

May
2023

Investigating Deep Neural Networks in the Context of Surface Defect Detection in Metal Casting

DISSERTATION

Word count: ≈14400

Theodore Smith Scott

Student ID: 3905438

Honours Computer Science Project

BSc Hons Computer Science

School of Engineering

London South Bank University

Declaration

This dissertation is my own original work and has not been submitted elsewhere in fulfilment of the requirements of this or any other award. Any passages taken from my own previous work or other people's work have been quoted and acknowledged by clear referencing to author, source and page(s). Any non-original illustrations are also referenced. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this dissertation and the degree as a whole.

A handwritten signature in black ink, appearing to read 'Theodore Smith Scott', with a long horizontal stroke extending to the right.

Theodore Smith Scott

Abstract

This report proposes a faster, more efficient approach to existing automated surface defect detection architectures, using a deep learning model based on R-CNN. The efficacy of machine learning and deep neural network models is highly reliant on being provided with a high-quality and representative dataset, therefore the research carried out in this report also proposes the use of relevant data pre-processing methods for the model to produce accurate results. The testing and modelling are carried out in the context of metal casting in submersible pump impellers. In addition to the production of a model, there is a comparison to existing machine learning architectures and a proof of concept in the form of a web-based graphical user interface that demonstrates the model's applicability to the industry.

Acknowledgements

First and foremost, I would like to extend my heartfelt gratitude to my supervisor, Dr. Bugra Alkan, whose invaluable guidance and support have been instrumental in making this project possible. I am immensely grateful for their expertise, encouragement, and unwavering dedication to my growth and development.

I would also like to express my deep appreciation to my incredible girlfriend for her unwavering support and understanding throughout my academic journey. Her encouragement and love have been a constant source of motivation, and I am truly fortunate to have her steadfast belief in my potential.

Furthermore, I would like to extend my thanks to my family and friends for their continuous support and encouragement. Their presence and positive influence have been instrumental in keeping me motivated and maintaining a positive mindset throughout my studies.

I am truly fortunate to have such a remarkable support system, and I am grateful for the opportunities and encouragement they have provided me along the way.

Table of Contents

Declaration	<i>i</i>
Abstract	<i>ii</i>
Acknowledgements	<i>iii</i>
Table of Contents	<i>iv</i>
List of Figures	<i>viii</i>
List of Tables	<i>x</i>
Glossary	<i>xi</i>
1. Introduction	1
1.1 Background	1
1.2 Research Motivation	2
1.3 Research Questions	3
1.4 Aim and Objectives	3
1.4.1 Project Aim	3
1.4.2 Objectives.....	4
1.5 Research Scope	4
1.6 Report Structure	5
2. Literature Review	6
2.1 Image Processing	6
2.1.1 Low-Level Processes	7
2.1.2 Mid-Level Processes	7
2.1.3 High-Level Processes	8
2.2 Data Pre-processing	8
2.3 Feature Extraction Methods	9
2.4 Surface Defect Detection Methods	11
2.4.1 Traditional Methods.....	12
Statistical Methods.....	12
Filter-based (Spectral) Methods.....	12
Model-based Methods	13
Learning-based Methods.....	13
2.4.2 Deep Learning-based Methods	13
2.5 Surface Defect Detection Applications	14
2.6 Existing Deep Learning Models	15
2.6.1 VGG16 vs Resnet50 [29]	15

2.6.2	VGG16 vs InceptionV3 vs ResNet50 [30]	18
2.7	Gap Analysis	20
3.	Technical Review.....	21
3.1	Data Acquisition Tools.....	21
3.1.1	Hardware Components	22
	Analogue-to-Digital Converter	22
	Sensors	22
3.1.2	Software Components.....	22
	WINDAQ.....	22
	ActiveX.....	22
3.2	Data Analytics Tools & Platforms (Backend).....	23
3.2.1	Open Source	23
	Python vs R (Programming Language).....	23
	OpenCV.....	23
	TensorFlow vs PyTorch	24
	NumPy vs Pandas	24
3.2.2	Commercial	25
	MATLAB.....	25
	LabelBox	25
3.3	Front-end Dashboard and Visualization Platforms.....	25
3.3.1	Flask.....	26
3.3.2	FastAPI.....	26
3.3.3	Django	26
3.4	Comparison of Front-end Technologies	27
4.	Research Methodology.....	28
4.1	Overview	28
4.1.1	Research	28
	Dataset	28
	Pre-processing.....	30
4.1.2	Development.....	30
	Action Plan	31
	Risk Assessment	32
4.2	Requirements	33
4.2.1	Model	33
4.2.2	Front-end Application	34
4.3	ML Models	36
4.3.1	VGG-16	36
4.3.2	InceptionV3	37
4.3.3	ResNet50	37
4.4	Performance Metrics.....	38

4.4.1	Accuracy	39
4.4.2	Precision	40
4.4.3	Recall	40
4.4.4	F1-Score.....	40
4.4.5	Summary of Metrics	41
4.4.6	Implementation.....	41
4.5	Hyperparameter Tuning	46
4.5.1	Manual Search.....	46
4.5.2	Automatic Search	47
	Grid Search	47
	Random Search	47
	Bayesian Optimization.....	48
	Evaluation of Techniques	48
4.5.3	Implementation.....	49
4.6	Sensitivity Analysis	52
4.6.1	Error Detection and Model Calibration	52
4.7	Activation Map	53
4.7.1	CAM.....	54
4.7.2	Grad-CAM.....	54
4.8	Front-end Design.....	55
4.8.1	Implementation	57
5.	Results and Discussion.....	64
5.1	Performance Analysis of ML Models	64
5.1.1	Experimental Comparison	64
	Dataset Split	64
	Input Image Dimensions.....	65
	Results (224x224 Images).....	66
	Discussion	67
	Results (128x128 Images).....	68
	Discussion	70
	Model File Sizes	71
5.2	Hyperparameter Tuning Results.....	72
5.2.1	Selected Hyperparameters.....	72
	Learning Rate.....	72
	Dropout Rate	72
	Adam Optimizer	73
	Justification for Selected Hyperparameters	73
5.2.2	Tuning Attempt 1	73
5.2.3	Tuning Attempt 2	75
5.2.4	Tuning Attempt 3 (Final)	77
	Result Verification	79
	Graph Results Discussion.....	82

5.3	GUI and User Experience	83
5.4	Discussion	87
6.	Conclusion and Future Work.....	89
6.1	Verification of the Study.....	89
6.2	Limitations	90
6.2.1	Unmet Requirements	90
6.2.2	Additional Limitations	91
6.3	Future Work.....	91
6.4	Conclusion	92
7.	References	93
Appendix	102
Miscellaneous	102
	Ethics Form	102
	Project Directory Structure	103
	requirements.txt	104
Application Code	106
	app.py	106
	localization.py	108
	train.py.....	110
	main.py	112
	index.html	113
	style.css.....	116
Experiment Code	117
	comparison.py	117
	hyperparameters.py	123
	graph.py.....	126
	dataset.py	127

List of Figures

Figure 1 - Contrast Enhancement Example using Histogram Equalization [89]	6
Figure 2 - Section of a Table Showing DL based Applications with Pre-processing Techniques [12]	9
Figure 3 - Feature Extraction Demonstration [91]	10
Figure 4 - Traditional (a) vs. Deep Learning (b) Computer Vision Workflow [18]	11
Figure 5 - Basic CNN Network Architecture [92]	14
Figure 6 - VGG16 vs ResNet50 Recall Table [29]	16
Figure 7 - VGG16 vs ResNet50 Precision Table [29]	16
Figure 8 - Expression Dataset Distribution [29]	18
Figure 9 - COVID-19 and Pneumonia Model Performance [30]	18
Figure 10 - Dataset Sample	29
Figure 11 - Jira Roadmap	31
Figure 12 - VGG-16 Architecture [98]	36
Figure 13 - InceptionV3 Architecture [99]	37
Figure 14 - ResNet50 Architecture [100]	38
Figure 15 - Confusion Matrix [56]	39
Figure 16 - train_model() Function	42
Figure 17 - build_model() Function	43
Figure 18 - build_custom_model() Function	44
Figure 19 - Model Evaluation Call	44
Figure 20 - evaluate_model() Function	45
Figure 21 - calculate_rates() Function	45
Figure 22 - hyperparameters.py build_model() Function	49
Figure 23 - hyperparameters.py optimize_model() Function	50
Figure 24 - hyperparameters.py Invokation	51
Figure 25 - Shoe Activation Map [101]	53
Figure 26 - Frontend Application Flowchart	55
Figure 27 - GUI Wireframe	56
Figure 28 - Implementation Default Endpoint	57
Figure 29 - Bootstrap and Custom CSS Link Tags	57
Figure 30 - Custom CSS Class	57
Figure 31 - Implementation Template Body	58
Figure 32 - JQuery 'on file change' Logic	58
Figure 33 - Implementation Template JQuery	59
Figure 34 - Implementation '/process-image' Check	60
Figure 35 - Implementation '/process-image' File Saving and Processing	60

Figure 36 - Implementation '/process-image' garbage_collection() Function	60
Figure 37 - localization.py 'is_defective()' Function	61
Figure 38 - Implementation '/process-image' Response	62
Figure 39 - localization.py 'save_activation_map()' Function.....	63
Figure 40 - Comparison Results Custom Model (224x224)	66
Figure 41 - Comparison Results VGG16 (224x224).....	66
Figure 42 - Comparison Results InceptionV3 (224x224)	66
Figure 43 - Comparison Results ResNet50 (224x224).....	67
Figure 44 - Comparison Results Custom Model (128x128)	68
Figure 45 - Comparison Results VGG16 (128x128).....	68
Figure 46 - Comparison Results InceptionV3 (128x128).....	69
Figure 47 - Comparison Results ResNet50 (128x128).....	69
Figure 48 - Model File Sizes with 128x128 Input.....	71
Figure 49 - Model File Sizes with 224x224 Input.....	71
Figure 50 - Hyperparameter Tuning Ranges (Attempt 1)	74
Figure 51 - Hyperparameter Tuning Output (Attempt 1)	74
Figure 52 - Hyperparameter Tuning Ranges (Attempt 2)	75
Figure 53 - Hyperparameter Tuning Output (Attempt 2)	76
Figure 54 - Hyperparameter Tuning Ranges (Attempt 3)	77
Figure 55 - Hyperparameter Tuning Output (Attempt 3)	78
Figure 56 - Final Training Output	79
Figure 57 - Training and Validation Accuracy Plot	80
Figure 58 - Training and Validation Loss Plot.....	81
Figure 59 - Implementation Landing Page.....	83
Figure 60 - Implementation GUI Loading GIF	83
Figure 61 - Implementation GUI Error Message	84
Figure 62 - Implementation GUI Result (Not Defective)	85
Figure 63 - Implementation GUI Result (Defective)	86
Figure 64 - Implementation Performance Logs	87

List of Tables

Table 1 - Front-end Framework Comparison	27
Table 2 - Risk Assessment	32
Table 3 - Model Requirements	33
Table 4 - Front-end Requirements	34
Table 5 - Comparison Results Table (224x224)	67
Table 6 - Comparison Results Table (128x128)	69
Table 7 - Initial Hyperparameters	74
Table 8 - Comparison between Tuning Attempts 1 and 2	76
Table 9 - Comparison between Tuning Attempts 1, 2, and 3	78
Table 10 - Unmet Requirements	90

Glossary

- **R-CNN** Region with Convolutional Neural Network
- **CNN** Convolutional Neural Network
- **ANN** Artificial Neural Network
- **DNN** Deep Neural Network
- **SPI** Submersible Pump Impeller
- **AI** Artificial Intelligence
- **CV** Computer Vision
- **LCD** Liquid Crystalline Display
- **LLP** Low-Level Process
- **MLP** Mid-Level Process
- **HLP** High-Level Process
- **SLAM** Simultaneous Localization and Mapping
- **ML** Machine Learning
- **DL** Deep Learning
- **AVI** Automated Visual Inspection
- **YOLO** You Only Look Once
- **CAM** Class Activation Mapping
- **TP** True Positive (rate)
- **TN** True Negative (rate)
- **FP** False Positive (rate)

-
- **FN** False Negative (rate)
 - **SSD** Single Shot Detector

1. Introduction

The main output of this project is to develop and train a deep learning model based on the existing R-CNN architecture, which can process images of submersible pump impellers and identify, categorize, and localize casting defects.

