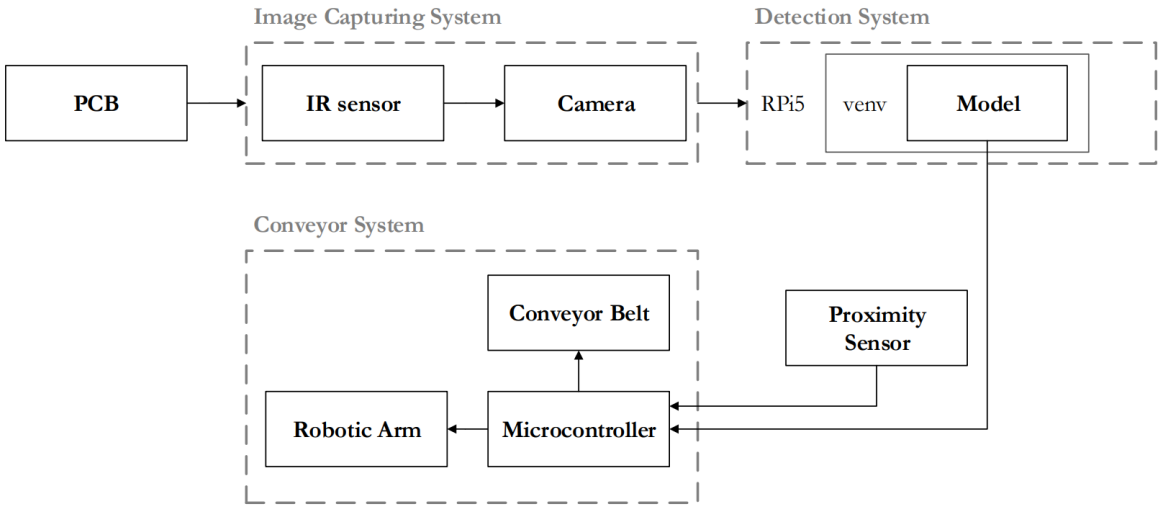


Figure 4.5 Proposed system block diagram

is the image capturing system, responsible for detecting the PCB when it approaches the station and captures its surface. The image is run directly through the defect detection software in the second stage and detection result (class) is sent to the conveyor system. Based on the index of the detected result, or lack thereof, the conveyor system decides the subsequent movement of the PCB. The conveyor system consists of a compartment for each defect type with an attached robotic arm at a horizontal closed position, as shown in Figure 4.5, and an IR proximity sensor placed at an adequate distance before each compartment. Each IR sensor is responsible for signaling the arm to open for its respective defective product and sort it into its intended compartment for further repair and/or suitable action.

4.3.2 Data acquisition

In this section, we review the different open-source PCB datasets obtained throughout this work and their respective acquisition procedures. The first dataset, DeepPCB, was provided by *S. Tang et al* [20] in 2019 during their research of a deep learning model with a novel pyramid pooling module to create an online PCB defect detector. Their dataset consists of 1,500 PCB image pairs with dimension 640 x 640 pixel containing six types of defects: mouse bite, missing hole, spur, spurious copper, shorts, and open circuits, as shown in Figure 4.7. Each pair consists of a defect-free template image and its defective tested counterpart. All images were obtained using a linear scan CCD in resolution around 48 pixels per 1 millimetre. The original images size was about 16k x 16k pixels which were later clipped into 640 x 640 sub-images and aligned through template matching techniques by reducing translation and rotation offsets between image pairs. After that the images underwent binarization by

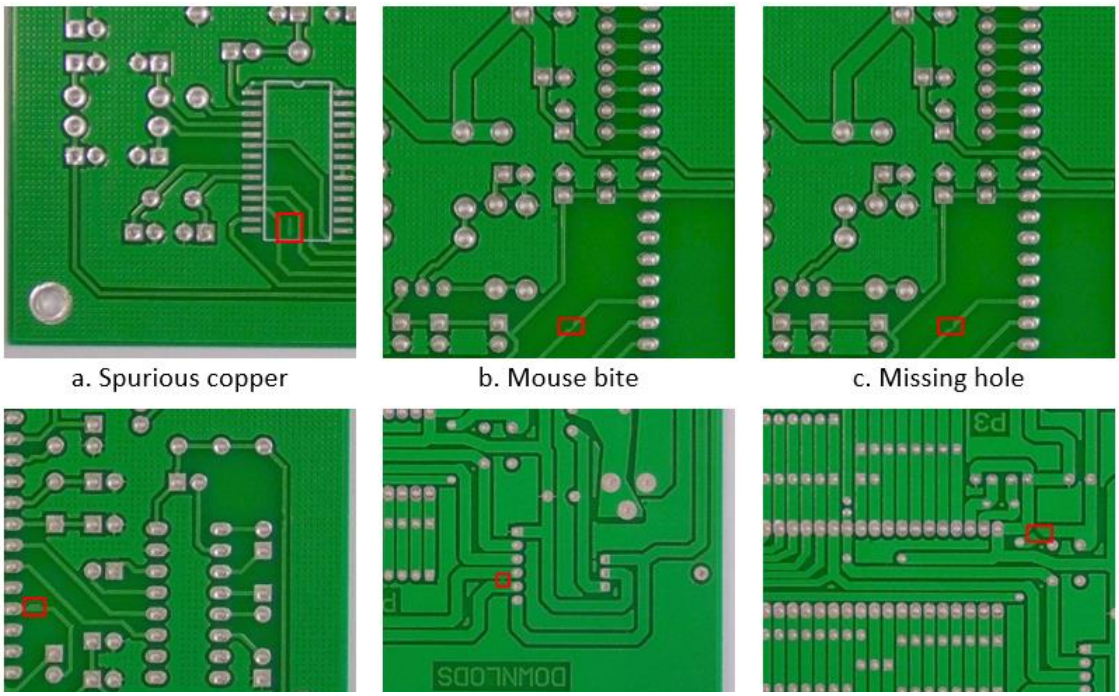


Figure 4.7 Types of PCB surface defects

selecting an appropriate threshold to minimize illumination noise for accurate localization and classification.

The second dataset is a synthesized PCB image collection that was published by *Huang et al* [21] in 2018 to address the lack of published data at the time which hindered the introduction and comparison of new models. Their dataset consists of 1386 images that were acquired by an imaging system of their own that mimics the practical AOI system used in inspection process. Their system consisted of a 16-Megapixel HD industrial camera equipped with a CMOS sensor mounted on an undistorted zoomable lens and a 2 frosted ring LED sources equipped with special diffuse matting to avoid possible consequences of uneven illumination. They were able to capture 693 naked PCB surface images of resolution 4608 x 3456 pixels, which was later adjusted according to the size of each board when creating the defects using Adobe Photoshop. Due to the limited number of defects per real board images, some defects were manually augmented on each tested image according to the defect patterns, which lead to around 3 to 12 defects per final image. The result of their work was a final set pf 1386 images. Their dataset, called HRIPCB, covers the same defect types covered by DeepPCB.

The third PCB defect dataset was provided by the authors of TDD-net [22] where they obtained the 693 PCB defective images obtained in HRIPCB dataset and performed more thorough manual data augmentation techniques after cropping the images into 600 x 600 sub-images randomly, such as gaussian noise, varying image brightness, image rotation, flipping and shifting. Their final dataset consists of 10668 images with complete PASCAL VOC annotation and a total of 21664 defects throughout the whole dataset. A breakdown of the dataset content is shown in Table 2.

Types of defects	Number of images	Number of defects
Missing hole	1832	3612
Mouse bite	1852	3684
Open circuit	1740	3548
Short circuit	1732	3508
Spur	1752	3636
Spurious copper	1760	3676
Total	10668	21664

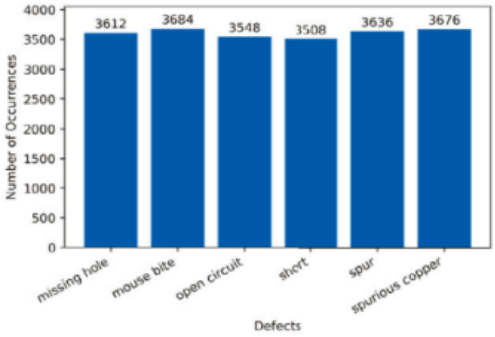
Table 2 Defect types count [23]

Dataset	Images	Size	Defects per image	Binarization
DeepPCB	1500	640	3 – 12	Yes
HRIPCB	1386	640	3 – 5	No
TDD-Net	10668	600	1 – 5	No

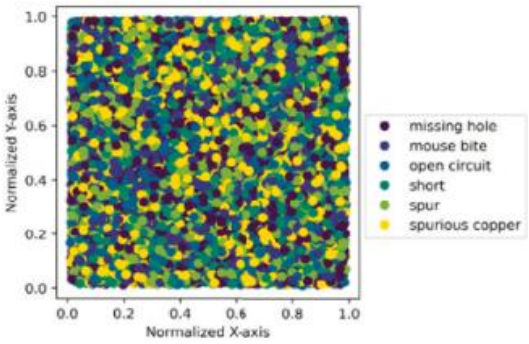
Table 3 Different datasets comparison [3]

Although image binarization increases the processing speed by reducing the size of the input data and improves the accuracy of the model by reducing the effects of noise and data variations, the binarized images produced in DeepPCB dataset were noticed to suffer from severe semantic degradation. In addition, these binarized images contain only one-color channel, both challenges of which can potentially hinder the

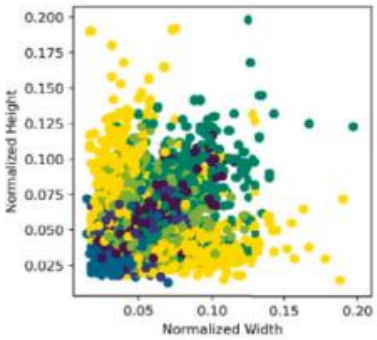
performance of the deep learning algorithms for image classification and detection unlike a standard three-channel RGB image which permits more feature handling. Another aspect to consider when selecting the dataset is size and balance. Since TDD-Net dataset consists of more image samples than HRIPCB, as shown in Table 3, a qualitative analysis of the dataset is needed to determine the inclusion and diversity of the dataset, since we are currently not further modifying the dataset. In this case, we refer to the work done by *J. Lim et al* [7] in performing qualitative analysis and



a. Defect occurrences plot



b. Normalized Location plot



c. Normalized Size plot

Figure 4.8 Visualization of dataset properties

visual representation of the TDD-Net dataset shown in Figure 4.8. The results show that the dataset doesn't suffer from severe class imbalance and contains an even random distribution of defects across each image. However, as indicated in Figure 4.8-c, there's a huge overlap in the aspect ratios of mouse bite and open circuit. Moreover, the majority of defect sizes occupy around 0.25% of the total image. Both problems pose challenges to accurate classification. As a result, the set used in this work is TDD-Net's dataset.

4.3.3 Data Preparation

Dataset Splitting

For the purpose of training the model and perform evaluation methods discussed later in this chapter, the dataset was split into several batches. This process was automated by writing python scripts to handle the splitting. First, the dataset was split into 80% for training purposes and 20% for evaluation, which was further split into 2 sets equally for validation and testing.

Second, the dataset was split into 5 equal batches. A python script, see Appendix A, was written to organize the dataset images and labels into pandas data frames, strip their paths and account for name variations. Often there should be no variation in file names between image file and its corresponding label. However, splitting the images produced a Key Error during execution and debugging the code by comparing file names showed that some image files were saved as [defect_name]_600 while their corresponding label files were saved as [defect_name]_256. This code was run on local machine instead of Colab environment since it is faster to manage files locally.

