EPOXY-RELATED DEFECT DETECTION ON PCB OF WIRELESS EARBUDS WITH TRANSFER LEARNING

YIN KAR KIN

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Science (Honours) Software Engineering

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

October 2023

DECLARATION

I hereby declare that this study report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :

Name : Yin Kar Kin

ID No. : 20UEB05657

Date : 9th September 2023

APPROVAL FOR SUBMISSION

I certify that this study report entitled **"EPOXY-RELATED DEFECT DETECTION ON PCB OF WIRELESS EARBUDS WITH TRANSFER LEARNING"** was prepared by **Yin Kar Kin** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Science (Honours) Software Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	kckhor
Supervisor	:	Khor Kok Chin
Date	:	14/9/2023

Signature	:	-
Co-Supervisor	:	-
Date	:	-

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my FYP supervisor, Dr. Khor Kok Chin, for his guidance throughout this project. When I embarked on this journey, my knowledge of both deep learning and computer vision was limited. However, Dr. Khor patiently and passionately guided me, providing the necessary methods to begin my study and delve deep into the world of machine learning. Without his dedicated efforts, I would not have been able to successfully complete this project.

In addition, I extend my thanks to ASPL Malaysia for their collaboration on this project. They generously provided a valuable real-world PCB dataset and offered technical training that greatly facilitated our research. Their involvement and resources greatly contributed to the success of my project, and I am thankful for the opportunity to work alongside such a reputable organization. I would also like to express my gratitude to my FYP moderator, Dr. Wong Whee Yen, for her valuable insights, constructive feedback, and contributions that helped improve the quality of my project.

Lastly, I wish to convey my appreciation to my family and friends who supported me throughout this project, providing the motivation and strength needed to overcome challenges and complete this endeavour.

ABSTRACT

Due to the increased manufacturing of wireless earbuds, the semiconductor industry's requirement for PCBs has increased drastically. As the manufacturing of PCBs grows, there is a need to improve the quality control process of the PCB, especially in the defect detection phase, by filtering out any defective PCBs and stopping them from being used in the manufacturing of wireless earbuds. This study evaluated three deep learning models that could perform defect detection for epoxy-related defects on the PCB of wireless earbuds with at least 90% accuracy. Transfer learning was applied to three pre-trained image classification deep learning models: ResNet50, Xception, and InceptionV3. The models were trained on a real-world PCB dataset provided by ASPL Malaysia after preprocessing the dataset images using OpenCV. ‘Epoxy Overflow on Die’ and ‘Epoxy Overflow on LED’ defects were detected by ResNet50 with an accuracy of 97.3% and 94.0% respectively, while Xception achieved an accuracy of 98.0% in detecting ‘Epoxy on Die’ and ‘FM on Die’ on the testing dataset.

TABLE OF CONTENTS

LIST OF APPENDICES

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS / ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Networks
AVI	Automated Visual Inspection
CNN	Convolutional Neural Networks
CUDA	Compute Unified Device Architecture
IDE	Integrated Development Environment
LED	Light-emitting Diode
mAP	mean Average Precision
ML	Machine Learning
OCT	Optical Coherence Tomography
PCB	Printed Circuit Board
ROI	region of interest

CHAPTER 1
INTRODUCTION

1.1 Project Background

The increasing demand for PCB (Printed-Circuit Board) is primarily driven by the rapid growth of consumer electronics such as smartphones, tablets, and wireless earbuds in recent years. PCBs have played a vital role in enhancing the functionality and performance of wireless earbuds. With the increasing demand for wireless earbuds, manufacturers in the semiconductor industry are experiencing mounting pressure to enhance their quality control processes, particularly during the defect detection phase on PCBs.

Among the various types of defects found on PCBs, epoxy-related defects are quite common. Since PCB manufacturing processes are automated, with machine robots responsible for applying epoxy, issues such as excess epoxy or accidental spills on unintended spots can occur during the epoxy application on the PCB.

Currently, there are several methods for detecting epoxy-related defects in the semiconductor industry. Some of the most widely used methods are: Firstly, visual inspections can be used to detect epoxy-related defects in chips. This can involve automated visual inspection (AVI) systems to inspect the PCB for any irregularities or anomalies in the epoxy layer (Haddad et al., 2018a). Secondly, using electron microscopy techniques makes it possible to examine semiconductor materials with extremely high precision, enabling the detection of defects at a microscopic level (Nakagaki, Honda and Nakamae, 2009). Thirdly, optical coherence tomography (OCT). OCT is an imaging technique that can be used to analyse the internal structure of the PCB and detect any epoxy-related defects. It measures the time delay and intensity of reflected light, providing high-resolution images of the chip (Serrels, Renner and Reid, 2010).

Besides all the methods mentioned above, deep learning is also one of the most commonly used techniques for detecting epoxy-related defects on PCBs (Devika and George, 2019a). Deep learning has proven highly effective across numerous domains and has achieved significant breakthroughs in various fields over the past decade.

This study aims to propose deep learning models that can achieve high accuracy in epoxy-related defect detection on the PCB of wireless earbuds by utilising transfer learning on renowned deep learning image classification models such as ResNet50, Xception, and InceptionV3. This study involves training these deep learning models on three different types of defects, resulting in three separate trained models for each defect. By the end of this study, the performance of these deep learning models will be evaluated and compared to select the best-performing model for each defect.

1.2 Problem Statement

Many existing methods can detect epoxy-related defects on PCBs in the semiconductor industry. However, there are limitations in the existing methods, which become a big challenge for big names in the industry to tackle the problems. After extensive discussions with Allied Solutions (ASPL) Malaysia Sdn Bhd and a thorough review of similar works conducted within the industry, this study has identified three specific problem statements:

1. Low Accuracy of Epoxy-Related Defect Recognition using existing AI Model

As mentioned by ASPL, the existing AI models within the semiconductor industry lack reliability in detecting epoxy-related defects on PCBs. The accuracy of detecting epoxy-related defects is low with the current AI models on the market.

2. Epoxy-Related Defects are Too Small in Size for Human Eyes to Spot

Epoxy-related defects on PCBs are too tiny for human eyes to detect, even with a microscope (Bellini et al., 2021a). The size of the defects, such as epoxy overflow and epoxy spill on the PCB, is usually in micrometres, making it difficult for humans to discern the difference in size accurately.

3. Time Cost is High

Existing methods for detecting epoxy-related defects in PCBs in the semiconductor industry can be time-consuming (Huang and Pan, 2015). Some methods, such as electrical testing and thermal imaging, require collecting and processing large amounts of data. This process can be particularly time-consuming, especially when the data needs to be analysed in real-time.

1.3 Aim and Objectives

This study aims to propose deep learning models that can detect epoxy-related defects of the PCB of wireless earbuds with the transfer learning approach.

Objective 1

To collect an image dataset of the PCB of wireless earbuds and pre-process them for deep learning model training.

Objective 2

To utilise transfer learning on pre-trained image classification deep learning models, i.e., ResNet50, InceptionV3, and Xception.

Objective 3

To evaluate the performance of the deep learning models trained on the collected image dataset and select the best-performing model for each epoxy-related defect type on the PCB with a minimum accuracy rate of 90%.

1.4 Proposed Solution

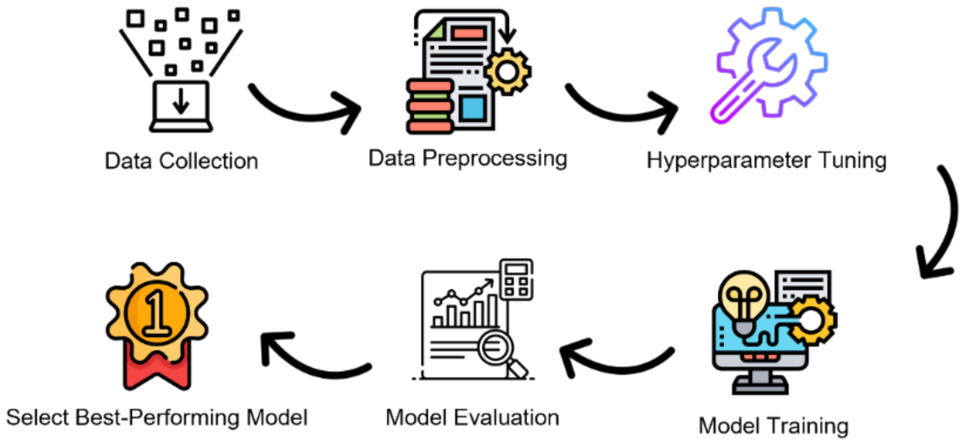


Figure 1.4.1: Proposed Solution Flowchart

Figure 1.4.1 shows the proposed solution flowchart of this study. The flowchart illustrates the phases necessary to fulfil the aim and objectives of this study.

1.5 Scope and Limitation of the Study

This study proposes three deep learning models using transfer learning that can achieve at least 90% accuracy in detecting epoxy-related defects on the PCB of wireless earbuds using a transfer learning approach. This study will focus on three distinct epoxy-related defect types found on PCBs.

For defect type 1, "Epoxy Overflow on Die," and defect type 2, "Epoxy Overflow on LED," binary classification will be performed to classify PCB images as either ‘Good’ or ‘Defect’.

For defect type 3, multi-class classification will be performed to classify the two similar defects: 'Epoxy on Die' and 'Foreign Material (FM) on Die,' resulting in three output classes of 'Good,' 'FM,' or 'Epoxy.' This study will train ResNet50, Xception, and InceptionV3 on each defect type and select the best-performing model. By the end of this study, one deep learning model will be proposed for each defect type, resulting in a total of three models.

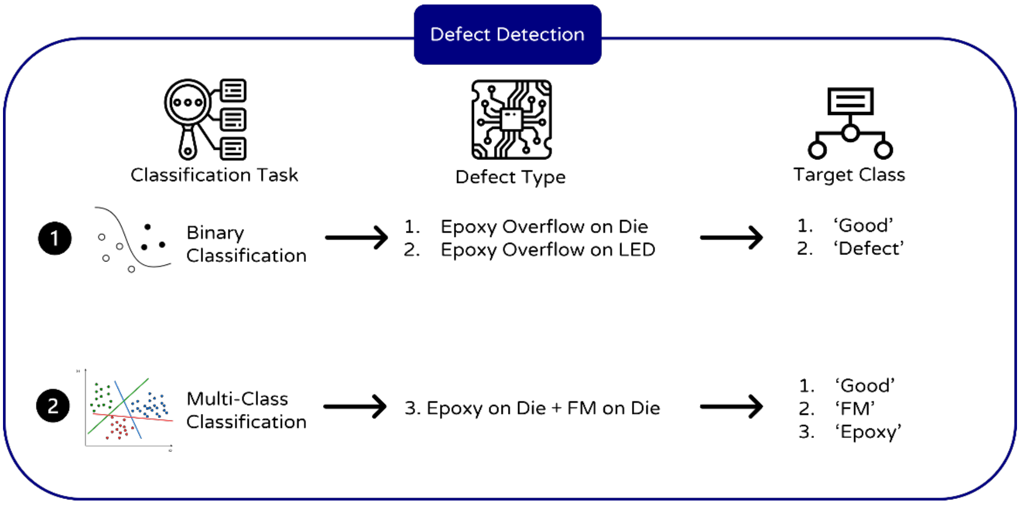


Figure 1.5.1: Classification Task with Defect Types

Table 1.5.1: Rejection Criteria for the Defects

Defect Type	Component	Rejection Criteria
1. Epoxy Overflow on Die	Die	• The excess epoxy overflows from edge of the component for more than 60 µm.
2. Epoxy Oveflow on LED	LED	
3. Epoxy on Die + FM on Die	Die	• The FM or epoxy on top of the component has more than 35 µm in length from the edge of the component. • The FM or epoxy has an accumulated area of 960 µm^2.

The research tools used in this study are Jupyter Notebook and Spyder as the IDE, machine learning libraries such as Keras, TensorFlow, and Scikit-learn, image preprocessing libraries like OpenCV, and visualisation libraries including Seaborn and Matplotlib.

This study has some limitations. Firstly, it did not consider external factors that could have affected the detection of the PCB. Secondly, it did not address the practical implementation of the deep learning model in a production environment. Acknowledging these limitations, the study emphasises the need for further research to improve the application of transfer learning in real-time defect detection on PCBs.

CHAPTER 2
LITERATURE REVIEW

2.1 Overview

Over the last decade, the semiconductor industry has experienced rapid growth due to the increased demand for more efficient and powerful electronic devices. However, as the manufacturing process of PCBs becomes more complex, detecting defects during production has become increasingly challenging. This poses a major challenge for the industry, leading to costly recalls and reduced product quality. To address this, the industry has employed several methods, including deep learning and machine learning techniques, which have become increasingly popular. This review of existing literature aims to thoroughly examine the latest advancements in machine learning and deep learning techniques for identifying defects in PCBs. This review will focus on the various techniques used, their effectiveness, and any constraints or limitations they may have. Furthermore, the review aims to pinpoint areas lacking the existing research and propose potential avenues for future studies. In addition, this paper will propose three deep learning models for detecting epoxy-related defects in the PCB of wireless earbuds using transfer learning.

2.2 Similar Works

Bellini et al. (2021) proposed a solution to detect defects in power semiconductors using an active deep learning method. The proposed solution can easily identify broken parts in the PCBs without requiring extensive manual image labelling, significantly reducing annotation effort. The patterns of the broken parts found by the computer are then used to create a list of different types of defects, facilitating further research into the underlying causes and prevention strategies. The key contribution of this work is the reduction of labelling costs through minimal image annotation effort. The trained neural network also demonstrates generalizability to new types of defects not present in the training sets. Additionally, the proposed solution is suitable for small datasets, achieving a precision of 0.99 with only 1,737 training examples.

Devika and George (2019b) introduced a deep learning model that employs convolutional neural networks (CNNs) to identify wafer defect patterns. CNNs use convolutional and pooling layers to extract features and fully-connected layers to classify them. CNNs exhibit robustness against random noise and efficiently detect single and multiple defects. Thus, the proposed CNN model can classify various defect patterns and outperform other models in terms of overall performance. The proposed CNN model can detect four types of defect patterns: circle, cluster, scratch, spot, and their combinations. The results of the proposed model show a 100% accuracy rate in detecting patterns of a single defect and an accuracy of 84% in detecting patterns of mixed defects.

Haddad et al. (2018b) presented a new method for detecting and classifying defectusing multiple features and sparse-based techniques. The proposed approach used the stacking concept to improve the accuracy of classification. Then, the stacking-based classifier was improved with a downsampling technique to handle imbalanced data. The approach also included a new pruning technique to remove bad base learners. The challenges of the approach included a shortage of defective units, similarities among different defect classes, wide variations within the same defect class, and a data imbalance. According to the experiment's results on actual data from Intel, the suggested method performed better than previous approaches and achieved an overall classification accuracy of 98.5%, which was very high.

Chen et al. (2020) introduced a new approach for detecting structural defects in wafers using convolutional neural networks (CNN) that can achieve high detection accuracy. They designed a set of imaging acquisition systems to capture wafer images, as there were not enough images available in open databases. To prepare the images, digital image preprocessing technology was used to divide each wafer image into several smaller images. The proposed model, WDD-Net, used depthwise separable convolutions to reduce the number of computations and parameters. The WDD-Net also incorporated multiple 1x1 standard convolutions to enhance network depth. Finally, the WDD-Net was designed to work well for edge computing, meaning that it could directly perform data collection and defect detection on local computing devices. WDD-

Net is five times faster in detection speed compared to VGG-16 and MobileNet-v2. Besides, it achieved more than 99% detection accuracy.

Huang et al. (2022) proposed a small object detection method called SO-YOLO. The research utilised CspDarknet53 as the main architecture of the network and improved the entire PANet through a novel feature fusion approach. This approach involved selecting suitable layers to fuse, thereby expanding the receptive field. The researchers also used the k-means++ method for anchor estimation as a priori. Lastly, they employed the mosaic data augmentation technique for dataset preprocessing. SO-YOLO achieved a mean average precision (mAP) score of 0.86 and an F1 score of 0.84.

Schlosser et al. (2019) proposed a novel deep neural network-based hybrid approach. The SH-CNN model proposed enables the identification and categorisation of very small structures within high-resolution images. The SH-CNN method combines traditional image processing techniques and artificial neural networks to recognise very small structures within images more effectively. SH-CNN achieved a mean accuracy of about 0.921 with a 0x augmentation level.

Tello et al. (2018) introduced a new method that used deep machine learning to detect and categorise different types of defects in semiconductor manufacturing. The method involved several steps, including noise reduction, differentiation between wafers with single or mixed defect patterns, classification of single defect patterns using a shallow-structured randomised general regression network, and identification of mixed defect patterns using a deep-structured convolutional network. The model was evaluated on real data and achieved an accuracy of 86.17% for detecting both single-defect and mixed-defect patterns.

Zhang et al. (2018) proposed a CNN-based model that acted as the configuration for the defect detection system on PCBs. The proposed solution used convolutional layers and a dense layer to obtain parameters for feature extraction and classification through a training process, from original images to detection results. The approach overcomes the complexity of traditional vision methods with multiple image processing steps. Additionally, it addressed a common challenge in the semiconductor industry: limited dataset availability. The proposed model achieved an mAP of 0.9959 in defect detection.

Raihan and Ce (2017) proposed a method for detecting defects in PCBs using a computer vision library called OpenCV. This method involves analysing images of PCBs using techniques called image subtraction and blob detection. The method is useful for identifying various visual defects in PCBs, especially complex patterns. By analysing images and highlighting differences, defective parts of PCBs can be quickly identified. However, the accuracy of detection depends on the resolution of the images, meaning low-resolution images have a lower accuracy rate of 40% with a faster detection time of 0.856 seconds, whereas high-resolution images have a higher accuracy rate of 80% with a slower detection time of 2.68 seconds.

Kim et al. (2021) proposed a PCB defect inspection system using a skip-connected convolutional autoencoder. The dataset of PCB images underwent preprocessing to remove unused areas and improve image quality through commonly used image enhancement techniques. To address the problem of an imbalanced training dataset, the datasets were also augmented. By using anaugmented datasets, the autoencoder model was effectively trained to differentiate non-defective images from potentially defective ones. However, the proposed method did not perform testing on actual PCB datasets, and the detection rate is low for untrained defect datasets. The skip-connected convolutional autoencoder achieved 0.9808 accuracy with its defect detection performance.

Xin, Chen and Wang (2021) proposed a new algorithm that improves the YOLOV4 method for detecting defects in printed circuit boards (PCBs). They used a dataset of PCB defect images provided by the Intelligent Robot Laboratory of Peking University, which had various defects, making the algorithm more reliable. The authors analysed the feature distribution of the CSPDarkNet53 structure layer and the size distribution of the detection target defects in the dataset. They improved the data preprocessing and input stage by automatically dividing the image based on the average size of the detection image's bounding box and increasing the probability of an anchor containing a detection target. The proposed algorithm achieved a high mean average precision (mAP) of 96.88% in detecting PCB defects.

Zeng et al. (2022) introduced a new method for detecting defects in PCBs called the IPDD framework, and within that framework, they proposed a new feature fusion method called the ABFPN. The ABFPN method uses a combination of atrous convolution operators and skips connections to consider context information and merge features from different levels. Additionally, a balanced module is used to improve features across different levels. The IPDD framework is particularly useful for detecting small defects on the surface of PCBs. The authors demonstrated that their framework achieved an average precision of 98.8% over intersection over union (IoU) at 0.5, outperforming other IPDD algorithms.

Ding et al. (2019) proposed a new model called TDD-Net to detect small defects in PCBs using deep learning algorithms and feature pyramid ConvNets. The model achieved the highest mAP of 98.90% compared to other models presented in the paper.

Khalilian et al. (2020) proposed a new method for detecting and locating defects in printed circuit boards (PCBs) using denoising convolutional autoencoders. The model used a neural network that receives a noisy input image and attempts to clean it up to produce a clear output image. By training the model on clean and noisy images, it learned to identify defects and locate them accurately. The results showed that the proposed method achieved a high accuracy of 97.5%, which is better than other existing defect detection methods.

Lu et al. (2018) proposed a new PCB defect inspection (PCBDD) framework to overcome the limitations of the traditional reference comparison approach. The proposed framework used LBP and HOG features to train two SVM models and combined them using Bayes feature fusion. The accuracy of the proposed method was compared to that of a single feature method and found to be much better at 89.22% while only slightly reducing the speed, indicating that it is a highly effective approach for detecting defects on PCB surfaces.

Adibhatla et al. (2020) presented a new method to detect defects in PCBs using a deep learning algorithm-based YOLO approach. The model consisted of 24 convolutional layers and two fully-connected layers. They tested the model on PCBs and achieved a high defect detection accuracy of 98.79% when using a batch size of 32. The study shows that a YOLO model with deep convolutional neural networks (CNNs) can achieve excellent results in detecting defects in PCBs.

The research papers mentioned are summarised in Table 2.2.1.

Table 2.2.1: Similar Works Summary

No	Author	Title	Technique used	Hyperparameter	Strength/Limitations	Result	Future Work
1.	Bellini et al. (2021)	An Active Deep Learning Method for the Detection of Defects in Power Semiconductors	- Image Annotation • Active learning		- Model generalise well to new kinds of defects, especially large size defects - Greatly reduce image annotation time - Novel morphological features not present in training set are misclassified as defects	- 5th Learning cycle (1737 examples) • Precision: 0.99 • Recall: 0.79	- Further reduce image annotation time - Increase training set size for rare defects
2.	Devika and George (2019b)	Convolutional Neural Network for Semiconductor Wafer Defect Detection	- Classification • CNN (8 convolutional layers)	- Adaptive moment estimation (Adam) • Learning rate: 0.0004 • Batch size: 32 • Epoch: 100	- Image preprocessing not needed - Able to detect defects over random defects	- 100% efficiency in single defect pattern detection - 84% accuracy in mixed defect pattern detection	
3.	Haddad et al. (2018b)	Multifeature, Sparse-Based Approach for Defects Detection and Classification in Semiconductor Units	- Feature Extraction • Bag-of-visual-words (BoW) model - Sparse coding - Classification • Stacked-based classfier - Adaptive data sampling technique • Adaptive downsampling • Syntethic oversampling		- Background feature subtraction which enhances the classification accuracy when datasets are small - Ensemble pruning and metadata oversampling effectively solve data imbalance and improve metaclassifier	- Average classification accuracy: • 30% training data: 95.88% • 80% training data:: 98.5%	
4.	Chen et al. (2020)	A Light-Weighted CNN Model for Wafer Structural Defect Detection	- Data preparation • Machine vision system - Data preprocessing • OpenCV Findcontours - Classification • CNN-based WDD-Net (3*3 standard convolution and 3 depthwise seperable convolution)	- Adaptive moment estimation (Adam) • Learning rate: 0.001 • Batch size: 32 • Epochs: 50	- High detection speed and small model size which makes WDD-Net is applicable in practical applications - Detection accuracy is slightly lower than VGG-16 and MobileNet-v2	- Overall detection accuracy and detection speed (FPS): • WDD-Net_28*28: 99.70%, 8719.3 • WDD-Net_224*224: 99.44%, 105.6	- Establishing wafer defect detection dataset - Research in unsupervised learning

5.	Huang et al. (2022)	Small object detection method with shallow feature fusion network for chip surface defect detection	- Feature extraction • Backbone network such as VGG Net, ResNet and Inception Net - Feature fusion • Modified fusion network in SO-YOLO (PANet) - Clustering • K-means ++ - Data preprocessing • Mosaic data augmentation method	• Learning rate: 0.001 • Attenuation coefficient: 0.0005 • Iteration: 10000 times	- Has less number of parameters and higher classification and detection accuracy compared to YOLOv4	- Detection accuracy • F1 score: 0.84 • mAP: 0.86 • BFLOPS: 53.624	- Higher detection accuracy and reduced model complexity
6.	Schlosser et al. (2019)	A Novel Visual Fault Detection and Classification System for Semiconductor Manufacturing Using Stacked Hybrid Convolutional Neural Networks	- Localisation of ROI • CNN - Classification • CNN		- enables the recognition of small structures with higher efficiency and accuracy at the pixel level.		- Enhance the proposed system in terms of audio and heat signatures - Has to deploy under production test conditions
7.	Tello et al. (2018)	Deep-Structured Machine Learning Model for the Recognition of Mixed-Defect Patterns in Semiconductor Fabrication Processes	- Classification • RGRN – single-defect pattern • DSCN – mixed-defect pattern - Image preprocessing • Spatial filter - Feature extraction • Splitter	• Fold: 10 • Learning rate: 0.001	- Improves the identification of spatial relationships in mixed-defect patterns - Can categorize patterns with a single defect as well as those with multiple defects	• Overall accuracy: 86.17%	- Improve the proposed method for real-time identification - The proposed method should be tested with large volume real datasets to be able to handle the increase in computational complexity
8.	Zhang et al. (2018)	Improved bare PCB defect detection approach based on deep feature learning	- Feature extraction • VGG16 - Deep feature learning • Data augmentation • Parameters transfer learning - Localisation • Sliding window approach - Classification	- Stochastic gradient descent (SGD) • Momentum: 0.9 • Learning rate: 0.0001 Epochs: 100	- During the training process, the values of the feature extractor and classifier parameters are obtained. - The learned deep feature possesses the capability to differentiate between defects effectively. - The system's high complexity enables its applicability in a wider range of situations.	• mAP: 0.9959	- Improve the top layers of the model - Focus on selecting better feature selection algorithms - Focus on network architecture