1.1 Background

For approximately every thousand tons of metallic castings, seventy-five tons must be re-melted due to defects [1]. The average lifespan of a metal is less than 10 years. Metal production accounts for roughly 8% of global greenhouse gas emissions [2], so detecting defects earlier on in the manufacturing process means: most metals won't need to be re-melted and cycled through the process again. This reduces greenhouse gas emissions, metal wastage, and the necessity to mine more. Accordingly, Helbig states: "the longer we use metals, the less we need to mine."

Using this artificial intelligence (AI), defects are detected earlier on, which will significantly reduce the chance of cast metals being lost to landfills or recycling plants, which currently accounts for 84% of cumulative metal loss globally [2].

In comparison to surface defect detection of other surfaces, such as liquid crystalline displays (LCDs), accurate detection of defects on metallic surfaces is much more prone to error, due to the high reflectance, resulting in skewed outcomes by slight changes in lighting conditions [3]. As a result, more advanced detection methods must be incorporated in place of traditional image processing techniques to increase precision and reliability across a wider range of environmental conditions.

According to Tabernik et al. (2020), deep learning techniques have become the best approach for this task. The leading factors are accuracy and the reduced need for domain knowledge required to identify and categorise anomalies in images. Despite their overall suitability in surface detection problems; the processing power and storage requirements of convolutional neural networks (CNNs) are significantly higher, majorly limiting hardware flexibility [4]. The mechanism is a ‘black box’, which makes it next to impossible to accurately troubleshoot specific issues.

Although in some domains – such as data mining applications with a high monetary stake (i.e., banking) – the lack of transparency in artificial neural networks (ANNs) is unacceptable. However, in the context of the scope of this report, this is not an issue as there is no chance of the architecture causing further revenue loss; its effectiveness is measured by its minimisation of loss – as will be discussed in Aims and Objectives.

1.2 Research Motivation

It is no secret that AI is one of the most exciting and controversial technological advancements of the 21st century. As of 2022, the global AI market is valued at over \$136 billion. This is projected to increase by over thirteen-times over the next 8 years. By 2025, as many as ninety-seven million people will work in the AI sector, and 83% of companies claim that AI is a top priority in their business plan. Netflix alone makes \$1 billion annually from automated personalised recommendations [5].

Many industries are trying to adopt Industry 4.0 and AI techniques to improve their production quality and efficiency. The process of quality control has remained unchanged for decades until recent developments in automated defect detection solutions. The introduction of systems

utilising more advanced technologies has greatly improved the accuracy of anomaly detection in manufacturing; however, these newer systems and their underlying algorithms (as previously mentioned) largely require a high degree of computational power, and a tedious calibration process [6]. Hence in this research, an efficient and lightweight solution is proposed to overcome this challenge.

1.3 Research Questions

This project aims to provide an answer to the following questions:

- How do the accuracy and speed of existing deep learning techniques compare to a novel deep neural network for surface defect detection problems?
- What are the important characteristics of an efficient and user-friendly surface defect detection application?
- How can R-CNN be adapted to surface defect detection to make it faster at an acceptable accuracy?

1.4 Aim and Objectives

1.4.1 Project Aim

The main aim of this project is to reduce the loss in revenue in the metal casting industry caused by casting defects: by providing companies and organisations with a deep learning (DL) architecture, which could be used by employees to quickly scan and log the defects. This data could be collated and statistically analysed to identify the most effective actions to reduce the frequency of the most commonly occurring defects.

1.4.2 Objectives

- **Research existing defect detection solutions:** Investigate and evaluate solutions provided using either traditional techniques or deep learning methods.
- **Obtain and divide a dataset:** Find a relevant dataset (images of the surface of SPIs) with enough samples and divide that into training and validation subsets.
- **Create labels for the dataset:** Using appropriate software, create categories for types of casting defects and manually classify them in images to aid in model training.
- **Create a design for the neural network:** Plan out the number of layers, arrangement, and number of output parameters for the network, based on the R-CNN architecture.
- **Program the network:** Translate the network design to code, using relevant languages, technologies, and modules.
- **Train the network:** Run the neural network using the training dataset as the input.
- **Assess the accuracy of the network:** Run the neural network using the validation dataset as the input and evaluate the accuracy of detection and categorisation.
- **Re-evaluate the design:** Repeat training and testing and modify the design until the accuracy is sufficient.
- **Create a functional front-end:** Provide a graphical user interface for user interaction with the model.

1.5 Research Scope

The project will consist of the development of a deep learning-based architecture derived from training, testing, and validation using a publicly available dataset of images consisting of the front

surface of submersible pump impellers. There will be a web application that will show a live feed of a camera, which will display bounding boxes around the defects with an appropriate categorical label.

1.6 Report Structure

This report will be structured accordingly: a **Literature Review** section, in which existing research on topics within the scope will be discussed and explored; a **Technical Review** section, which will explore and compare relevant technologies for the development of the project; a **Research Methodology** section, which outlines the research and development methodology, as well as defining requirements; a **Results and Discussion** section, providing an overview of the output of the project and discussing findings; and a **Conclusion and Further Work** section, which verifies the study, outlines limitations encountered, and an overview of potential further work.

2. Literature Review

2.1 Image Processing

According to Petrou (2010), the purposes of image processing are image enhancement, compression, restoration, and feature extraction. Apart from compression, these processes all aim to make a digital image more interpretable by either human or automatic analysis [7]. Image enhancement is an important task in computer vision applications, as it provides necessary pre-processing to facilitate the improvement or removal of low-quality data – increasing the effectiveness of further processing such as feature extraction.



Figure 1 - Contrast Enhancement Example using Histogram Equalization [89]

As can be observed in the figure, the contrast of the image on the right has been enhanced. As stated by Patel et al. (2013), histogram equalization is not the most effective method for contrast enhancement, however, the benefit of using similar low-level techniques is readily apparent.

The techniques used to process images can be classified into 3 main categories: low-level, mid-level, and high-level. Low-level processes (LLPs) and Mid-level processes (MLPs) both accept an image as input; however, the output for LLPs is an image, and the output for MLPs is a set of attributes. High-level processes (HLPs), on the other hand, take a set of attributes and output understanding [8].

2.1.1 Low-Level Processes

The purpose of this level of process is to carry out two main divisions of tasks: Image Enhancement, and Image Restoration [9]. Enhancements include operations such as contrast or brightness adjustment, or other techniques focused on highlighting image details, which produce subjective results. Image restoration, on the other hand, outputs an objectively measurable result, as the issue of image degradation can be addressed using a mathematical or probability-based model [10].

2.1.2 Mid-Level Processes

According to Sood, these processes are responsible for Object Recognition and Segmentation; however, this is contradicted by Azimi, who believes these tasks should be categorized as high-level processes. Instead, the tasks belonging to this class are Transforms and Compression [9].

2.1.3 High-Level Processes

These processes are most associated with computer vision (CV), and as such – in accord with Azimi – the tasks at this level are Segmentation, Feature Extraction, and Classification or Recognition, which will be further discussed later in sections 2.3 and 2.4.

2.2 Data Pre-processing

Pre-processing of images for DL applications is necessary to successfully train a network because the characteristics of the defect and non-defect regions are not easily discernible. Due to the presence of noise in unprocessed images, they do not ensure the successful training of a DL model [11]. This is supported by Ranganathan (2021), who suggests that although DL systems are capable of being trained using noisy data, it may have an impact on the applications' accuracy [12].

Several data pre-processing techniques are commonly used in machine learning CV projects: including but not limited to grayscale conversion, normalisation, data augmentation, and image standardization. They are used to achieve a higher accuracy whilst also reducing a model's complexity [13].

Citation	Area	Approach	Preprocessing	Application	Accuracy
Poloni et al., (2021)	Image classification	SVM	Non-Local Means technique	Alzheimer's disease diagnosis	69.44%
Beeravolu et al., (2021)	Image classification	Deep CNN	Sobel filter	Breast cancer classification	99.06%
Wang et al., (2021)	Image classification	Random Implication Image Classifier	Median filter	Iris disease detection	96.7%
Akhter et al., (2021)	Data classification	Multisize Filters CNN	Stemming	Document classification	95.4%
Lichouri et al., (2021)	Data Classification	BiLSTM	Lemmatization, stemming and POS tagging	Sentiment detection	88.29%

Figure 2 - Section of a Table Showing DL based Applications with Pre-processing Techniques [12]

The figure above demonstrates the range of learning techniques, pre-processing techniques, and their respective accuracies. The table illustrates the applicability of pre-processing techniques and how they can be utilised in both image and standard data DL problems.

2.3 Feature Extraction Methods

The goal of feature extraction, a crucial stage in the development of any pattern classification, is to obtain the pertinent data that defines each class and is performed after the data has been pre-processed through techniques previously discussed. This procedure creates feature vectors by extracting appropriate features from detected objects; this feature vector represents the identity of a class of objects [14]. Image features can be divided into local features and global features. They can be defined accordingly [15]:

- **“Local features:** Features calculated over the results of subdivision of the image band on image segmentation or edge detection.”

- “**Global features:** Features calculated over the entire image or just regular sub-area of an image.”

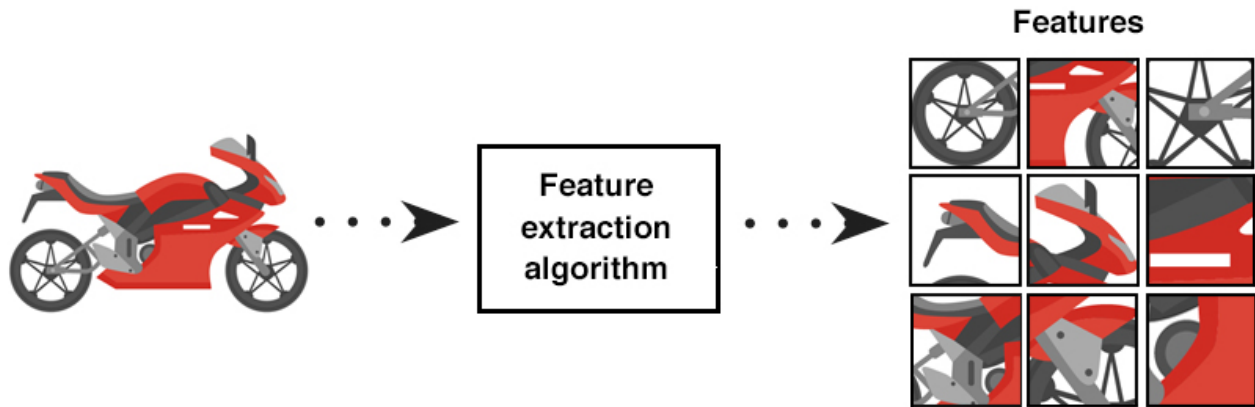


Figure 3 - Feature Extraction Demonstration [91]

According to Kumar & Bhatia (2008), a good feature set includes discriminating data that can separate one object from others. To avoid producing various feature codes for objects belonging to the same class, it must be as stable as possible. The features chosen should be a limited subset whose values effectively distinguish between patterns of various classes while being comparable for patterns belonging to the same class.