Image Segmentation Trials

In an attempt to further enhance the performance of the model, we attempted to apply segmentation techniques to the dataset images. Segmentation is a preprocessing technique that aims to partition an input image into distinct regions. The output of segmentation is a mask that assigns each pixel to a specific object class or instance. The purpose of segmentation in the context of this project is to generalize the model detection capabilities of detection to include all PCB surface types regardless of background colour. Segmentation was chosen as an alternative approach for binarization to maintain the colour information and quality of image without the noise occurring through binarization and thresholding methods. Two main approaches were conducted:

K-means clustering

It is a machine learning algorithm that divides the set of data points into k number of groups so that the data points within each group are more comparable to one another and different from the data points within other groups. The way this works is by randomly initializing k points, called means or cluster centroids. Each image pixel is categorized based on the minimum distance between itself and the initialized centroids. After all image points are classified, the centroid locations are updates and points are redistributed among clusters. This process is repeated for as long as the changes in new formed clusters are significant.

Based on the visual characteristics of our utilized dataset, the optimal number of clusters was determined to be 3, encompassing the background, traces and pads. However, this doesn't work as intended due to the very close values of the green channel in both background and traces, making it hard to completely isolate the traces and background as shown in Figure 4.9. This directly affects the detection capability of spur and short circuit defect types.

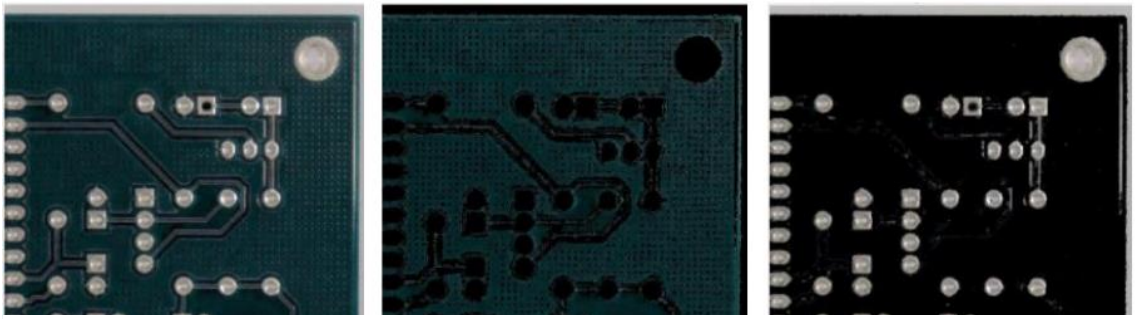


Figure 4.9 K-means clustering segmentation output

Adaptive Histogram-Based Thresholding

Thresholding is one of the most commonly known and simplest segmentation technique. Thresholding simply picks a value to act as the border for segmenting between classes. This works well with extrememly high contrast images and binary images, in color images with close colour ranges, however, determining a single threshold proves to be next to impossible. Thus adaptive thresholding is used to solve this problem by determining the threshold for segmentation based on the information of each image. This is attained by using the histogram of images. A histogram plots the pixel values of each channel with respect to their frequency in the image. In the images we use, the background takes up the biggest area with respect to the rest of components in the image. Using this information, we can adapt the MATLAB code to find the lowest frequencies on either side of the peaks and then replace all that range of values with zeros (black).

This approach faces some challenges due to the different distribution of pixel values among all three challenges, where abundance of green values doesn't necessarily mean an abundance in red and blue channels which makes determining the range quite more complex. Another observation made throughout experimenting with this code is that the images used in this dataset don't have the same background and track colors. Instead, some images contain light green backgrounds with dark traces while others have the backgrounds dark green with the traces in the light color, which makes picking the range harder. Also, the quality of images within the dataset suffers degradation and noise that is affecting the performance of thresholding. This resulted in segmentation only working on some images as shown in Figure 4.10.
Due to the segmentation difficulties emerging from the dataset image properties, it was decided to use the correctly segmented samples to test the model performance on non-green-background PCBs to test the model's generalizability.

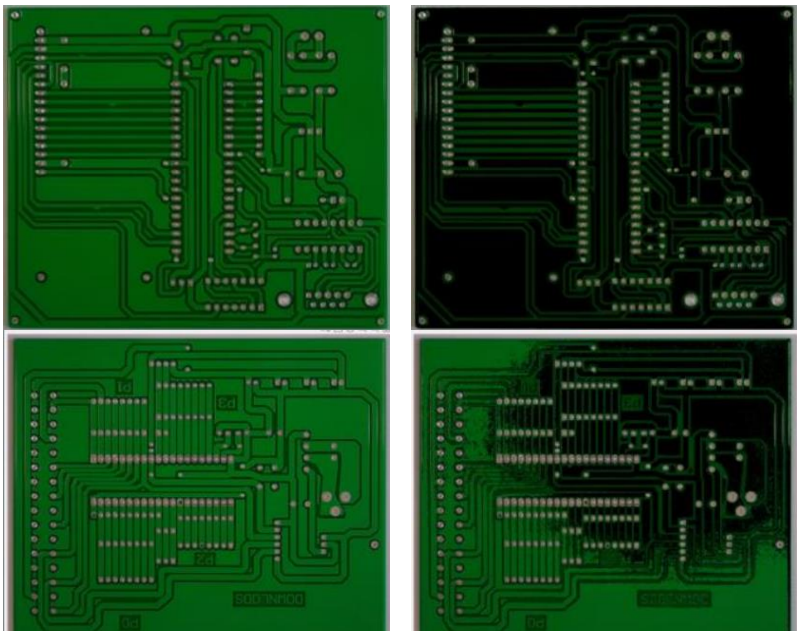


Figure 4.10 Adaptive thresholding segmentation output

4.3.4 Model Building

All YOLO versions are originally trained on the COCO dataset, a large-scale dataset that consists of more than 200K labelled images and has several redeemable features including 1.5 million instances and 80 object categories. For each version, different model variants exist to offer a trade-off between model size, computational complexity, and performance. Based on the model size at each checkpoint, the model weight is determined. The available weights for each YOLO version are nano, small, medium, large and extra-large. For all the pretrained checkpoints of YOLO versions, the number of training epochs were set to 300 epochs with default settings. However, different hyperparameters were set for different checkpoints where the nano and small models were trained with batch size: 64 and image size: 640, while the rest of the models were trained with batch size: 32 and image size: 1280 [24].

Training

Since the small and nano models of both versions 5 and 8 have the smallest depth and feature map width with respect to the other models of the same series, a performance evaluation of YOLOv5s, YOLOv5n, YOLOv8s, and YOLOv8n was performed on our dataset. All models were trained for 50 epochs with batch size: 16, and image size: 416. The dataset was divided into train, test, and validation sets in the ratio 80:10:10 respectively, where the training set is used by the model to extract the important features and use them for adjusting the model weights. The validation set is used to fine-tune the model during training, where they are used to evaluate the model performance and thus adjust the hyperparameters of the model accordingly. The test set is kept separate from the rest of the data and is used to evaluate the model's performance on unseen data to assess its generalization to real-world scenarios.

k-fold cross validation

Based on the resulting analysis, k-fold cross validation, a data resampling method, was performed on the two best-performing models to asses the generalizability of the predictive models to prevent overfitting. Overfitting typically occurs when the model is too familiar and well trained on the training the dataset that it performs poorly on unseen data. Cross validation [25] is used to estimate the true prediction error of models in order to tune the model parameters which in turn prevents overfitting. In k-fold cross validation, a python script is written to partition the dataset into subsets of approximately equal size. The number of resulting subsets is denoted by the k in k-fold. The first subset is held out and serves as a validation set while the model is trained on the remaining k - 1 subsets. Then the second subset is used as validation and the model is trained on the remaining k - 1 subsets. This process is repeated k times where each time a different subset is held out for validation when the model the trained on the remaining data until all subsets are used as validation exactly one time,

as shown in Figure 4.11. The accuracy of the model is measured at each fold and the results are used to select the optimal model, which is YOLOv8.

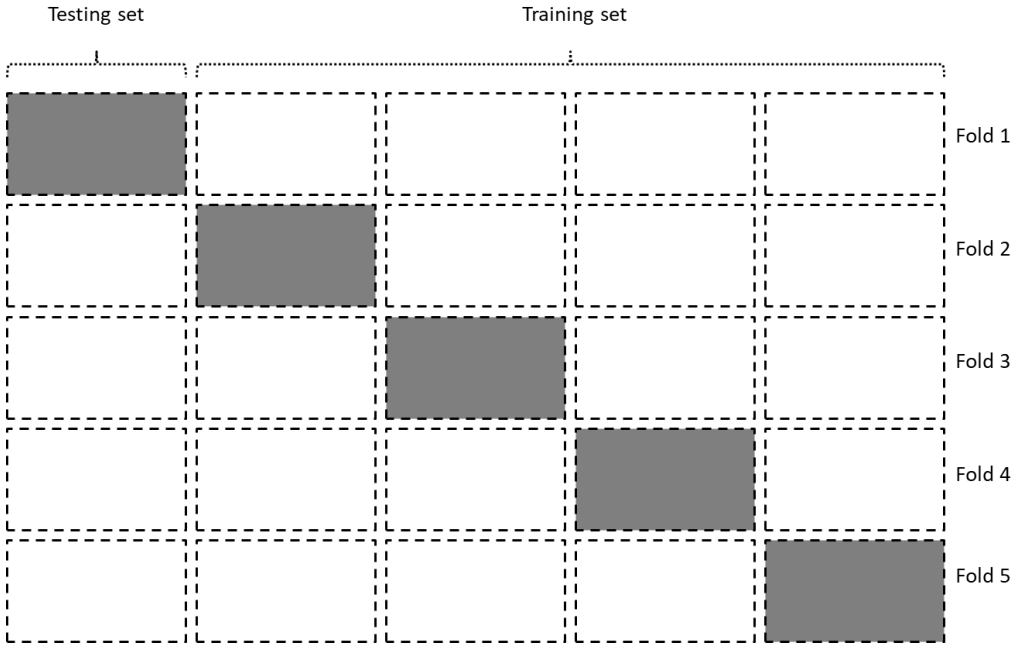


Figure 4.11 Cross validation working principle

Evaluation metrics

1. Accuracy: It represents the fraction of predictions that are actually true.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad (1)$$

Where, TP refers to the correctly identified defect, TN refers to the correctly identified lack of defect, FP refers to the misclassifying a region as defective of certain class, and FN refers to the failure to spot defects.

2. Precision: It is the metric responsible for measuring the accuracy of a model in correctly identifying objects. It can be calculated through the ratio:

$$Precision = \frac{TP}{TP+TN+FP+FN} \qquad (2)$$

3. Recall: It is a quantity describing the model's ability to detect all relevant objects. It is the measure of true positive predictions to the total number of actual positive instances in the dataset and can be represented by the ratio:

$$Recall = \frac{Total\ predictions}{Total\ labels} \qquad (3)$$

4. Mean Average Precision (mAP): This metric is responsible for quantifying the accuracy of object localization through computing the average precision across all classes and taking their mean value. It can be calculated from the area under the PR curve.

5. Inference speed or inference time: It is a metric used for measuring the time taken by the model to make predictions on a given dataset. This metric is crucial for such real-time and resource constrained applications PCB production lines.