			• Combine SVM with LBP and HOG feature				
9.	Raihan and Ce (2017)	PCB defect detection USING OPENCV with image subtraction method	- Image processing • OpenCV		- Short detection time but low accuracy for low resolution PCB images - Long detection time but high accuracy for high resolution PCB images	- High resolution image • Accuracy: 40% • Time: 0.856 seconds - Low resolution image • Accuracy: 80% • Time: 2.68 seconds	- Improve blob detection and morphology algorithm - Apply hough-transform process and image segmentation - Research any method that can optimise the system
10.	Kim et al. (2021)	Printed Circuit Board Defect Detection Using Deep Learning via A Skip-Connected Convolutional Autoencoder	- Image preprocessing • Histogram equalisation • Median filtering method - Data augmentation • Geometric transformation • Noise injection - Classification • Skip-Connected Convolutional Autoencoder	• Batch size: 128 • Optimiser: ○ Weight decay: 5 x 10^4 ○ Momentum: 0.9 • Learning rate: ○ 0.1 – 60 epochs ○ 0.02 – 120 epochs ○ 0.004 – 160 epochs ○ 0.0008 – 300 epochs	- The proposed method is tested using artificial dataset - The proposed method does not generalise well to new unseen data	• Accuracy: 0.9808 • TPR: 09773 • TNR: 0.9843 • Precision: 0.9830 • F1: 0.9801 • BCR: 0.9806 • SSIM: 0.9749	- Test the models on real dataset
11.	Xin, Chen and Wang (2021)	PCB Electronic Component Defect Detection Method based on Improved YOLOv4 Algorithm	- Improved YOLO V4 structure • Backbone (Feature extraction) ○ 5 CSP modules (CSPDarkNet53) • Neck (Feature fusion) ○ SPP module ○ FPN+PAN module - Object detection	• Batch size: 64 • Learning rate: 0.001	- Suitable for small object detection - Faster detection speed and accuracy compared to YOLOv4 method	• Average detection accuracy: 96.88%	- Provide possibilities in developing object detection application

			• K-means clustering				
12.	Zeng et al. (2022)	A Small-Sized Object Detection Oriented Multi-Scale Feature Fusion Approach With Application to Defect Detection	- Feature extraction • Backbone ○ ResNeXt structure - Feature fusion • Neck ○ Skip-ASPP module ○ Balanced module - Data augmentation • Mixup method • AutoAugmetImage method - Data visualisation • TDD-Net	- Same for all 3 datasets • Batch size: 2 • Learning rate decay factor: 0.1 • Optimiser: SGD+Momentum • Regularisation method: L2 weight decay - Different for each datasets - MS COCO2017 • Iterations: 709716 • Initial learning rate: 0.0025 • Learning rate decay iterations: [473144, 650573] - VOC07+12 • Iterations: 270000 • Initial learning rate: 0.02 • Learning rate decay iterations: [180000, 240000] - VisDrone2019 • Iterations: 120000 • Initial learning rate: 0.02 • Learning rate decay iterations: [90000, 110000]	- The proposed IPDD framework performs better than existing methods in terms of localisation and classification in small object detection.	- MS COCO2017 • Average Precision: 38.6% - VOC07+12 • mAP: 85.59% - VisDrone2019 • Average Precision: 17.1%	- Utilize the suggested IPDD framework in other detection tasks involving small objects. - Improve positioning performance of IPDD framework with better localisation method - Tune hyperparameters of IPDD framework
13.	Ding et al. (2019)	TDD-net: a tiny defect detection network for printed circuit boards	- Object detection • Faster R-CNN - Data augmentation		- TDD-Net is strong which it can be applied to other fields	• mAP over tIoU of 0.5: 98.90%	- Investigating zero-shot learning techniques because

			• Gaussian noise • Change light • Rotate image • Flipping • Random chop • Shift - Feature extraction • ResNet-101 - Feature fusion • Feature pyramid architecture - Reasonable anchors • K-mean clustering - Improve quality of ROI • Online hard example mining		- Combines structurally strong features with semantically strong features - Structurally strong feateres are combined with semantically strong features		of the limited size of the training dataset. - Optimise the network - Optimise the post-process methods - Utilize TDD-Net to different kinds of PCB defects
14.	Khalilian et al. (2020)	PCB Defect Detection Using Denoising Convolutional Autoencoders	- Proposed network structure • Encoder (3 layers) ○ Convolution layer ○ Batch normalisation ○ Activation layer ○ Max pooling • Decoder (3 layers) ○ Convulsion layers ○ Batch normalisation ○ Activation layer ○ Up sampling	• Batch size: 2 • Epochs: 4 & 17	- Can detect defects and recover them - Can be applied to other products	- Best result when threshold = 100 • Recall: 0.97 • Precision: 0.983 • Selectivity: 0.983 • Accuracy: 0.975 • F-score: 0.976	- Improve subtracting algorithm to locate defects more accurately
15.	Lu et al. (2018)	Defect detection of PCB based on Bayes feature fusion	- Image preprocessing • Image segmentation technology • Median filter and mean filter - Features extraction • LBP • HOG - Feature fusion • Bayes fusion - Classification		- The proposed method solved uneven illumination which occurs in traditional reference comparison method	• Accuracy: 89.22% • Detection speed: 20 seconds	- Transform a binary classification problem into a multi-class classification problem - Develop a method to detect the spatial location of defects - Attempt to combine multiple types of features for evaluation purposes.

			• SVM				
16.	Adibhatla et al. (2020)	Defect Detection in Printed Circuit Boards Using You-Only-Look-Once Convolutional Neural Networks	- Object detection • Tiny-YOLO-V2 - Classification • CNN - Data collection • Novel user interface	• Batch size: 32	- Suitable for large-scale PCB quality inspection as it can accurately detect defects. - CNNs can learn the target task automatically with proper tuning of model parameters. - The proposed model accuracy may be low with unbalanced datasets	• Mean Accuracy: 98.79%	- Enhance CNN performance by conducting experimental evaluation and performance analysis

Table 2.2.1 shows that deep learning methods are a current trend in defect detection problems in the semiconductor industry. CNNs have been proven to be effective in detecting defective PCBs. Many papers have proposed novel approaches that address the issue of high time costs and automate the defect detection process using deep learning methods. These deep learning models were trained and tested on fine and defected PCB images, achieving high accuracy in detecting a single defect pattern. Pre-trained models such as VGG16, ResNet, YOLO, and GoogleNet are frequently utilised in computer vision, and they have demonstrated the ability to generalise well to new image datasets, making them popular choices for detecting defective PCBs in the semiconductor industry. Although many innovative solutions have been suggested to address the issue of defect detection, there are still shortcomings in the approaches and issues in the semiconductor industry that need to be resolved.

The shortage of open-source image datasets for fine and defected PCBs is a frequently cited issue in many papers, which presents a challenge for training deep learning models to perform well on new, unseen data. Moreover, imbalanced class distributions can make it difficult for the model to learn about the minority class effectively. Although some papers have proposed methods to address these issues, they still exist and continue to pose significant challenges to the semiconductor industry.

While some novel models can detect multiple defect patterns in combination, most can only detect one defect type at a time. Furthermore, many proposed solutions are not yet practical in real-world settings due to high costs, a lack of diversity in training data, and insufficient computational resources. As a result, real-time detection of defective PCBs remains a challenging problem.

To address these challenges, future research efforts should focus on creating more diverse and comprehensive image datasets, developing more effective algorithms to handle imbalanced class distributions, and exploring innovative ways to reduce the cost of training and deployment. Furthermore, the practical implementation of deep learning models in production environments should also be considered to ensure their usefulness in real-world settings.

2.3 Image Preprocessing Techniques

Open Source Computer Vision Library (OpenCV) is popular for preprocessing images for training deep learning models. OpenCV is mainly used for real-time operations such as object and face recognition or handwriting classification by processing images and videos. With more than 2,500 computer vision and machine learning algorithms, OpenCV is widely used by major companies like Google, Yahoo, and Microsoft. The library has a Python interface and supports Windows, making it compatible with the study's environmental setup.

While OpenCV contains many useful functions for image processing, ASPL found the following five OpenCV functions to be useful and vital in processing images of PCBs:

cv2.threshold()

The process of thresholding converts an image from grayscale to binary to isolate regions of interest from the background. Each pixel of the input grayscale image is assigned a binary value based on a threshold value, which is compared to the pixel's intensity value. The function requires four parameters: the input grayscale image, the threshold value, the maximum value above the threshold, and the threshold type. To achieve a better output image and improve accuracy in detecting defects when applied to deep learning models, it is essential to find the optimal threshold value for extracting a good region of interest.

cv2.Canny()

The Canny edge detection algorithm is a useful tool for detecting edges in an image. This function first applies Gaussian blurring to the input image to reduce noise. It then computes the intensity gradients of the image and looks for potential edges by searching for local maxima in the gradient magnitude. Non-maximum suppression is applied next to thin out the edges and keep only the strongest ones. Finally, hysteresis thresholding is applied to link edges that belong to the same feature in the image. The function requires three parameters: the input grayscale image, the first threshold value, and the second threshold value. These threshold values are used as the minimum and maximum intensity gradients, respectively, that the function uses to detect edges.

<u>cv2.findContours()</u>

The "findContours" function is used to identify contours in a binary image. This function processes a binary image and produces a set of contours represented as simplified or full coordinates. It requires three parameters: the input image, the retrieval mode of the contour, and the contour approximation method. The retrieval mode can be adjusted to retrieve all the contours, only the outer contours, or all contours and their hierarchical structure. The contour approximation method reduces the number of points in the contour.

<u>cv2.connectedComponentsWithStats()</u>

The connectedComponentsWithStats function is a useful tool for extracting connected components from a binary image. Connected components refer to sets of pixels that share the same pixel value and are connected. This function is critical for extracting regions from the input image that are connected, which is often necessary in image processing tasks.

To use this function, three parameters are required: the binary image, the connectivity value (which can be set to 4 or 8), and a constant representing the data type used for storing pixel values.

<u>cv2.matchTemplate()</u>

The matchTemplate function is used to match an image with a template. It takes an input image and a template image and returns a grayscale image where each pixel represents the match score of the template at that specific position in the input image. The function functions by locating a region in the input image that resembles the template. As recommended by ASPL, this function serves as a crucial backup option in case other image processing techniques fail to produce the desired results.

2.4 Traditional Machine Learning

The term "machine learning" describes teaching computers to learn from data. It involves the convergence of computer science and statistics and the utilisation of algorithms that are designed to execute a task without explicit programming.

Instead, these algorithms identify patterns in the data and make predictions when new data is available (Wolfewicz, 2023). Typically, the process of using these algorithms to acquire knowledge can be categorised into supervised and unsupervised learning based on the data type used to train the algorithms (Wolfewicz, 2023). Statistics is the main underlying concept behind machine learning, where the algorithm is trained to make predictions by identifying data patterns without explicit programming. To put it briefly, machine learning is an interdisciplinary field that merges computer science and statistics to empower computers to learn from data without being explicitly programmed (Wolfewicz, 2023). There are two primary categories of problems in machine learning: supervised and unsupervised.

2.5 Supervised Learning

Supervised learning, also known as supervised machine learning, falls under the broader categories of machine learning and artificial intelligence. It encompasses a machine learning process that utilises a labelled dataset, wherein both the data and corresponding labels are provided to the model for training, enabling it to accurately classify data or predict outcomes. Throughout the training process, the model continually adjusts its weights until it achieves high accuracy in making predictions (Anon., 2023b).

2.6 Classification

Classification is a type of supervised learning. Classification can be described as identifying, comprehending, and categorising objects and concepts into predefined groups, often called "sub-populations. Machine learning programmes employ various algorithms and utilise pre-categorised training datasets to classify upcoming datasets into appropriate and meaningful categories (Banoula, 2023).

A classification algorithm trains models using input sample data to predict the probability or likelihood that incoming data will belong to predefined categories. A widely recognized application of classification is the filtering of emails into "spam" or "non-spam" categories, a functionality commonly employed by leading email service providers today (Banoula, 2023).

In this study, there are two classification tasks: binary classification, which classifies the data into two categories, and multi-class classification, which classifies the data into three categories. Defect Type 1 and Defect Type 2 will be predicted with binary classification, while Defect Type 2 will be predicted with multi-class classification.

2.7 Deep Learning

Deep learning algorithms are considered a more advanced and mathematically intricate version of machine learning algorithms. In recent times, this field of research has gained much attention, and for valid reasons; the recent developments have generated results that were once thought to be unachievable (Wolfewicz, 2023). Deep learning algorithms are a more sophisticated and intricate machine learning form that employs a logical structure similar to human reasoning to analyse data. They can use both supervised and unsupervised learning techniques. Deep learning algorithms utilise supervised and unsupervised learning approaches and are more sophisticated versions of machine learning. They incorporate a layered structure of algorithms, an artificial neural networks (ANN), which are modelled on the biological neural network of the human brain. This results in a learning process that is much more powerful than traditional machine learning models (Wolfewicz, 2023).

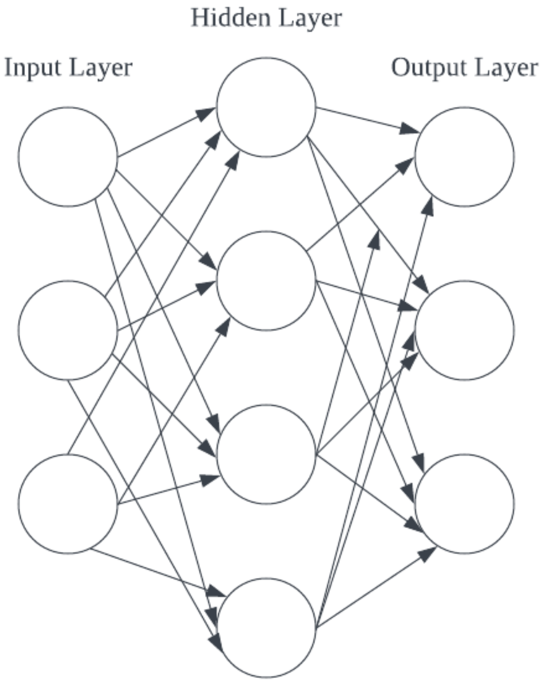


Figure 2.7.1: A simple Artificial Neural Network (ANN)

As shown in Figure 2.7.1, an artificial neural network (ANN) consists of an input layer, an output layer, and hidden layers in between. The hidden layers are not visible or directly accessible in the training set. A neural network with more hidden layers is considered "deeper". Typically, a neural network with two or more hidden layers is considered deep (Wolfewicz, 2023). The ANN initially performs a feature identification process, which involves identifying particular structures in an input. The first hidden layer of the network may learn how to detect edges, while the subsequent layers may learn how to differentiate colours and so forth until the final layer learns how to detect more intricate shapes relevant to the object being recognised. Basically, the earlier hidden layers learn simple features like edges while progressively learning task-specific features such as the shape of the LED while moving to later layers. Finally, the output layer processes the high-level features extracted from previous layers to perform classification or regression tasks. During training with input data, the deep learning algorithm learns from its errors to determine whether its prediction is accurate or needs adjustment (Wolfewicz, 2023).

As ANN slowly became more popular, researchers began researching building deeper neural networks to solve more complicated tasks. One of the most popular deep neural networks invented was the Convolutional Neural Network (CNN). A typical CNN consists of multiple layers: a convolutional layer, a non-linearity layer, a pooling layer, and a fully-connected layer (Bayat et al., n.d.). CNN has gained significant popularity in computer vision and natural language processing. CNNs tackle the inefficiency of fully connecting input pixels to neurons in traditional neural networks due to the large number of weight parameters required. CNNs employ local connections, meaning neurons in one layer are connected only to a corresponding local region in the previous layer. This drastically reduces the number of weight connections required. Additionally, CNNs often employ weight sharing, where the same weights are used for local connections across different neurons in the next layer. These two simplifications significantly reduce the number of parameters while enabling the network to detect and recognise features regardless of their position in the image. This process is akin to sliding a filter over the input image and mapping the results to the next layer, which is why these networks are termed convolutions (Bayat et al., n.d.). To improve the efficiency of the method, convolutional layers were stacked with each layer associated with different layers in which they can extract different features from the image.

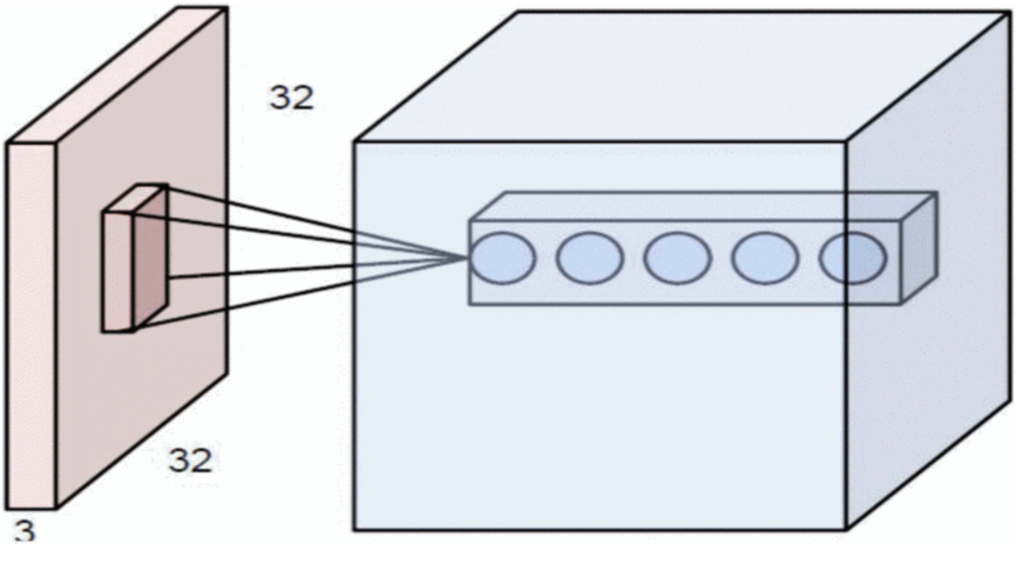


Figure 2.7.2: Multiple layers where each of them corresponds to a different filter but looking at the same region in the given image (Bayat et al., n.d.)

"Stride" can be applied to the convolution layer, where it refers to how the convolutional filter moves across the input image. Stride refers to how the convolutional filter moves across the input image. In the example provided in Figure 2.x, a 7x7 image is used as an illustration. When the filter is moved one node at a time, it results in a 5x5 output. Importantly, there is an overlap between the output regions. However, the stride can be manipulated. For example, if the stride is set to 2, the filter moves every two nodes, resulting in a 3x3 output. This not only affects the overlap but also reduces the size of the output (Bayat et al., n.d.).

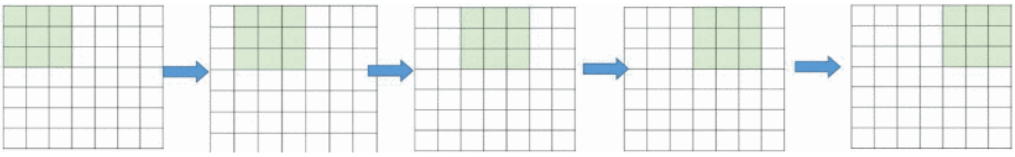


Figure 2.7.3: Movement of Filter Window with stride "1" (Bayat et al., n.d.)

Equation (1) is presented to formalise this relationship, where N denoting the dimension of input image, F represents the size of the filter, and S represents the stride.

$$O = 1 + \frac{N-F}{S} \tag{2.7.1}$$

The next layer after convolution is the non-linearity layer. This layer introduces non-linearity into the network and adjusts or controls the generated output. Non-linearity refers to a property where changes in input do not result in directly proportional or additive changes in output. Traditionally, sigmoid and tanh functions were commonly used as non-linearities in neural networks. However, in recent years, the Rectified Linear Unit (ReLU) has gained popularity (Bayat et al., n.d.). ReLU can be defined as:

$$ReLU(x) = \max(0, x) \tag{2.7.2}$$

$$\frac{d}{dx}(x) = \{1\ if\ x > 0; 0\ otherwise\} \tag{2.7.3}$$

where

x = input value to ReLu function

The pooling layer served as a down-sampling technique to reduce the complexity for further layers. It is important to note that pooling does not alter the number of filters but rather focuses on reducing the spatial dimensions of the data. Among various pooling methods, max-pooling is one of the most commonly used. Pooling, particularly max-pooling, is a technique in image processing that reduces spatial dimensions while retaining important features, and its parameters, like filter size and stride, can be adjusted to suit specific requirements (Bayat et al., n.d.).

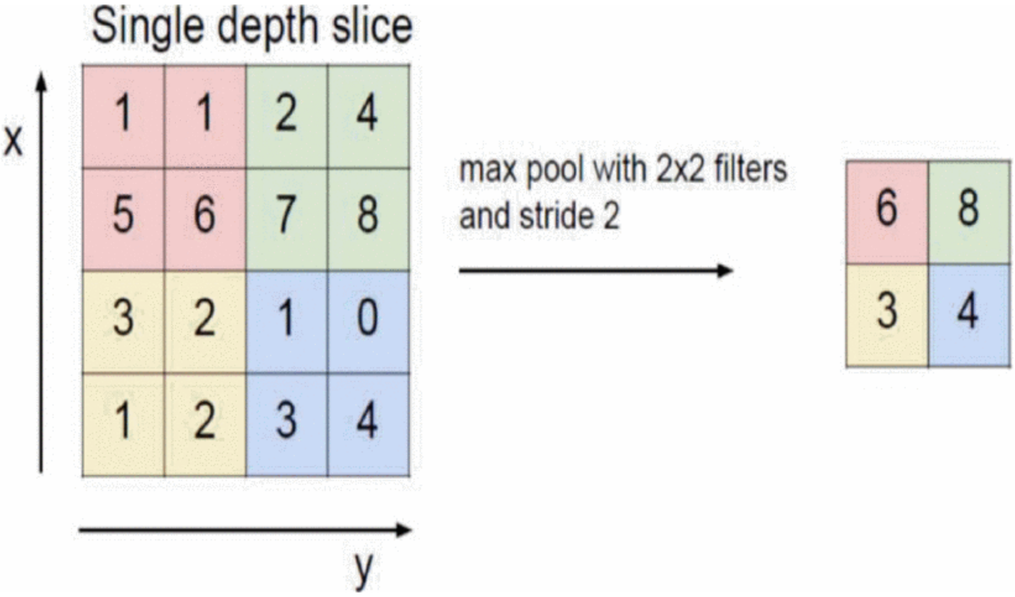


Figure 2.7.4: Demonstration of Max Pooling with 2x2 filters (Bayat et al., n.d.)

Finally, the fully connected layer in CNN works like traditional neural networks. In this layer, each node is directly connected to every node in the preceding and subsequent layers. This means that each node in the last pooling layer is connected to every node in the first layer of the fully connected layer. The fully-connected layer typically contains the highest number of parameters within a CNN, and training it can be time-consuming. In essence, the core of a CNN lies in its convolutional operations, and the introduction of nonlinearity and pooling layers significantly contributes to its effectiveness (Bayat et al., n.d.).

2.8 Transfer Learning

This study used transfer learning, a deep learning technique that requires less training data but can still achieve high performance. In transfer learning, a pre-trained model is used as the foundation for a new task. This approach can perform significantly better than training a model from scratch. Transfer learning is a widely used technique, and it is not common to train a model for image processing tasks from scratch nowadays. Conventional machine learning models need to be trained from the beginning, which is time-consuming and computationally expensive. Moreover, a considerable amount of data is required to achieve high performance. In contrast to traditional machine learning, transfer learning is a more efficient approach that utilises knowledge gained from pre-existing models to improve performance and achieve better results using smaller datasets (Baheti, 2023). Transfer learning involves leveraging knowledge learned from a pre-existing model rather than starting the training process from scratch to improve performance and speed up the learning process. In contrast, traditional machine learning models are typically trained independently and require enormous data and heavy computational resources to achieve comparable performance (Baheti, 2023). Transfer learning is faster than training neural networks from scratch because models that use features and weights from pre-trained models already understand the underlying features, which speeds up the training process (Baheti, 2023).

In the context of computer vision, transfer learning works by freezing the hidden layers of a deep learning model, and we replace the output layer for our specific task. It leverages the trained weights on the hidden layers, which were already trained to extract features from the image (Donges, 2022).