There are many feature extraction methods, which must be chosen based on several factors: computational complexity, implementation difficulty, adaptability to translated objects and shapes, etc. [16]. However, these feature extraction methods are only relevant whilst considering traditional analysis. Deep learning is now widely used for image and video analysis, and it has received recognition for being able to analyse raw image data without first extracting any features from it [17]. In this regard, DL solutions provide a large advantage over traditional computer vision systems, in that there is greater accuracy of classification, segmentation, object detection, and

Simultaneous Localization and Mapping (SLAM). Kernels, often referred to as filters, are used by CNNs to identify features, such as edges, throughout a picture [18].

The figure below shows an overview of the difference in workflow comparing traditional CV to deep learning CV.

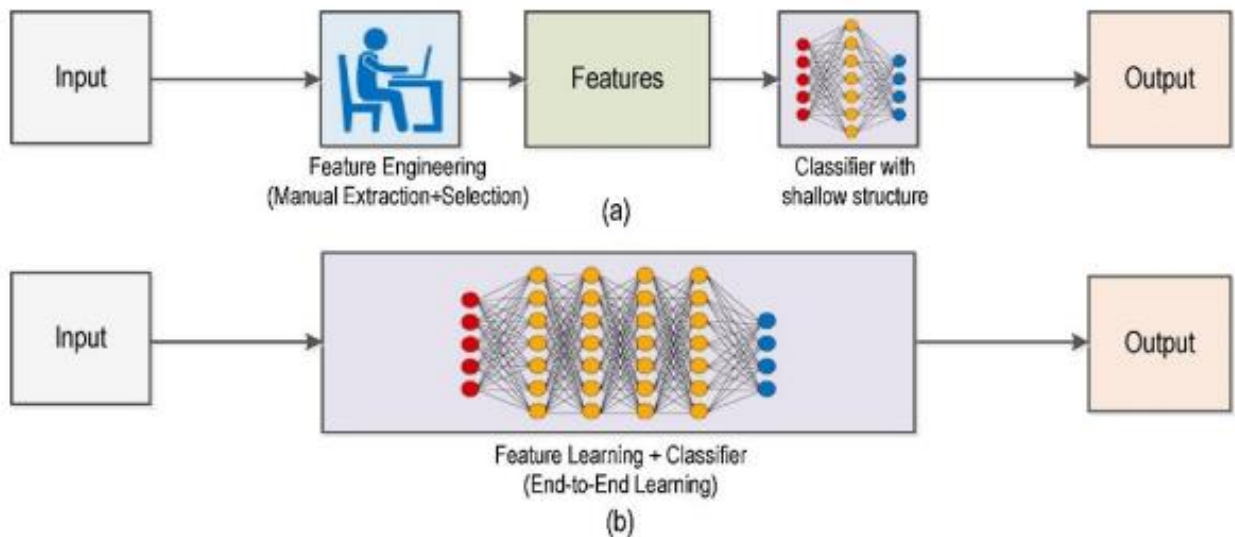


Figure 4 - Traditional (a) vs. Deep Learning (b) Computer Vision Workflow [18]

2.4 Surface Defect Detection Methods

The main goal of surface defect detection – also referred to as automated visual inspection (AVI) – is to use classification techniques to group defects into specified classes [19]. There are two categories of methods that are going to be compared: traditional approaches, and deep learning approaches. The benefits and drawbacks of both will be compared, in addition to their sub-categories.

2.4.1 Traditional Methods

According to Kumar & Bhatia (2008), feature extraction and decision-making techniques should be divided into three groups: statistical, spectral, and model-based. This has more recently been accepted, but modified; Sun et al. (2018), believe the techniques should be divided into statistical methods, filter-based methods, model-based methods, and learning-based methods.

Statistical Methods

The statistical approach involves developing a mathematical model by applying mathematical statistics and probability theory [20]. Histogram properties, co-occurrence matrices, mathematical morphology, and local binary patterns are common statistical techniques (LBP) [21]. These methods have a low computational cost and can be highly accurate, however, the accuracy over a wide range of scenarios can be detrimentally affected by a variety of factors such as grey value, noise, and irregularities in texture [20].

Filter-based (Spectral) Methods

To extract features, these methods mathematically transfer data from the spatial domain to the frequency domain [21]. The filter-based methods considered by Sun et al. (2018) include spatial domain, frequency analysis, Gabor transform, wavelet transform, and multiscale geometric analysis. Modern high-dimensional datasets require the development of faster algorithms, and as a result, spectral approaches have grown in popularity [22].

Model-based Methods

Ren et al. [21] state that “model-based approaches construct representations of images by modeling multiple properties of defects”. The Markov random field (MRF) and the auto-regressive model are the most often used model-based techniques. For modern purposes, auto-progressive models are becoming more favoured due to a lower computational cost compared to non-linear models [21].

Learning-based Methods

Derived from statistical pattern recognition theories, linear support vector machine (SVM) algorithms have been in use for over two decades. Compared to Naïve Bayes and neural networks, the SVM classification approach is thought to be a better option for noisy datasets when considering accuracy and computational complexity [23].

2.4.2 Deep Learning-based Methods

Deep learning approaches have rapidly become the standard for AVI, which can be attributed to the efficiency of its feature extraction capabilities. There are a variety of DL methods that are in use for a range of tasks, however, all the most used rely on CNNs [24]. According to Yang et al. (2019), currently, the most widely used object identification techniques are the Faster R-CNN, You Only Look Once (YOLO), and Single Shot Detector (SSD) – which are all CNN-based. R-CNN is a specific type of CNN that is specialised for object detection in images, as opposed to general image classification [25]; hence it is suitable for defect detection, as defects can be classified as objects.

The figure below outlines the overall structure of a very basic CNN. The main features of a CNN are convolutional layers, pooling layers, fully connected layers, dropout layers, and activation functions [26].

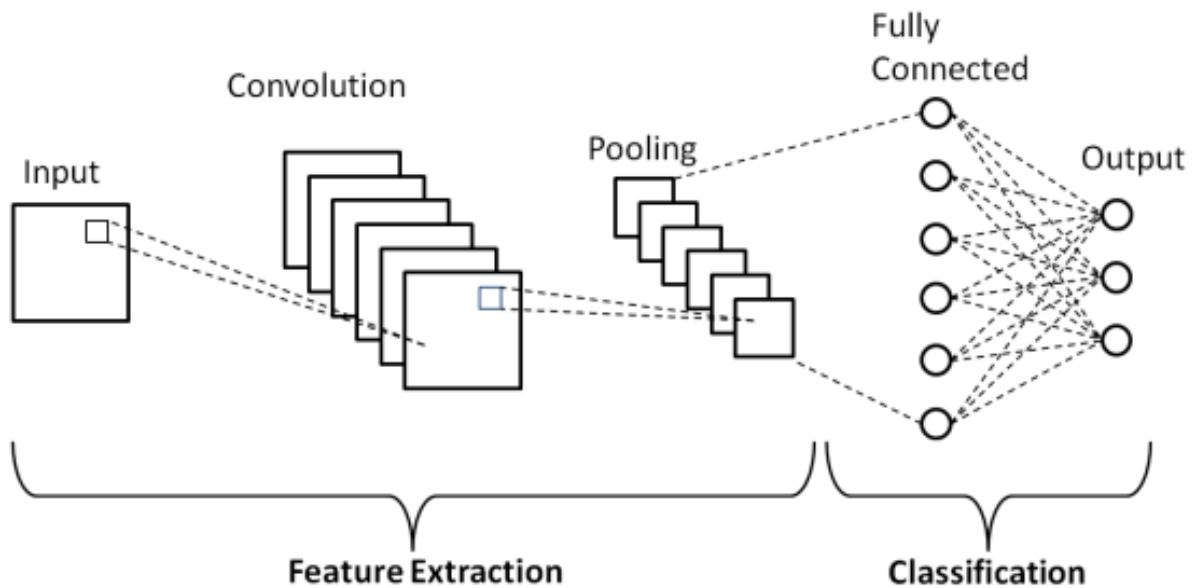


Figure 5 - Basic CNN Network Architecture [92]

2.5 Surface Defect Detection Applications

Overall, the employment of an AVI system reduces the loss of revenue compared to manual inspection not only by reducing the frequency of product returns or waste, but because improving product quality can lead to more sales and revenue, customer confidence and trust, repeat business, and even improved brand reputation [27].

According to Ravikumar et al. (2011), machine vision is mostly used for visual inspection of component surfaces. However, the applications of this technology cover a wide range of domains

and industries. AVI systems have a wide range of usage; in the medical field to detect defects in cardiovascular stents, in food processing to grade lentils, and the manufacturing of glass and plastics, among others [28].

2.6 Existing Deep Learning Models

2.6.1 VGG16 vs Resnet50 [29]

In a study exploring the application of convolutional neural networks (CNNs) for classifying affect states, specifically facial expressions, the performance of two transfer learning models, VGG16 and ResNet50, were compared. The experimental results demonstrated validation accuracies of 96.8% for VGG16 and an impressive 99.47% for ResNet50 [29]. These findings showcase the superior performance of ResNet50 in accurately identifying and classifying facial expressions, indicating its potential as a powerful tool in affect state recognition tasks.

The study also included the calculation of precision and recall as additional performance measures. The corresponding results for precision and recall can be observed in the subsequent figures, providing a comprehensive evaluation of the models' performance in terms of these metrics. These results offer valuable insights into the precision (ability to avoid false positives) and recall (ability to capture all relevant positives) of the models, further enhancing the understanding of their classification capabilities.

	VGG16	ResNet50
Angry	0.99	1.00
Contempt	1.00	1.00
Disgust	0.88	1.00
Fear	0.98	0.95
Happy	0.96	1.00
Sad	0.99	0.99
Surprise	0.99	1.00

Figure 7 - VGG16 vs ResNet50 Precision Table [29]

	VGG16	ResNet50
Angry	0.95	0.99
Contempt	0.93	1.00
Disgust	1.00	0.99
Fear	1.00	1.00
Happy	1.00	1.00
Sad	0.88	0.98
Surprise	0.98	0.91

Figure 6 - VGG16 vs ResNet50 Recall Table [29]

The results presented in the tables demonstrate the overall effectiveness of both models in accurately identifying images from each class. However, a comparison between the two models reveals that VGG16 had a slightly lower performance compared to ResNet50, particularly in terms of false positives.

In terms of precision, VGG16 demonstrated a higher rate of false positives, particularly for the 'disgust' and 'happy' classes, with precision scores of 0.88 and 0.96, respectively. This indicates that VGG16 tended to incorrectly classify negative instances as positive for these specific classes. In contrast, ResNet50 achieved better precision, suggesting a lower rate of false positives for these classes.