4.3.5 Hardware Implementation Trials

Iteration 1:

Design:

Our initial design for the conveyor belt aimed for simplicity. It utilized a single wooden plank as the base, with rollers positioned at opposing ends. To achieve this,

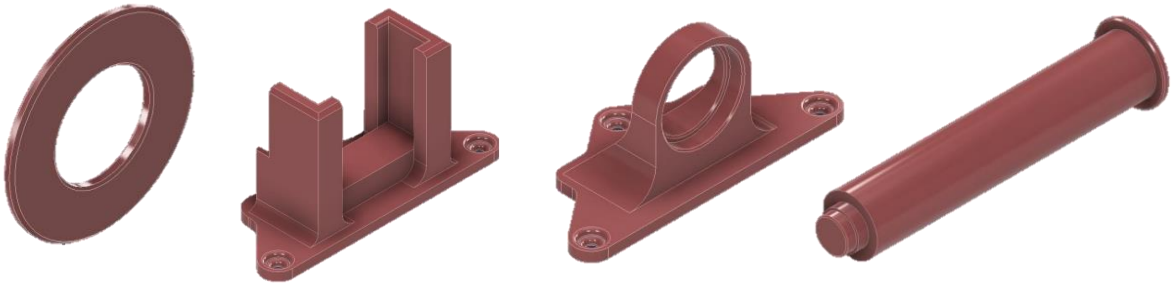


Figure 4.12 Iteration 1: 3D parts design

we designed custom rollers, shown in figure 4.12. Each roller had a hole on one side to house the motor's axis and a capped end on the other side that connected to the roller holder. The motor itself was secured to the wooden plank by screwing it onto a dedicated motor mount.

This design allowed for tension adjustment in the conveyor belt. By strategically positioning the motor mounts at opposite ends of the plank, we could control the amount tension in the belt.

Assembly:

However, this first iteration presented several limitations. Firstly, the conveyor belt, shown in Figures 4.13 and 4.14, width was significantly wider than the size of the PCBs it was intended to transport. Secondly, the design offered minimal to no elevation from the ground, making it more likely to come in contact with the floor and potentially causing damage. Thirdly, and perhaps most importantly, this initial design lacked any sorting compartments for separating the defective PCBs. Finally, the complete absence of support structures along the length of the conveyor belt meant it would be unable to handle any significant weight or heavier loads.

Figure 4.13 Assembled final product of iteration 1

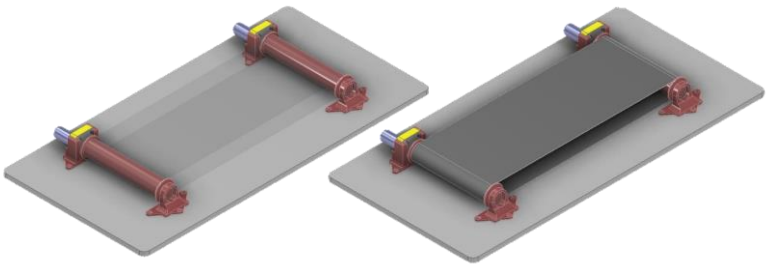


Figure 4.14 Assembled model of conveyor belt

Iteration 2:

Design:

Supports:

Our conveyor belt design process began by analysing existing industrial and conventional conveyor belts. We aimed to apply this foundation while incorporating modifications to optimize affordability and modify the design to our specific project needs. Conventional conveyor belts typically utilize wide belts, pulleys, and roller supports throughout their length. However, due to cost constraints and the specialized fabrication process required for the Artelon rollers which needed to be custom-made in a specific workshop, we opted for an alternative approach that eliminated the use of rollers. Instead, we designed a wooden plank, Figure 4.15, that supports the weight of the belt with only two rollers at the beginning and end of the belt.

In addition to the main design, our conveyor belt needed placements for various components: servo motor mounts, IR sensor mounts, bearing mounts, compartments for sorting stations, and stations for the Raspberry Pi camera and stepper motor mount.

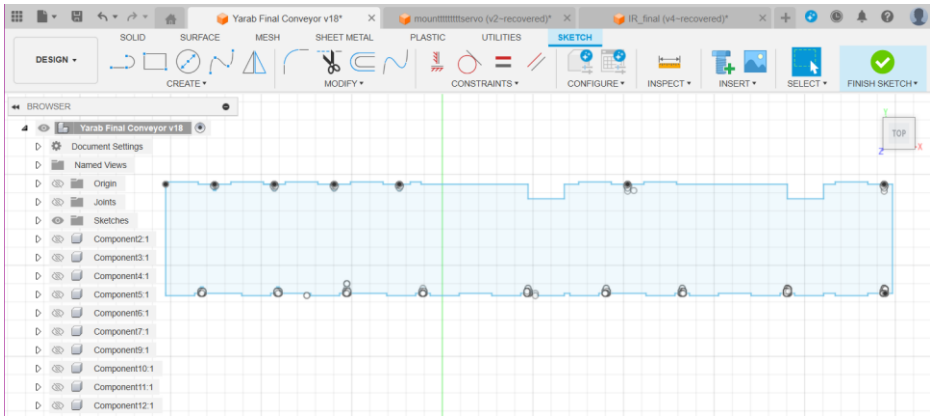


Figure 4.15 Design sketch of wooden support plank

Servo Motor Mount:

Attaching the servo motor to the conveyor belt involved a two-part process. The first part was a wooden mount, slightly larger than the servo motor (by a couple of millimetres), that crammed the motor. It contained two holes for later screwing the mount to the conveyor belt's side.

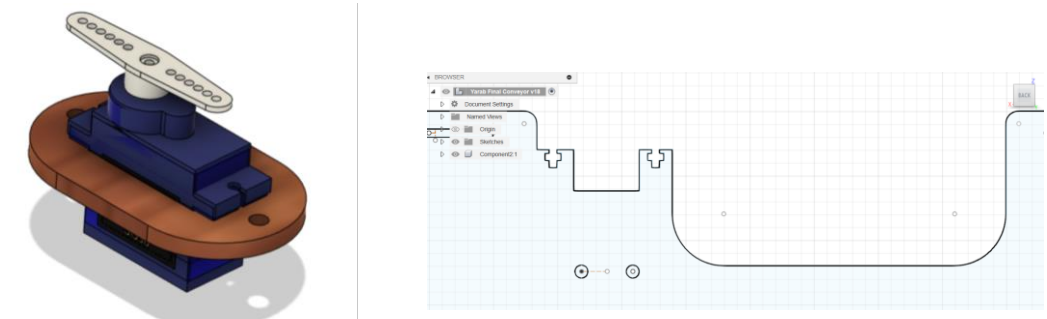


Figure 4.16 Motor Mound 3D design

The second part involved adjusting the servo motor mount's position. We needed to lower the mount below the conveyor belt's level to prevent the servo motor arm from obstructing the PCB and hindering its sorting function. Additionally, a gap was created in front of each servo mount to ensure the servo arms could freely sort defective PCBs. The designs are depicted in Figure 4.16.

IR Sensor Mount:

The primary challenge with placing the IR sensor arose from the very thin thickness of the PCB. Placing the sensor at the same level as the conveyor belt was impractical, as the PCB would slide under the IR's coverage range without being detected. To address this, we designed a 30-degree angled mount, shown in Figure 4.17. positioned at a higher elevation. This mount attached to the station using a tab and slot joint method.

Figure 4.18 IR sensor mount 3D design

Figure 4.17 Bearing mount 3D design

Bearing Mount:

In order to attach the rollers to each side of the conveyor belt which avoiding friction that could slow down the conveyor belt's movement. We secured bearings to the rollers and then fastened the bearing mounts, shown in Figure 4.18, to the sides of the conveyor belt. To ensure consistent sizing for the bearing mounts, we downloaded a bearing step file from GrabCAD and projected the bearing's diameter onto the sketch.

Camera Station:

Similar to the bearing mount, we downloaded the Raspberry Pi camera's step file from GrabCAD and used the projection tool to incorporate its design into the station's sketch, as shown in Figure 4.19. This included ensuring the camera's holes and cable slot were accounted for in our design.

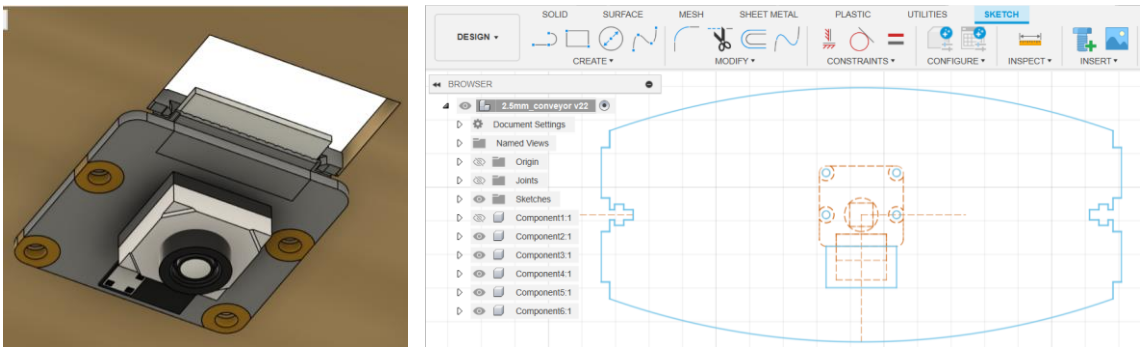


Figure 4.19 Camera mount 3D design

Stepper Motor Mount:

Designing a stable mount to withstand the 0.24 kg weight of the stepper motor presented a significant challenge. Initially, we focused on ensuring the motor's axis aligned with the roller's axis. To achieve sufficient stability, we then incorporated two triangular pillars, refer to Figure 4.20, on each side of the front motor panel, connecting them to the base for added support.

Figure 4.20 Motor mount 3D design

Assembly:

While this second design represented a significant improvement over the first iteration, it still presented some minor design flaws that caused major delays later on. The first issue stemmed from how the roller was initially attached. Only the axis of the roller protruded from the conveyor belt's side, connecting directly to the motor's axis. This design created significant friction between the exposed axis and the wooden side of the conveyor belt, hindering smooth operation and potentially causing delays. Secondly, despite mounting the IR sensor on an angled mount, it remained unreliable in detecting PCBs. The sensor's detection zone was limited to the area directly below it. Since the PCBs typically sat in the middle of the conveyor belt, most of them weren't being properly detected.

Finally, this design lacked any provisions for cable management or component organization. Both sides of the conveyor belt were solid wood without any openings for cables. This made it difficult to manage and organize the wiring for the various components, leading to potential tangles and hindering future maintenance.

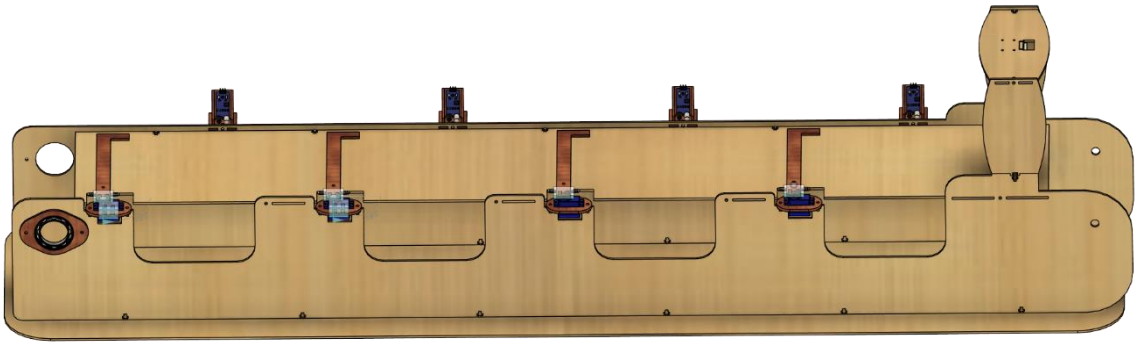


Figure 4.21 Assembled 3D design of conveyor belt: iteration 2

Iteration 3:

Design:

The third iteration of the conveyor belt design took the successful elements of the second iteration and addressed its limitations with key modifications that significantly improved overall functionality.