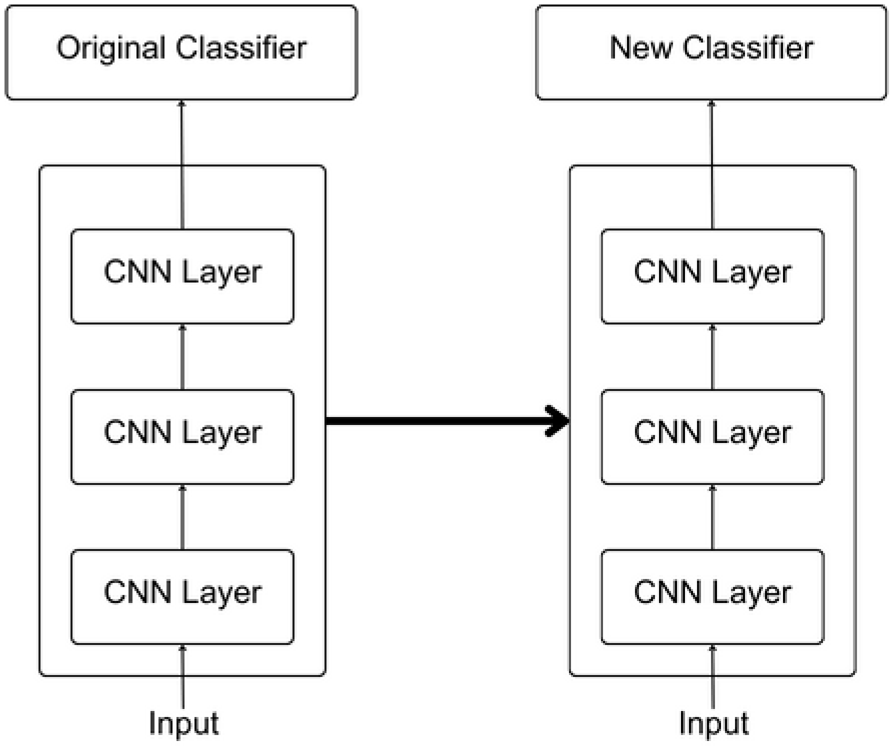


Figure 2.8.1: Visualisation of Tranfer Learning (Donges, 2022)

As shown in Figure 2.8.1, the CNN layers were trained on their original task, and the trained layers will not be modified or trained on the new task; instead, only the output layer will be changed according to the task. In this case, transfer learning allows us to leverage the trained layers from the base model, and we only have to train the new output layer for our task. The frozen layers act as the feature extraction layers, and the replaced output layer will be used to perform classification tasks in our project.

2.9 Pre-trained Image Classification Deep Learning Models

This study planned to use pre-trained image classifiers from Keras based on the ImageNet dataset. These models can classify input images into 1,000 object categories with high accuracy. Additionally, the models generalise well to other images other than the ImageNet dataset through transfer learning techniques (Rosebrock, 2017). A few available pre-trained image classifiers from Keras will be discussed, including VGG16, VGG19, ResNet50, InceptionV3, and Xception (Rosebrock, 2017).

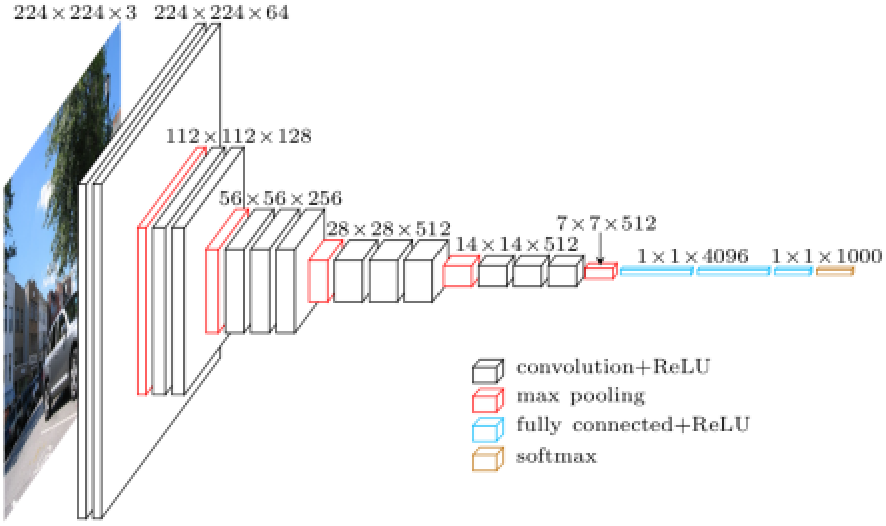


Figure 2.9.1: VGG architecture visualization (Rosebrock, 2017)

(Simonyan and Zisserman, 2014) introduced the VGG network architecture in their paper. The network architecture includes multiple 33 convolutional layers arranged in increasing depth. To decrease the size of the volume, max pooling is used. The architecture also comprises two fully connected layers, each with 4,096 nodes. Finally, a softmax classifier is applied (Simonyan and Zisserman, 2014). The naming convention of VGG is based on its number of layers. For example, VGG16 has 16 layers, while VGG19 has 19 layers. Despite their high accuracy in image classification, these models have some limitations, such as being slow to train and having heavy network architecture weights that require significant bandwidth (Rosebrock, 2017).

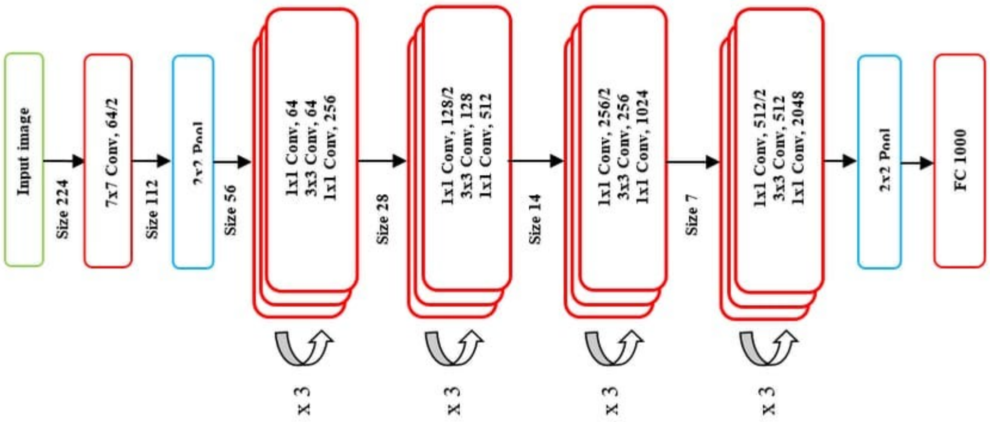


Figure 2.9.2: ResNet-50 Architecture (Mukherjee, 2022)

He et al. (2015) introduced the ResNet network architecture in their paper, "Deep Residual Learning for Image Recognition". ResNet is considered an unusual type of architecture that employs microarchitecture modules. Micro-architecture refers to the individual "building blocks" that are used to create the network. A combination of these building blocks, along with the standard layers like convolution and pooling, make up the overall macro-architecture of the network. Compared to VGG16 and VGG19, ResNet50 has a smaller size because it uses global average pooling instead of fully connected layers (Rosebrock, 2017).

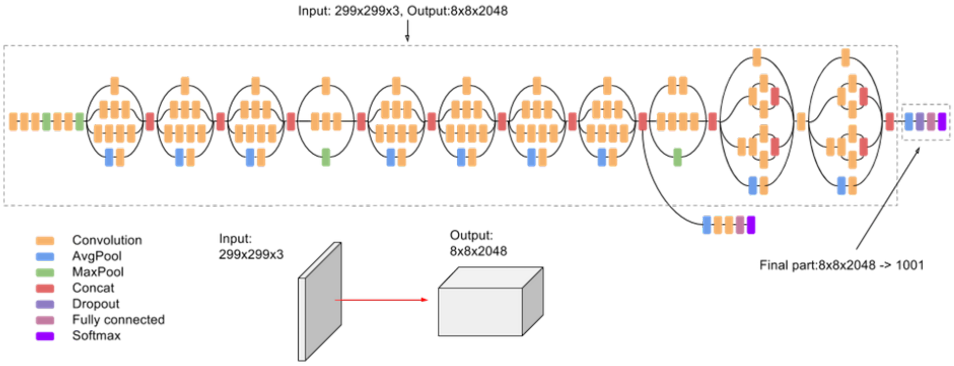


Figure 2.9.3: InceptionV3 architecture (T, 2023)

Szegedy et al. (2014) first introduced "Inception" micro-architecture in their paper "Going Deeper with Convolutions". Then, they proposed the improved InceptionV3 architecture in their next paper, "Rethinking the Inception Architecture for Computer Vision," in 2015. The inception module is designed to extract features at various levels by using 1x1, 3x3, and 5x5 convolutions in a single module. The resulting outputs are then combined and passed on to the next network layer. Google originally introduced this architecture as GoogLeNet, and later versions were named Inception vN, where N denotes the version number released by Google (Rosebrock, 2017).

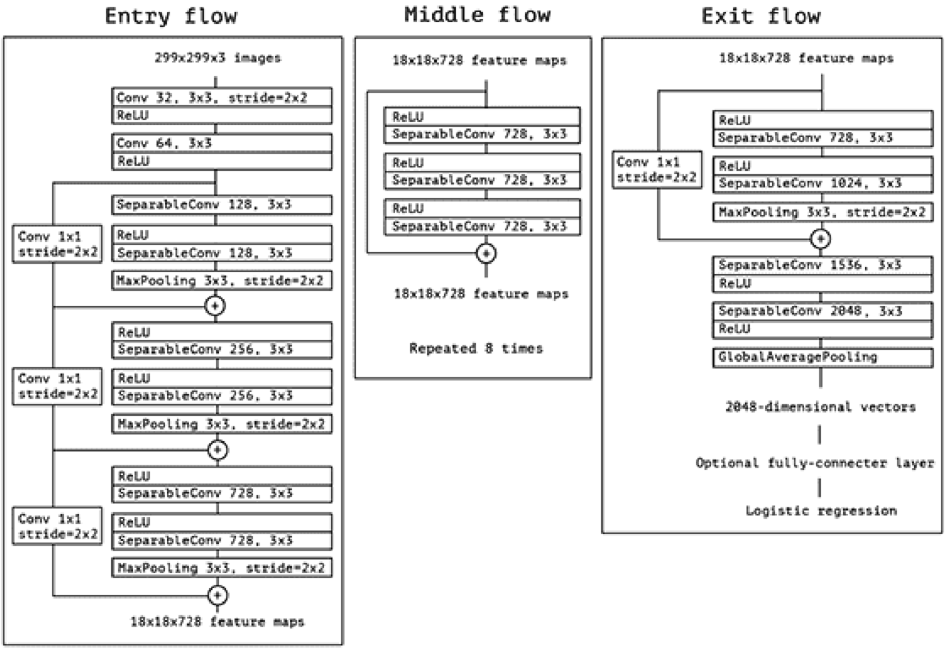


Figure 2.9.4: Xception architecture (Rosebrock, 2017)

Chollet (2016) proposed Xception, which is a type of architecture that builds on Inception. Instead of using standard Inception modules, it uses depth-wise separable convolutions. This process divides the convolution operation into two stages: the first is depthwise convolution, which individually applies a filter to each input channel, and the second is pointwise convolution, which amalgamates the results from the depthwise convolution through a 1x1 convolution. It has a small model size of 91 MB (Rosebrock, 2017).

Table 2.9.1: Comparison of models based on ImageNet dataset (Papers With Code, n.d.)

Pre-trained Model	**Approximate Model Size (MB)**	**Model top-5 accuracy on 1000 ImageNet classes**	**Number of Parameters**	**Advantages**	**Disadvantages**
VGG16	528	91.1%	138M	- Simple architecture which leads to simple fine-tune and implementation - Easy to understand and interpret	- Deep architecture which makes it computationally expensive - Requires a lot of memory - Large number of parameters
ResNet50	102	92.2%	23M	- Deep architecture which allows learning of complex features - Residual connections help alleviate the vanishing gradient problem - Good performance on large datasets	- Deep architecture which makes error detection difficult
InceptionV3	96	96.9%	25M	- Designed to be computationally efficient - Good performance on large datasets - Relatively low number of parameters	- Complex architecture which makes it difficult to understand and interpret - Comparatively poor performance on small datasets
Xception	91	96.5%	22.8M	- Designed to be even more computationally efficient than InceptionV3 - Good performance on large datasets - Relatively low number of parameters	- Complex architecture which makes it difficult to understand and interpret - Comparatively poor performance on small datasets

Table 2.9.1 compares the pre-trained models based on their performance on the ImageNet dataset. The performance results are obtained from the Papers With Code website based on published journal articles. The top-5 accuracy measures the number of times the correct label is included among the five most probable predictions made by the network (Papers With Code, n.d.).

Through the comparison of the pre-trained models, this study adapted three deep learning models: (i) ResNet50, (ii) InceptionV3, and (iii) Xception for epoxy-related defect detection on PCBs of wireless earbuds.

ResNet50 has a deep network architecture that allows the learning of complex features in data by organising multiple layers of processing units. As the depth of the network increases, it can learn and represent more abstract and complex features of the data. This is because each layer in a deep network extracts higher-level features from the previous layer's output, enabling the network to learn hierarchical representations of the input data. Additionally, deep networks can reduce the need for hand-engineered features as they can automatically learn relevant features directly from the data, saving time and resources in developing deep learning models (Mukherjee, 2022).

InceptionV3 and Xception are designed with computational efficiency in mind, achieved by reducing the number of computations required without compromising on accuracy. InceptionV3 utilises "Inception modules", allowing parallel computation of convolutional filters with varying kernel sizes. This approach results in a reduction in parameters and computations compared to traditional convolutional layers. Additionally, InceptionV3 incorporates batch normalisation to enhance generalisation and accelerate training (Szegedy et al., 2015).

Xception takes this concept further by using depthwise separable convolutions, combining depthwise and pointwise convolutions. Depthwise convolutions filter each input channel separately before being combined through pointwise convolutions, which perform 1x1 convolutions across all channels. This technique effectively reduces the number of computations while still capturing important features (Chollet, 2016).

In general, the major reason for choosing ResNet50, InceptionV3, and Xception for this study is their ability to achieve high accuracy while being computationally efficient. This makes them ideal for this study, which has limited computational resources and requires a small model size.

2.10 Hyperparameters

Hyperparameters can be described as special parameters that oversee and control the learning process, ultimately dictating the values of the model parameters learned by a machine learning algorithm. The term 'hyper_' is prefixed to emphasise that these are higher-level parameters responsible for governing the learning process and shaping the resulting model parameters (Nyuytiymbiy, 2020).

Hyperparameter values are defined before the model training process, and the values cannot be changed during training. Hyperparameters are not included in the final output of a model training process. We would not know the hyperparameter values used to train the model just by looking at it (Nyuytiymbiy, 2020).

Table 2.10.1 shows some common hyperparameters and their functions in model training.

Table 2.10.1: Hyperparameters Description

Hyperparameter	Description
Learning Rate	How fast the neural network adjusts its weight during training
Batch Size	Number of training examples in each iteration
Epoch	Number of times the training dataset updates the weights of neural network
Loss Function	Measures the difference between the predicted output of a model and the actual output
Activation Function	Introduce nonlinearity into the output of a neuron
Optimiser Algorithm	Find optimal set of weights and biases to minimise loss functions

CHAPTER 3
METHODOLOGY

3.1 Overview

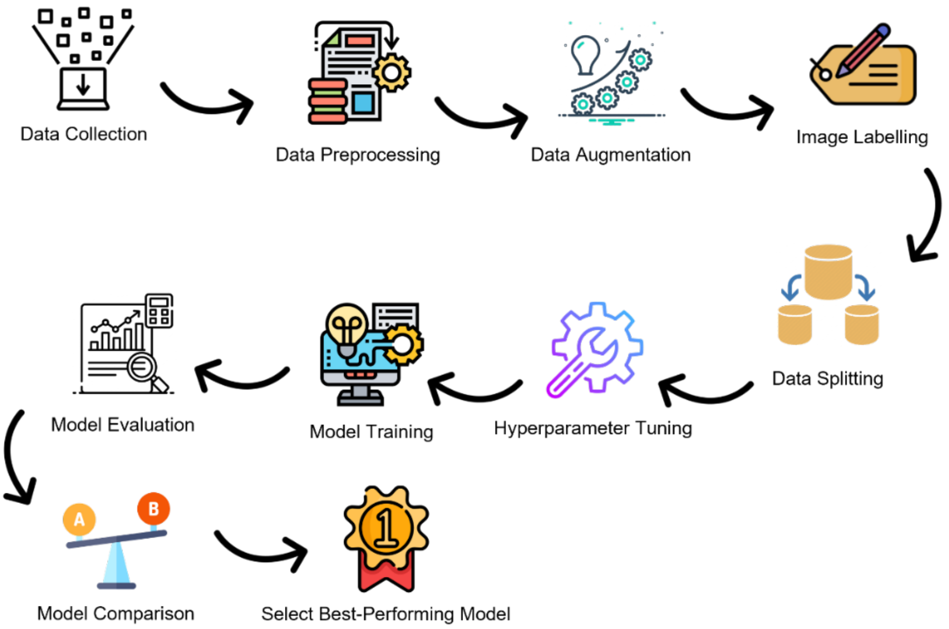


Figure 3.1.1: Project Workflow Summary

This study workflow is summarised in Figure 3.1.1. First, this study began by collecting a PCB image dataset of wireless earbuds provided by ASPL. The dataset underwent preprocessing, which included image segmentation to extract the region of interest from the images and data augmentation to generate new sample data using existing data. Next, the dataset was labelled according to its classes. After labelling, the dataset was divided into training, testing, and validation sets, ready for use in model training and evaluation. After preparing the dataset, hyperparameter tuning was performed to find the best hyperparameter combinations for model training. Then, the study trained the deep learning models and evaluated their performance in predicting the testing set. Following the evaluation, the study compared the models and selected the best-performing model as the proposed model.

The same workflow was applied in training ResNet50, Xception, and InceptionV3 for all three defect types. In this section, only a small segment of the original PCB images is allowed to be shown as they are protected by copyright and are confidential.

3.2 Project Plan

This study plan was displayed in the Work Breakdown Structure (WBS) to break down the works into smaller components for better understanding and management. Then, the project timeline was shown in a Gantt Chart in managing the project activities. The Gantt chart displayed the work completion date along with the duration taken to complete the activities. The project cost included the computational resources required for model development, such as hardware and software.

Work Breakdown Structure (WBS)

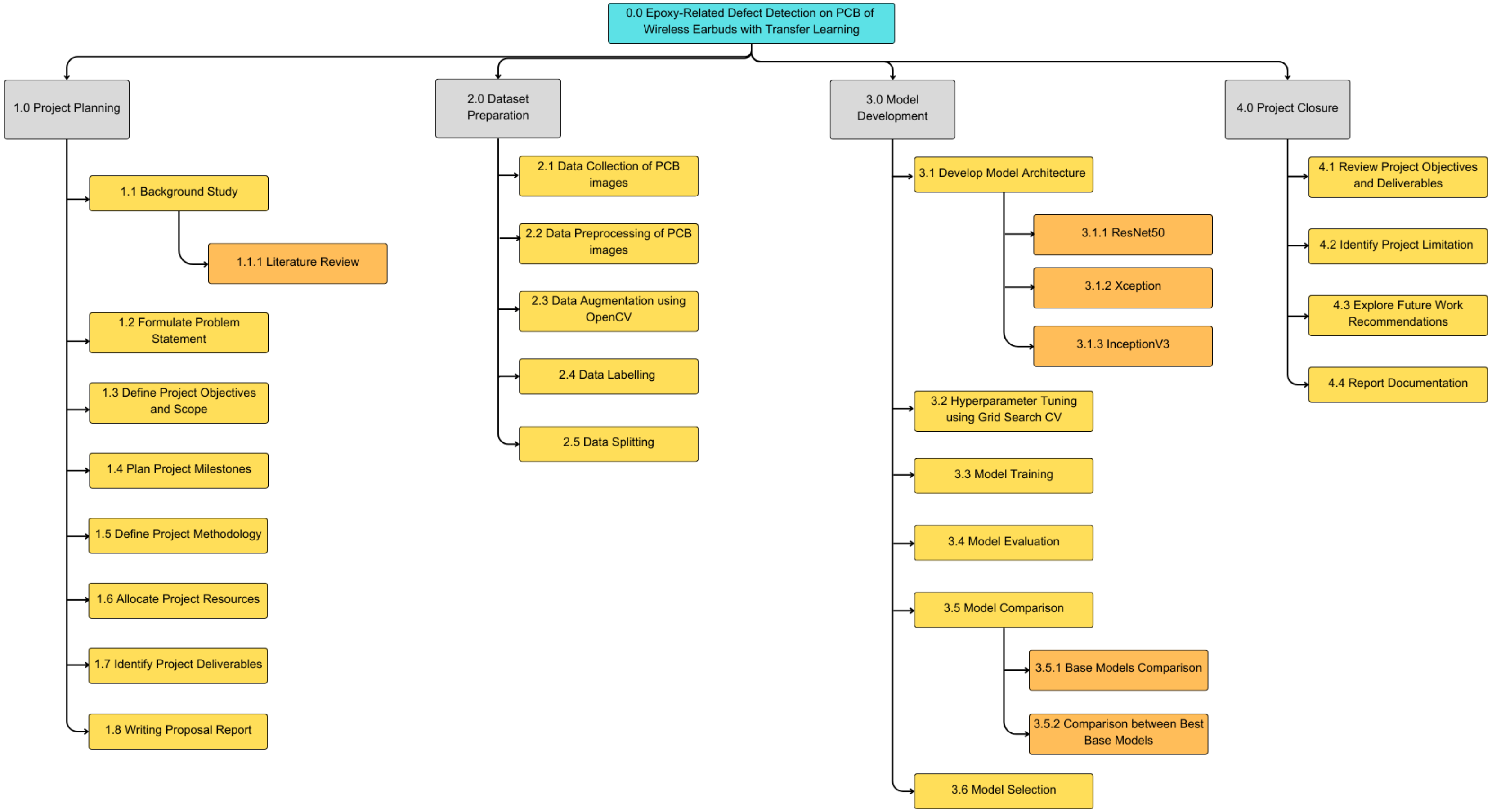
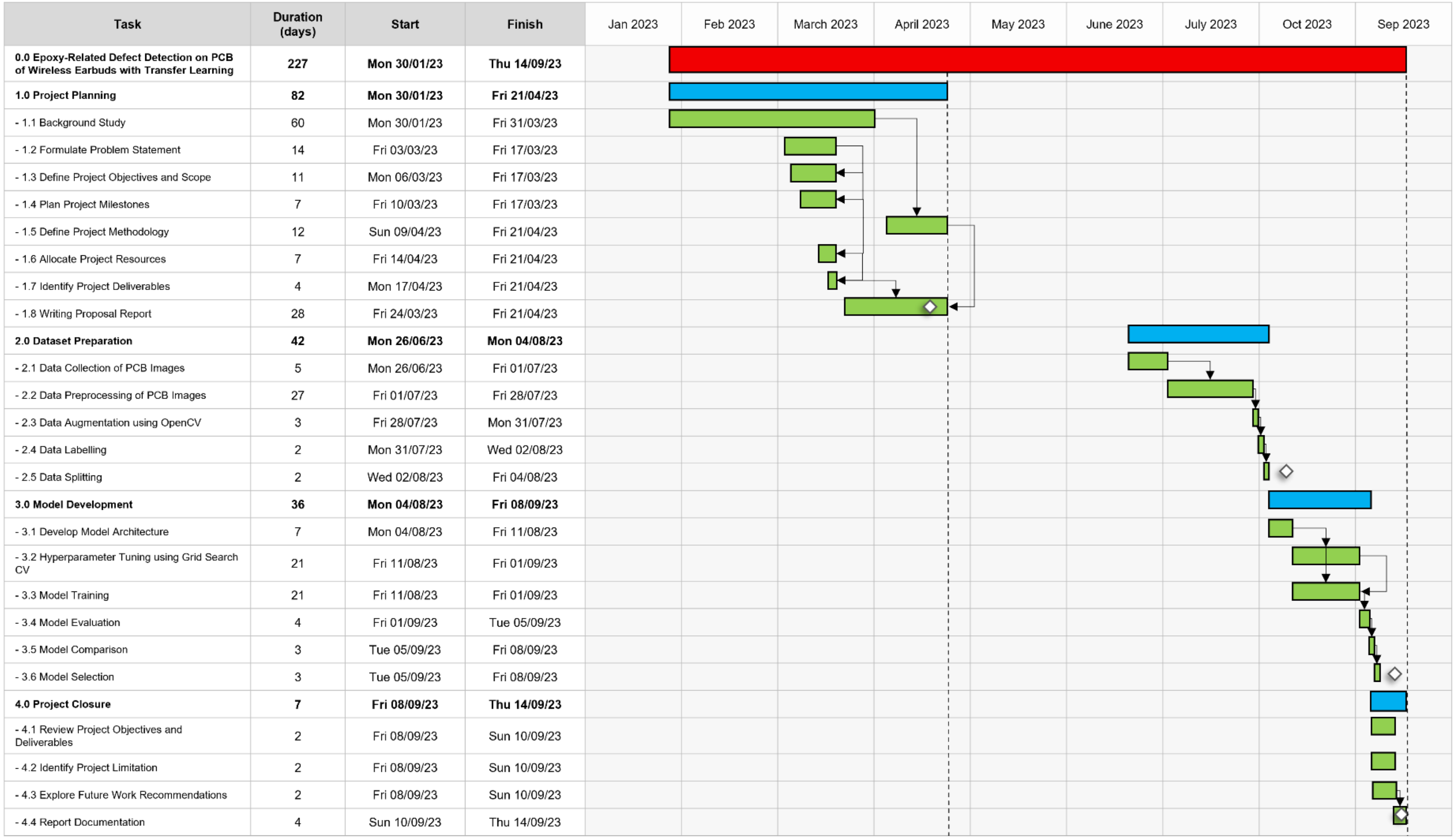