Regarding recall, the two models exhibited relatively similar performance, with average recall values of approximately 0.96 for VGG16 and 0.98 for ResNet50. However, it's worth noting that VGG16's recall value for the 'sad' class, scoring 0.88, stands as somewhat of an outlier. This suggests that a lower proportion of 'sad' images were correctly identified by VGG16 compared to other classes.

Considering both precision and recall provides a more comprehensive assessment of the models' performance. While VGG16 struggled with false positives, particularly for 'disgust' and 'happy', it still achieved a relatively high average recall. On the other hand, ResNet50 exhibited better precision and consistent recall rates. These observations shed light on the models' strengths and weaknesses in accurately classifying the different image classes.

A possible explanation for some classes yielding lower rates is an imbalance in image class distribution, as can be seen in the following figure, which contains the number of images belonging to each class.

Affect states	Number of images
Angry	412
Contempt	105
Disgust	325
Fear	220
Happy	604
Sad	313
Surprise	523
Total	2502

Figure 8 - Expression Dataset Distribution [29]

2.6.2 VGG16 vs InceptionV3 vs ResNet50 [30]

A study compares the performance of various models, including InceptionV3, VGG16, and ResNet50 in the classification of COVID-19 and pneumonia [30]. The dataset used for the study consisted of 1536 chest X-ray images depicting COVID-19 cases and 5629 images displaying pneumonia cases. However, it's worth noting that the dataset was unbalanced, with a larger number of pneumonia images. To address this issue, under-sampling techniques were employed to balance the dataset, ensuring fair comparison among the different classes. The results of the study are shown in the figure below and are discussed on the next page.

	Best epoch	Training loss	Training accuracy	Validation loss	Validation accuracy
VGG16	21	0.0100	0.9990	0.0000e+00	1.0000
VGG19	22	0.0122	0.9994	0.0000e+00	1.0000
DenseNet121	20	6.1363e-09	1.0000	7.2103e-07	1.0000
Inception-ResNet-V2	9	0.0022	0.9993	0.0011	1.0000
InceptionV3	3	0.0472	0.9943	0.0554	0.9963
ResNet50	4	1.1188e-04	1.0000	0.0633	0.9982

Figure 9 - COVID-19 and Pneumonia Model Performance [30]

In terms of overall performance, VGG-16 emerged as the top-performing model among the three evaluated models, achieving a remarkable validation accuracy of 100%. In a separate study conducted by Shazia et al. (2021), focusing on the classification of COVID-19, pneumonia, and normal images, VGG16 exhibited the highest overall accuracy, reaching an impressive 95.88%. On the other hand, InceptionV3 and ResNet50 achieved slightly lower validation accuracies of 99.63% and 99.82%, respectively.

Interestingly, all three models demonstrated excellent performance even with a relatively low number of epochs. InceptionV3 and ResNet50 achieved optimal accuracy in just 3 and 4 epochs, respectively, while VGG-16 required the most epochs, specifically 21, to achieve optimal results. Despite the variation in the number of epochs required, each model demonstrated their effectiveness in accurately classifying the images.

These findings highlight the superior performance of VGG-16 in the given classification task and emphasize the efficiency of InceptionV3 and ResNet50 in achieving optimal accuracy with fewer epochs.

2.7 Gap Analysis

As stated by Zheng et al. [21], “manual surface inspection methods performed by quality inspectors have the disadvantages of low efficiency, high labor intensity, low accuracy, low real-time performance, etc.” Manual visual inspection methods necessitate a significant amount of time and are very subjective. AVI approaches are intended to supplement or completely replace human judgement to overcome the limitations of human inspection [31], however, this is yet to be improved; according to Damacharla (2021), due to the training requirements and inaccuracies associated with AVI systems, most steel production sectors continue to use manual visual inspection.

To remedy this situation, an AVI system with the following features must be developed: rapid detection speed, sufficiently high accuracy, a high level of abstraction to reduce the cost of employee training, and an ANN which can be precisely trained with a minimal dataset.

3. Technical Review

The purpose of this technical review is to evaluate and select relevant technologies required for different steps in the development of the DL model and web application. This review has been divided into three main sections: Data Acquisition Tools, Data Analytics Tools & Platforms, and Frontend. However, Data Analytics Tools & Platforms has been further partitioned into Open Source and Commercial options. Using a set of criteria derived from research into relevant technologies, the suitability of each will be contrasted and compared.

3.1 Data Acquisition Tools

The process of measuring aspects of the physical environment, such as pressure, temperature, sound, and electricity, is known as data acquisition [32]. This is accomplished by using a variety of sensors that capture analogue signals from the environment and convert them to digital signals [32]. In the context of machine learning, data acquisition tools relate to sensing hardware and systems used to acquire data to form a dataset, such as a camera. However, due to the scope of this project, it is infeasible to use manual data acquisition methods; it instead utilizes a dataset found on Kaggle, an open dataset repository. Despite this, the components of a data acquisition system are discussed in this section, as they are relevant to the research.

3.1.1 Hardware Components

Analogue-to-Digital Converter

An analogue-to-digital converter (ADC) is a device that transforms analogue signals into digital signals, such as sound captured by a microphone or light entering a digital camera [33]. This component is vital in allowing digital systems to communicate with real-time analogue signals. An ADC provides a data acquisition system with the ability to collect information from the real world, which is usually in the form of a voltage, which is then converted to a binary number [34].

Sensors

Sensors are another important aspect of a data acquisition system. Their role is to convert physical properties such as temperature, pressure, humidity, and light intensity into analogue or digital electrical signals [35]. They are a subset of transducers, which are broadly defined devices that convert energy from one form to another [36].

3.1.2 Software Components

WINDAQ

WINDAQ is software used for real-time data acquisition, signal processing, and data analysis. It provides a platform for acquiring and analysing data from a variety of sources, such as USB and PCI devices which may utilise sensors and analogue-to-digital converters [37].

ActiveX

Microsoft's ActiveX framework enables programmers to construct reusable software components. Numerous applications, including data-acquisition systems, can use these components [38].

3.2 Data Analytics Tools & Platforms (Backend)

This section of the technical review covers all the technologies required to develop & train the model, as well as to develop other elements of the backend such as performance measurement.

3.2.1 Open Source

Python vs R (Programming Language)

R is a programming language specifically purposed for statistical computation and graphics [39], whereas Python is a general-purpose high-level programming language with large extensibility in the form of modules. Python is more versatile and consistent than R due to its broad community support, accessible documentation, and code standardisation. Additionally, since Python is a general-purpose language; it can seamlessly integrate with other development tasks, notably for this project the creation of a web application utilizing one of its robust third-party frameworks. R, on the other hand, despite being a difficult language to learn, offers more models for statistical analysis [40].

OpenCV

OpenCV is a computer vision library compatible with several programming languages, such as C++, Python, and MATLAB [41]. In Python, OpenCV is a module capable of image processing and facilitating CV tasks. In the context of the project, it provides the ability to perform data pre-processing and annotate images for use in the frontend.

TensorFlow vs PyTorch

Both TensorFlow and PyTorch are deep learning frameworks that can utilise a discrete GPU to perform tasks [42]. Due to their ability to use parallel processing, GPUs are more suited to complex tasks such as the training of a deep-learning network. For this project, training will take place on a computer equipped with an NVIDIA RTX 3070, which will significantly reduce training time when combined with a GPU-accelerated framework.

TensorFlow, developed by Google, is a much more mature framework than PyTorch, with superior debugging and visualisation capabilities. PyTorch, on the other hand, takes a more 'pythonic' approach, making it easier for Python developers to use [42]. However, TensorFlow includes a package called Keras, which is a high-level API for deep learning. It can be used to write more concise, readable code while retaining the TensorFlow GPU back-end's power.

NumPy vs Pandas

Through the virtue of locality of reference, NumPy provides a Python array implementation that is 50 times faster than native Python lists. It also provides a set of mathematical functions for image pre-processing, such as the Fourier transform [43]. TensorFlow includes NumPy as an API, meaning NumPy structures can be easily integrated into a TensorFlow environment without adding complexity [44].

Pandas has a broader set of capabilities than NumPy, but it is more focused on delivering insights into big data problems than image data. However, Pandas includes a 'DataFrame' structure, which is a 2D array implementation that could be used in storing image data. However, a NumPy array outperforms Pandas DataFrames when working with lesser than five-hundred-thousand rows [45]. Furthermore, because Pandas lacks TensorFlow integration, NumPy is the superior choice.

3.2.2 Commercial

MATLAB

MATLAB is a high-level programming platform and language designed for engineers and scientists [46]. It has many features and can complete most of the essential tasks; nevertheless, it is unsuitable for this project due to its steep learning curve, and for the sake of integration simplicity, it is a better choice to use only one programming language for development.

LabelBox

LabelBox is a data training platform most typically used for annotating or labelling data [47], which can then be exported and utilised to train a TensorFlow (or other ML) model more quickly.

*In conclusion, the primary back-end technologies used were **Python, OpenCV, TensorFlow, Keras,** and **NumPy**. It was also planned to use **LabelBox**; however, after evaluation, it was deemed infeasible to manually annotate a sufficient-sized subset of a dataset.*

3.3 Front-end Dashboard and Visualization Platforms

As a proof-of-concept, an interface in the form of a barebones web application will be developed. Due to the impracticality of acquiring samples of defective SPI castings, the front-end will be demonstrated using images from the dataset. Since Python is the most suitable language for the project, only Python frameworks and libraries will be considered in this section.

3.3.1 Flask

Flask is a microframework for developing web applications and APIs. It requires little boilerplate code, resulting in a codebase that is very concise and readable, allowing for rapid development. Flask, like Python, has a multitude of third-party libraries for a wide range of tasks, as well as robust community support. It is also extremely fast and comes with a development server, which eliminates the need for a separate web server.

3.3.2 FastAPI

FastAPI is a Python web framework with excellent documentation and features. However, unlike Flask, it lacks an integrated development server, necessitating additional steps to begin programming. Furthermore, its design is focused on producing APIs rather than web apps [48].

3.3.3 Django

Django is a full-stack framework suited to building complex web applications and is widely adopted in commercial applications. It allows for rapid, organized development; however, it is excessive for this project.

3.4 Comparison of Front-end Technologies

The front-end technologies were evaluated based on six criteria: user adoption, community support, project suitability, the inclusion of a development web server, the speed of development, and prior knowledge of the framework. Additionally, these criteria were weighed by determining whether it was deemed required or optional.

Flask and Django had the highest adoption rate, with FastAPI having fewer online resources referencing it, likely because it is a newer framework. In terms of community support, Flask had the highest number of online resources and forum responses compared to the other two frameworks. However, it was found that Flask and FastAPI were more suitable for the required task as they are minimal frameworks, as opposed to Django’s bundled and complex nature. Based on these three required criteria, Flask scored the highest, as can be seen in the table below.

Table 1 - Front-end Framework Comparison

Importance	Description	<i>Flask</i>	<i>Django</i>	<i>FastAPI</i>
<i>Required</i>	Adoption	Highest	Highest	Lower
<i>Required</i>	Community Support	High	Medium	Low
<i>Required</i>	Project Suitability	Highest	Medium	High
<i>Preferred</i>	In-built Web Server	Yes	Yes	No
<i>Preferred</i>	Development Speed	High	High	High
<i>Preferred</i>	Prior Knowledge	Highest	Medium	Medium

*In conclusion, **Flask** was chosen as the front-end framework.*

4. Research Methodology

4.1 Overview

4.1.1 Research

Research for this project primarily utilised an experimental design, however, some analytic methods were also used. In addition to the experimental output, pre-conducted comparisons of ML models have been analysed (in the **Literature Review**) to provide further insight and to guide the development of the model.