The most critical change involved the roller attachment. Instead of a small hole for the roller axis, a larger circle with a diameter matching the bearing size was cut into each side of the conveyor belt. This modification allowed for the insertion of bearings on both sides, eliminating the friction issue experienced in the second iteration and ensuring smoother operation.

We designed a dedicated elevated station for the IR sensor mount, offering two placement options for optimal performance testing. The first option involved slots on the side of the station for securing the mount horizontally. The second option placed the IR sensor vertically at the top of the station, with its transmitter and receiver pointing downwards towards the conveyor belt. This dual placement provided flexibility during testing to determine the most effective location for accurate PCB detection.

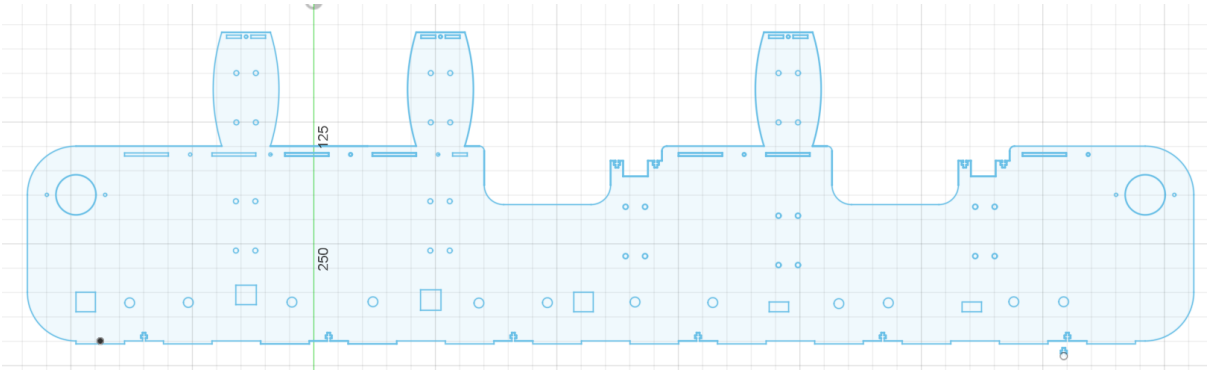


Figure 4.22 Modified side design: iteration 3

To address the cable management challenges, we implemented two major changes. Firstly, the overall height of the conveyor belt was increased by 5 cm, from 15 cm to 20 cm. This additional space allowed for all the electronic components to be positioned underneath the conveyor belt without hindering belt movement. Secondly, we strategically placed holes and rectangles roughly 3 cm from the base of the conveyor belt. These openings facilitated the routing of cables under the belt and connection to the various components. The size and placement of these holes and rectangles were carefully chosen to accommodate the specific types of electronic components used in the system.

Assembly:

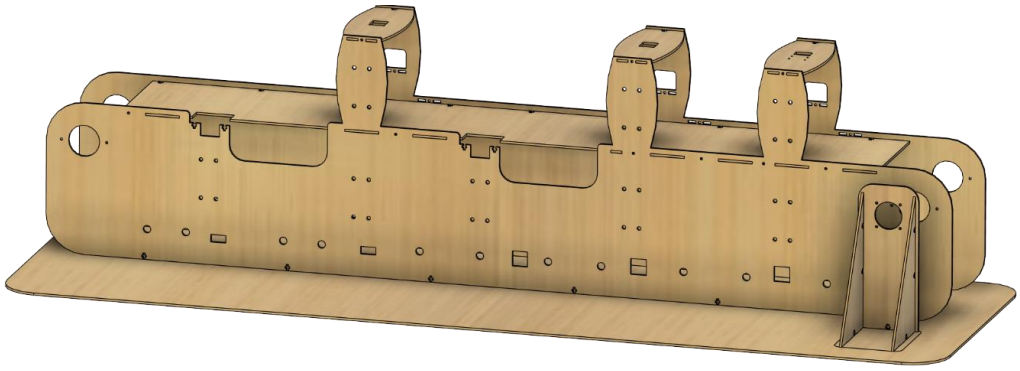


Figure 4.23 Assembled 3D design of conveyor belt: iteration 3

After the design was reworked and reviewed, it was converted to DXF files and laser cut for assembly. It successfully solved the previous challenges faced by the previous attempts. In addition, a minor modification was implemented where the rollers were changed from wood to ABS Artelon. Artelon is the commercial name for plastic-based sheets or poles used mainly in mechanical industries and piping. It can be made from several plastic types, the one we used in this project is made from Acrylonitrile Butadiene Styrene (ABS), a common type of thermoplastic polymer known for its versatility and good balance of properties. ABS Artelon offer desirable characteristics including:

1. High impact resistance, highlighting the material's ability to provide a high level of grip or resistance to sliding. This is crucial for keeping the roller in place and provide stability for the conveyor belt when in motion.
2. Abrasion resistance, emphasizing the material's ability to resist wear and tear from rubbing or scraping. This is desirable for keeping its quality through long hours of production line motions.
3. Extremely hard. The material's hardness and resistance to deformation makes it ideal for use in heavy machinery.

Figure 4.24 ABS Artelon

Chapter Five

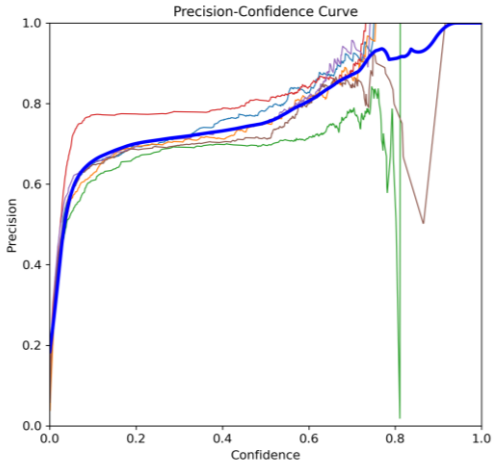
5 RESULTS AND DISCUSSION

5.1 RESULTS

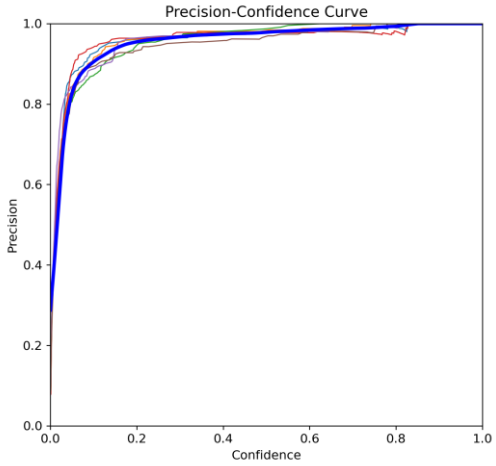
5.1.1 YOLO versions comparison

Model	Precision %	Recall %	mAP 50 %	Accuracy %
YOLOv5s	96.7	97.7	98.4	84.165
YOLOv5n	71.9	93.5	81.8	81.831
YOLOv8s	97.7	97.9	98.4	84.022
YOLOv8n	97.4	97.3	98.3	83.644

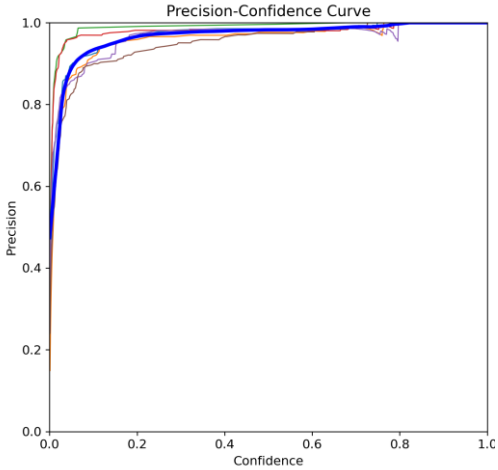
Table 4 Performance comparison of all custom-trained models



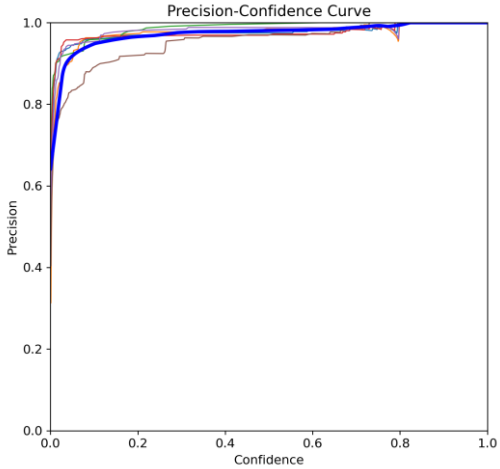
a. custom-trained YOLOv5n



b. custom-trained YOLOv5s



c. custom trained YOLOv8n



d. custom-trained YOLOv8s

Figure 5.1 Precision confidence Curves of all custom-trained models

5.1.2 Cross validation

Model	Accuracy %				
	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
YOLOv8s	83.5616	84.1655	82.008	84.3839	84.2407
YOLOv5s	82.091	81.974	81.857	81.054	80.818

Table 5 Cross validation accuracy output

5.1.3 Model performance on validation set

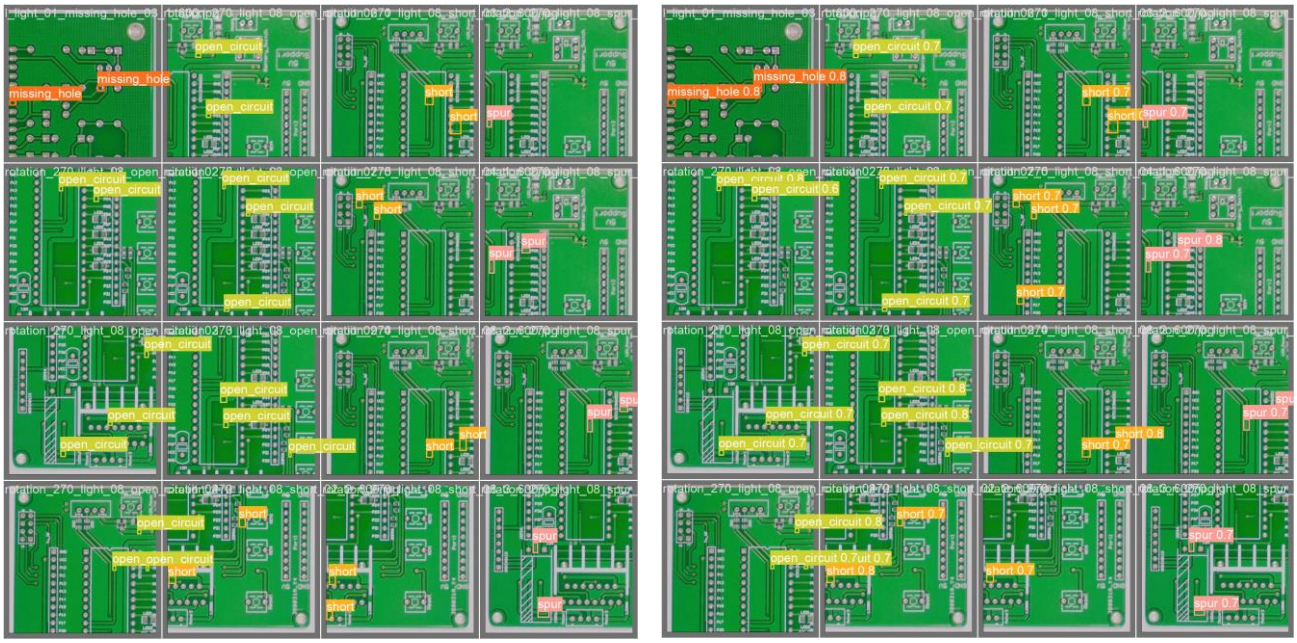


Figure 5.2 Validation batch results for YOLOv8s custom-trained model

5.1.4 Inference Speed Benchmarks

Format	Size (MB)	mAP50-95	Inference time (ms)
PyTorch	21.4	0.5816	270.87
TorchScript	42.8	0.5755	228.71
ONNX	42.6	0.55755	194.60
OpenVINO	42.7	0.5755	157.00
TensorRT	NaN	NaN	NaN
CoreML	21.4	NaN	NaN
TensorFlow SavedModel	NaN	NaN	NaN
TensorFLow GraphDef	NaN	NaN	NaN
TensorFlow Lite	NaN	NaN	NaN
TensorFlow Edge TPU	NaN	NaN	NaN
TensorFlow.js	NaN	NaN	NaN
PaddlePaddle	NaN	NaN	NaN
ncnn	42.5	0.5755	174.99