0.0 Epoxy-Related Defect Detection on PCB of Wireless Earbuds with Transfer Learning
1.0 Project Planning
1.1 Background Study
1.1.1 Literature Review
1.2 Formulate Problem Statement
1.3 Define Project Objectives and Scope
1.4 Plan Project Milestones
1.5 Define Project Methodology
1.6 Allocate Project Resources
1.7 Identify Project Deliverables
1.8 Writing Proposal Report
2.0 Dataset Preparation
2.1 Data Collection of PCB images
2.2 Data Preprocessing of PCB images
2.3 Data Augmentation using OpenCV
2.4 Data Labelling
2.5 Data Splitting
3.0 Model Development
3.1 Develop Model Architecture
3.1.1 ResNet50
3.1.2 Xception
3.1.3 InceptionV3
3.2 Hyperparameter Tuning using Grid Search CV
3.3 Model Training
3.4 Model Evaluation
3.5 Model Comparison
3.5.1 Base Models Comparison
3.5.2 Comparison between Best Base Models
3.6 Model Selection
4.0 Project Closure
4.1 Review Project Objectives and Deliverables
4.2 Identify Project Limitation
4.3 Explore Future Work Recommendations
4.4 Report Documentation

Figure 3.2.1: Project Work Breakdown Structure (WBS)

Gantt Chart



Task	Duration (days)	Start	Finish
0.0 Epoxy-Related Defect Detection on PCB of Wireless Earbuds with Transfer Learning	**227**	**Mon 30/01/23**	**Thu 14/09/23**
1.0 Project Planning	**82**	**Mon 30/01/23**	**Fri 21/04/23**
- 1.1 Background Study	60	Mon 30/01/23	Fri 31/03/23
- 1.2 Formulate Problem Statement	14	Fri 03/03/23	Fri 17/03/23
- 1.3 Define Project Objectives and Scope	11	Mon 06/03/23	Fri 17/03/23
- 1.4 Plan Project Milestones	7	Fri 10/03/23	Fri 17/03/23
- 1.5 Define Project Methodology	12	Sun 09/04/23	Fri 21/04/23
- 1.6 Allocate Project Resources	7	Fri 14/04/23	Fri 21/04/23
- 1.7 Identify Project Deliverables	4	Mon 17/04/23	Fri 21/04/23
- 1.8 Writing Proposal Report	28	Fri 24/03/23	Fri 21/04/23
2.0 Dataset Preparation	**42**	**Mon 26/06/23**	**Mon 04/08/23**
- 2.1 Data Collection of PCB Images	5	Mon 26/06/23	Fri 01/07/23
- 2.2 Data Preprocessing of PCB Images	27	Fri 01/07/23	Fri 28/07/23
- 2.3 Data Augmentation using OpenCV	3	Fri 28/07/23	Mon 31/07/23
- 2.4 Data Labelling	2	Mon 31/07/23	Wed 02/08/23
- 2.5 Data Splitting	2	Wed 02/08/23	Fri 04/08/23
3.0 Model Development	**36**	**Mon 04/08/23**	**Fri 08/09/23**
- 3.1 Develop Model Architecture	7	Mon 04/08/23	Fri 11/08/23
- 3.2 Hyperparameter Tuning using Grid Search CV	21	Fri 11/08/23	Fri 01/09/23
- 3.3 Model Training	21	Fri 11/08/23	Fri 01/09/23
- 3.4 Model Evaluation	4	Fri 01/09/23	Tue 05/09/23
- 3.5 Model Comparison	3	Tue 05/09/23	Fri 08/09/23
- 3.6 Model Selection	3	Tue 05/09/23	Fri 08/09/23
4.0 Project Closure	**7**	**Fri 08/09/23**	**Thu 14/09/23**
- 4.1 Review Project Objectives and Deliverables	2	Fri 08/09/23	Sun 10/09/23
- 4.2 Identify Project Limitation	2	Fri 08/09/23	Sun 10/09/23
- 4.3 Explore Future Work Recommendations	2	Fri 08/09/23	Sun 10/09/23
- 4.4 Report Documentation	4	Sun 10/09/23	Thu 14/09/23

Figure 3.2.2: Project Gantt Chart

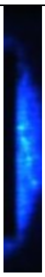
As shown in Figure 3.2.2, Project 1 primarily focused on the preparatory phase of the project. This phase involved defining essential project elements, such as the project scope, objectives, problem statement, methodology, and project deliverables. In Project 2, the project shifted its focus towards the technical aspect. Programming codes were written to prepare the dataset for model training and evaluation. Subsequently, the project advanced to the model development stage, where models were constructed, trained, evaluated, and compared. Hyperparameter tuning was carried out for each model using Grid Search CV.

Once the best-performing models were identified, the project moved on to the project closure phase. During this phase, conclusions were drawn, and the Project 2 report was completed.

3.3 Data Collection

This study collected a dataset of PCB images of wireless earbuds provided by ASPL. The dataset contains three images for each PCB: (i) the yellow channel, which displays the original colour; (ii) the blue channel, which shows dark field blue light; and (iii) the red channel, which displays dark field white light. Each colour channel provides a distinct view of different defects on the PCB. In this study, only the blue channel images were used since they emphasised the epoxy on the PCBs. Table 3.3.1 shows the sample images of different colour channel.

Table 3.3.1: Sample images of different colour channels

Colour channel	Sample Image
Yellow	
Blue	
Red	

The collected images are real-world images produced during the PCB manufacturing process of wireless earbuds. Since the dataset represents a real-world dataset, it brings benefits to this study as it trains models on actual data, which can increase the generalisation and reliability of the models. Table 3.3.2 shows the number of PCB images in the dataset.

Table 3.3.2: Number of Images Collected

Type	Images	Number of images
Good	Good (Non-defective)	2433
Defect Type 1	Epoxy Overflow on Die	785
Defect Type 2	Epoxy Overflow on LED	217
Defect Type 3	Foreign Material (FM) on Die	111
	Epoxy on Die	11

3.4 Data Preprocessing

In the data preprocessing step, the dataset underwent image segmentation to extract the regions of interest from the images. Extracting the region of interest (ROI) from the image helps the model focus on the important features on the images, improving the model's accuracy and generalisation ability.

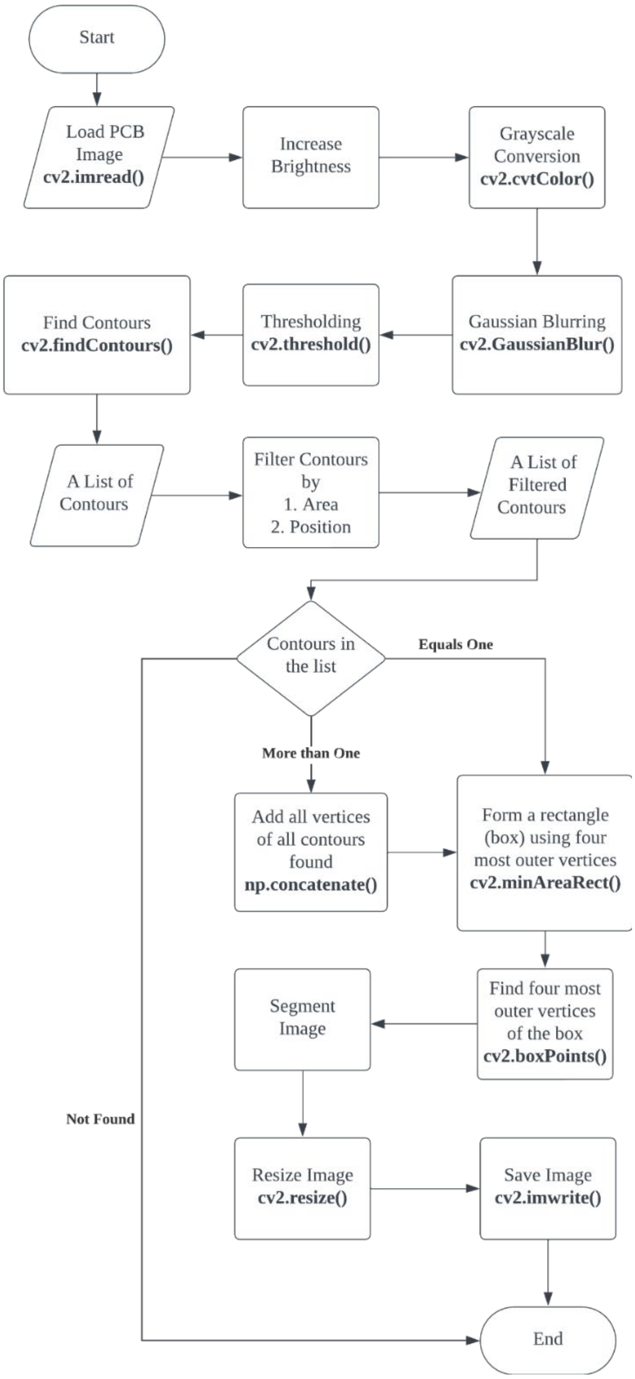
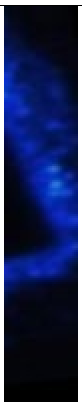
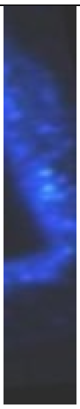

Start
Load PCB Image cv2.imread()
Increase Brightness
Grayscale Conversion cv2.cvtColor()
Gaussian Blurring cv2.GaussianBlur()
Thresholding cv2.threshold()
Find Contours cv2.findContours()
A List of Contours
Filter Contours by 1. Area 2. Position
A List of Filtered Contours
Contours in the list
Equals One
More than One
Not Found
Add all vertices of all contours found np.concatenate()
Form a rectangle (box) using four most outer vertices cv2.minAreaRect()
Find four most outer vertices of the box cv2.boxPoints()
Segment Image
Resize Image cv2.resize()
Save Image cv2.imwrite()
End

Figure 3.4.1: Image Segmentation Flowchart

All images in the dataset were looped using the approach shown in Figure 3.4.1 to segment the ROI from the PCB images. The image preprocessing techniques used were from OpenCV library.

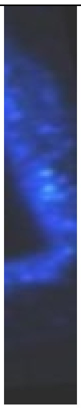
First, the PCB image was loaded using the **cv2.imread()** function. This function reads the image to be preprocessed. To begin the preprocessing, this study first increased the image's brightness by adding 45 to every pixel value. The purpose of increasing brightness was due to the dimmed original images in the dataset; increasing brightness helped to improve the quality of the following preprocessing steps. Table 3.4.1 shows the sample image before and after increasing brightness.

Table 3.4.1: Image Brightness Adjustment Comparison

Original Brightness	After Increasing Brightness
	

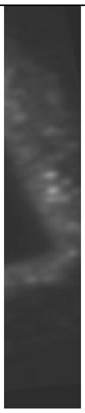
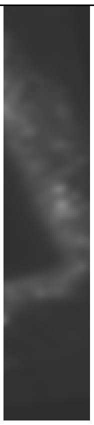
The image was then converted into a grayscale image using **cv2.cvt (image, cv2.COLOR_BGR2GRAY).** Converting an image into grayscale is a common practice in preprocessing because it simplifies the process. A coloured image has three channels (red, green, and blue), while a grayscale image has only one channel. A grayscale image also helps to intensify the contrast in pixel intensity between different regions and objects in the image. Table 3.4.2 showsthe sample image before and after applying grayscale conversion.

Table 3.4.2: Image Grayscale Comparison

Before Grayscale	After Grayscale
	

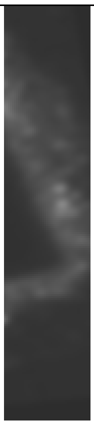
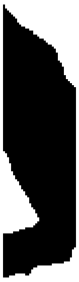
The image was further pre-processed by applying Gaussian blurring using **CV2. GaussianBlur(image, (7,7), 0)**. Gaussian blurring helps reduce the image's noise by helping the functions focus on important features. Besides, it enhances the performance of edge detection algorithms. Table 3.4.3 shows the sample image before and after applying Gaussian blurring.

Table 3.4.3: Image Gaussian Blurring Comparison

Before Gaussian Blurring	After Gaussian Blurring
	

Thresholding is a technique for turning every pixel to white or black depending on the pixel threshold. Thresholding was applied to the image using **cv2.threshold(image, 55, 255, cv2.THRESH_BINARY_INV) [1]**. For every pixel in the image, the pixel value below 55 is turned into 0, while the pixel above 55 is turned into 255, resulting in a black-and-white image. Thresholding improves the performance of edge detection algorithms as it helps to intensify edges. Table 3.4.4 shows the sample image before and after applying threshold. The image showed distinct differences between regions after applying the threshold.

Table 3.4.4: Image Thresholding Comparison

Before Thresholding	After Thresholding
	

After all the above steps, the black-and-white image was ready for image segmentation. The epoxy area had already turned black and the surrounding area had turned white. This study used **cv2.findContours(image, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)** to locate all the contours found in the image. The function output was a list of contours found within the images. A contour is a connected region on the image. The list of contours was sorted in descending order and looped to filter out the important contours, which in this case were PCB components such as LEDs or dies and regions of epoxy-related defects such as areas of epoxy overflow.

The list of contours found was looped to filter out the unwanted contours. In the loop, the contour area was calculated using **cv2.contourArea(contour)**. The function returned the area of the contour provided. Then, the position of the centre point of the contour was calculated using **cv2.moments(contour)**. The function returned the middle point as the x-coordinate and y-coordinate of the whole contour. After successfully obtaining the middle point and area of the contour, the contour was filtered by comparing the values obtained to the threshold defined. The threshold to filter the contours was obtained by the trial-

and-error method. Table 3.4.5 shows the threshold values used to filter the contours.

Table 3.4.5: Contours Filtering Criteria

Component	Position	Area
Die	160 <= x-coordinate <= 640 160 <= y-coordinate <= 630	Contour area <= 110000.
LED	160 <= x-coordinate <= 630 600 <= y-coordinate <= 1150	Contour area <= 150000.

Once the list of contours found on the image was filtered, the remaining list of contours consisted of only the interested objects, such as the PCB component and the area of defects. For the list of contours, a conditional statement to check on the number of remaining contours in the list was applied.

If only one contour remained, it indicated that the contour represented the main PCB component with the surrounding epoxy that was connected, such as a region of die surrounded by epoxy.

If the remaining number of contours was more than one, it indicated that the contours contained some small contours of epoxy that were not connected to the main PCB component, such as epoxy spillage.

If no contour remained, it indicated that no contours fulfilled the filtering criteria, and thus, the image was not pre-processed and not used in model training.

The remaining contours formed a rectangle box of regions on the image using the four most outer vertices in the contour or the list of contours with **cv2.minAreaRect(contour).** Once the rectangle box was formed, the four most outer vertices of the rectangle box were found using **cv2.boxPoints(box).** For the case where the number of contours was more than one, the vertices obtained after forming the rectangle box were stored in an array. The list of arrays containing vertices of all contours was then used to locate the four most outward vertices.

The four vertices obtained were used to segment the region of interest, including the major PCB component and the epoxy and defects region. This step used a custom module provided by ASPL to segment the image region from the original image and rotate it back to a bird's-eye view.R

Finally, the segmented image was resized to the size of 299x299 pixels by using **cv2.resize(image, (299, 299)).** This step ensured that all the pre-processed images had the same size, which the model training process required.

By the end of the image segmentation phase, all the successfully segmented images contained only the region of interest, which was the PCB component of either a die or an LED and epoxy connected to or around the component. These images were ready to be used in model training and validation processes.

Table 3.4.6: Number of PCB Images after Image Segmentation

Type	Images	Number of un-processed images	Number of images
Good	Good (Non-defective)	0	2433
Defect Type 1	Epoxy Overflow on Die	11	774
Defect Type 2	Epoxy Overflow on LED	2	215
Defect Type 3	Foreign Material (FM) on Die	0	111
	Epoxy on Die	0	11

Referring to Table 3.4.6, the collected dataset faced one common obstacle when dealing with real-world data: class imbalance. Class imbalance happens when the number of images used to train the model in each class differs greatly. For instance, ‘Defect Type 1’ has only 774 images compared to ‘Good's 2433 images. The other two defect types faced a more serious class imbalance problem where the difference between the number of images in each class was even bigger.

Class imbalance brings several disadvantages to the model training process, such as bias towards the majority class, as the majority class has more sample data to train the model on, and the model will perform better in the majority class. Then, an imbalanced dataset can lead to poor generalisation performance as the model struggles to recognise and correctly identify instances from the minority class. Lastly, when the minority class is underrepresented, the model may not learn sufficient information about it, potentially missing important patterns or characteristics of that class.

3.5 Data Augmentation

Due to the class imbalance problem in the dataset, data augmentation was applied to the dataset to improve the quality of the model training process. Data augmentation refers to generating new training samples by randomly applying transformations to the dataset's original images. This helped make the dataset larger and more diverse and reduce the impact of the class imbalance problem in the PCB dataset. Besides, data augmentation helped to improve model robustness and reduce overfitting by introducing variability in the dataset. In short, data augmentation was used to increase the size of the dataset and improve the model's generalisation ability by reducing the impact of the class imbalance issue.

Table 3.5.1: Data Augmentation Transformation

Transformation	Value Range
Rotation	-5 (anti-clockwise) to 5 (clockwise) degree
Increase Brightness	Multiply by 1.2 to 1.3
Increase Contrast	Multiply by 0 to 0.5

The transformations were randomly applied to the images by generating a random value within the range stated in Table 3.5.1. This approach introduced variability in the dataset without using the same images for training purposes. By applying random transformations, the dataset was expanded with new samples that were different from the original images, which helped to improve the model generalisation while increasing the dataset size.

Table 3.5.2: Number of Images throughout Data Augmentation Process

Defect Type	Defect	Number of Original Images Used for Data Augmentation	Number of New Images Generated per Original Image	Number of New Image Generated
1	Epoxy Overflow on Die	500	1	500
2	Epoxy Overflow on LED	200	2	400
3	Epoxy on Die	11	5	55
	FM on Die	111	2	222

Table 3.5.2 shows the number of images throughout the data augmentation process. The choice of the number of original images used for data augmentation was justified by aiming to increase the number of images in the minority class, thereby reducing the impact of the class imbalance issue without oversampling the minority class. This decision was based on the fact that only basic transformations using OpenCV were applied during the data augmentation process, resulting in new sample images that were not significantly different from the original ones. By avoiding oversampling the minority class, this study prevented the model from overfitting during the training process.

Ultimately, the techniques used in data augmentation were mainly used to increase the sample size in minority classes to reduce the impact of class imbalance but were unable to completely tackle the issue. Table 3.5.3 shows the finalised number of images in the dataset.

Table 3.5.3: Total Number of Images after Data Augmentation

Type	Images	Number of Images Added	Total Number of Images
Good	Good (Non-defective)	0	2433
Defect Type 1	Epoxy Overflow on Die	500	1274
Defect Type 2	Epoxy Overflow on LED	400	615
Defect Type 3	Foreign Material (FM) on Die	222	333
	Epoxy on Die	55	66

3.6 Data Labelling

For defect types 1 and 2, the good images were labelled with a value of 0 and the defective images were labelled with 1. There are only two classes in which the binary classification will classify the images as 0 or 1. The samples were labelled with a single integer, either 0 or 1.

For defect type 3, the good images were labelled with value 0, 'Epoxy on Die' images were labelled with value 1, and 'FM on Die' images were labelled with value 2. The samples were labelled using one-hot labelling. For example, a good image that has a class value of 0 was labelled as (1, 0, 0), an 'epoxy' image that has a class value of 1 was labelled as (0, 1, 0), and an FM image that has a class value of 2 was labelled as (0, 0, 1). Encoding multi-class data using integer labels can introduce unintended ordinal relationships that do not exist in the original data. One-hot encoding eliminates this issue by providing binary columns, helping to reduce bias in the model.

3.7 Data Splitting

The generated augmented sample images were only used within the training set. This distinction arises from the necessity to maintain the validation and testing sets as repositories of authentic, real-world data. This strategic decision ensures that the evaluation of the model's performance relies solely on genuine data. This approach aligns with the ultimate goal of deploying the trained model for predictions on the novel, unseen real-world data, warranting that the validation and testing phases accurately simulate these real-world conditions and scenarios.

The dataset was split into three sets: training, testing, and validation, with a ratio of 70:15:15. Then, the images generated by data augmentation were all added to the training set.

During the process of training a machine learning model, the training set was used for training the model, the validation set was used for monitoring the performance of the model during the training process, and the testing set was used to assess the final performance of the trained model on new, unseen data.

The validation set helped to avoid a problem called overfitting. This happens when the model is too focused on the training data and cannot perform well on new data. By using the validation set to check the model's performance, the study can monitor the model's performance at every epoch by observing the validation loss and validation accuracy.

The dataset was split using the **train_test_split()** function in the scikit-learn library. The function helped split the data and corresponding labels into different sets for model training.

Table 3.7.1: Total Number of Images after Splitting

Defect Type	Training	Validation	Testing
1	2744 (2244 + 500)	482	481
2	2253 (1853 + 400)	398	397
3	1062 (785 + 277)	169	168

Table 3.7.1 shows the number of images in each split set. By successfully splitting the data into different sets, the study can proceed to model training and evaluation.

3.8 Hyperparameter Tuning

Hyperparameter tuning plays a crucial role in efficiently training deep learning models. Thus, Grid Search CV with 3-fold cross-validation was employed to identify the optimal hyperparameters for ResNet50, Xception, and InceptionV3. A 3-fold cross-validation is an approach where the algorithm splits the data used for grid search into three folds. It trains and evaluates a model three times, each time using a different fold as the validation set and the remaining data as the training set. The results will be the mean score of the three model evaluation process, which provides a more robust evaluation of the model's generalisation performance than a single train-test split.

The best combination of the number of epoch, learning rate value and optimiser were determined using Grid Search CV. Then, other hyperparameters like batch size were fixed due to resource constraints, and the loss function was the commonly used loss function in the respective classification tasks.

Table 3.8.1: Hyperparameters for Grid Search CV

Hyperparameters	Value
Epoch	• 10 • 20 • 50
Optimiser	• SGD • RMSprop • Nadam • Adam
Learning Rate	• 0.0001 • 0.001 • 0.01

This study defined a specific set of hyperparameters for use in Grid Search CV, which are detailed in Table 3.8.1. The choice of optimisers was made using a trial-and-error approach. The number of epochs was determined through a pilot study, which indicated that a small number of epochs sufficed thanks to the benefits of transfer learning. Additionally, a relatively small learning rate was selected to prevent excessive adjustments to the model's weights and biases during the training process. The learning rate controls the step size at which these updates occur.

Table 3.8.2: Other hyperparameters

Hyperparameters	Value
Loss Function	Binary-crossentropy (Binary Classification)
	Categorical-crossentropy (Multi-class Classification)
Batch Size	16

Tabl 3.8.2 shows the other hyperparameters used in model training. In this study, a total of nine grid searches were done, as each model required a hyperparameter tuning for each defect type. It is often necessary to perform a grid search on each dataset separately for each model. This is because the optimal hyperparameters may indeed differ across datasets due to the differences in data distribution, complexity, and other factors associated with each defect type. Tailoring hyperparameters to each specific dataset helps optimise model performance for the unique challenges posed by each dataset.

Aside from the common hyperparameter used for model training, this study tuned the decision threshold that was only required for binary classification tasks. The model's output in predicting a sample would result in a probability between 0 and 1. Thus, it is required to convert the probability value using a threshold value such as 0.5, where all values falling under 0.5 are mapped to 0, and all the other values are mapped to 1 (Brownlee, 2021). Using a value of 0.5 is a common practice for binary classification, but in cases where there is a severe imbalance in the distribution of the classes, the model will perform poorly on prediction. Due to the class imbalance issue in the dataset, this study incorporated decision threshold adjustments in the model evaluation phase for defect types 1 and 2 to evaluate the model’s performance on different decision thresholds. It was a straightforward and simple approach to improve the performance of classification models while facing an imbalanced classification problem (Brownlee, 2021).

3.9 Model Architecture

To fully explore the potential of transfer learning, this study decided to make the minimum modifications to the pre-trained models to retain their original architecture and trained weights.

ResNet50, Xception, and InceptionV3 models were imported from the Keras library. These models were loaded without the fully connected dense output layer, typically responsible for making predictions using trained weights from 'ImageNet.' Then, all of the hidden layers were frozen to prevent weight updates during the model training process. These frozen hidden layers worked as the feature extractor for our classification task.

Next, A GlobalAveragePooling2D layer was added to the model. GlobalAveragePooling2D is a technique commonly used in CNN for feature extraction and dimensionality reduction. In a CNN, after multiple convolutions and pooling layers, a set of feature maps were produced. Each feature map represents certain learned features in the input data. Traditional pooling layers diminish the spatial dimensions of the feature maps by selecting the highest or mean value within a small region., usually 2x2 or 3x3 and moving a filter over the feature map. This results in a downsampled representation of the features. In GlobalAveragePooling 2D, instead of using small regions and downsampled representations, it takes the average of the entire feature map. The average value of each feature map is calculated, resulting in a single value for each feature map. After applying a GlobalAveragePooling2D layer, the output will be a 1D vector of values that was connected to the fully connected layer for classification tasks.

Following that, a custom fully-connected dense layer was appended to the model. In the case of binary classification, the dense layer comprised a single neuron with a sigmoid activation function. Using a single neuron in the output layer with a sigmoid activation function is a common and effective approach for binary classification tasks, providing clear probabilistic outputs and maintaining model simplicity. The output of a sigmoid function is always between 0 and 1, combined with a decision threshold that maps any output values into distinct classes. The formula of a sigmoid function is:

$$f(x) = \frac{1}{1+e^{-(x)}} \quad (3.9.1)$$

where

x = input to the sigmoid function

e = the base of the natural logarithm, approximately equal to 2.71828.