The goal of the research was to create a model which is sufficiently accurate whilst providing an adequate level of efficiency, with relevant performance measures and visualisations used to evaluate and improve.

Dataset

The dataset used for model training, development, and comparison was obtained externally from Kaggle [49]. This dataset played a crucial role in conducting the research and gathering the research output. It consists of 1300 images, each with a resolution of 512x512 pixels, representing the front surface of submersible pump impellers. Out of the total images, 781 of them are labelled as 'defective', indicating that these impellers have some kind of damage or flaw. The remaining 519 images are labelled as 'ok', implying that these impellers are in good condition without any noticeable defects. The dataset encompasses a diverse range of defects that can occur in SPIs. These defects can include various types of damages or abnormalities that affect the impeller's functionality or performance. Examples of such defects may include cracks, erosion, corrosion,

wear, imbalance, or any other anomalies that could potentially impair the impeller's efficiency, as can be ascertained from the figure below.

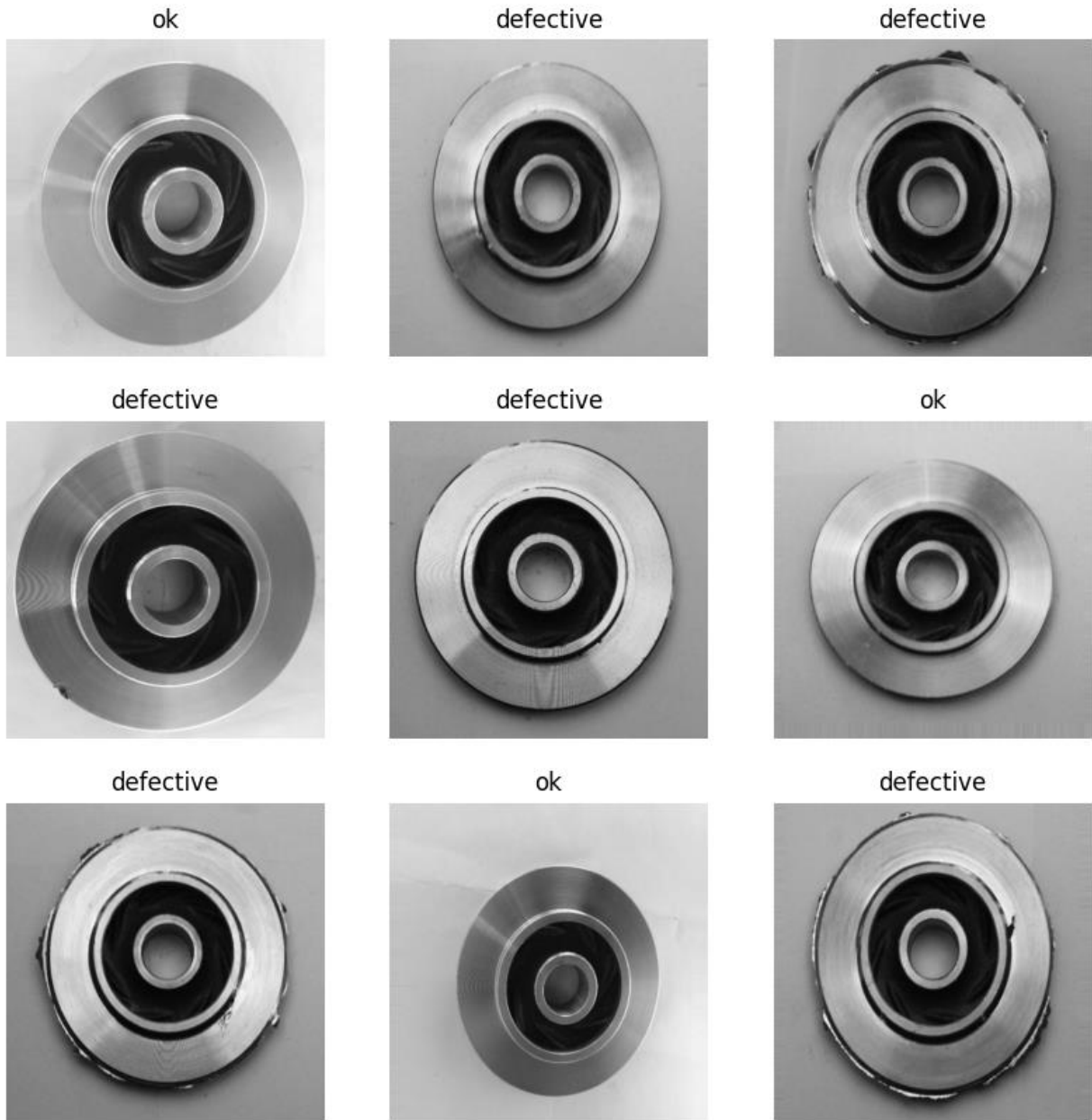


Figure 10 - Dataset Sample

Pre-processing

Two pre-processing techniques were employed in the analysis. The first technique involved rescaling the pixel values of the images to normalize them within a range of 0 to 1. This process of rescaling ensured that all pixel values fell within a standardized and consistent range, facilitating subsequent computations.

The second pre-processing technique focused on resizing the images. By downsizing the images to a smaller resolution, computational efficiency was improved. This resizing step reduced the dimensions of the images while retaining important features and patterns, enabling faster processing during model training and inference.

4.1.2 Development

The creation of this deep learning model necessitated the use of an agile development methodology; the pre-processing, prediction, comparison, and front-end modules also benefitted from the use of such methodology. The reason for this is that modelling such an architecture can require many iterations with constantly changing specifications to fine-tune the model's parameters to an acceptable level of efficiency, speed, and accuracy. Additionally, refining the other modules required modifications based on changes to the structure of the model.

The project management methodology used was based on Kanban because the workflow structure is much more flexible than Scrum, allowing for more easily permitted changes to the development process. However, because there was only one developer on the project, Kanban was only used loosely, as the methodology is designed for larger projects and teams; Kanban features were used as required.

Jira was the project management tool used to plan and track the project's progress. It has features like a Kanban board, issue linking to source control, and a 'roadmap,' which is similar to a Gantt chart but designed for agile methodologies [50]. GitHub was additionally used for code versioning.

Action Plan

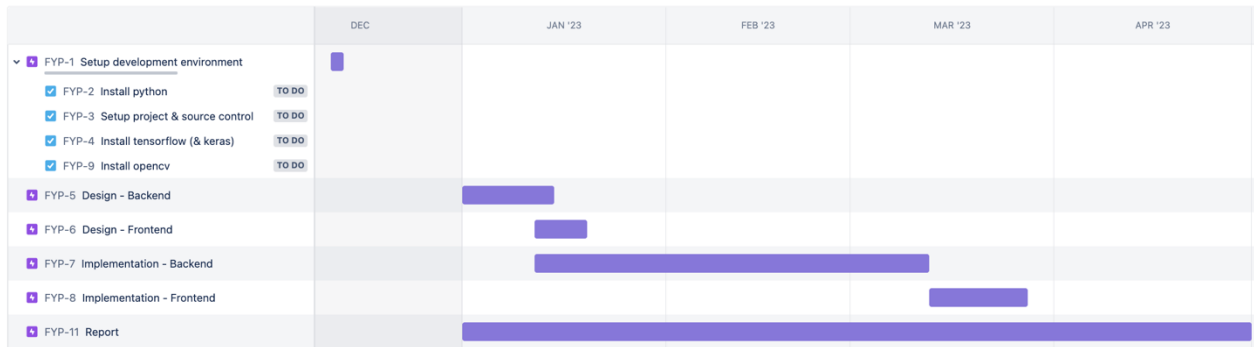


Figure 11 - Jira Roadmap

The action plan for this project was divided into six agile epics, as observable in the figure above:

- Setting up the development environment
- Designing the model
- Designing the web application
- Implementing, training, and testing the model
- Implementing the web application and testing
- Writing the report

The roadmap includes a timeframe for each epic based on reasonable estimation after task analysis. It should be noted that the implementation of the backend was allotted the most time – two months. This is regarding the process being a series of rapid prototyping, testing, and redesigning iterations. The implementation of the frontend was only allotted two weeks because of familiarity with the front-end framework. The two design phases were only given two weeks

to complete, as it was clear that a large portion of the design would take place during the implementation.

Risk Assessment

To ensure a comprehensive and successful project, a risk assessment was undertaken to help identify and avoid potential blockers. The results for which can be viewed in the table below.

Table 2 - Risk Assessment

Hazard	Affected Group(s)	Existing Controls	Risk
Development & report take more time than expected	Author	<ul style="list-style-type: none"> • Plan for the worst-case scenario • Follow the roadmap as closely as possible • Use Jira effectively 	Low
Project workload causing burnout and low productivity	Author	<ul style="list-style-type: none"> • Realistic goals and timeframes set • Follow Jira roadmap 	Low
Speed and efficiency of model insufficient	Author	<ul style="list-style-type: none"> • Continuous cycles of iterative development 	Low
Product strays from the scope of the project	Author	<ul style="list-style-type: none"> • Follow Jira roadmap • Intermittently refer to report scope and objectives sections 	Low
Losing track of code history	Author	<ul style="list-style-type: none"> • Use git/GitHub version control • Refer to Jira issues 	Low
The program doesn't work as intended (bugs)	Author	<ul style="list-style-type: none"> • Frequent and thorough unit testing 	Low
The computer or graphics card becomes damaged	Author	<ul style="list-style-type: none"> • Use a surge-protected extension cable • Keep liquids away from the computer/desk 	Low

The subsequent sections (4.2 to 4.6) discuss the methodologies implemented for each aspect of the research.

4.2 Requirements

The requirements are divided into model requirements and front-end application requirements.

For each of these, the functional and non-functional requirements are listed in the respective subsequent tables, with justification provided for each.

4.2.1 Model

Table 3 - Model Requirements

Requirement ID	Requirement Type	Description	Justification
M-FR1	Functional	The model shall be able to classify surface defects of SPI castings in images.	The primary goal of the deep learning model is to accurately identify and classify surface defects for quality control purposes.
M-FR2	Functional	The system shall support multiple classes of surface defects.	Surface defects can vary in type and severity, so the model should be able to classify different defect categories.
M-FR3	Functional	The model shall accept input images in various formats (e.g., JPEG, PNG).	The model should handle different image formats commonly encountered in real-world scenarios.
M-FR4	Functional	The model shall achieve a minimum classification accuracy of 90%.	High accuracy is crucial to ensure reliable detection and minimize false positives or false negatives.
M-NFR1	Non-functional	The model shall process an image for defect classification within 1 second.	Real-time or near-real-time processing is necessary to enable efficient defect detection in production environments.

M-NFR2	Non-functional	The model shall be able to handle variations in lighting conditions and camera angles.	The model should be able to detect defects accurately, regardless of lighting variations or different capture angles.
M-NFR3	Non-functional	The model shall be capable of handling a large number of images for batch processing.	The model should be capable of processing a significant number of images efficiently to support high-volume defect detection.
M-NFR4	Non-functional	The model shall provide insights into the decision-making process.	Understanding how the model arrives at its classifications is important for building trust and debugging potential issues.