Table 6 Inference speed benchmarks on Colab CPU

Format	Size (MB)	mAP50-95	Inference time (ms)
PyTorch	21.5	0.6254	945.54
TorchScript	42.8	0.6186	822.08
ONNX	42.6	0.6186	561.06
OpenVINO	42.7	0.6186	442.90
TensorRT	0.0	NaN	NaN
CoreML	0.0	NaN	NaN
TensorFlow SavedModel	106.5	0.6186	523.26
TensorFLow GraphDef	42.6	0.6186	506.48
TensorFlow Lite	42.6	0.6186	705.20
TensorFlow Edge TPU	0.0	NaN	NaN
TensorFlow.js	0.0	NaN	NaN
PaddlePaddle	0.0	NaN	NaN
Ncnn	42.5	0.6186	264.77

Table 7 Inference speed benchmarks on RPi5 CPU

More detailed results are available in Appendix B.

5.2 DISCUSSION

Taking into account the dataset analysis, refer to 4.8, showing an even distribution of defects across images with respect to count and size, the results in Table 4 show high values in most metrics which is desirable. The high precision values indicate the models' capability of identifying the defect for what it actually is. In other words, it focuses on all the predicted instances of a certain class and calculates how accurate their prediction is compared to their ground truth. This is useful in our case since the cumulative precision value means that the model doesn't incorrectly identify the lack of defect in each class as defective. As a result, it saves a lot in production time as well as avoids unnecessary inspections and potential losses. Based on this metric, YOLOv5n can be safely eliminated due to the inefficiency of its misidentifications. The results of the other metrics are closely aligned which makes it hard to make a decision based on only one metric alone. Thus, other considerations were factored in such as the inference speed, F1-score, and model size. This resulted in eliminating the nano weight due to inaccurate predictions in both versions.

Cross validation was then conducted to evaluate the performance of the remaining two models, YOLOv5s and YOLOv8s. The results shown in Table 5 of k-fold cross validation shows a close range of accuracy across all models in the different folds. This confirms an even distribution of data and unbiased classification of classes across the test, train, and validation sets in different combinations. It also indicates a similar performance across all models. We chose to deploy the one with highest accuracy.

It was observed that the accuracies range are significantly below that mAP ranges. This high mAP with low accuracy performance shown in all models across the same dataset can be attributed to the way accuracy is calculated in each class. Although the number of images for each defect is roughly the same across the dataset, the accuracy calculated for each defect is skewed due to the much more numerous TN (all other classes with respect to each class alone).

Chapter Six

6 CONCLUSION AND RECOMMENDATIONS

6.1 CONCLUSION

Within the electronics manufacturing industry, even minor imperfections on PCBs can significantly impact product performance, potentially leading to malfunction or complete failure. While Automated Optical Inspection (AOI) is a widely used quality control method, its complex algorithms and operation suffer from a high misjudgement rate, rendering it unsuitable for fully automated environments. This necessitates manual reinspection by human resources, which introduces the risk of human error, production delays, and increased costs.

Our project proposes a solution – a fully automated PCB defect detection system that eliminates reliance on AOI. We introduce a dedicated station along the production line equipped with a camera connected to a powerful processing unit (Raspberry Pi 5). This unit runs a pre-trained model. Through extensive testing and evaluation, we identified the optimal model configuration: YOLOv8s trained on the TDD-Net dataset, which includes a broad spectrum of PCB defects. Our model achieves a remarkable mAP50 of 98.4% and an accuracy of 84.022%. This indicates an exceptional ability to recognize six types of defects (mouse bites, spurs, missing holes, short circuits, open circuits, and spurious copper) on PCBs, even with imperfect bounding boxes. Upon detecting a defect, the system automatically removes the faulty PCB from the production line using robotic arms. These arms sort the PCBs by defect type for potential repair (if feasible) or recycling. This approach effectively reduces overall production costs by minimizing the number of undetected defective PCBs entering the line and eliminates the need for human intervention during inspection.

6.2 RECOMMENDATIONS

We envision our PCB defect detection system having widespread application across various electronics factories. However, for a smooth industrial transition, a few modifications are necessary. In our prototype, the model ran on a Raspberry Pi 5, introducing a latency of around 1500ms. For high-volume industrial settings, this hardware might not be sufficient. A more suitable option would be hardware with a dedicated GPU, such as the Jetson Nano equipped with an Nvidia Tegra GPU, to handle the processing demands. Additionally, the Raspberry Pi Camera V3, with its 12-megapixel resolution and focus acquisition time of ~2000ms (due to its area scan nature), wouldn't be practical for real-time defect detection without halting production. Instead, a line scan camera excels at capturing objects in continuous motion without delays, making it a better fit for this application. Our simulation used an IR sensor to detect PCBs reaching the sorting robot arm. However, due to the

PCB's thin profile, a laser sensor that detects the reflected beam would be a more reliable alternative.

Despite our model's impressive performance, there's room for improvement. Currently, it can only detect defects on green PCBs due to the limitations of the TDD-Net dataset we used. This could be addressed by training the model on datasets encompassing a wider variety of PCB colours and shapes. Alternatively, segmentation techniques could be employed to isolate PCB traces from the background, although our initial attempts at this method haven't yielded perfect background elimination. Further exploration of hyperparameter tuning and potentially modifying the model architecture could also enhance performance.

Ultimately, we hope our system serves as a valuable contribution to PCB defect detection research and offers significant benefits to the electronics industry.

Chapter Three

7 CITATION AND REFERENCING

[1] UNDP, "Sustainable development goals," THE SDGS IN ACTION, https://www.undp.org/sustainable-development-goals (accessed Nov. 7, 2023).

[2] The Egyptian State Information Service, "Egypt: Enhancing the Technology Industry and Electronics Design.," Local Studies and Research, https://t.ly/fqm1X (accessed Nov. 22, 2023).

[3] D. S. Koblah, O. P. Dizon-Paradis, J. Schubeck, U. J. Botero, D. L. Woodard, and D. Forte, "A Comprehensive Taxonomy of Visual Printed Circuit Board Defects," Journal of Hardware and Systems Security. Springer Science and Business Media LLC, Apr. 24, 2023. doi: 10.1007/s41635-023-00132-4.

[4] "PCB Design Automation 101 - static.sw.cdn.siemens.com," A complete guide to PCB design automation, https://static.sw.cdn.siemens.com/siemens-disw-assets/public/72lwInmhwvJFtWlkii7ffD/en-US/Siemens-SW-PCB-design-automation-101-EB-85518-D2.pdf (accessed Sep. 26, 2023).

[5] P. Chen and F. Xie, "A machine learning approach for automated detection of critical PCB flaws in optical sensing systems," Photonics, vol. 10, no. 9, p. 984, Aug. 2023. doi:10.3390/photonics10090984.

[6] J. Chen, E. Bao, and J. Pan, "Classification and positioning of circuit board components based on improved Yolov5," Procedia Computer Science, vol. 208, pp. 613–626, 2022. doi:10.1016/j.procs.2022.10.085.

[7] J. Lim, J. Lim, V. M. Baskaran, and X. Wang, "A deep context learning based PCB defect detection model with anomalous trend alarming system," Results in Engineering, vol. 17. Elsevier BV, p. 100968, Mar. 2023. doi: 10.1016/j.rineng.2023.100968.

[8] J. Tang, S. Liu, D. Zhao, L. Tang, W. Zou, and B. Zheng, "PCB-YOLO: An Improved Detection Algorithm of PCB Surface Defects Based on YOLOv5," Sustainability, vol. 15, no. 7. MDPI AG, p. 5963, Mar. 29, 2023. doi: 10.3390/su15075963.

[9] Y. Yang and H. Kang, "An Enhanced Detection Method of PCB Defect Based on Improved YOLOv7," Electronics, vol. 12, no. 9. MDPI AG, p. 2120, May 06, 2023. doi: 10.3390/electronics12092120.

[10] F. Chen, M. Deng, H. Gao, X. Yang, and D. Zhang, "NHD-Yolo: Improved yolov8 using optimized neck and head for product surface defect detection with data augmentation," IET Image Processing, vol. 18, no. 7, pp. 1915–1926, Mar. 2024. doi:10.1049/ipr2.13073.

[11] J. Lu and S. H. Lee, "Real-time defect detection model in industrial environment based on Lightweight Deep Learning Network," Electronics, vol. 12, no. 21, p. 4388, Oct. 2023. doi:10.3390/electronics12214388.

[12] B. Hu and J. Wang, "Detection of PCB Surface Defects With Improved Faster-RCNN and Feature Pyramid Network," IEEE Access, vol. 8. Institute of Electrical and Electronics Engineers (IEEE), pp. 108335–108345, 2020. doi: 10.1109/access.2020.3001349.

[13] “PCB defect detection with computer vision - raspberry pi,” Expert Projects, https://docs.edgeimpulse.com/experts/image-projects/pcb-defect-detection-with-computer-vision-raspberry-pi

[14] F. Chollet, Deep Learning with Python. Shelter Island: Manning Publications, 2021.

[15] Google Research, “Colaboratory: Frequently Asked Questions,” Google colab, https://research.google.com/colaboratory/faq.html).

[16] G. Rodola, “psutil: Cross-platform LIB for process and system monitoring in Python,” GitHub, https://github.com/giampaolo/psutil

[17] T. Carneiro, R. V. Medeiros Da Nobrega, T. Nepomuceno, G.-B. Bian, V. H. C. De Albuquerque, and P. P. R. Filho, “Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications,” IEEE Access, vol. 6. Institute of Electrical and Electronics Engineers (IEEE), pp. 61677–61685, 2018. doi: 10.1109/access.2018.2874767.

[18] Raspberry Pi Foundation, “Raspberry pi documentation - raspberry pi hardware,” Raspberry Pi, https://www.raspberrypi.com/documentation/computers/raspberry-pi.html (accessed Jun. 27, 2024). G. Jocher et al., ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation. Zenodo, 2022. doi: 10.5281/ZENODO.7347926.

[19] S. Tang, F. He, X. Huang, and J. Yang, “Online PCB Defect Detector On A New PCB Defect Dataset.” arXiv, 2019. doi: 10.48550/ARXIV.1902.06197.