For multi-class classification involving three classes to be predicted, the dense layer consisted of three neurons with a softmax activation function, with each neuron associated with each class. In multi-class classification, a common encoding scheme for the target labels is one-hot encoding. Each class is represented by a unique neuron, and the output for each sample should have a single high value (1) in the neuron corresponding to the true class and low values (0) in the other neurons. This setup ensures that each sample is assigned to only one class. A multi-class classification task usually requires a softmax activation function. Softmax converts the raw output scores into class probabilities, which the total probabilities across all classes equals 1 for each sample. The number of neurons in the output layer corresponds to the number of classes. The formula of a softmax function is:

$$\sigma(\underset{z}{\rightarrow})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad (3.9.2)$$

where

σ = softmax

$(\underset{z}{\rightarrow})$ = input vector

e^{z_i} = standard exponential function for input vector

K = number of classes in the multi-class classifier

e^{z_j} = standard exponential function for output vector

In summary, the number of neurons used in the output layer depends on the task's requirements. On binary classification, one neuron will output the value as a probability value, like 0.6, indicating the sample might belong to the positive class. On the other hand, three neurons are required for three-class classification, with each neuron representing one class. The output of the layer aligns with the one-hot labelling used in this study, where the output of a predicted sample will be (0.55, 0.4, 0.05), where the highest value indicates the prediction of the model; in this case, it is the class with value 0.

By adding the final dense layer to the models, they were ready for model training and evaluation.

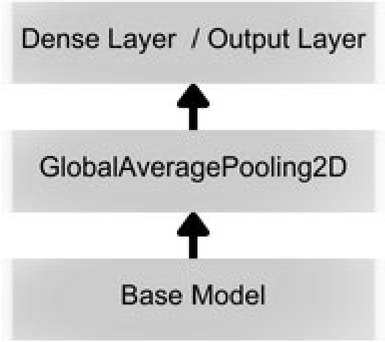


Figure 3.9.1: Model Architecture

Figure 3.9.1 shows the overall model architecture used in this study. Base models are ResNet50, Xception, or InceptionV3, with a pooling layer using GlobalAveragePooling2D and an output layer which is a fully connected layer that was trained on classification tasks.

3.10 Model Evaluation

A graph illustrating the training and validation losses was plotted to monitor the models' performance throughout the training phase. The study utilised the pyplot functions provided in the Matplotlib library to plot a line chart to visualise the information required. A line chart provides a simple and straightforward method to monitor losses during the training phase.

This study adopted the strategy of saving the model at every epoch, providing better control over the training process. This approach provided the flexibility to load the model from a particular epoch if indications of overfitting emerged. Overfitting is indicated by the rise in validation loss while training loss continues to decrease. The approach ensured that the model's generalisation capabilities are effectively managed and maintained.

Then, the results of the performance of the deep learning models on the testing set were evaluated using a confusion matrix, precision, recall, F1 score, and accuracy.

Table 3.10.1: Evaluation Metrics Description

Metric	Description
Confusion Matrix	A table that displays the number of correct and incorrect predictions made by the models on the given dataset.
Accuracy (Main)	Percentage of correctly predicted samples in the total dataset.
Precision	Percentage of correctly predicted positive samples out of all positive predictions.
Recall (Secondary)	Percentage of correctly predicted positive samples out of all actual positive samples.
F1 score	The harmonic mean of precision and recall, which combines both metrics into a single value.

Table 3.10.2: Confusion Matrix for Binary Classification

	Predicted Good	Predicted Defected
Actual Good	TN	FP
Actual Defected	FN	TP

where

True Positive (TP) = Correctly predicted a defected PCB

True Negative (TN)= Correct prediction a good PCB

False Negative (FN)= Incorrectly predicted a defected PCB as good PCB

False Positive (FP)= Incorrectly predicted a good PCB as defected PCB

Table 3.10.3: Confusion Matrix for Three-Class Classification

	Predicted Good	Predicted Epoxy	Predicted FM	
Actual Good	TP_1	a	b	$FN_1 = a + b$
Actual Epoxy	c	TP_2	d	$FN_2 = c + d$
Actual FM	e	f	TP_3	$FN_3 = e + f$
	$FP_1 = c + e$	$FP_2 = a + f$	$FP_3 = b + d$	

where

TP_1 = Correctly predicted good PCBs

TP_2 = Correctly predicted defected PCBs with 'Epoxy on Die' defect

TP_3 = Correctly predicted defected PCBs with 'FM on Die' defect

FP_1 = Incorrectly predicted samples as good PCB but the samples are from other two classes

FP_2 = Incorrectly predicted samples as PCB with 'Epoxy on Die' defect but the samples are from other two classes

FP_3 = Incorrectly predicted samples as PCB with 'FM on Die' defect but the samples are from other two classes

FN_1 = Incorrectly predicted samples as other two classes but the samples are good PCBs

FN_2 = Incorrectly predicted samples as other two classes but the samples are defected PCBs with 'Epoxy on Die' defect

FN_3 = Incorrectly predicted samples as other two classes but the samples are defected PCBs with 'FM on Die' defect

The formulas for accuracy, weighted recall, weighted precision, and weighted F1-score are:

$$accuracy = \frac{TP1+TP2+TP3}{TP1+TP2+TP3+FN1+FN2+FN3+FP1+FP2+FP3} \quad (3.10.1)$$

$$weighted\ recall = \frac{\sum_{i=1}^{3} TPi}{\sum_{i=1}^{3}(TPi+FNi)} \quad (3.10.2)$$

$$weighted\ precision = \frac{\sum_{i=1}^{3} TPi}{\sum_{i=1}^{3}(TPi+FPi)} \quad (3.\ 10.3)$$

$$weighted\ F1 - Score = \frac{2(weighted\ precision * weighted\ recall)}{weighted\ precision + weighted\ recall} \quad (3.\ 10.4)$$

where

TP_i = True Positives for class i

TN_i = True Negatives for class i

FP_i = False Positives for class i

i = 1,2,3

During model evaluation, the confusion matrix showed the prediction on each testing sample, providing a better understanding of the model's performance. Confusion matrix is a useful tools to help visualising the outcome of the prediction which can assist the study in analysing and understanding the model’s behaviour. The study used the heatmap function in Seaborn library to create the confusion matrix.

The results obtained through confusion matrix were used to calculate the accuracy, weighted recall, weighted precision, and weighted F1-score. Weighted metrics were used due to the class imbalance issues in the dataset, in which the calculated metrics for each class were assigned a weight to them to emphasise the minority classes without being overwhelmed by the majority class. Accuracy served as the primary metric, reflecting the overall correctness of predictions, and recall served as the secondary metric. While accuracy measured the overall percentage of correct predictions, recall specifically assessed the model's capability to accurately identify positive instances within a dataset, which in our context are the defective images, thereby emphasising its performance in capturing all relevant information. This dual evaluation approach ensured a balanced assessment of our model's effectiveness, particularly in tasks where the correct identification of specific instances was paramount.

3.11 Model Comparison

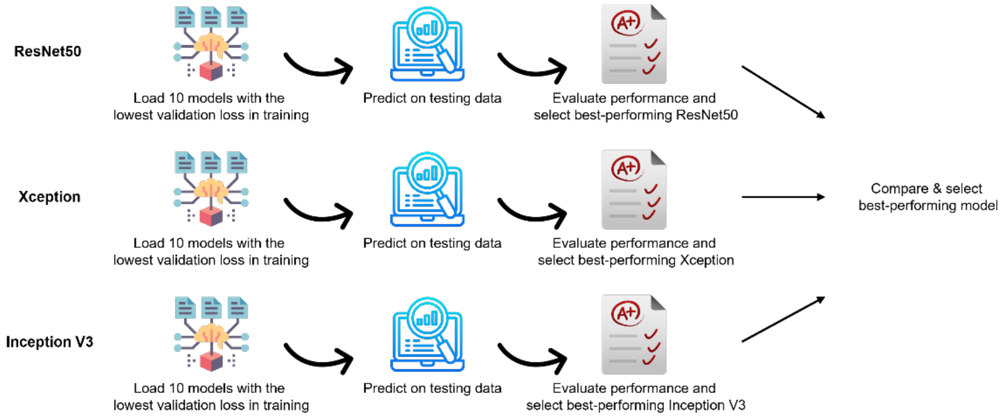

ResNet50
Load 10 models with the lowest validation loss in training
Predict on testing data
Evaluate performance and select best-performing ResNet50
Xception
Load 10 models with the lowest validation loss in training
Predict on testing data
Evaluate performance and select best-performing Xception
Compare & select best-performing model
Inception V3
Load 10 models with the lowest validation loss in training
Predict on testing data
Evaluate performance and select best-performing Inception V3

Figure 3.11.1: Workflow Model Evaluation and Comparison

The approach shown in Figure 3.11.1 was used to select the best-performing model of the same base model. The study loaded ten models with the lowest validation loss during the training and had them predict the testing data. The model that achieved the highest accuracy was selected for final model comparison.

For final model comparison, the study compared the best-performing ResNet50, best-performing Xception, and best-performing InceptionV3 and determined the model with the highest accuracy as the proposed model for the particular defect type.

3.12 Software Environment

The programming language used in this study was Python 3. This study used ResNet50, InceptionV3, and Xception, which are pre-trained deep learning models available in the Keras deep learning library.

This study utilised Compute Unified Device Architecture (CUDA) for model training. Using CUDA for this study's model training significantly improved performance and efficiency. CUDA, developed by NVIDIA, allowed the study to leverage the computational power of GPU (Graphics Processing Unit) for our deep learning tasks. This choice accelerated our model training process, resulting in faster training times than traditional CPUs. The study carefully selected a compatible NVIDIA GPU and ensured seamless integration with the deep learning frameworks, such as TensorFlow. This strategic decision led to more efficient model training and optimised resource utilisation.

The integrated development environment (IDE) used was Spyder and Jupyter Notebook. Spyder was used to write code for data preprocessing, data augmentation, data labelling, and data splitting. Spyder was used due to its support of the IPython console, which allowed for interactive computing and data exploration. Spyder allowed the study to execute code line by line, view results, and inspect variables in real time. This feature was very helpful in preparing the data for model training. On the other hand, Jupyter Notebook was used for hyperparameter tuning, model evaluation, and model comparison. Jupyter Notebook allowed the study to visualise the data easily and document the entire process, making it easier to take control of the model training process. With its documentation feature, the results could be referred back at anytime.

Table 3.12.1: Software Specifications

Package	**Version**
Python	3.9.17
Imutils	0.5.4
OpenCV	4.7.0
NumPy	1.24.2
TensorFlow	2.11.0
Matplotlib	3.7.0
Keras	2.10.0
Scikit-learn	1.3.0
Seaborn	0.12.2
Scikeras	0.11.0
Pandas	1.5.3

The versions of each Python library used are listed in Table 3.12.1. Imutils and OpenCV were responsible for image preprocessing in this study. These two libraries provide useful and convenient functions to simplify the image preprocessing step.

Meanwhile, NumPy and Pandas were required to manipulate data. NumPy provides a high-performance multidimensional array object called 'ndarray'. NumPy is valuable for numerical and array-based operations, making it essential for computer vision and machine learning tasks. On the other hand, Pandas is useful for handling structured data, making it a good choice for data labelling and organisation in these projects. These two libraries were used to save images and contours in an array for operations while also enabling me to label the data and export the labels into an Excel file.

Seaborn and Matplotlib were used to visualise graphs and the confusion matrix. TensorFlow, Scikit-learn, and Scikeras are libraries that provide convenient functions such as pre-trained deep learning models and data splitting, which were the main libraries that contributed to the success of the project.

3.13 Hardware Environment

The hardware used in this study was an Acer Nitro 5 AN515-57 gaming laptop. The hardware specifications are shown in Table 3.13.1.

Table 3.13.1: Hardware Specifications

Hardware Component	**Description**
CPU	11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz 2.69 GHz
GPU	NVIDIA GeForce RTX 3060
RAM	24GB DDR4 @ 2933MHz
Storage	2 x 512 GB NVMe SSD
Operating System	Windows 11 Pro 64-bit, x64 based processor

CHAPTER 4
RESULTS & DISCUSSION

4.1 Overview

The main goal of this study is to propose deep learning models that can detect epoxy-related defects on the PCB of wireless earbuds with high accuracy using a transfer learning approach. ResNet50, Xception, and InceptionV3 from the Keras library were selected for this defect detection task. First, hyperparameter tuning was done using Grid Search CV. Then, the hyperparameters obtained were used to train the deep learning models. After that, the performance of the deep learning models was evaluated by predicting the testing data set. Finally, the model that achieved the highest accuracy for each defect was selected. The main evaluation metric used is accuracy, followed by recall.

4.2 Grid Search Result

Hyperparameter tuning was done using Grid Search CV with three cross-validations. The grid search was mainly used to determine the best combination of hyperparameters, which will be used to train the models. In grid search, the learning rate (0.0001, 0.001, and 0.01), optimiser (SGD, RMSprop, Nadam, and Adam), and the number of epochs (10, 20, 50) were used for defect types 1 and 2. There were a total of 36 combinations of hyperparameters.

For Detect Type 3, some hyperparameter values were discarded to reduce the time needed for grid search. By filtering out hyperparameter values that always achieved a low mean score in the grid search of defect type 1 and defect type 2, learning rate (0.0001, 0.001, 0.01), optimiser (RMSprop, Nadam, and Adam), and number of epochs (20, 50) were used. There were a total of 18 combinations of hyperparameters.

4.2.1 Defect Type 1 – Epoxy Overflow on Die

1. **ResNet50**

Table 4.2.1: ResNet50 Grid Search Result on Defect Type 1

#	Epochs	Optimiser	Learning Rate	Mean Score
1	50	RMSprop	0.01	0.969591
2	50	Nadam	0.01	0.969201
3	20	Adam	0.01	0.967251
4	50	Adam	0.001	0.966082
5	50	Adam	0.01	0.966082
6	50	Nadam	0.001	0.965302
7	20	Nadam	0.01	0.964912
8	50	RMSprop	0.001	0.964522
9	20	RMSprop	0.01	0.963743
10	20	Adam	0.001	0.963353
11	20	RMSprop	0.001	0.962573
12	20	Nadam	0.001	0.962573
13	10	Adam	0.01	0.961793
14	10	Nadam	0.01	0.960234
15	10	Nadam	0.001	0.959844
16	10	RMSprop	0.001	0.959064
17	20	SGD	0.01	0.956335
18	10	RMSprop	0.01	0.953606
19	10	Adam	0.001	0.953606
20	10	SGD	0.01	0.953216
21	50	SGD	0.01	0.952437
22	50	Adam	0.0001	0.951267
23	50	RMSprop	0.0001	0.950487
24	50	Nadam	0.0001	0.949318
25	50	SGD	0.001	0.937232
26	20	RMSprop	0.0001	0.933723
27	20	Nadam	0.0001	0.930214
28	20	Adam	0.0001	0.928265

29	20	SGD	0.001	0.913450
30	10	RMSprop	0.0001	0.904094
31	10	Nadam	0.0001	0.901365
32	10	Adam	0.0001	0.892788
33	10	SGD	0.001	0.887329
34	50	SGD	0.0001	0.831969
35	10	SGD	0.0001	0.777388
36	20	SGD	0.0001	0.776218

2. Xception

Table 4.2.2: Xception Grid Search Result on Defect Type 1

#	Epochs	Optimiser	Learning Rate	Mean Score
1	50	Adam	0.001	0.958285
2	50	Nadam	0.001	0.957505
3	20	Adam	0.001	0.954386
4	50	Nadam	0.01	0.952827
5	50	RMSprop	0.001	0.950487
6	20	Nadam	0.001	0.950097
7	20	RMSprop	0.001	0.947758
8	50	Adam	0.01	0.945809
9	50	RMSprop	0.01	0.944639
10	10	Nadam	0.001	0.943860
11	20	Nadam	0.01	0.940741
12	20	Adam	0.01	0.938402
13	10	Adam	0.001	0.938402
14	50	SGD	0.01	0.935283
15	10	RMSprop	0.001	0.933333
16	50	SGD	0.001	0.933333
17	50	RMSprop	0.0001	0.932554
18	50	Nadam	0.0001	0.931774
19	20	RMSprop	0.01	0.930604
20	50	Adam	0.0001	0.928655

21	10	RMSprop	0.01	0.925146
22	20	SGD	0.01	0.924366
23	20	SGD	0.001	0.920468
24	10	Adam	0.01	0.914620
25	20	RMSprop	0.0001	0.911111
26	10	Nadam	0.01	0.909552
27	50	SGD	0.0001	0.902924
28	20	Nadam	0.0001	0.900195
29	10	SGD	0.001	0.899805
30	20	Adam	0.0001	0.899025
31	10	RMSprop	0.0001	0.873294
32	10	Adam	0.0001	0.867446
33	10	Nadam	0.0001	0.863158
34	20	SGD	0.0001	0.862768
35	10	SGD	0.01	0.848343
36	10	SGD	0.0001	0.846004

3. InceptionV3

Table 4.2.3: InceptionV3 Grid Search Result on Defect Type 1

#	Epochs	Optimiser	Learning Rate	Mean Score
1	20	Nadam	0.001	0.951333
2	50	Adam	0.001	0.951333
3	50	Adam	0.01	0.951
4	20	RMSprop	0.001	0.949
5	20	Nadam	0.01	0.948333
6	20	Adam	0.001	0.948333
7	20	RMSprop	0.01	0.943
8	10	Nadam	0.01	0.940666
9	50	Nadam	0.0001	0.940333
10	50	RMSprop	0.001	0.936
11	50	Adam	0.0001	0.935333
12	10	Nadam	0.001	0.935

13	10	RMSprop	0.001	0.934
14	50	RMSprop	0.0001	0.933667
15	50	Nadam	0.001	0.933
16	50	SGD	0.0001	0.921667
17	50	Nadam	0.01	0.920333
18	10	SGD	0.01	0.919
19	10	RMSprop	0.01	0.916333
20	20	RMSprop	0.0001	0.916
21	20	Adam	0.0001	0.915667
22	20	SGD	0.001	0.914667
23	20	SGD	0.01	0.913
24	20	SGD	0.0001	0.911333
25	20	Nadam	0.0001	0.910667
26	50	RMSprop	0.01	0.908333
27	10	SGD	0.001	0.9
28	20	Adam	0.01	0.897
29	50	SGD	0.01	0.895667
30	10	RMSprop	0.0001	0.890667
31	10	Nadam	0.0001	0.885333
32	50	SGD	0.001	0.884333
33	10	Adam	0.0001	0.883
34	10	SGD	0.0001	0.880667
35	10	Adam	0.01	0.803333
36	10	Adam	0.001	0.802667

4.2.2 Defect Type 2 – Epoxy Overflow on LED

1. **ResNet50**

Table 4.2.4: ResNet50 Grid Search Result on Defect Type 2

#	Epochs	Optimiser	Learning Rate	Mean Score
1	50	RMSprop	0.01	0.969068
2	50	Nadam	0.01	0.968313
3	50	Adam	0.001	0.968310

4	20	Nadam	0.01	0.967937
5	20	Adam	0.01	0.967934
6	50	Adam	0.01	0.967183
7	50	RMSprop	0.001	0.965295
8	20	RMSprop	0.01	0.964917
9	50	Nadam	0.001	0.964917
10	20	Nadam	0.001	0.962654
11	10	Nadam	0.01	0.961526
12	10	Adam	0.01	0.961523
13	10	RMSprop	0.01	0.960013
14	20	Adam	0.001	0.959263
15	20	RMSprop	0.001	0.956620
16	10	Nadam	0.001	0.955488
17	50	Nadam	0.0001	0.953601
18	10	Adam	0.001	0.953226
19	50	SGD	0.01	0.952093
20	50	Adam	0.0001	0.952092
21	50	RMSprop	0.0001	0.951715
22	10	RMSprop	0.001	0.950586
23	20	SGD	0.01	0.944931
24	10	SGD	0.01	0.935497
25	50	SGD	0.001	0.932476
26	20	Nadam	0.0001	0.930592
27	20	RMSprop	0.0001	0.929841
28	20	Adam	0.0001	0.926821
29	20	SGD	0.001	0.910981
30	10	Adam	0.0001	0.889099
31	10	RMSprop	0.0001	0.887593
32	10	Nadam	0.0001	0.880798
33	10	SGD	0.001	0.861941
34	50	SGD	0.0001	0.837797
35	20	SGD	0.0001	0.800824

36	10	SGD	0.0001	0.781593

2. Xception

Table 4.2.5: Xception Grid Search Result on Defect Type 2

#	Epochs	Optimiser	Learning Rate	Mean Score
1	20	Adam	0.01	0.953602
2	20	Adam	0.001	0.945304
3	20	Nadam	0.001	0.944925
4	50	Nadam	0.001	0.944545
5	50	Adam	0.01	0.944168
6	50	Adam	0.001	0.941524
7	50	RMSprop	0.001	0.940773
8	50	Nadam	0.01	0.940404
9	10	Nadam	0.01	0.940020
10	10	Adam	0.001	0.939270
11	50	RMSprop	0.0001	0.938893
12	10	Nadam	0.001	0.937759
13	10	RMSprop	0.001	0.935497
14	50	Adam	0.0001	0.934362
15	50	Nadam	0.0001	0.933609
16	20	RMSprop	0.001	0.933234
17	50	RMSprop	0.01	0.930580
18	50	SGD	0.001	0.923427
19	20	Nadam	0.01	0.920042
20	10	RMSprop	0.01	0.913221
21	20	RMSprop	0.01	0.912482
22	20	RMSprop	0.0001	0.910225
23	20	SGD	0.001	0.907584
24	20	SGD	0.01	0.905307
25	20	Nadam	0.0001	0.903814
26	20	Adam	0.0001	0.901175
27	10	SGD	0.001	0.896648

28	50	SGD	0.0001	0.891366
29	50	SGD	0.01	0.889484
30	10	RMSprop	0.0001	0.884576
31	10	Adam	0.0001	0.883819
32	10	Adam	0.01	0.875167
33	20	SGD	0.0001	0.866465
34	10	SGD	0.01	0.858891
35	10	Nadam	0.0001	0.855910
36	10	SGD	0.0001	0.847610

3. InceptionV3

Table 4.2.6: InceptionV3 Grid Search Result on Defect Type 2

#	Epochs	Optimiser	Learning Rate	Mean Score
1	50	Adam	0.001	0.955489
2	50	Nadam	0.001	0.950584
3	50	RMSprop	0.01	0.950208
4	50	Adam	0.01	0.948700
5	50	RMSprop	0.001	0.948319
6	20	Adam	0.001	0.947569
7	50	Nadam	0.01	0.944927
8	20	Adam	0.01	0.940781
9	20	Nadam	0.001	0.939643
10	50	Nadam	0.0001	0.935493
11	50	RMSprop	0.0001	0.935122
12	10	Adam	0.01	0.931351
13	50	Adam	0.0001	0.929835
14	50	SGD	0.01	0.929462
15	10	Nadam	0.001	0.929090
16	10	RMSprop	0.01	0.924552
17	10	RMSprop	0.001	0.923428
18	50	SGD	0.0001	0.918518
19	20	RMSprop	0.001	0.916252

20	50	SGD	0.001	0.914379
21	20	RMSprop	0.0001	0.913619
22	20	RMSprop	0.01	0.911713
23	20	Adam	0.0001	0.911351
24	20	Nadam	0.0001	0.910221
25	10	Nadam	0.01	0.901532
26	20	SGD	0.01	0.898149
27	10	Adam	0.001	0.895526
28	20	Nadam	0.01	0.888337
29	20	SGD	0.0001	0.886841
30	10	SGD	0.01	0.883823
31	10	SGD	0.001	0.881934
32	10	Adam	0.0001	0.878914
33	20	SGD	0.001	0.872877
34	10	RMSprop	0.0001	0.869857
35	10	Nadam	0.0001	0.864583
36	10	SGD	0.0001	0.844209

4.2.3 Defect Type 3 – Epoxy on Die + FM on Die

1. **ResNet50**

Table 4.2.7: ResNet50 Grid Search Result on Defect Type 3

#	Epochs	Optimiser	Learning Rate	Mean Score
1	50	Adam	0.01	0.941511
2	20	Adam	0.01	0.934202
3	50	Nadam	0.01	0.933391
4	50	RMSprop	0.01	0.930948
5	50	Adam	0.001	0.926885
6	50	Nadam	0.001	0.924456
7	20	RMSprop	0.01	0.924444
8	20	Nadam	0.01	0.924442
9	50	RMSprop	0.001	0.918788
10	20	Adam	0.001	0.918766

11	20	Nadam	0.001	0.915514
12	20	RMSprop	0.001	0.896000
13	50	RMSprop	0.0001	0.869207
14	50	Nadam	0.0001	0.853773
15	50	Adam	0.0001	0.848897
16	20	RMSprop	0.0001	0.790398
17	20	Nadam	0.0001	0.775792
18	20	Adam	0.0001	0.759547

2. Xception

Table 4.2.8: Xception Grid Search Result on Defect Type 3

#	Epochs	Optimiser	Learning Rate	Mean Score
1	20	Adam	0.01	0.906575
2	50	Adam	0.01	0.900097
3	50	Adam	0.001	0.896004
4	20	Adam	0.001	0.873272
5	50	Nadam	0.001	0.871667
6	20	Nadam	0.001	0.861088
7	50	RMSprop	0.001	0.846504
8	50	RMSprop	0.0001	0.823714
9	50	Nadam	0.0001	0.822903
10	50	Adam	0.0001	0.810727
11	20	RMSprop	0.001	0.809127
12	50	RMSprop	0.01	0.800269
13	20	Nadam	0.01	0.793749
14	50	Nadam	0.01	0.784068
15	20	Nadam	0.0001	0.770113
16	20	RMSprop	0.0001	0.769292
17	20	Adam	0.0001	0.767670
18	20	RMSprop	0.01	0.741814

3. InceptionV3

Table 4.2.9: InceptionV3 Grid Search Result on Defect Type 3

#	Epochs	Optimiser	Learning Rate	Mean Score
1	20	Adam	0.001	0.874892
2	50	Adam	0.001	0.862671
3	20	Nadam	0.001	0.849728
4	50	Adam	0.0001	0.834273
5	20	RMSprop	0.01	0.831872
6	50	RMSprop	0.0001	0.831836
7	50	Nadam	0.0001	0.826163
8	20	Nadam	0.01	0.815552
9	50	RMSprop	0.01	0.806739
10	20	Adam	0.01	0.803515
11	50	RMSprop	0.001	0.787217
12	50	Adam	0.01	0.785469
13	20	RMSprop	0.0001	0.779034
14	50	Nadam	0.001	0.774273
15	20	Nadam	0.0001	0.770908
16	20	Adam	0.0001	0.764435
17	50	Nadam	0.01	0.725648
18	20	RMSprop	0.001	0.706975

4.2.4 Grid Search Summary

The optimal hyperparameters were summarised in Table 4.2.10. These hyperparameters were used in the model training.