4.2.2 Front-end Application

Table 4 - Front-end Requirements

Requirement ID	Requirement Type	Description	Justification
F-FR1	Functional	The front-end shall provide a user interface for uploading images for defect detection.	Users should be able to easily upload images from their devices to initiate the defect detection process.
F-FR2	Functional	The front-end shall display the classification results for each uploaded image.	Users need to see the detected surface defects and their corresponding classifications for further analysis and decision-making.
F-FR3	Functional	The front-end shall support real-time streaming of	Real-time streaming allows for continuous monitoring

		images from a connected camera.	and immediate detection of defects in production environments.
F-FR4	Functional	The front-end shall provide a visual representation of the detected defects overlaid on the original image.	Visual overlays help users locate and understand the precise location and nature of the detected defects.
F-FR5	Functional	The front-end shall allow users to download or save the results of defect detection for further analysis.	Users may want to save or export the results for reporting, documentation, or additional processing purposes.
F-NFR1	Non-functional	The front-end shall have a UI that is simple and easy to use.	Users may simply access and interact with the application without difficulty or irritation because there is a user-friendly design.
F-NFR2	Non-functional	The front-end shall provide a smooth and responsive user experience.	Even when executing difficult image analysis tasks, users want the programme to respond promptly and offer frictionless interactions.
F-NFR3	Non-functional	The front-end shall be suitable for widely used web browsers (such as Chrome, Firefox, and Safari).	To maximise accessibility and user reach, the application should function flawlessly on a variety of browsers.
F-NFR4	Non-functional	The front-end shall adapt to various screen dimensions and resolutions.	Regardless of the user's device or screen size, the application should offer a consistent an optimised experience.

4.3 ML Models

The ML models selected for use in comparison were VGG-16, InceptionV3, and ResNet50. They were chosen based on their suitability, efficiency, and their inclusion in Keras as pre-trained models [51], allowing for a fair comparison. Their designs and characteristics are discussed below.

4.3.1 VGG-16

The CNN-based model composed of 16 convolution and fully connected layers performs strongly in image classification tasks. According to Rohini (2021), VGG-16 is still considered to be one of the best models for computer vision since its inception in 2014. It performs well in capturing fine-grained details; however, it is computationally expensive due to its large number of parameters.

Its architecture is outlined in the figure below.

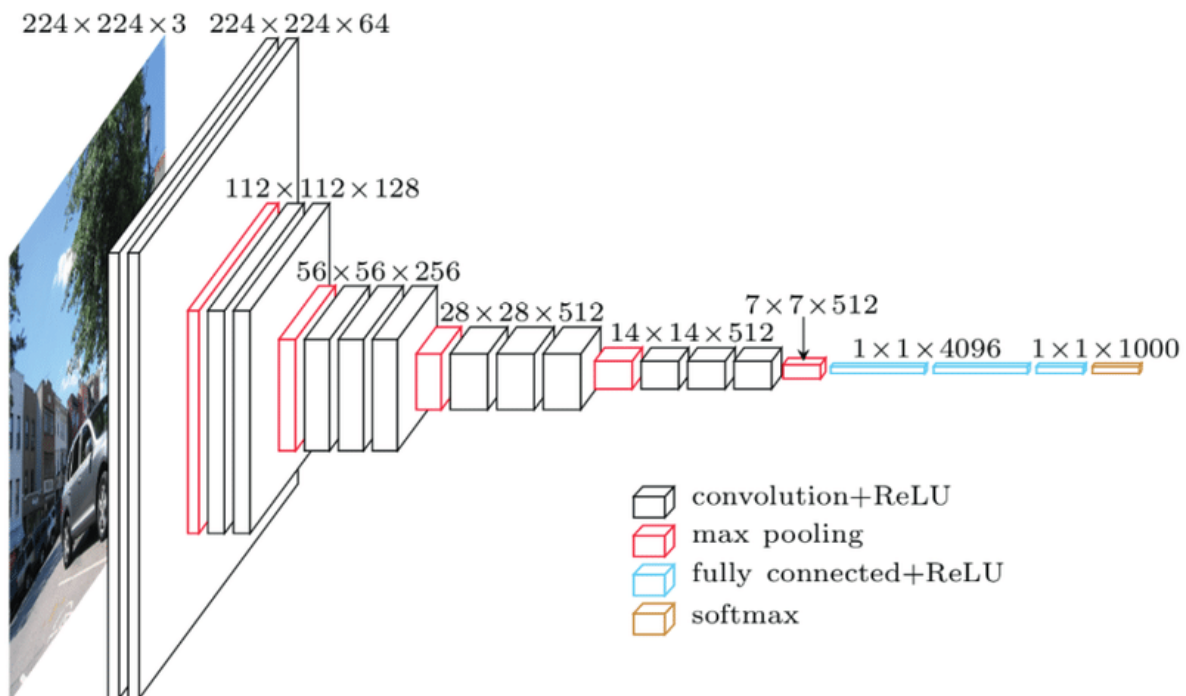


Figure 12 - VGG-16 Architecture [98]

4.3.2 InceptionV3

The InceptionV3 model provides a balance of model size and computational efficiency, whilst retaining excellent performance in image classification and defect detection tasks. To yield this efficiency, its architecture utilizes Inception Modules, which were primarily designed as a solution to the issue of computational expense and overfitting; they rely upon the principle of using multiple filter sizes within the same layer [52]. The architecture for the model can be observed in the figure below.

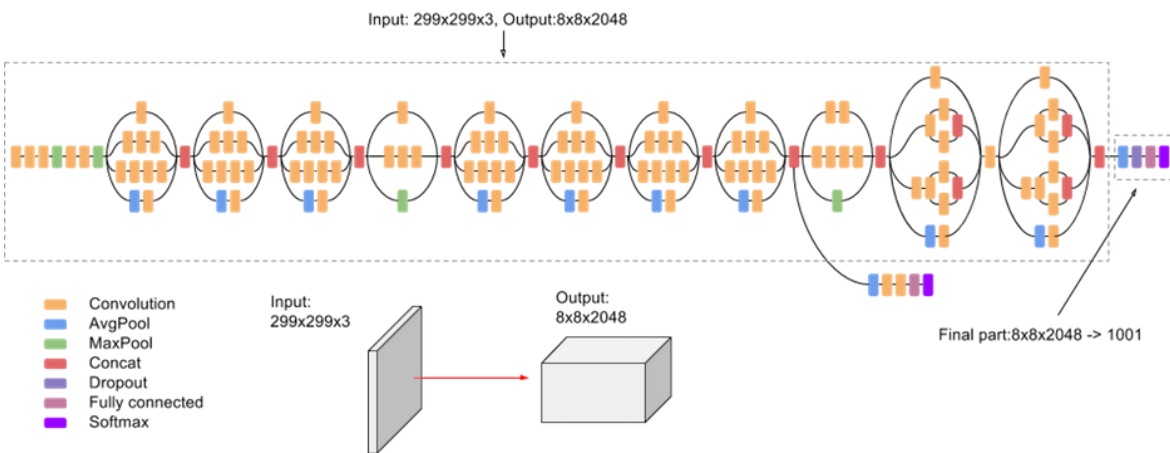


Figure 13 - InceptionV3 Architecture [99]

4.3.3 ResNet50

Deep neural networks often encounter the difficulty of training past a certain number of iterations; this is because of the vanishing gradient problem. ResNet50 provides a solution for this problem through the introduction of skip connections, which provide shortcuts for an input to an activation layer [53], and make model training less computationally intensive. It also performs well in image classification and has been widely used for defect detection tasks. In one such defect

detection task, hot-rolled steel strip defect detection using ResNet50 – along with other methods – achieved a 94.11% accuracy rate [54]. The model’s design can be derived from the figure below.

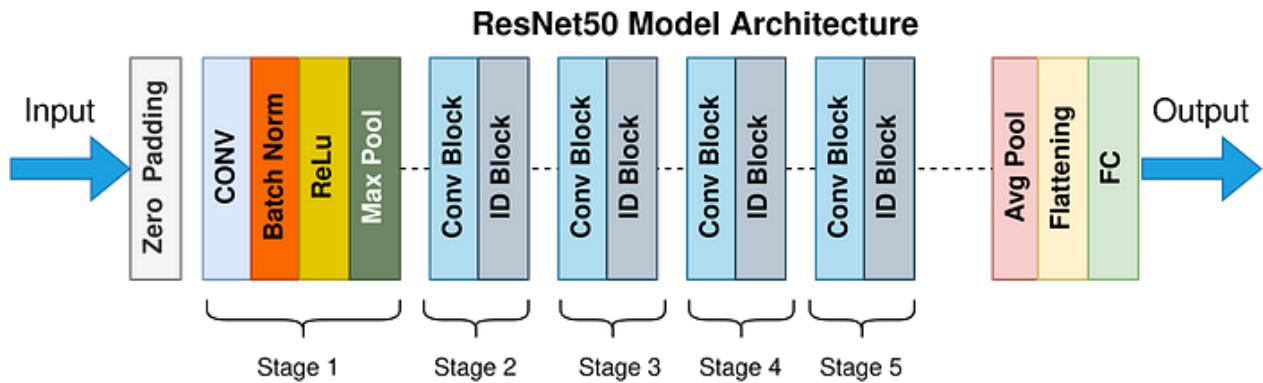


Figure 14 - ResNet50 Architecture [100]

4.4 Performance Metrics

The metrics decided upon for analysis were accuracy, precision, recall, and F1-score. This decision was made based on the need for a comprehensive evaluation of the model's performance in a binary classification task. Each of these metrics provides unique insights into different aspects of the model's performance, allowing for a well-rounded assessment.

All these metrics rely upon the involvement of true positive, true negative, false positive, and false negative rates, which are respectively the counts of correctly identified positives, correctly identified negatives, incorrectly identified negatives, and incorrectly identified positives. A commonly used method for summarizing these rates is a confusion matrix, which can be visualised in the figure below.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 15 - Confusion Matrix [56]

The matrix allows for a detailed analysis of model performance by providing insights into the types of errors it makes. The metrics mentioned above are derived from the rates defined in the confusion matrix. Their respective formulae along with a description of their use cases and relevance to this project are covered by sections 4.4.1 to 4.4.4.

4.4.1 Accuracy

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

Accuracy is a commonly used metric that measures the overall correctness of predictions and provides a reasonably accurate indication of its performance; however, it can be misleading in imbalanced datasets where classes are unequally represented [55]. Nevertheless, there was no need to take this weakness into account, as the dataset utilized for this project was appropriately balanced; approximately 60% of the images depict 'defective' components, while the remaining 40% represent 'ok' components.

4.4.2 Precision

$$\mathbf{Precision} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FP}}$$

Precision, on the other hand, measures the correctness of positive predictions by calculating the ratio of true positives to the total number of positive predictions. This metric can be particularly useful to detect when the number of false positives is high, acting as an indicator of the model's ability to avoid false alarms.

4.4.3 Recall

$$\mathbf{Recall} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}}$$

Recall, also known as sensitivity, measures the proportion of actual positive instances correctly identified by the model. It is used to assess the model's ability to capture all positive instances and can be valuable when the cost of false negatives is high.

4.4.4 F1-Score

$$\mathbf{F1\ Score} = 2 \left(\frac{\mathbf{Precision} \times \mathbf{Recall}}{\mathbf{Precision} + \mathbf{Recall}} \right)$$

F1-score is a combined metric that builds upon its simpler components: precision and recall. It represents the harmonic mean of these two components and produces reliable results even when dealing with imbalanced data, in addition to giving extreme values a lesser weighting [56]. As it is

an average of the two metrics, an F1-score will be low when both are low, high when both are high, and medium when one is high, and one is low [57]. It provides a single value that summarizes the performance of the model, considering both aspects of class-wise accuracy.