[20] W. Huang, P. Wei, M. Zhang, and H. Liu, “HRIPCB: a challenging dataset for PCB defects detection and classification,” The Journal of Engineering, vol. 2020, no. 13. Institution of Engineering and Technology (IET), pp. 303–309, May 22, 2020. doi: 10.1049/joe.2019.1183.Paper 1

[21] R. Ding, L. Dai, G. Li, and H. Liu, “TDD-net: a tiny defect detection network for printed circuit boards,” CAAI Transactions on Intelligence Technology, vol. 4, no. 2. Institution of Engineering and Technology (IET), pp. 110–116, May 31, 2019. doi: 10.1049/trit.2019.0019.

[22] R. Ding, L. Dai, G. Li, and H. Liu, “TDD-net: a tiny defect detection network for printed circuit boards,” CAAI Transactions on Intelligence Technology, vol. 4, no. 2. Institution of Engineering and Technology (IET), pp. 110–116, May 31, 2019. doi: 10.1049/trit.2019.0019.

[23] G. Jocher et al., ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation. Zenodo, 2022. doi: 10.5281/ZENODO.7347926.

[24] D. Berrar, “Cross-Validation,” Encyclopedia of Bioinformatics and Computational Biology. Elsevier, pp. 542–545, 2019. doi: 10.1016/b978-0-12-809633-8.20349-x.

[25] D. Schier, “Release - raspberry pi 5,” blog.while, https://blog.while-true-do.io/release-raspberry-pi-5/ (accessed Jun. 28, 2024).

[26] “Laser cutting machine for wood acrylic,” Triumphlaser, https://www.triumphlaser.com/laser-cutting-machine/# (accessed Jun. 29, 2024).

8 APPENDICES

APPENDIX A: SCRIPTS

- Dataset splitting into k folds

```
import datetime
import shutil
import pathlib
import os
from pathlib import Path
from collections import Counter
import yaml
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
dataset_path = Path("E:/final-grad-project/v8_attempt/kfold-
cv/pcb-defect-dataset")
sup_ext = [".txt"]
labels = []
for ext in sup_ext:
    labels.extend(sorted((dataset_path / "train" /
"labels").rglob(f"*{ext}")))
    labels.extend(sorted((dataset_path / "val" /
"labels").rglob(f"*{ext}")))
with open(
    "E:/final-grad-project/v8_attempt/kfold-cv/pcb-defect-
dataset/data.yaml",
    "r",
    encoding="utf8",
) as y:
    classes = yaml.safe_load(y)["names"]
cls_idx = list(range(len(classes)))

indx = [l.stem for l in labels]  # uses base filename as ID (no
extension)
labels_df = pd.DataFrame([], columns=cls_idx, index=indx)
for label in labels:
    lbl_counter = Counter()
    with open(label, "r") as lf:
        lines = lf.readlines()
    for l in lines:
        lbl_counter[int(l.split(" ")[0])] += 1
    labels_df.loc[label.stem] = lbl_counter
labels_df = labels_df.fillna(0.0)  # replace `nan` values with
`0.0`
ksplit = 5
kf = KFold(
    n_splits=ksplit, shuffle=True, random_state=20
)  # setting random_state for repeatable results
kfolds = list(kf.split(labels_df))
folds = [f"split_{n}" for n in range(1, ksplit + 1)]
folds_df = pd.DataFrame(index=indx, columns=folds)
for idx, (train, val) in enumerate(kfolds, start=1):
    folds_df[f"split_{idx}"].loc[labels_df.iloc[train].index] =
"train"
    folds_df[f"split_{idx}"].loc[labels_df.iloc[val].index] =
"val"
```

```
fold_lbl_distrb = pd.DataFrame(index=folds, columns=cls_idx)
for n, (train_indices, val_indices) in enumerate(kfolds,
start=1):
    train_totals = labels_df.iloc[train_indices].sum()
    val_totals = labels_df.iloc[val_indices].sum()
    ratio = val_totals / (train_totals + 1e-7)
    fold_lbl_distrb.loc[f"split_{n}"] = ratio
supported_extensions = [".jpg", ".jpeg", ".png"]
images = []
for ext in supported_extensions:
    images.extend(sorted((dataset_path / "train" /
"images").rglob(f"*{ext}")))
    images.extend(sorted((dataset_path / "val" /
"images").rglob(f"*{ext}")))

# Create the necessary directories and dataset YAML files
save_path = Path(
    dataset_path /
f"{datetime.date.today().isoformat()}_{ksplit}-Fold_Cross-val"
)
save_path.mkdir(parents=True, exist_ok=True)
ds_yamls = []
for split in folds_df.columns:
    # Create directories
    split_dir = save_path / split
    split_dir.mkdir(parents=True, exist_ok=True)
    (split_dir / "train" / "images").mkdir(parents=True,
exist_ok=True)
    (split_dir / "train" / "labels").mkdir(parents=True,
exist_ok=True)
    (split_dir / "val" / "images").mkdir(parents=True,
exist_ok=True)
    (split_dir / "val" / "labels").mkdir(parents=True,
exist_ok=True)
    dataset_yaml = split_dir / f"{split}_dataset.yaml"
    ds_yamls.append(dataset_yaml)
    with open(dataset_yaml, "w") as ds_y:
        yaml.safe_dump(
            {
                "path": split_dir.as_posix(),
                "train": "train",
                "val": "val",
                "names": classes,
            },
            ds_y,
        )
for image, label in zip(images, labels):
    modified_stem = image.stem[:-3]
    matching_entries =
folds_df[folds_df.index.str.startswith(modified_stem)].index[0]
    for split, k_split in
folds_df.loc[matching_entries].items():
        # Destination directory
        img_to_path = save_path / split / k_split / "images"
        lbl_to_path = save_path / split / k_split / "labels"

        shutil.copy(image, img_to_path / image.name)
        shutil.copy(label, lbl_to_path / label.name)
```

- Training and cross validation for YOLOv5

```
!pip install torch torchvision torchaudio
!git clone https://github.com/ultralytics/yolov5.git
from google.colab import drive
drive.mount('/content/drive/')
import zipfile
with zipfile.ZipFile('/content/drive/MyDrive/grad
project/v8/TDD-Net/tddcvdataset.zip', 'r') as zip_ref:
    zip_ref.extractall('/content/')
cd yolov5
!pip install -r requirements.txt
!python train.py --img 416 --batch 16 --epochs 50 --data
split_3_dataset.yaml --weights yolov5n.pt --cache --name
v5_pcb_tdd
!python val.py --weights
"/content/yolov5/runs/train/v5_pcb_tdd/weights/best.pt" --data
split_3_dataset.yaml --img 416
!python benchmarks.py --weights
"/content/yolov5/runs/train/v5_pcb_tdd/weights/best.pt" --data
split_3_dataset.yaml --imgsz 416 --device cpu
```

- Training and cross validation for YOLOv8

```
import datetime
import shutil
import os
import glob
from pathlib import Path
from collections import Counter
from ultralytics import YOLO
import yaml
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from google.colab import drive
drive.mount('/content/drive/')
!unzip "/content/drive/MyDrive/gradproject/dataset/TDD-
Net/archive.zip" -d /content/
dataset_path = Path('./pcb-defect-dataset')
model = YOLO("yolov8s.pt", task='detect')
results = {}
for k in range(ksplit):
  dataset_yaml = ds_yamls[k]
  model.train(data=dataset_yaml, epochs=50, batch=16,
imgsz=416, project='v8-kfold-cv')
  results[k] = model.metrics
def zip_directory(directory_path, output_zip_path):
    with zipfile.ZipFile(output_zip_path, 'w',
zipfile.ZIP_DEFLATED) as zipf:
        for root, dirs, files in os.walk(directory_path):
            for file in files:
                file_path = os.path.join(root, file)
                arcname = os.path.relpath(file_path,
directory_path)
                zipf.write(file_path, arcname=arcname)
directory_to_zip = "/content/v8-kfold-cv"
output_zip_file = "/content/v8-kfold-cv.zip"
zip_directory(directory_to_zip, output_zip_file)
shutil.copy("/content/v8-3fold-cv.zip",
"/content/drive/MyDrive/gradproject/v8/")
```

- Deploying model on Raspberry Pi

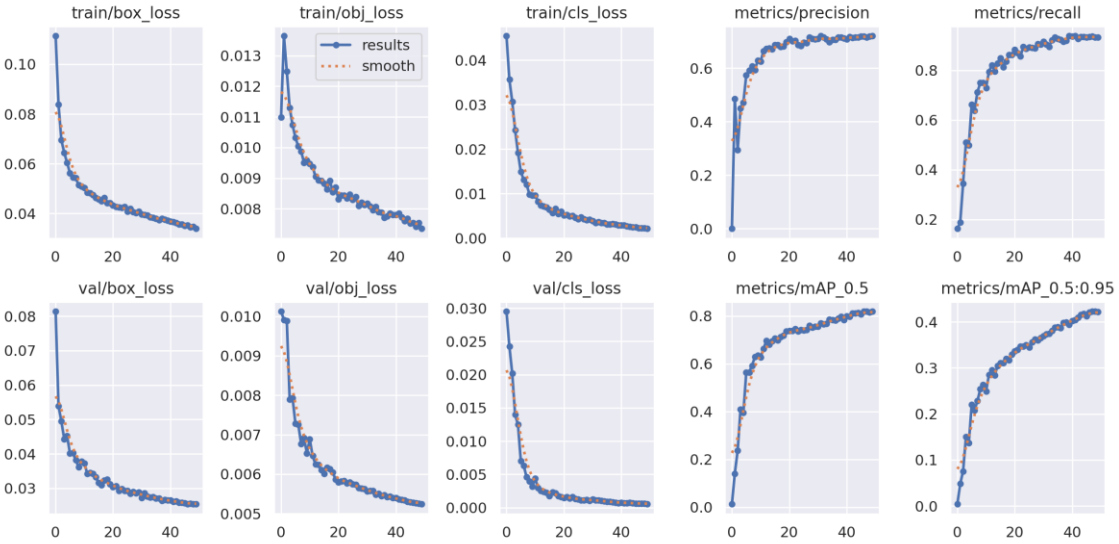
```
from ultralytics import YOLO
import argparse
import math, serial, time
temp_arr = []
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("--model", required=True,
default='./models/best.pt')
    args = parser.parse_args()
    classNames = ['mouse_bite', 'spur', 'missing_hole',
'short', 'open_circuit', 'spurious_copper']
    ser = serial.Serial('/dev/ttyACM0', 115200, timeout=1)
    ser.reset_input_buffer()
    model = YOLO(args.model)
    results = model.predict('tcp://192.168.137.36:8080',
stream=True, imgsz=416)
    while True:
        for result in results:
            boxes = result.boxes
            for box in boxes:
                probs = math.ceil((box.conf[0]*100))/100
                print(f"Confidence = {probs}")
                # class name
                cls = int(box.cls[0])
                cls = str(cls) + "\n"
                if len(temp_arr) < 1:
                    x = str.encode(temp_arr[-1])
                    print(x)
                    ser.write(x)
                    temp_arr = []
                elif cls != temp_arr[-1]:
                    temp_arr.append(cls)
                    x = str.encode(temp_arr[-1])
                    print(x)
                    ser.write(x)
                    temp_arr = []
```