Table 4.2.10: Summary of Optimal Hyperparameters

Defect Type	**Model**	**Epoch**	**Learning Rate**	**Optimiser**	**Loss Function**	**Batch Size**
1	ResNet50	50	0.01	RMSprop	Binary Cross-entropy	16
	Xception	50	0.001	Adam		
	InceptionV3	20	0.001	Nadam		
2	ResNet50	50	0.01	RMSprop		
	Xception	20	0.01	Adam		
	InceptionV3	50	0.001	Nadam		
3	ResNet50	50	0.01	Adam	Categorical Cross-entropy	
	Xception	20	0.01	Adam		
	InceptionV3	20	0.001	Adam		

The results show that Adam optimiser appears more frequently selected as the optimal choice across different defect types and models. However, the learning rates for these optimisers differ. The number of epochs also varies across different configurations, suggesting that the convergence rate and training dynamics differ for each model and defect type. We can see that RMSprop and Nadam were selected as the optimisers for defect types 1 and 2, which are binary classification tasks with a larger dataset size, compared to defect type 3, which is a multi-class classification task with a smaller dataset size. All three models selected Adam as the optimiser in defect type 3, indicating Adam is suitable for small dataset sizes or multi-class classification.

By looking at the differences in hyperparameters for the same model on different defect types, it was shown that it is crucial to consider the specific characteristics of the dataset and problem when selecting hyperparameters. What works well for one defect type or model may not necessarily be the best choice for another. These results highlight the importance of hyperparameter tuning and the need to experiment with various combinations to find the optimal setup for each specific problem.

4.3 Model Evaluation

The deep learning models were evaluated on the same distribution to ensure a fair comparison between the models.

The deep learning models underwent two comparison stages: the first stage was after training to select the best-performing model from the same base model, and the second stage was after selecting the best-performing model from a different base model on the same defect type. Then, the study compared the performance of the models from different base models. The number of images in testing dataset of each defect type are shown in Table 4.3.1.

Table 4.3.1: Number of Images in each Testing Dataset

Defect Type	Number of Images in Testing Dataset
1. Epoxy Overflow on Die	481
2. Epoxy Overflow on LED	397
3. Epoxy on Die + FM on Die	168

The results in the following sections rank the models at epoch from the lowest validation loss to the highest validation loss. In the case where the accuracy of the models was tallied, this study selected a model with a higher ranking or a lower validation loss as it indicated better generalisation ability.

For comparison between different base models, the study selected the model with the highest accuracy, and if the accuracy of the models was equal, the model with a higher recall was selected.

Even though the study adopted the approach of loading models based on low validation losses and this strategy did not directly rely on monitoring the convergence rate, it analysed the training and validation losses during training to potentially gain insights into the model behaviour and hyperparameter tuning, contributing to a deeper understanding of hyperparameter tuning and deep learning principles.

4.3.1 Defect Type 1 – Epoxy Overflow on Die

1. **ResNet50**

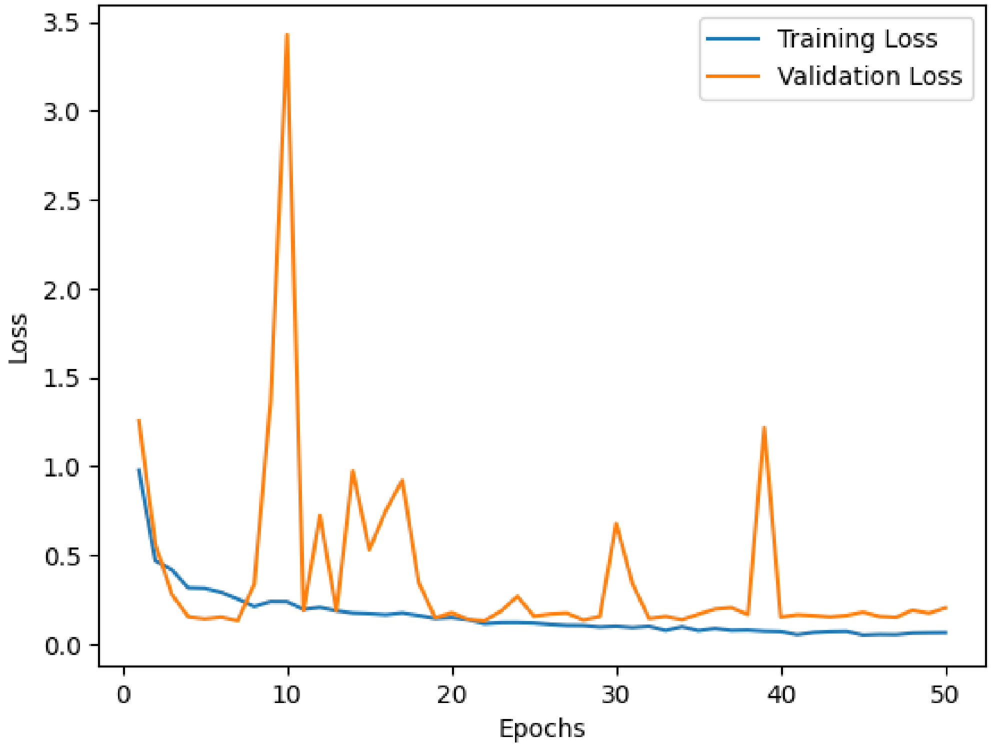


Figure 4.3.1: ResNet50 Training and Validation Loss Graph on Defect Type 1

Table 4.3.2: Accuracy of ResNet50 for Defect Type 1

Ranking	Epoch	Decision Threshold	Accuracy
1	22	0.80	0.931
2	7	0.20	0.958
3	28	0.20	0.971
4	34	0.20	0.961
5	**21**	**0.35**	**0.973**
6	5	0.20	0.969
7	32	0.20	0.946
8	19	0.20	0.965
9	47	0.20	0.963
10	40	0.20	0.969

2. Xception

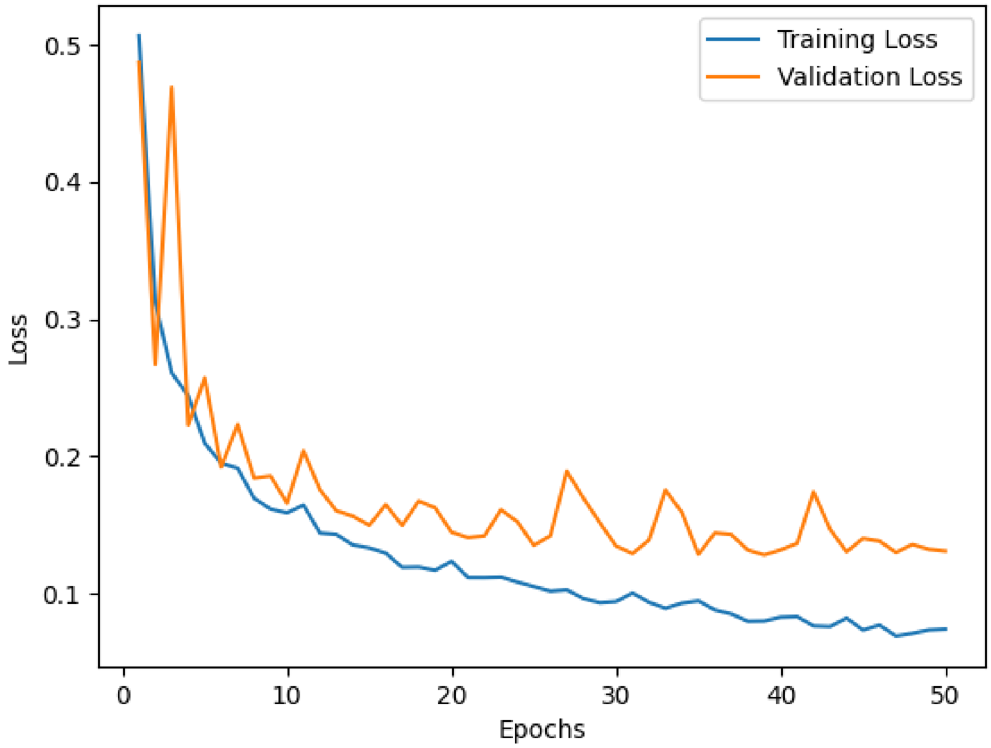


Figure 4.3.2: Xception Training and Validation Loss Graph on Defect Type 1

Table 4.3.3: Accuracy of Xception for Defect Type 1

Ranking	Epoch	Decision Threshold	Accuracy
1	39	0.30	0.969
2	35	0.50	0.965
3	31	0.45	0.963
4	47	0.60	0.961
5	**44**	**0.35**	**0.973**
6	50	0.40	0.973
7	38	0.60	0.969
8	40	0.40	0.973
9	49	0.35	0.971
10	30	0.45	0.963

3. InceptionV3

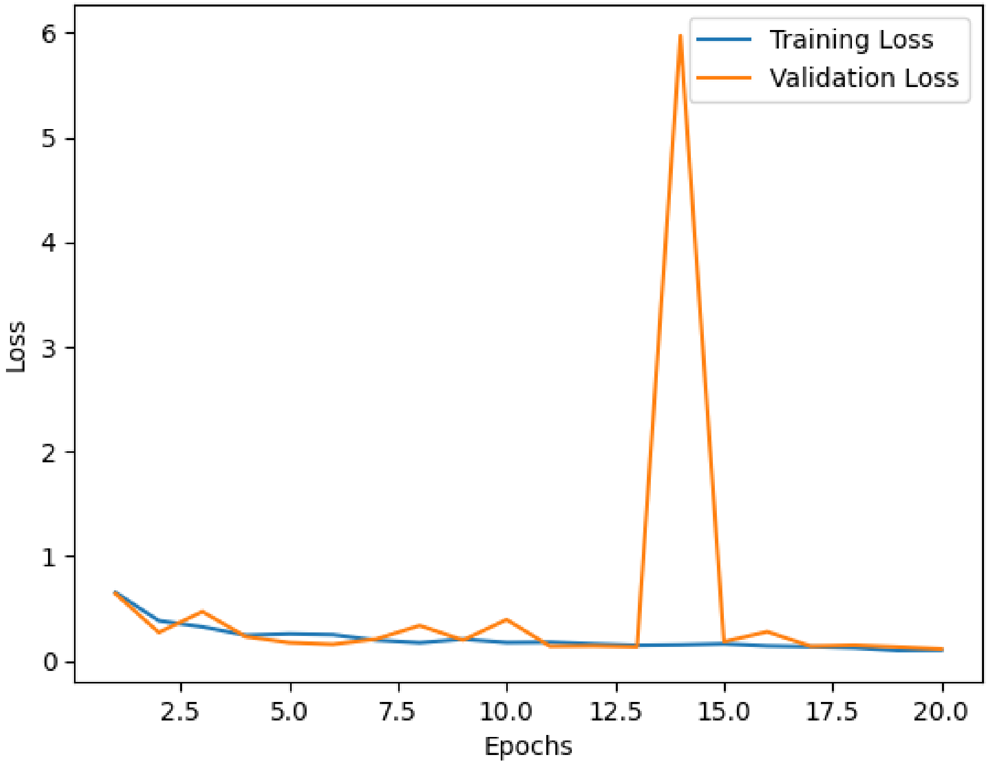


Figure 4.3.3: InceptionV3 Training and Validation Loss Graph on Defect Type 1

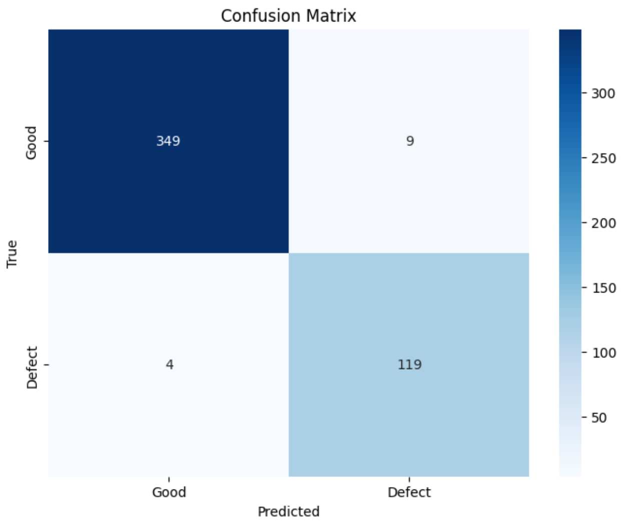
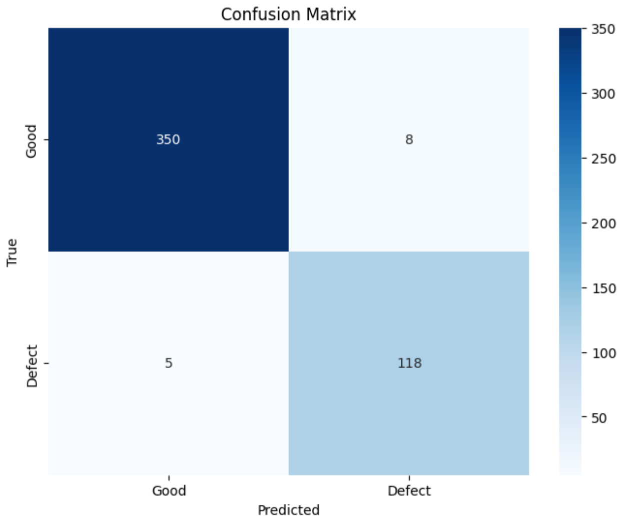
Table 4.3.4: Accuracy of InceptionV3 for Defect Type 1

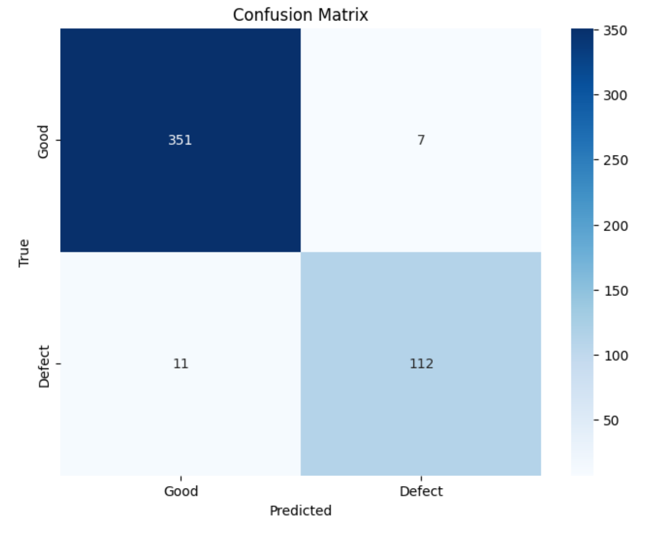
Ranking	Epoch	Decision Threshold	Accuracy
1	20	0.75	0.954
2	19	0.55	0.958
3	**13**	**0.55**	**0.963**
4	11	0.55	0.963
5	17	0.20	0.958
6	12	0.75	0.958
7	18	0.75	0.958
8	6	0.75	0.958
9	5	0.65	0.950
10	15	0.80	0.950

Table 4.3.5: Comparison of Models for Defect Type 1

Models	Accuracy	Recall	Precision	F1-Score
ResNet50	**0.973**	**0.967**	**0.930**	**0.948**
Xception	0.973	0.937	0.959	0.948
InceptionV3	0.963	0.941	0.911	0.926

Table 4.3.6: Confusion Matrix for Defect Type 1

<table>
<tr><td>ResNet50</td><td>
</td></tr>
<tr><td>Xception</td><td>
</td></tr>
</table>

InceptionV3	

Figures 4.3.1, 4.3.2, and 4.3.3 show the training loss and validation loss of the models during training for defect type 1. For ResNet50, there were spikes in the validation loss frequently. The spikes indicated that the model might have been overfitted during the training process. Xception had a steady and smooth convergence, with inversely proportional training and validation losses, which is a positive sign. It suggested that the model is learning well and generalising to the validation data effectively. InceptionV3 also had a steady and smooth convergence except for a huge spike of validation loss for a few epochs. Despite the spikes in validation loss, all three models showed a good convergence rate on learning the dataset, which indicated that the models were not overfitting and were converging towards an optimal state.

By referring to Table 4.3.5, all three models achieved high accuracy in detecting defect type 1, indicating that they are generally good at making correct predictions. InceptionV3 achieved the lowest accuracy among the three models, scoring only 96.3%. ResNet50 and Xception achieved the same accuracy of 97.3%, but ResNet50 scored a recall of 96.7%, and Xception scored a recall of 93.7%. ResNet50 was the best-performing model for defect type 1.

4.3.2 Defect Type 2 – Epoxy Overflow on LED

1. ResNet50

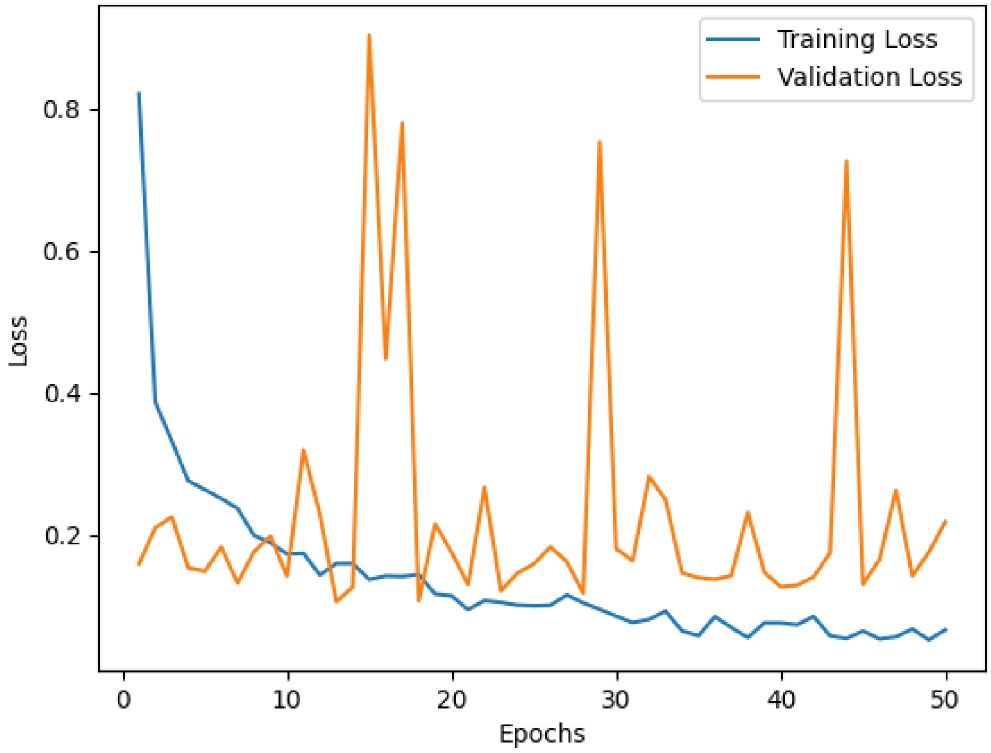


Figure 4.3.4: ResNet50 Training and Validation Loss Graph for Defect Type 2

Table 4.3.7: Accuracy of ResNet50 for Defect Type 2

Ranking	Epoch	Decision Threshold	Accuracy
1	13	0.80	0.972
2	**18**	**0.55**	**0.977**
3	28	0.80	0.970
4	23	0.80	0.970
5	14	0.80	0.967
6	40	0.80	0.975
7	41	0.80	0.972
8	21	0.80	0.972
9	45	0.80	0.972
10	7	0.80	0.972

2. Xception

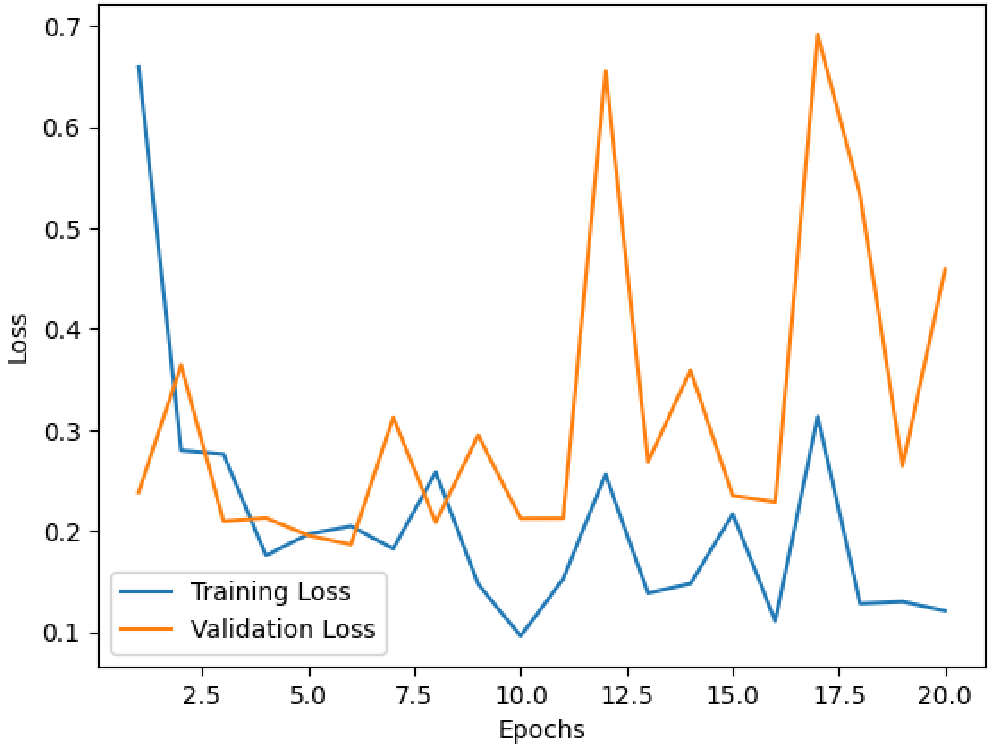


Figure 4.3.5: Xception Training and Validation Loss Graph for Defect Type 2

Table 4.3.8: Accuracy of Xception for Defect Type 2

Ranking	Epoch	Decision Threshold	Accuracy
1	6	0.80	0.9723
2	5	0.50	0.9748
3	8	0.80	0.9748
4	3	0.60	0.9723
5	10	0.80	0.9748
6	11	0.80	0.9748
7	4	0.50	0.9748
8	16	0.50	0.9748
9	**15**	**0.75**	**0.9798**
10	1	0.75	0.9798