4.4.5 Summary of Metrics

Using multiple metrics to measure and evaluate the performance of ML models is crucial. Each metric has its advantages and limitations, providing valuable insights into specific aspects of a model's strengths and weaknesses. For instance, the F1-score offers a balanced perspective of precision and recall, but it presents an aggregated measure that may lack intuitive interpretability; whilst accuracy, the simplest measure, provides a general overview of the performance, but offers less value when the dataset is unbalanced.

4.4.6 Implementation

This section explains the Python-based approach used to extract model performance metrics. To ensure organized code structuring, various tasks were separated into distinct scripts. In this case, the 'comparison.py' script was dedicated to comparing performance metrics. The primary objective of this task was to generate predictions using models trained on the validation dataset and assess their performance by comparing the predicted labels with the true labels of the images. This label comparison enabled the calculation of key metrics such as accuracy, precision, recall, and f1-score, which are essential for evaluating the model's effectiveness. These metrics, as mentioned above, provide valuable insights into the model's predictive capabilities.

To ensure fairness in the evaluation process, an additional step was taken to maintain consistency among the models; a dedicated function was implemented within the script to uniformly train each model. These trained models were then saved as '.h5' files in the local directory, enabling easy access for subsequent analysis and comparisons. The function responsible for training the models was appropriately named `train_model()`, as depicted in the figure below. This systematic approach helps to streamline the training process and promote transparency in the evaluation of the models.

```
115 def train_model(name: str, model):
116
117     model.fit(train_generator,
118             validation_data=validation_generator,
119             steps_per_epoch=train_generator.samples/train_generator.batch_size,
120             epochs=10)
121     model.save(f'trained-{name}.h5')
122     print(f"Saved model: trained-{name}.h5")
```

Figure 16 - train_model() Function

The provided figure illustrates the utilization of both training and validation datasets to evaluate accuracy throughout the training process (lines 117 to 118). To ensure that every image in the training dataset is covered within a single epoch, the steps per epoch were defined as the total number of samples in the training dataset divided by the batch size (line 119).

Given that multiple models were required for comparison in this training scenario, the number of epochs was set to ten (line 120). This selection strikes a balance between training time and achieving satisfactory model accuracy, as will be demonstrated in section 5.1. Once the model training is complete, it is saved to the disk (line 121).

To facilitate the compilation of models, a builder function was implemented (`build_model()` – as seen in the figure below), designed to accommodate two parameters: the name of the model and the desired shape of the transformed images. This builder function allows for flexible customization by enabling the specification of the model's name and the desired shape of the input images. By encapsulating these parameters within the builder function, the process of constructing models becomes more streamlined and adaptable to different requirements.

```
83 def build_model(name: str, SHAPE: tuple):
84     set_seed(33)
85
86     if name == "custom":
87         return build_custom_model(SHAPE)
88
89     if name == "vgg16":
90         base_model = VGG16(weights='imagenet', include_top=False, input_shape=SHAPE)
91         base_model.summary()
92     elif name == "inceptionv3":
93         base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=SHAPE)
94         base_model.summary()
95     elif name == "resnet50":
96         base_model = ResNet50(weights='imagenet', include_top=False, input_shape=SHAPE)
97         base_model.summary()
98     else:
99         raise Exception("Invalid model name")
100
101     for layer in base_model.layers[:-5]:
102         layer.trainable = False
103
104     x = base_model.output
105     x = tf.keras.layers.GlobalAveragePooling2D()(x)
106     x = tf.keras.layers.Dense(2, activation="softmax")(x)
107
108     model = tf.keras.Model(base_model.input, x)
109     model.compile(optimizer=Adam(learning_rate=0.0001),
110                 loss="categorical_crossentropy",
111                 metrics=['accuracy'])
112     return model
```

Figure 17 - build_model() Function

To harness the benefits of transfer learning, the design employed a strategy of freezing all layers except the last five in the base model (lines 101 to 102) and adding an additional pooling layer and a dense layer. This approach allows the pre-trained weights and feature extraction

capabilities of the base model to be leveraged while adapting it for binary classification. The added layers also enhance the model's ability to learn and generalize from the data.

The final model design, before hyperparameter tuning, was established and built for training purposes using the `build_custom_model()` function in the figure below. Note that the hyperparameters for this model are the same as with the `build_model()` function.

```
61 def build_custom_model(SHAPE: tuple):
62     model = Sequential([
63         Conv2D(32, 3, padding='same', activation='relu', input_shape=SHAPE),
64         MaxPooling2D(),
65         Conv2D(64, 3, padding='same', activation='relu'),
66         Conv2D(64, 3, padding='same', activation='relu'),
67         MaxPooling2D(),
68         Conv2D(128, 3, padding='same', activation='relu'),
69         Conv2D(128, 3, padding='same', activation='relu'),
70         MaxPooling2D(),
71         Flatten(),
72         Dense(1024, activation='relu'),
73         Dense(1024, activation='relu'),
74         Dense(2, activation='sigmoid')
75     ])
76     model.compile(optimizer=Adam(learning_rate=0.0001),
77                 loss="categorical_crossentropy",
78                 metrics=['accuracy'])
79
80     return model
```

Figure 18 - build_custom_model() Function

After training the models, they were loaded and then evaluated through the invocation of the following methods – `load_trained_models()` and `evaluate_models()`, respectively.

```
212 custom, vgg16, inceptionv3, resnet50 = load_trained_models()
213 evaluate_models(custom, vgg16, inceptionv3, resnet50)
```

Figure 19 - Model Evaluation Call

```

169     def evaluate_model(model, y_true, name: str):
170         y_pred = model.predict(validation_generator, verbose=0)
171         y_pred = np.argmax(y_pred, axis=1)
172         calculate_metrics(y_true, y_pred, name)

```

Figure 20 - evaluate_model() Function

The evaluation process relied on the utilization of the evaluate_model() function, depicted in the figure above. It should be noted that the array of true image labels, denoted as y_true, and the array of model predictions, denoted as y_pred, served as the key inputs for this function.

Following the prediction (line 170), 'argmax()' is called to one-hot encode the class labels (line 171). The calculate_metrics() function was subsequently invoked (line 172), with the aforementioned arrays being passed as arguments. The calculate_metrics() function then calculates the accuracy, precision, recall, and f1-score using the TP, TN, FP, and FN rates derived using the function in the figure below. It then prints these performance metrics, along with the confusion matrix.

```

126     def calculate_rates(y_true_labels, y_pred_labels):
127         # Calculate the confusion matrix metrics
128         tn = tf.keras.metrics.TrueNegatives()
129         tn.update_state(y_true_labels, y_pred_labels)
130         fp = tf.keras.metrics.FalsePositives()
131         fp.update_state(y_true_labels, y_pred_labels)
132         fn = tf.keras.metrics.FalseNegatives()
133         fn.update_state(y_true_labels, y_pred_labels)
134         tp = tf.keras.metrics.TruePositives()
135         tp.update_state(y_true_labels, y_pred_labels)
136         # Get the TP, TN, FP, FN rates
137         tp_rate = tp.result().numpy()
138         tn_rate = tn.result().numpy()
139         fp_rate = fp.result().numpy()
140         fn_rate = fn.result().numpy()
141
142         return tp_rate, tn_rate, fp_rate, fn_rate

```

Figure 21 - calculate_rates() Function

4.5 Hyperparameter Tuning

In the context of machine learning models, internal parameters refer to the weightings of the neurons determined by training a neural network; hyperparameters, on the other hand, refer to the parameters which are pre-configured before the training process [58]. The selection and configuration of an ML model's hyperparameters have a significant impact on how well it performs, as they establish the model's organisational structure [59]. Some common hyperparameters are [60]:

- The number of layers in the model
- The number of neurons in each layer
- The activation function used in each layer
- The optimizer used to minimize the loss function
- The learning rate of the optimizer
- The batch size used during training
- The number of epochs used during training

There are two main categories of approaches for hyperparameter tuning: manual search, which includes a person changing certain hyperparameters by hand; and automatic search, which uses functions to iteratively improve without the requirement for user input past setting the bounds.

4.5.1 Manual Search

The success of this method of tuning is highly reliant on the professional expertise of the user. It becomes easy for a person to misread trends and relationships due to high dimensionality. Additionally, the results of a manual tuning process may not be reproducible [61].

4.5.2 Automatic Search

Automatic search algorithms, such as grid search, random search, and Bayesian optimization are solutions to this problem. They reduce the need for in-depth knowledge of the process and instead implement an iterative method [61]. Each algorithm has its advantages and disadvantages, which are discussed below.

Grid Search

Grid search is the simplest automatic hyperparameter tuning method, which utilizes an exhaustive evaluation approach; therefore, all combinations will be compared, and the best performing will be identified. However, this comes at the cost of efficiency; this method is limited due to its computational expense, and the ‘curse’ of dimensionality [61]. As a result of its inefficiency, it is primarily useful when dealing with a small hyperparameter space.

Random Search

The random search algorithm improves upon grid search by instead running iterations with random values within the defined range, which saves on computation. Aside from its efficiency, this strategy can be especially effective when the hyperparameter space is large or there is little prior knowledge about the ideal hyperparameter values [62]. Although this technique may find the optimal hyperparameters faster than grid search, it may also be used exhaustively.

Bayesian Optimization

In contrast, Bayesian optimization is a sequential model-based optimization technique that works by obtaining posterior information of function distribution using the Bayesian formula [63]. This information allows the algorithm to make informed decisions about the next hyperparameter distribution [63]. It is especially helpful when evaluating the objective function requires a lot of time or resources [64].

Evaluation of Techniques

In summary, grid search is straightforward but computationally expensive, random search offers efficiency and is suitable for large hyperparameter spaces, while Bayesian optimization combines a probabilistic model with intelligent search decisions, making it particularly useful when the evaluation of the objective function is resource-intensive or time-consuming. The choice of method depends on the characteristics of the hyperparameter space and the available computational resources.

*Based on these factors, the selected hyperparameter tuning technique was **Bayesian optimization**, as it provides an efficient and effective manner of handling a large hyperparameter space. This is ideal because the optimal values are uncertain and require exploration.*

4.5.3 Implementation

Hyperparameter tuning was implemented similarly to performance metric evaluation and was cast into 'hyperparameters.py'. Firstly, the dataset split was the same with 80% training and 20% validation. Secondly, it utilized a different model building function, also named build_model(), which can be viewed below.

```
34 def build_model(learning_rate, dropout_rate):
35     model = Sequential([
36         Rescaling(1./255, input_shape=(512, 512, 3)),
37         Resizing(128, 128),
38         Conv2D(32, 3, padding='same', activation='relu'),
39         MaxPooling2D(),
40         Conv2D(64, 3, padding='same', activation='relu'),
41         Conv2D(64, 3, padding='same', activation='relu'),
42         MaxPooling2D(),
43         Conv2D(128, 3, padding='same', activation='relu'),
44         Conv2D(128, 3, padding='same', activation='relu'),
45         MaxPooling2D(),
46         Flatten(),
47         Dense(1024, activation='relu'),
48         Dense(1024, activation='relu'),
49         Dense(2, activation='sigmoid')
50     ])
51
52     # Compile the model with the specified learning rate and dropout rate
53     optimizer = Adam(learning_rate=learning_rate)
54     model.compile(optimizer=optimizer,
55                 loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
56                 metrics=['accuracy'])
57
58     # Set the dropout rate for applicable layers
59     for layer in model.layers:
60         if isinstance(layer, tf.keras.layers.Dropout):
61             layer.rate = dropout_rate
62
63     return model
```

Figure 22 - hyperparameters.py build_model() Function

Line 34 shows the parameters for the function as the learning rate and dropout rate, allowing for simple modification of the hyperparameters. In addition to what was in the build_custom_model() function, there is also dropout which is implemented on lines 58 to 61.