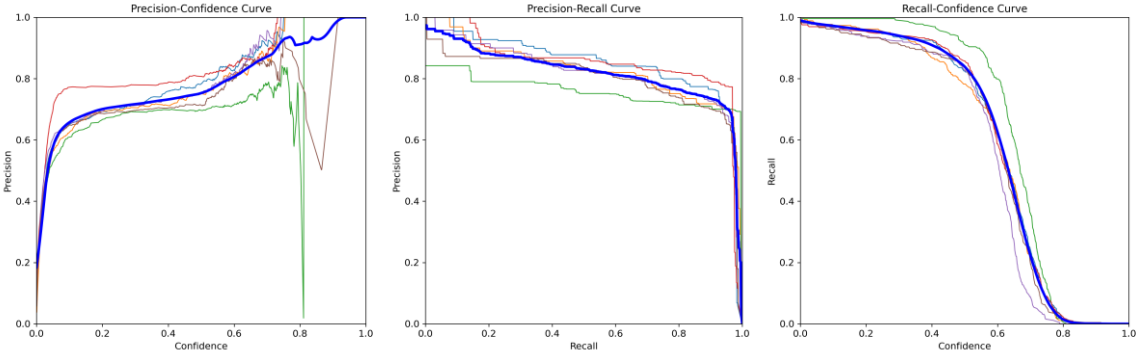
- Deploying model on Raspberry Pi

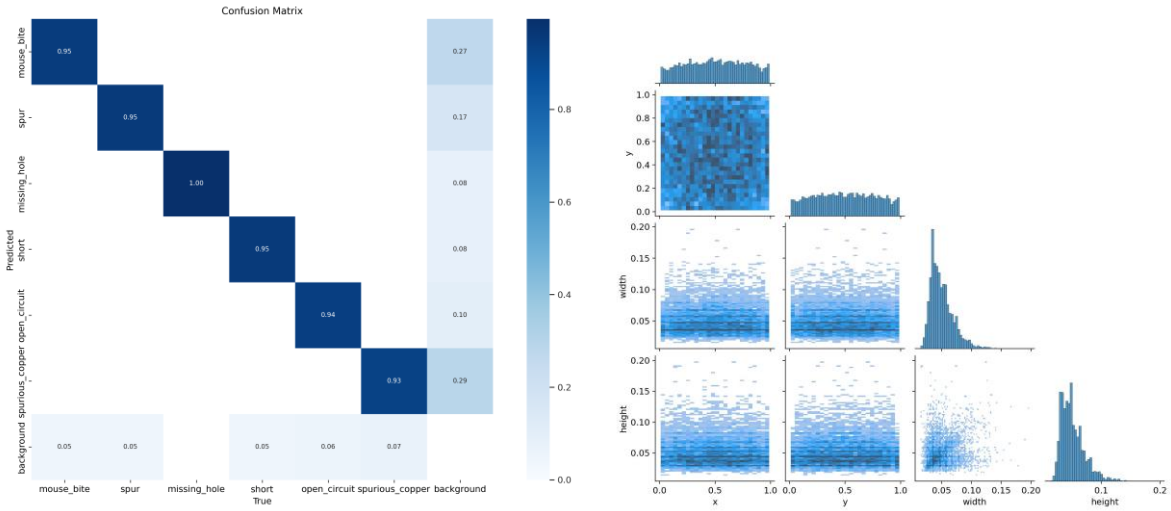
```
#!/bin/bash
# enter venv and project dir
current_dir=$(pwd)
if [[ "$current_dir" != "/home/pi/RPi5-pcb" ]]; then
  # Change directory to "RPi5-pcb"
  cd RPi5-pcb || {
    echo "Directory 'RPi5-pcb' not found." >&2
    exit 1
  }
echo "Changed directory to RPi5-pcb"
else
  echo "RPi5-pcb is the current directory"
fi

# start camera stream
nohup rpicam-vid --width 640 --height 640  --framerate 25 --
codec mjpeg  -t 0 --inline --listen -o tcp://0.0.0.0:8080 &
# run detection on stream
python main.py --model=./models/best.onnx &
```

APPENDIX B: ANALYSIS

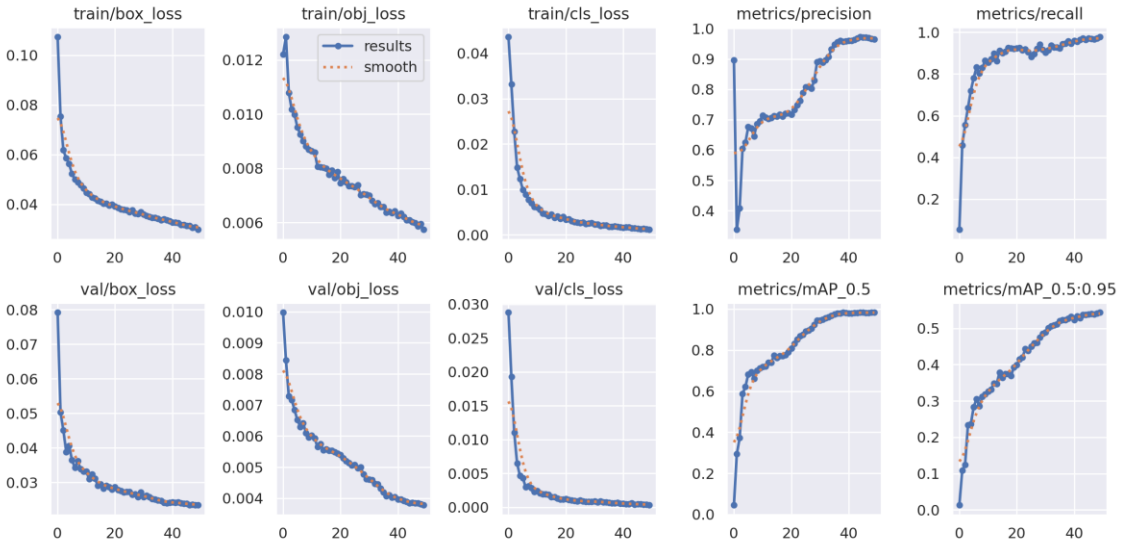
a. YOLOv5n

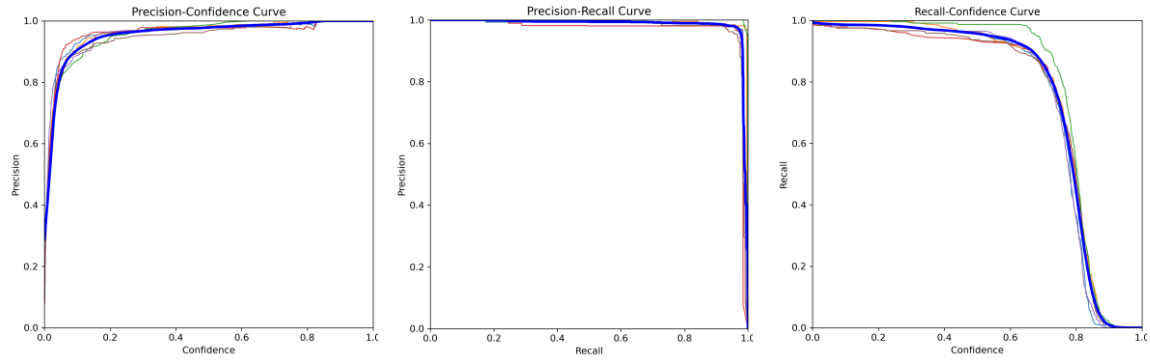


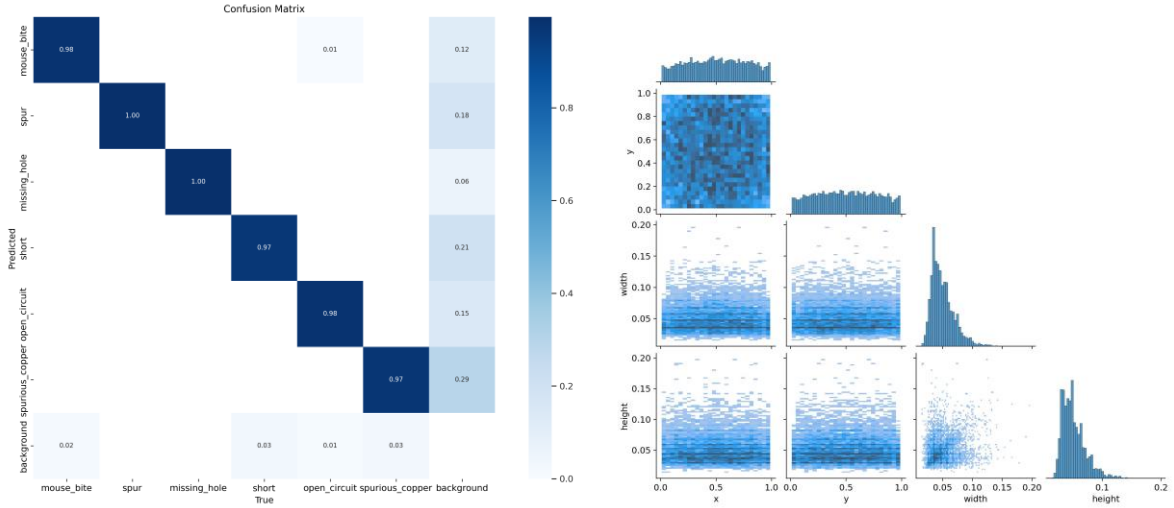




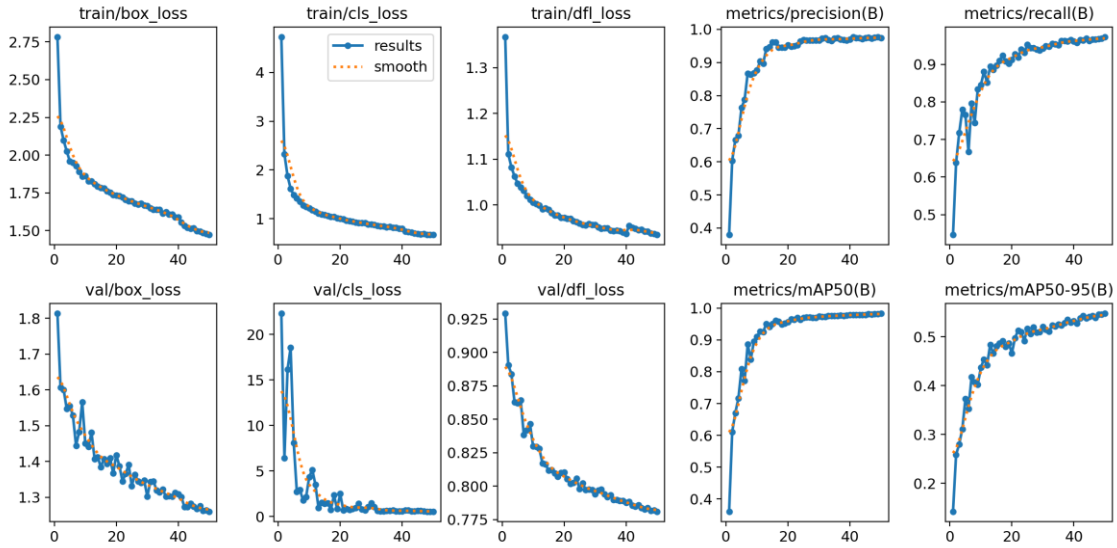
b. YOLOv5s

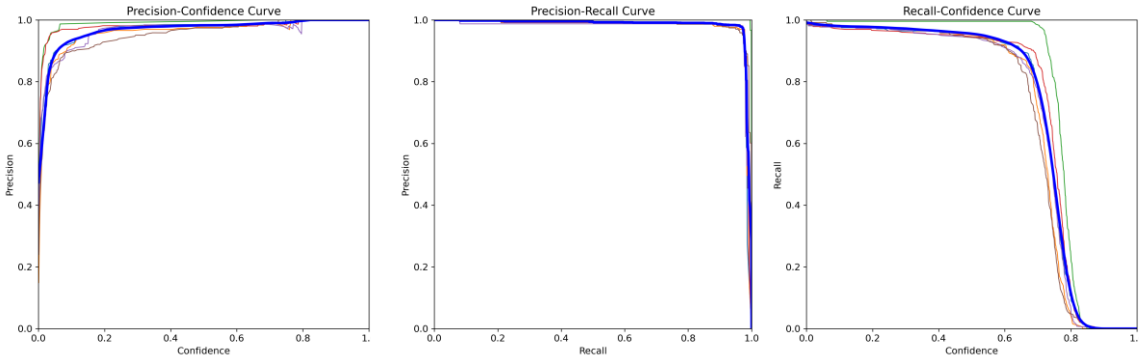

train/box_loss
train/obj_loss
train/cls_loss
metrics/precision
metrics/recall
results
smooth
val/box_loss
val/obj_loss
val/cls_loss
metrics/mAP_0.5
metrics/mAP_0.5:0.95