3. InceptionV3

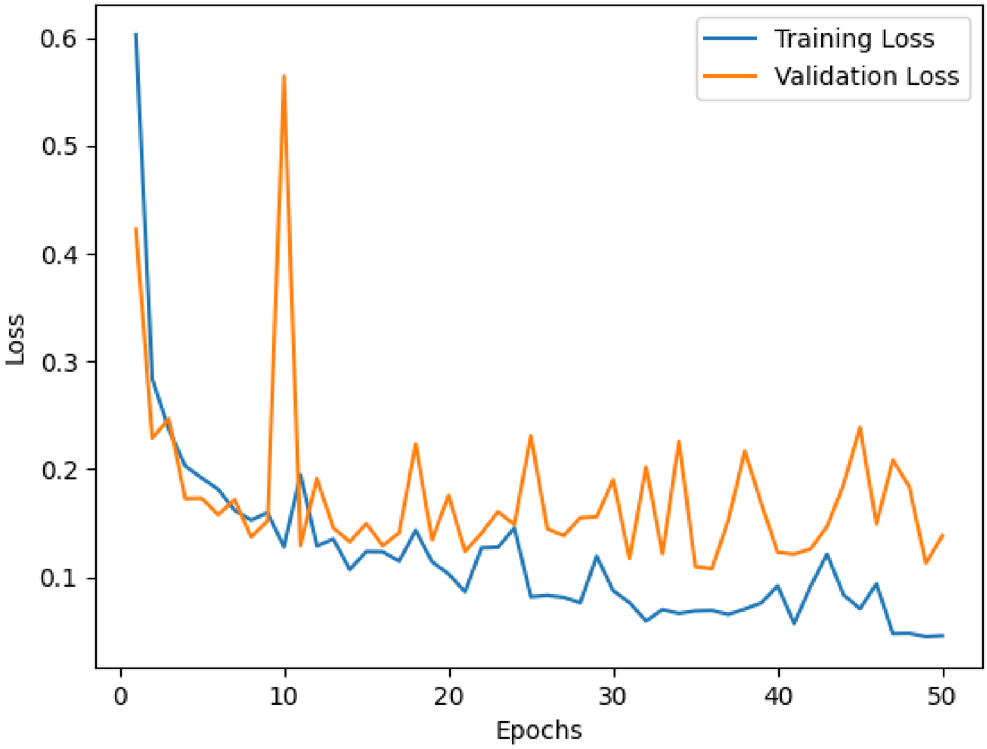


Figure 4.3.6: InceptionV3 Training and Validation Loss Graph for Defect Type 2

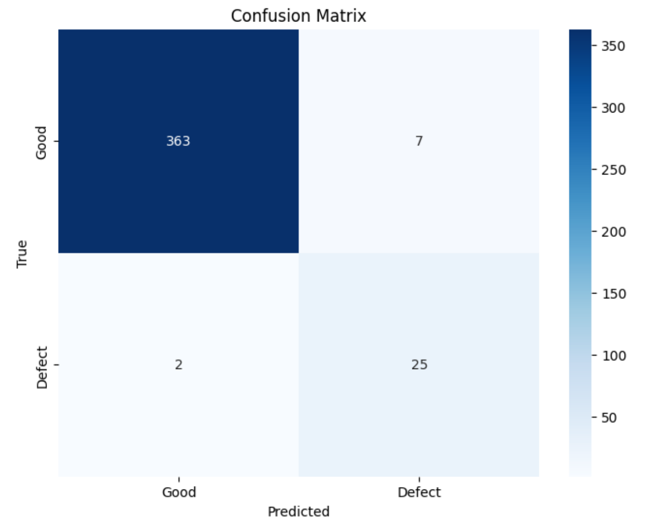
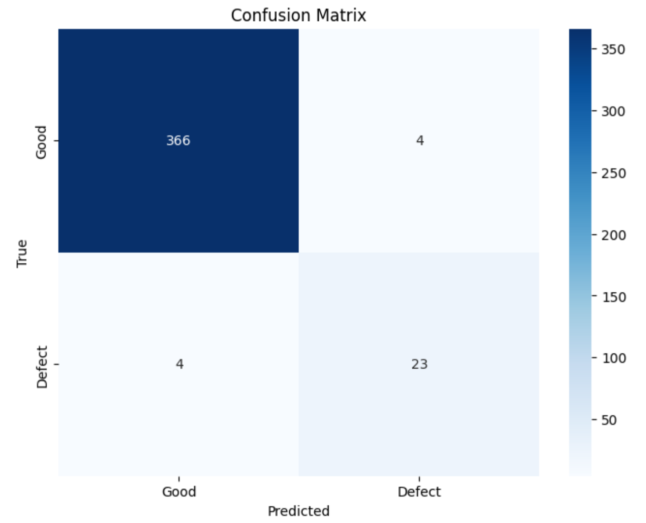
Table 4.3.9: Accuracy of InceptionV3 for Defect Type 2

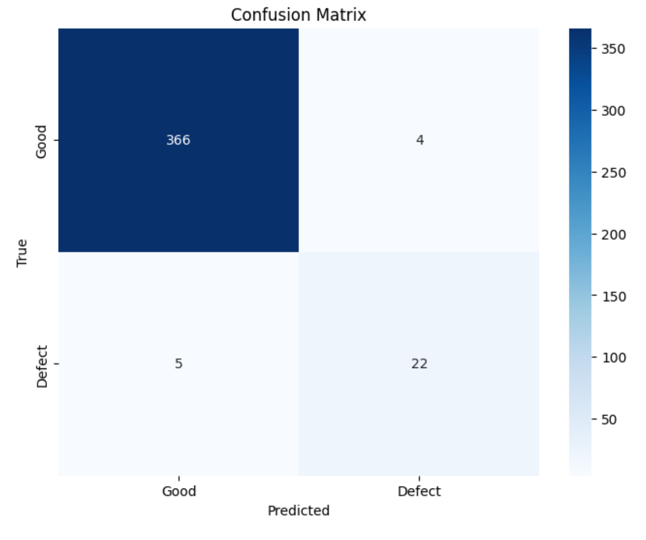
Ranking	Epoch	Decision Threshold	Accuracy
1	36	0.60	0.965
2	35	0.65	0.965
3	49	0.45	0.967
4	31	0.80	0.962
5	41	0.80	0.967
6	33	0.80	0.967
7	40	0.75	0.965
8	21	0.80	0.975
9	**42**	**0.65**	**0.977**
10	16	0.65	0.970

Table 4.3.10: Comparison of Models for Defect Type 2

Models	Accuracy	Recall	Precision	F1-Score
ResNet50	0.977	0.926	0.781	0.847
Xception	**0.980**	**0.852**	**0.852**	**0.852**
InceptionV3	0.977	0.815	0.846	0.830

Table 4.3.11: Confusion Matrix for Defect Type 2

ResNet50	
Xception	

InceptionV3	

Figures 4.3.4, 4.3.5, and 4.3.6 show the training loss and validation loss of the models during training for defect type 2. For ResNet50, the spikes in validation loss were more serious than in training for defect type 1. Besides, the validation loss slowly increased towards the later epochs in the training, showing that ResNet50 was overfitting. The same goes for Xception, where it also faced the issue of spiked validation losses, which indicated that the model was not converging optimally. InceptionV3 showed the best convergence rate in validation loss among the models, suggesting that the model has good generalisation ability.

All three models accurately detected defect type 2: epoxy overflow on the LED. ResNet50 and InceptionV3 achieved the same accuracy of 97.7%. Xception outperformed the other two models by achieving an accuracy score of 98.0%. We can see that Xception only had eight wrong predictions in 397 test samples. Xception is the best-performing model for defect type 2.

4.3.3 Defect Type 3 – Epoxy on Die + FM on Die

1. ResNet50

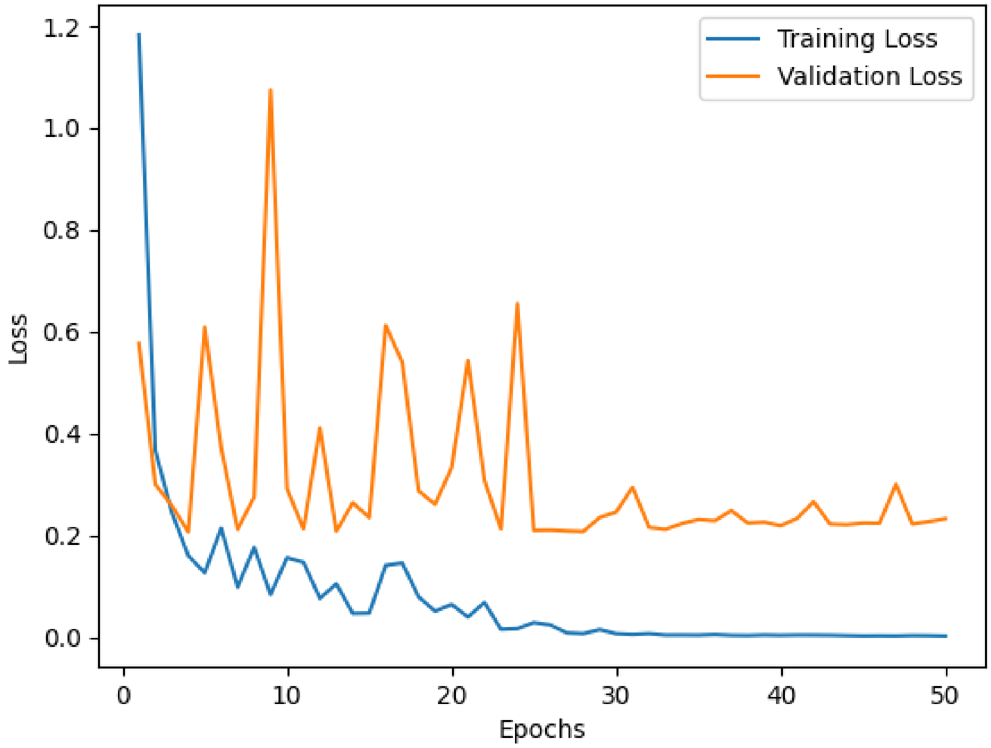


Figure 4.3.7: ResNet50 Training and Validation Loss Graph for Defect Type 3

Table 4.3.12: Accuracy of ResNet50 for Defect Type 3

Ranking	Epoch	Accuracy
1	4	0.893
2	28	0.935
3	13	0.923
4	27	0.917
5	25	0.935
6	26	0.917
7	7	0.905
8	**33**	**0.940**
9	23	0.929
10	11	0.929

2. Xception

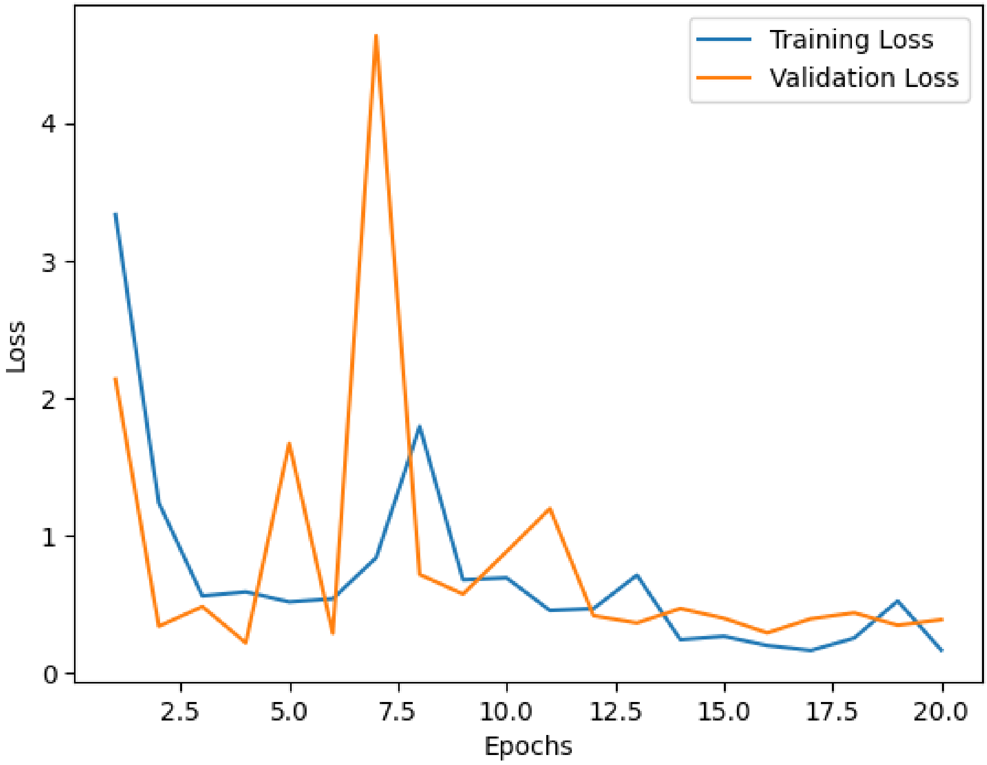


Figure 4.3.8: Xception Training and Validation Loss Graph for Defect Type 3

Table 4.3.13: Accuracy of Xception for Defect Type 3

Ranking	Epoch	Accuracy
1	4	0.875
2	6	0.863
3	16	0.881
4	2	0.881
5	19	0.887
6	13	0.887
7	20	0.863
8	**17**	**0.905**
9	15	0.905
10	12	0.893

3. InceptionV3

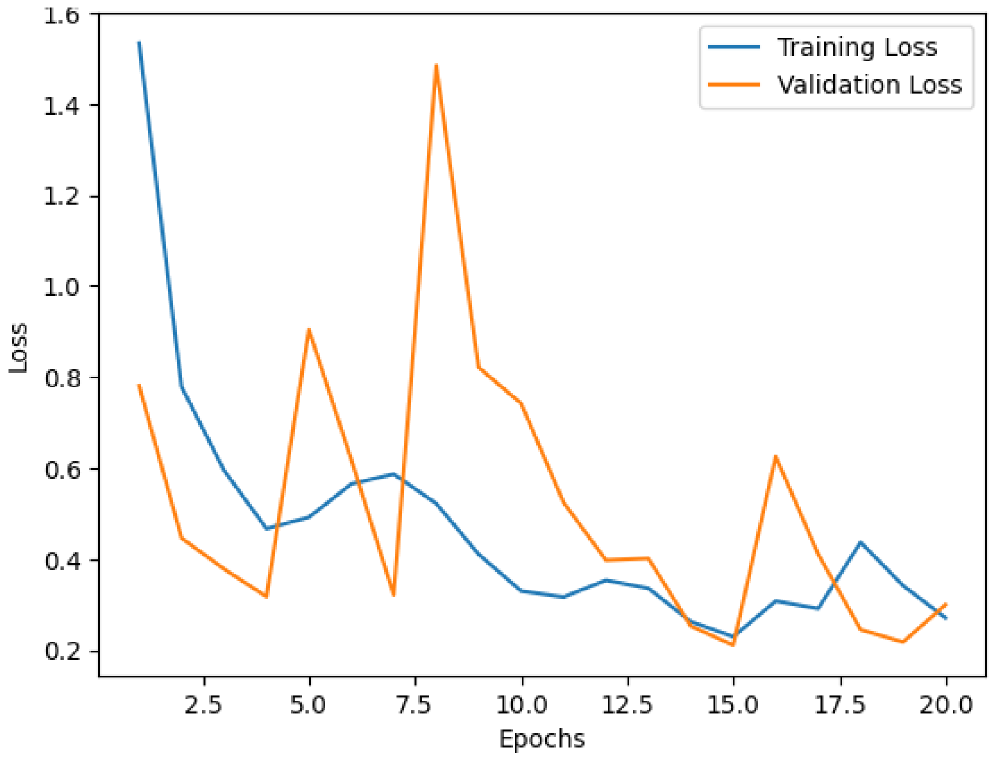


Figure 4.3.9: InceptionV3 Training and Validation Loss Graph for Defect Type 3

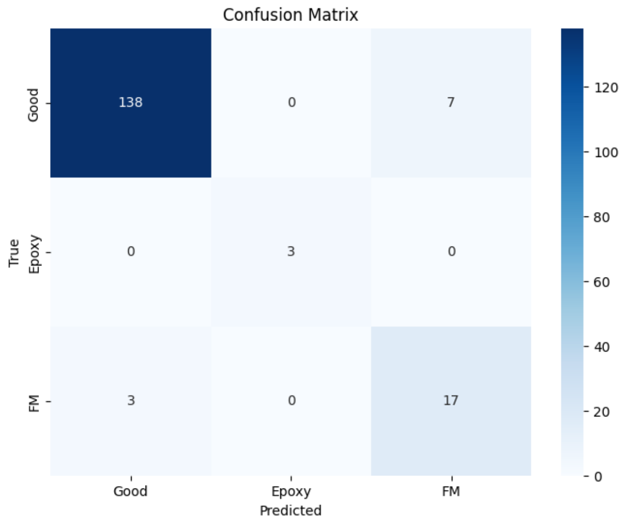
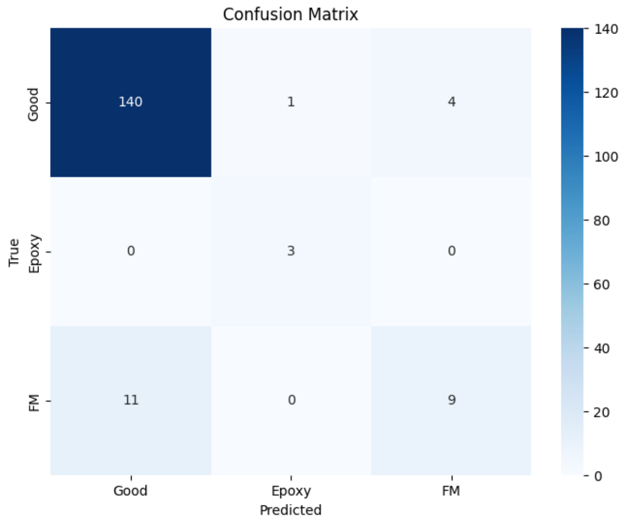
Table 4.3.14: Accuracy of InceptionV3 for Defect Type 3

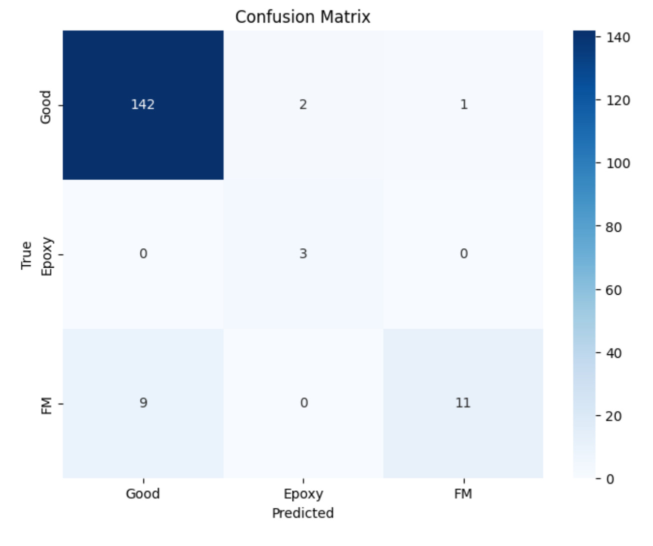
Ranking	Epoch	Accuracy
1	**15**	**0.929**
2	19	0.905
3	18	0.887
4	14	0.881
5	20	0.845
6	4	0.851
7	7	0.851
8	3	0.792
9	12	0.887
10	13	0.798

Table 4.3.15: Comparison of Models for Defect Type

Models	Accuracy	Recall	Precision	F1-Score
ResNet50	**0.940**	**0.940**	**0.947**	**0.942**
Xception	0.905	0.905	0.896	0.897
InceptionV3	0.929	0.929	0.931	0.923

Table 4.3.16: Confusion Matrix for Defect Type 3

ResNet50	
Xception	

InceptionV3	

Figures 4.3.7, 4.3.8, and 4.3.9 show the training loss and validation loss of the models during training for defect type 3. The spike in validation loss for Xception and InceptionV3 indicated that the model was not converging well and might need adjustments on hyperparameters. On the other hand, ResNet50 had a stabilised line of validation loss starting around epoch 25, showing that the model was learning effectively without overfitting.

All three models accurately detected defect type 3: epoxy and FM on the die. Xception had the lowest accuracy of 90.5%, and InceptionV3 ranked second with 92.9%. ResNet50 scored the highest accuracy of 94.0%. From Table 4.3.16, we can see that only ResNet50 could correctly predict all ‘Epoxy’ class samples, while the other two models incorrectly predicted samples from other classes as ‘Epoxy’, further proving ResNet50 as the most robust model among the three.

4.4 Model Evaluation Summary

Table 4.4.1: Model Evaluation Summary

Defect Type	Models	Accuracy	Recall	Precision	F1-Score
1	**ResNet50**	**0.973**	**0.967**	**0.930**	**0.948**
	Xception	0.973	0.937	0.959	0.948
	InceptionV3	0.963	0.941	0.911	0.926
2	ResNet50	0.977	0.926	0.781	0.847
	Xception	**0.980**	**0.852**	**0.852**	**0.852**
	InceptionV3	0.977	0.815	0.846	0.830
3	**ResNet50**	**0.940**	**0.940**	**0.947**	**0.942**
	Xception	0.905	0.905	0.896	0.897
	InceptionV3	0.929	0.929	0.931	0.923

The performance of the nine models is recorded in Table 4.4.1.

For defect type 1, ResNet50 and Xception both performed exceptionally well, with an accuracy of 97.3%. This indicated that they were highly reliable in correctly classifying defect type 1. The recall for ResNet50 was slightly higher at 96.7%, while Xception had a recall of 93.7%. This meant that ResNet50 was better at identifying most of the instances of defect type 1. Precision for Xception was notably higher at 95.9%, suggesting fewer false positives. InceptionV3 also performed well with an accuracy of 96.3% but had a slightly lower recall of 94.1% compared to the other two models. As the project prioritised accuracy and recall, ResNet50 was the best-performing model for defect type 1.

For defect type 2, Xception achieved the highest accuracy among all models and defect types at 98.0%, making it a top choice for defect type 2 classification when accuracy was paramount. However, its recall was 85.2%, indicating that it missed some instances of defect type 2. Despite this, Xception maintained a high precision of 85.2%. ResNet50 also delivered strong results, with an accuracy of 97.7%, a recall of 92.6%, and a precision of 78.1%. InceptionV3 offered a competitive option with an accuracy of 97.7%, a recall of

81.5%, and a precision of 84.6%. It struck a balance between accuracy and recall. Xception was the best-performing model for defect type 2 due to its highest accuracy score of 98.0%.

For defect type 3, ResNet50 excelled in defect type 3 classification with an accuracy of 94.0%. It achieved a high recall of 94.0%, indicating its effectiveness in identifying instances of defect type 3. The model also had the highest precision of 94.7%, meaning it made fewer false-positive predictions. Xception had an accuracy of 90.5%, a recall of 90.5%, and a precision of 89.6%. It provided consistent results, but with slightly lower accuracy. Inception V3 achieved an accuracy of 92.9%, a recall of 92.9%, and a precision of 93.1%. It performed competitively and balanced accuracy and recall effectively. ResNet50 was the best-performing model for defect type 3, with the highest accuracy of 94.0%. In short, ResNet50 performed the best for defect types 1 and 3, while Xception performed the best for defect type 2, and InceptionV3 performed slightly poorly compared to the other two models.

CHAPTER 5
CONCLUSIONS

This study successfully applied transfer learning to three pre-trained deep learning models, ResNet50, Xception, and InceptionV3, for epoxy-related defect detection on the PCB of wireless earbuds. This study has achieved all objectives for which an image dataset was collected and preprocessed to train the deep learning models, and the models achieved a minimum of 90% accuracy in the defect detection task. After evaluating the models, ResNet50 was the best-performing model for defect types 1 and 3 with an accuracy of 97.7% and 94.0%, respectively, while Xception was the best-performing model for defect type 2 with an accuracy of 98.0%. This study successfully proposed three deep learning models that achieved high accuracy in detecting epoxy-related defects on PCBs that were difficult for human eyes to see. The proposed model successfully achieved an accuracy of up to 98.0% in detecting epoxy-related defects on real-world wireless earbud PCB images.

This study also addressed the time-cost issue by utilising transfer learning in model training to shorten the time taken while achieving high accuracy. InceptionV3 was capable of classifying 168 images at a speed of 0.6913 seconds per image, ResNet50 took 0.6211 seconds, and Xception took 0.8465 seconds. This means that they can process a batch of 168 images within one second demonstrating their efficiency.

5.1 Limitations

Optimising Hyperparameters

This study did not fully discover the most optimal hyperparameters for the models due to computational resource constraints. In this case, there might be better hyperparameter combinations that could further improve the model training process and improve the accuracy of defect detection. This limitation highlights a future improvement for the study by researching potentially better hyperparameter combinations.

Class Imbalance Issue

The class imbalance issue was pervasive throughout the dataset, with the most severe imbalance observed in defect type 3. In this category, the 'Good' class contained 1,000 images, while the 'Epoxy' class had only 11 images, and the 'FM' class comprised 111 images. The imbalanced class distribution posed a significant challenge to the model's performance, as it struggled to effectively learn from the minority classes. For instance, ResNet50 performed the best in detecting defect types 1 and 3, achieving an accuracy of 97.3% for defect type 1 while only having an accuracy of 94.0% for defect type 3.

Decision Threshold

This study adjusted the decision threshold value for binary classification for defect types 1 and 2. While tuning the threshold can enhance model performance and accuracy, it also introduces manual intervention and complexity. Ideally, models should be capable of self-adaptation without requiring manual threshold adjustments, and this limitation highlights an area for potential future improvements in automation and model robustness.

5.2 Recommendations for Future Work

Utilisation of Synthetic Data Generation Technique

To mitigate the class imbalance issue observed in the dataset, one potential avenue for improvement is the application of synthetic data generation techniques, particularly using advanced deep learning models such as generative adversarial networks (GANs). GANs can generate high-quality synthetic data that closely resembles real samples. By introducing synthetic data for minority classes, the study can potentially rebalance the dataset and provide the model with more examples of underrepresented classes. This can lead to improved generalisation and performance, especially for defect type 3, which had a severe class imbalance issue.

Further Enhancement on Hyperparameters and Model Architecture

There is potential for further enhancement in model performance through an exhaustive hyperparameter tuning process or by exploring modifications to the model architecture. A more comprehensive grid search, spanning a wider range of hyperparameters and combinations, may help fine-tune the model to discover optimal settings. Besides, exploring alternative model architectures or even more advanced neural network architectures can open up opportunities to capture complex patterns and features within the dataset, potentially leading to improved model performance.

Applying the Methodology to Other Types of PCB Defects

While the current focus of this study has been on epoxy-related defects in PCBs, the methodology and techniques developed herein possess a versatile applicability that can be extended to other types of defects commonly encountered in PCB manufacturing. The robust framework established in this study, including data collection, preprocessing, and deep learning model development, can be readily adapted to investigate and address a broader spectrum of PCB defects. This adaptability highlights the approach's versatility and underscores its potential to contribute to comprehensive quality control and assurance efforts within the PCB industry.

REFERENCE

Adibhatla, V.A., Chih, H.-C., Hsu, C.-C., Cheng, J., Abbod, M.F. and Shieh, J.-S., 2020. Defect Detection in Printed Circuit Boards Using You-Only-Look-Once Convolutional Neural Networks. *Electronics*, [online] 9(9), p.1547. https://doi.org/10.3390/electronics9091547.

Anon. 2023a. *Papers With Code*. [online] Papers With Code. Available at: <https://paperswithcode.com/> [Accessed 10 April 2023].

Anon. 2023b. *What is supervised learning?* [online] IBM. Available at: <https://www.ibm.com/topics/supervised-learning> [Accessed 11 September 2023].

Baheti, P., 2023. *A Newbie-Friendly Guide to Transfer Learning*. [online] V7. Available at: <https://www.v7labs.com/blog/transfer-learning-guide#:~:text=Traditional%20machine%20learning%20models%20require,using%20a%20small%20data%20set> [Accessed 10 April 2023].

Banoula, M., 2023. *Classification in Machine Learning: What it is & Classification Models*. [online] simplilearn. Available at: <https://www.simplilearn.com/tutorials/machine-learning-tutorial/classification-in-machine-learning#what_is_classification> [Accessed 11 September 2023].

Bayat, O., Aljawarneh, S., Carlak, H.F., International Association of Researchers, Institute of Electrical and Electronics Engineers and Akdeniz Üniversitesi, n.d. *Proceedings of 2017 International Conference on Engineering & Technology (ICET'2017) : Akdeniz University, Antalya, Turkey, 21-23 August, 2017*.

Bellini, M., Pantalos, G., Kaspar, P., Knoll, L. and De-Michielis, L., 2021. An Active Deep Learning Method for the Detection of Defects in Power Semiconductors. In: *2021 32nd Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*. [online] IEEE. pp.1–5. https://doi.org/10.1109/ASMC51741.2021.9435657.

Brownlee, J., 2021. *A Gentle Introduction to Threshold-Moving for Imbalanced Classification*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/> [Accessed 13 September 2023].

Chen, X., Chen, J., Han, X., Zhao, C., Zhang, D., Zhu, K. and Su, Y., 2020. A Light-Weighted CNN Model for Wafer Structural Defect Detection. *IEEE Access*, [online] 8, pp.24006–24018. https://doi.org/10.1109/ACCESS.2020.2970461.

Chollet, F., 2016. Xception: Deep Learning with Depthwise Separable Convolutions. [online] Available at: <http://arxiv.org/abs/1610.02357>.

Devika, B. and George, N., 2019a. Convolutional Neural Network for Semiconductor Wafer Defect Detection. In: *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. [online] IEEE. pp.1–6. https://doi.org/10.1109/ICCCNT45670.2019.8944584.

Devika, B. and George, N., 2019b. Convolutional Neural Network for Semiconductor Wafer Defect Detection. In: *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. [online] IEEE. pp.1–6. https://doi.org/10.1109/ICCCNT45670.2019.8944584.

Ding, R., Dai, L., Li, G. and Liu, H., 2019. TDD-net: a tiny defect detection network for printed circuit boards. *CAAI Transactions on Intelligence Technology*, [online] 4(2), pp.110–116. https://doi.org/10.1049/trit.2019.0019.

Donges, N., 2022. *What Is Transfer Learning? Exploring the Popular Deep Learning Approach.* Built In.

Haddad, B.M., Yang, S., Karam, L.J., Ye, J., Patel, N.S. and Braun, M.W., 2018a. Multifeature, Sparse-Based Approach for Defects Detection and Classification in Semiconductor Units. *IEEE Transactions on Automation Science and Engineering*, [online] 15(1), pp.145–159. https://doi.org/10.1109/TASE.2016.2594288.

Haddad, B.M., Yang, S., Karam, L.J., Ye, J., Patel, N.S. and Braun, M.W., 2018b. Multifeature, Sparse-Based Approach for Defects Detection and Classification in Semiconductor Units. *IEEE Transactions on Automation Science and Engineering*, [online] 15(1), pp.145–159. https://doi.org/10.1109/TASE.2016.2594288.

He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. [online] Available at: <http://arxiv.org/abs/1512.03385>.

Huang, H., Tang, X., Wen, F. and Jin, X., 2022. Small object detection method with shallow feature fusion network for chip surface defect detection. *Scientific Reports*, [online] 12(1), p.3914. https://doi.org/10.1038/s41598-022-07654-x.

Huang, S.-H. and Pan, Y.-C., 2015. Automated visual inspection in the semiconductor industry: A survey. *Computers in Industry*, [online] 66, pp.1–10. https://doi.org/https://doi.org/10.1016/j.compind.2014.10.006.

Khalilian, S., Hallaj, Y., Balouchestani, A., Karshenas, H. and Mohammadi, A., 2020. PCB Defect Detection Using Denoising Convolutional Autoencoders. In: *2020 International Conference on Machine Vision and Image Processing (MVIP)*. [online] IEEE. pp.1–5. https://doi.org/10.1109/MVIP49855.2020.9187485.

Kim, J., Ko, J., Choi, H. and Kim, H., 2021. Printed Circuit Board Defect Detection Using Deep Learning via A Skip-Connected Convolutional Autoencoder. *Sensors*, [online] 21(15), p.4968. https://doi.org/10.3390/s21154968.

Lu, Z., He, Q., Xiang, X. and Liu, H., 2018. Defect detection of PCB based on Bayes feature fusion. *The Journal of Engineering*, [online] 2018(16), pp.1741–1745. https://doi.org/10.1049/joe.2018.8270.

Mukherjee, S., 2022. The Annotated ResNet-50. *Towards Data Science*. [online] 18 Aug. Available at: <https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758> [Accessed 10 April 2023].

Nakagaki, R., Honda, T. and Nakamae, K., 2009. Automatic recognition of defect areas on a semiconductor wafer using multiple scanning electron microscope images. *Measurement Science and Technology*, [online] 20(7), p.75503. https://doi.org/10.1088/0957-0233/20/7/075503.

Nyuytiymbiy, K., 2020. *Parameters and Hyperparameters in Machine Learning and Deep Learning*. Medium.

Raihan, F. and Ce, W., 2017. PCB defect detection USING OPENCV with image subtraction method. In: *2017 International Conference on Information Management and Technology (ICIMTech)*. [online] IEEE. pp.204–209. https://doi.org/10.1109/ICIMTech.2017.8273538.

Rosebrock, A., 2017. *ImageNet: VGGNet, ResNet, Inception, and Xception with Keras*. [online] pyimagesearch. Available at: <https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/> [Accessed 10 April 2023].

Schlosser, T., Beuth, F., Friedrich, M. and Kowerko, D., 2019. A Novel Visual Fault Detection and Classification System for Semiconductor Manufacturing Using Stacked Hybrid Convolutional Neural Networks. In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. [online] IEEE. pp.1511–1514. https://doi.org/10.1109/ETFA.2019.8869311.

Serrels, K.A., Renner, M.K. and Reid, D.T., 2010. Optical coherence tomography for non-destructive investigation of silicon integrated-circuits. *Microelectronic Engineering*, [online] 87(9), pp.1785–1791. https://doi.org/10.1016/j.mee.2009.10.011.

Simonyan, K. and Zisserman, A., 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. [online] Available at: <http://arxiv.org/abs/1409.1556>.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2014. Going Deeper with Convolutions. [online] Available at: <http://arxiv.org/abs/1409.4842>.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z., 2015. Rethinking the Inception Architecture for Computer Vision. [online] Available at: <http://arxiv.org/abs/1512.00567>.

T, A.N., 2023. *Inception V3 Model Architecture*. [online] OpenGenus. Available at: <https://iq.opengenus.org/inception-v3-model-architecture/> [Accessed 10 April 2023].

Tello, G., Al-Jarrah, O.Y., Yoo, P.D., Al-Hammadi, Y., Muhaidat, S. and Lee, U., 2018. Deep-Structured Machine Learning Model for the Recognition of Mixed-Defect Patterns in Semiconductor Fabrication Processes. *IEEE Transactions on Semiconductor Manufacturing*, [online] 31(2), pp.315–322. https://doi.org/10.1109/TSM.2018.2825482.

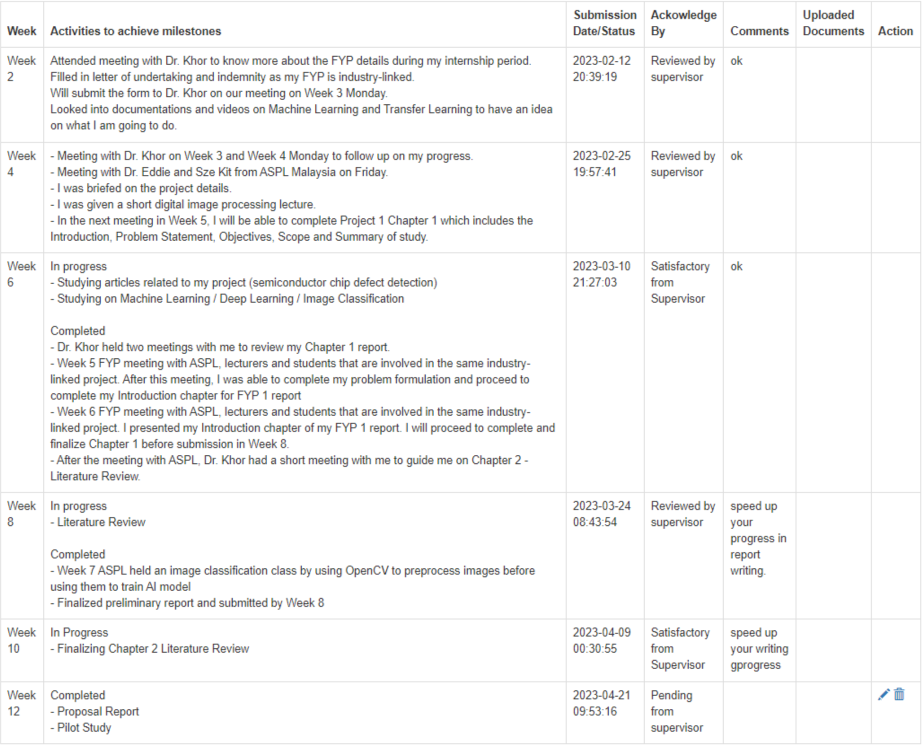
Wolfewicz, A., 2023. *Deep Learning vs. Machine Learning – What's The Difference?* [online] Levity. Available at: <https://levity.ai/blog/difference-machine-learning-deep-learning#:~:text=Machine%20Learning%20means%20computers%20learning,documents%2C%20images%2C%20and%20text> [Accessed 10 April 2023].

Xin, H., Chen, Z. and Wang, B., 2021. PCB Electronic Component Defect Detection Method based on Improved YOLOv4 Algorithm. *Journal of Physics: Conference Series*, [online] 1827(1), p.012167. https://doi.org/10.1088/1742-6596/1827/1/012167.

Zeng, N., Wu, P., Wang, Z., Li, H., Liu, W. and Liu, X., 2022. A Small-Sized Object Detection Oriented Multi-Scale Feature Fusion Approach With Application to Defect Detection. *IEEE Transactions on Instrumentation and Measurement*, [online] 71, pp.1–14. https://doi.org/10.1109/TIM.2022.3153997.

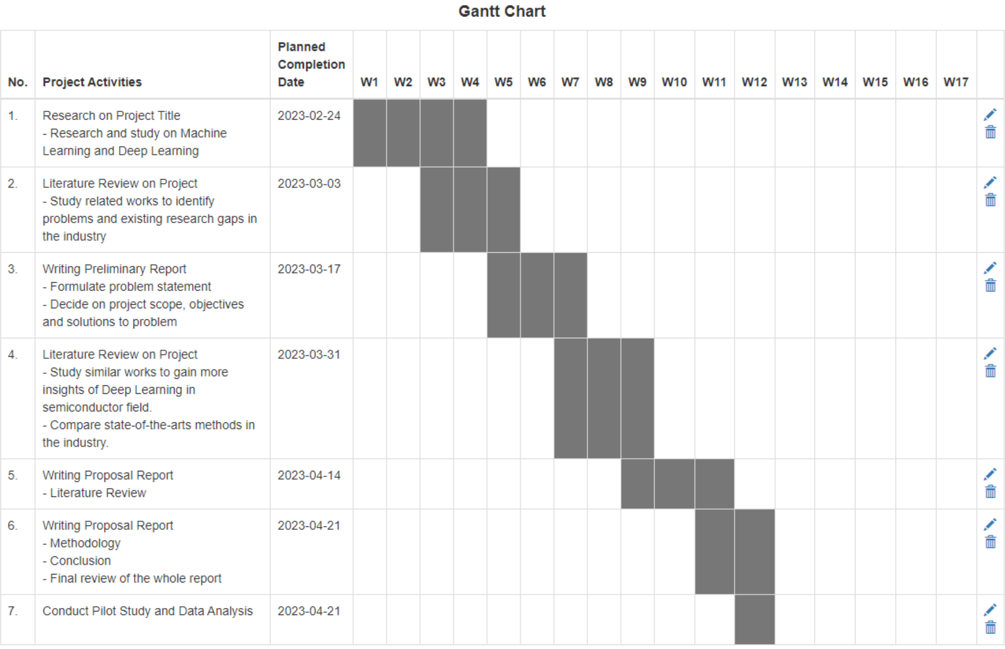
Zhang, C., Shi, W., Li, X., Zhang, H. and Liu, H., 2018. Improved bare PCB defect detection approach based on deep feature learning. *The Journal of Engineering*, [online] 2018(16), pp.1415–1420. https://doi.org/10.1049/joe.2018.8275.

APPENDICES



Week	Activities to achieve milestones	Submission Date/Status	Ackowledge By	Comments	Uploaded Documents	Action
Week 2	Attended meeting with Dr. Khor to know more about the FYP details during my internship period. Filled in letter of undertaking and indemnity as my FYP is industry-linked. Will submit the form to Dr. Khor on our meeting on Week 3 Monday. Looked into documentations and videos on Machine Learning and Transfer Learning to have an idea on what I am going to do.	2023-02-12 20:39:19	Reviewed by supervisor	ok		
Week 4	- Meeting with Dr. Khor on Week 3 and Week 4 Monday to follow up on my progress. - Meeting with Dr. Eddie and Sze Kit from ASPL Malaysia on Friday. - I was briefed on the project details. - I was given a short digital image processing lecture. - In the next meeting in Week 5, I will be able to complete Project 1 Chapter 1 which includes the Introduction, Problem Statement, Objectives, Scope and Summary of study.	2023-02-25 19:57:41	Reviewed by supervisor	ok		
Week 6	In progress - Studying articles related to my project (semiconductor chip defect detection) - Studying on Machine Learning / Deep Learning / Image Classification Completed - Dr. Khor held two meetings with me to review my Chapter 1 report. - Week 5 FYP meeting with ASPL, lecturers and students that are involved in the same industry-linked project. After this meeting, I was able to complete my problem formulation and proceed to complete my Introduction chapter for FYP 1 report - Week 6 FYP meeting with ASPL, lecturers and students that are involved in the same industry-linked project. I presented my Introduction chapter of my FYP 1 report. I will proceed to complete and finalize Chapter 1 before submission in Week 8. - After the meeting with ASPL, Dr. Khor had a short meeting with me to guide me on Chapter 2 - Literature Review.	2023-03-10 21:27:03	Satisfactory from Supervisor	ok		
Week 8	In progress - Literature Review Completed - Week 7 ASPL held an image classification class by using OpenCV to preprocess images before using them to train AI model - Finalized preliminary report and submitted by Week 8	2023-03-24 08:43:54	Reviewed by supervisor	speed up your progress in report writing.		
Week 10	In Progress - Finalizing Chapter 2 Literature Review	2023-04-09 00:30:55	Satisfactory from Supervisor	speed up your writing gprogress		
Week 12	Completed - Proposal Report - Pilot Study	2023-04-21 09:53:16	Pending from supervisor			✎ 🗑

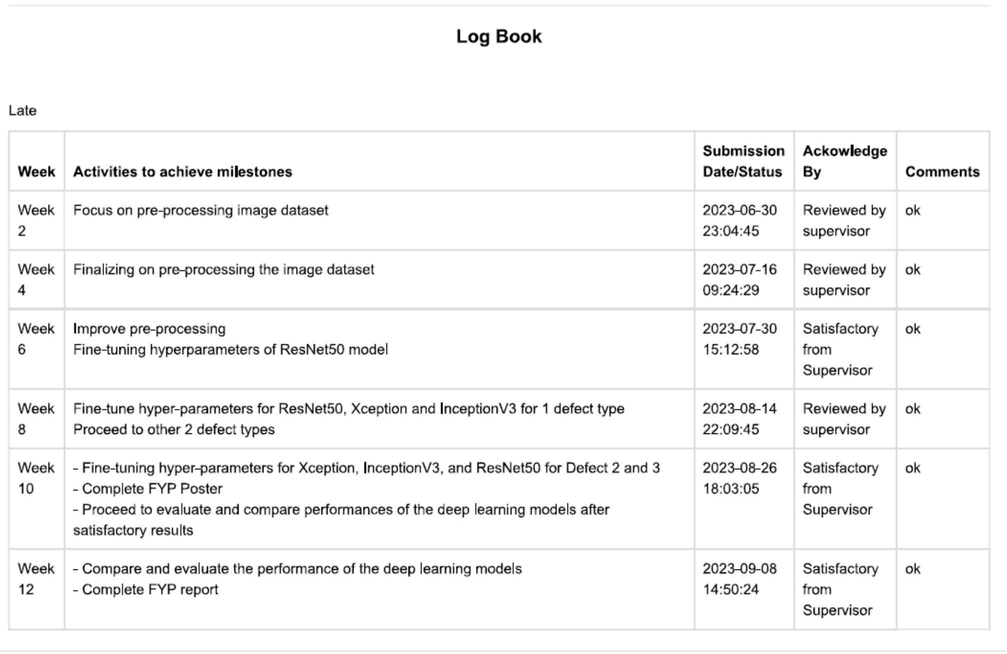
Appendix 1: FYP 1 Log Book



Gantt Chart

No.	Project Activities	Planned Completion Date	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17	
1.	Research on Project Title - Research and study on Machine Learning and Deep Learning	2023-02-24	▨	▨	▨	▨														✎ 🗑
2.	Literature Review on Project - Study related works to identify problems and existing research gaps in the industry	2023-03-03			▨	▨	▨													✎ 🗑
3.	Writing Preliminary Report - Formulate problem statement - Decide on project scope, objectives and solutions to problem	2023-03-17					▨	▨	▨											✎ 🗑
4.	Literature Review on Project - Study similar works to gain more insights of Deep Learning in semiconductor field. - Compare state-of-the-arts methods in the industry.	2023-03-31							▨	▨	▨									✎ 🗑
5.	Writing Proposal Report - Literature Review	2023-04-14									▨	▨	▨							✎ 🗑
6.	Writing Proposal Report - Methodology - Conclusion - Final review of the whole report	2023-04-21											▨	▨						✎ 🗑
7.	Conduct Pilot Study and Data Analysis	2023-04-21												▨						✎ 🗑

Appendix 2: FYP 2 Gantt Chart

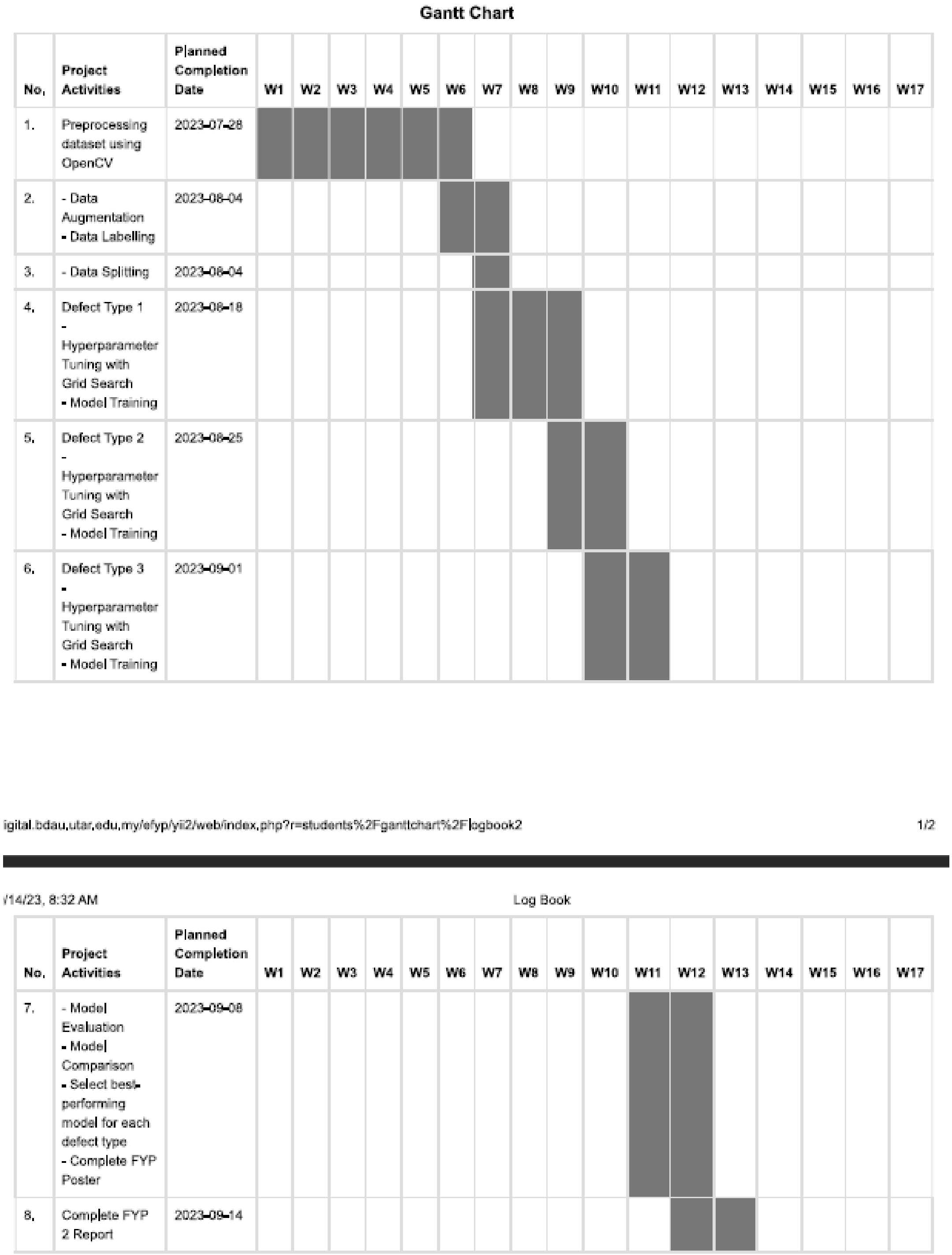


Log Book

Late

Week	Activities to achieve milestones	Submission Date/Status	Ackowledge By	Comments
Week 2	Focus on pre-processing image dataset	2023-06-30 23:04:45	Reviewed by supervisor	ok
Week 4	Finalizing on pre-processing the image dataset	2023-07-16 09:24:29	Reviewed by supervisor	ok
Week 6	Improve pre-processing Fine-tuning hyperparameters of ResNet50 model	2023-07-30 15:12:58	Satisfactory from Supervisor	ok
Week 8	Fine-tune hyper-parameters for ResNet50, Xception and InceptionV3 for 1 defect type Proceed to other 2 defect types	2023-08-14 22:09:45	Reviewed by supervisor	ok
Week 10	- Fine-tuning hyper-parameters for Xception, InceptionV3, and ResNet50 for Defect 2 and 3 - Complete FYP Poster - Proceed to evaluate and compare performances of the deep learning models after satisfactory results	2023-08-26 18:03:05	Satisfactory from Supervisor	ok
Week 12	- Compare and evaluate the performance of the deep learning models - Complete FYP report	2023-09-08 14:50:24	Satisfactory from Supervisor	ok

Appendix 3: FYP 2 Log Book



Gantt Chart

No.	Project Activities	Planned Completion Date	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17
1.	Preprocessing dataset using OpenCV	2023-07-28	■	■	■	■	■	■											
2.	- Data Augmentation - Data Labelling	2023-08-04						■	■										
3.	- Data Splitting	2023-08-04							■										
4.	Defect Type 1 - Hyperparameter Tuning with Grid Search - Model Training	2023-08-18							■	■	■								
5.	Defect Type 2 - Hyperparameter Tuning with Grid Search - Model Training	2023-08-25									■	■							
6.	Defect Type 3 - Hyperparameter Tuning with Grid Search - Model Training	2023-09-01										■	■						
7.	- Model Evaluation - Model Comparison - Select best-performing model for each defect type - Complete FYP Poster	2023-09-08											■	■					
8.	Complete FYP 2 Report	2023-09-14												■	■				

igital.bdau.utar.edu.my/efyp/yii2/web/index.php?r=students%2Fganttchart%2Flogbook2 1/2

/14/23, 8:32 AM Log Book

Appendix 4: FYP 2 Gantt Chart