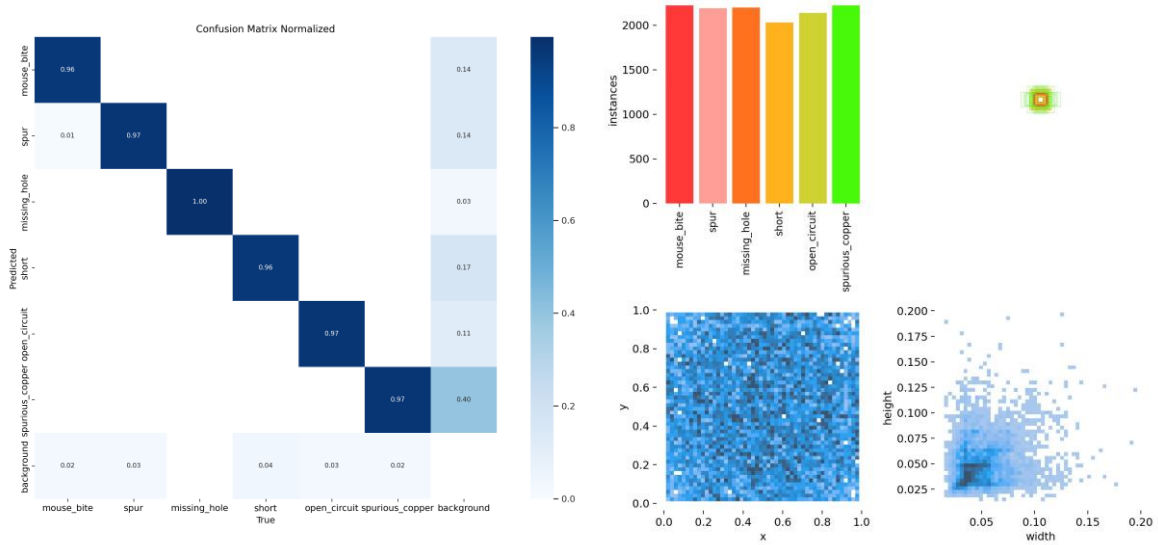

Precision-Confidence Curve
Precision-Recall Curve
Recall-Confidence Curve
Precision
Recall
Confidence


Confusion Matrix
mouse_bite
spur
missing_hole
short
open_circuit
spurious_copper
background
Predicted
True
x
y
width
height

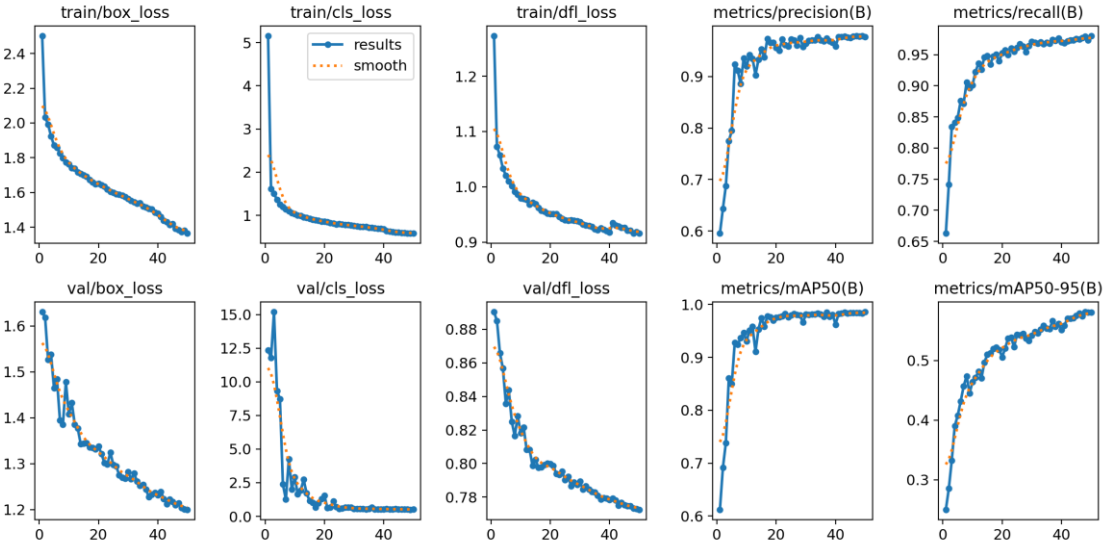
c. YOLOv8n

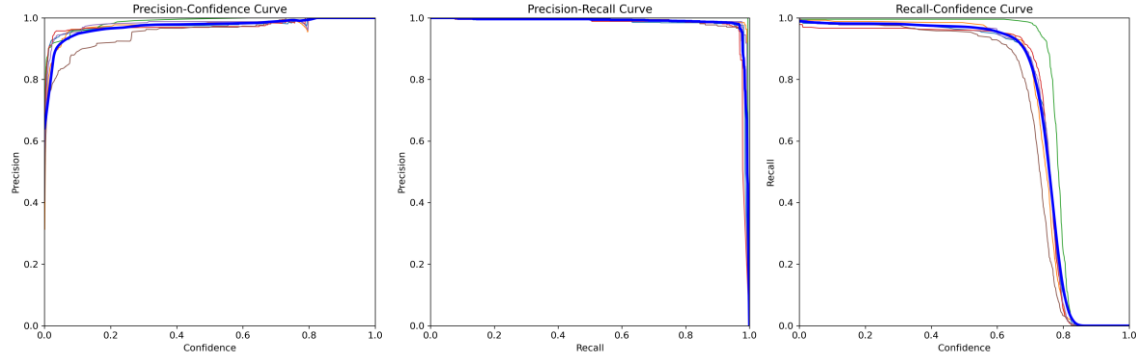

train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)

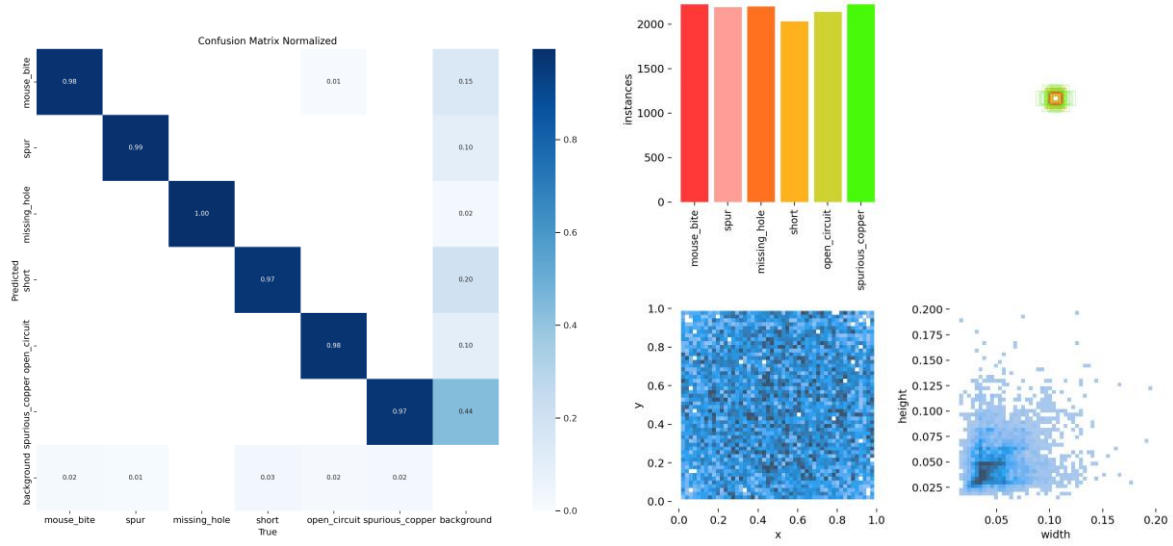

Precision-Confidence Curve
Precision
Confidence
Precision-Recall Curve
Precision
Recall
Recall-Confidence Curve
Recall
Confidence


Confusion Matrix Normalized
Predicted
True
mouse_bite
spur
missing_hole
short
open_circuit
spurious_copper
background
instances
x
y
width
height

d. YOLOv8s


train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)


Precision-Confidence Curve
Precision
Confidence
Precision-Recall Curve
Precision
Recall
Recall-Confidence Curve
Recall
Confidence


Confusion Matrix Normalized
mouse_bite
spur
missing_hole
short
open_circuit
spurious_copper
background
Predicted
True
instances
x
y
width
height

الملخص

تعتمد معظم الصناعات اليوم بشكل رئيسي على لوحات الدوائر المطبوعة (PCB) كمكون أساسي في الأجهزة الإلكترونية. ومع إدراك أهمية لوحات الدوائر المطبوعة واستخدامها الواسع في مختلف المنتجات الإلكترونية، تم تكليف مصانع متخصصة بإنتاج هذه المكونات بكفاءة لتصنيعها بشكل جماعي. ومع ذلك، تم اكتشاف أن العيوب في لوحات الدوائر المطبوعة قد تبقى غير مكتشفة أثناء الفحص اليدوي، مما يؤدي إلى تعطل المنتج. نقترح محطة متخصصة في خط الإنتاج حيث تخضع لوحات الدوائر المطبوعة لفحص دقيق لأنواع مختلفة من العيوب. بعد ذلك، يتم فرز لوحات الدوائر المطبوعة المعيبة تلقائيًا لإصلاحها أو إعادة استخدامها المحتملة. لتحقيق هدفنا، قمنا بتطوير نموذج دقيق للغاية لاكتشاف العيوب استنادًا إلى بنية YOLOv8 المتقدمة. تم تدريب نموذجنا بعناية باستخدام مجموعة بياناتTDD-Net ، والتي تشمل مجموعة واسعة من تصاميم لوحات الدوائر المطبوعة بما في ذلك 6 أنواع من العيوب السطحية. من خلال اختبارات صارمة، واستعمال إصدارات مختلفة من إطار عمل YOLO وتجربة مختلف التوازنات، قمنا بتقييم أداء كل نموذج بدقة وحققنا دقة بنسبة 84.38% و mAP بنسبة 98.4% مع نموذج YOLOv8 الذي تم نشره بعد ذلك على Raspberry Pi لأداء الكشف في الوقت الفعلي عن لوحات الدوائر المطبوعة على خط الإنتاج من خلال البث المباشر المستمر.

الأكاديمية العربية للعلوم والتكنولوجيا والنقل البحري

كلية الهندسة والتكنولوجيا
قسم الالكترونيات والاتصالات

مشروع التخرج لدرجة البكالوريوس

الكشف عن عيوب الدوائر الكهربية المطبوعة و تصنيفهم

باستخدام خوارزمية YOLOv8

الطلبة القائمون بالمشروع:

منة الله يسري

نورهان مجدي

عمر احمد

محمد مجدي

محمد صلاح

تحت اشراف:

د. هشام حمدي علي

د. محمد عاطف الخريبي

يونيو – ٢٠٢٤