Next is the `optimize_model()` function which was responsible for returning the best validation accuracy for a full training cycle, as in the figure below. It builds the model (line 68), trains it and assigns its history to a variable (line 71) and finds and returns the 'max' validation accuracy from the history (lines 74 and 76).

```
65 # Define the function to optimize
66 def optimize_model(learning_rate, dropout_rate):
67     # Build the model
68     model = build_model(learning_rate, dropout_rate)
69
70     # Train the model
71     history = model.fit(train_ds, validation_data=val_ds, epochs=50, verbose=0)
72
73     # Get the best validation accuracy
74     best_val_accuracy = max(history.history['val_accuracy'])
75
76     return best_val_accuracy
```

Figure 23 - hyperparameters.py optimize_model() Function

Finally, the hyperparameter space is defined in lines 81-84, specifying the bounds for each hyperparameter. These bounds are then passed to the Bayesian optimization object while defining the `optimize_model()` function in line 87. Additionally, the number of initial points and iterations for the optimization process is determined, and the optimizer is executed in line 88. Once the optimization is complete, the best-found parameters are stored and printed to the console in lines 90-97. This process can be observed in the subsequent figure.

```
79 if __name__ == '__main__':
80     # Define the hyperparameter search space
81     hyperparameter_space = {
82         'learning_rate': (0.0005, 0.001),
83         'dropout_rate': (0.2, 0.5)
84     }
85
86     # Perform Bayesian optimization
87     optimizer = BayesianOptimization(f=optimize_model, pbounds=hyperparameter_space, verbose=2)
88     optimizer.maximize(init_points=3, n_iter=10)
89
90     # Get the best hyperparameters and the corresponding validation accuracy
91     best_hyperparameters = optimizer.max['params']
92     best_val_accuracy = optimizer.max['target']
93
94     print("Best Hyperparameters:")
95     print(best_hyperparameters)
96     print("Best Validation Accuracy:")
97     print(best_val_accuracy)
```

Figure 24 - hyperparameters.py Invokation

4.6 Sensitivity Analysis

Sensitivity analysis is a method where the model input is altered in a controlled manner and the consequences of the changes are assessed. According to Saliccioli, et al. (2016), its purpose is to quantify how the uncertainty of model input is related to the uncertainty of outputs. Through uncertainty quantification, it enables researchers to evaluate how various input uncertainties propagate to the output uncertainty to determine the model's robustness and dependability [65]. The procedure is crucial because it may aid in error detection, model parameter calibration, and a deeper comprehension of the connection between the model's inputs and outputs [66].

4.6.1 Error Detection and Model Calibration

Sensitivity analysis for error detection and model calibration works by identifying the parameters that are most susceptible to changes in input data and calculating the effect of these changes on the model's output [66]. By systematically varying the values of specific parameters or inputs, sensitivity analysis provides insights into the relationship between these inputs and the corresponding outputs. This helps in understanding the model's behaviour and identifying potential discrepancies or errors [67]. The parameters that have the most effects on the model's output are highlighted via sensitivity analysis, which also helps with model calibration. Researchers can prioritise their efforts in fine-tuning and calibrating these important parameters to enhance the performance of the model by focusing on them. The sensitivity analysis results are used to iteratively alter the parameter values, improving the accuracy and dependability of the model by ensuring that predictions match observed data [68].

4.7 Activation Map

In image localization, the goal is to identify and localize specific objects or regions of interest within an image. CNNs are powerful models that excel at capturing hierarchical representations of visual data. Activation maps play a crucial role in this process by highlighting the areas of the image that contribute most to the network's decision-making. According to Wharton, et al. (2021), this is primarily because CNNs can dissect a picture into smaller parts, extract multi-scale localised features, and then combine them to create extremely comprehensible representations for decision-making. An example of an activation map can be seen in the figure below.

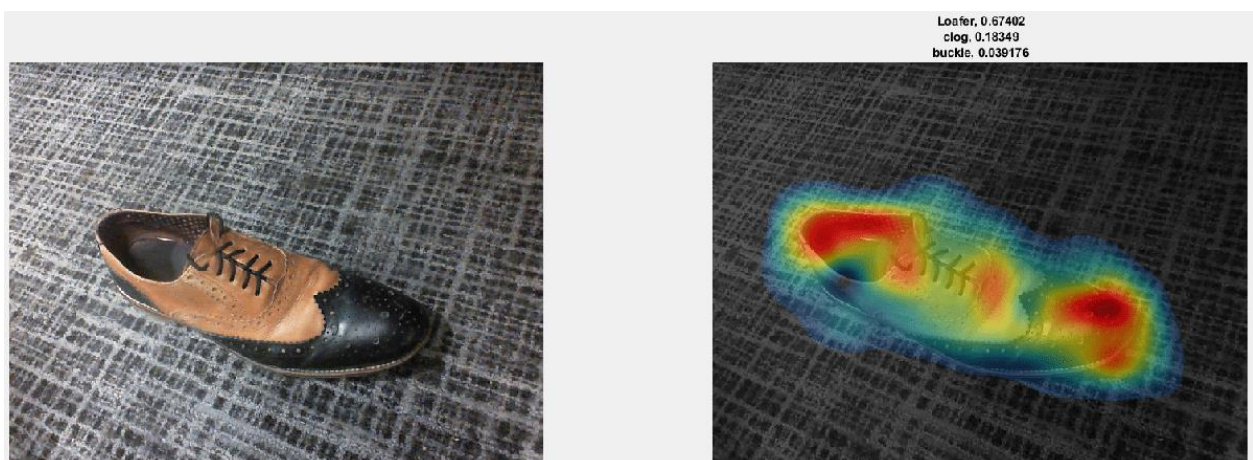


Figure 25 - Shoe Activation Map [101]

As can be observed, the activation map provides an expressive output of the model's internal representations and highlights the most decisive sections of the image for class determination. An activation map has many purposes, such as providing interpretability and transparency for a model, visualization of its localization capabilities, comparison to other models, and exposure of errors for analysis, which provides the basis for more informed decisions in fine-tuning and refinement. This provides some remediation for a model's black-box nature [69].

The activation map is generated by applying a technique such as Grad-CAM (Gradient-weighted Class Activation Mapping) or CAM (Class Activation Mapping), which allows for visualizing the areas of the image that are most relevant to the model's decision-making process.

4.7.1 CAM

CAM is an approach for producing feature maps from a DNN that is simple yet effective. It achieves this by extracting the weightings from the final convolutional layer of the model and performing a dot product between the reshaped activation maps and class weights [70].

4.7.2 Grad-CAM

On the other hand, Grad-CAM considers gradients from throughout the network, providing more class discrimination and more fine-grained visual explanations [71]. However, due to the gradient-averaging step, Grad-CAM may not always be reliable in that it can sometimes highlight sections that were not used [72].

*Considering its simplicity of implementation and effectiveness, the decision was made to employ **CAM** as the preferred method for visualizing the activation map. Grad-CAM was not chosen because its gradient averaging tends to highlight unused sections of the image, reducing localization reliability.*

4.8 Front-end Design

In a production environment, such as an engineering facility, the web-based GUI would allow workers the ability to seamlessly monitor a live feed and swiftly scan components using a connected camera when the user presses a submit button. The system would then promptly generate a processed image of the component, which would display information such as its defect status, alongside localization and classification of defects. However, since physical access to SPIs was not available, and annotation of a sufficiently sized subset of the dataset was infeasible, certain design adjustments were made in the front-end. An overview of the proposed process can be ascertained from the figure below.

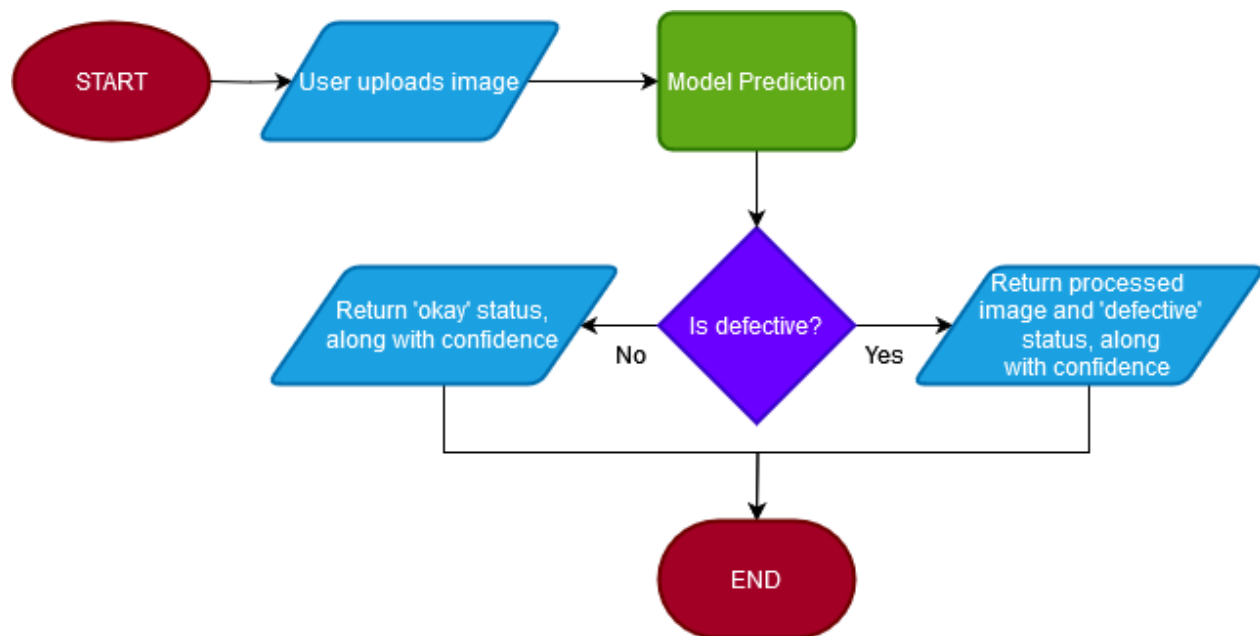


Figure 26 - Frontend Application Flowchart

Instead of relying on a live camera feed, the GUI was tailored to utilize a file selection button, granting users the ability to upload pre-existing images composed of the front surface of SPIs. Due to the lack of dataset annotation, classification of distinct types of defects (*M-FR2*) was

impossible, therefore the returned processed image would consist of a heatmap localization instead. Considering the proof-of-concept purpose of the system, it was determined that this approach would suffice. By opting for the file upload functionality, only minimal adjustments to the existing code would be necessary to transition to a solution aligned with the production design.

For minimalism and functionality, a card design has been utilized for the GUI. As in the figure below, the contents of the card consist of an initial 'upload image' button. Depending on the inference of the model, after processing the card will display either 'Defective' or 'Not Defective' as the result, along with the model's confidence in the prediction and a displayed image that can be either processed or unprocessed.

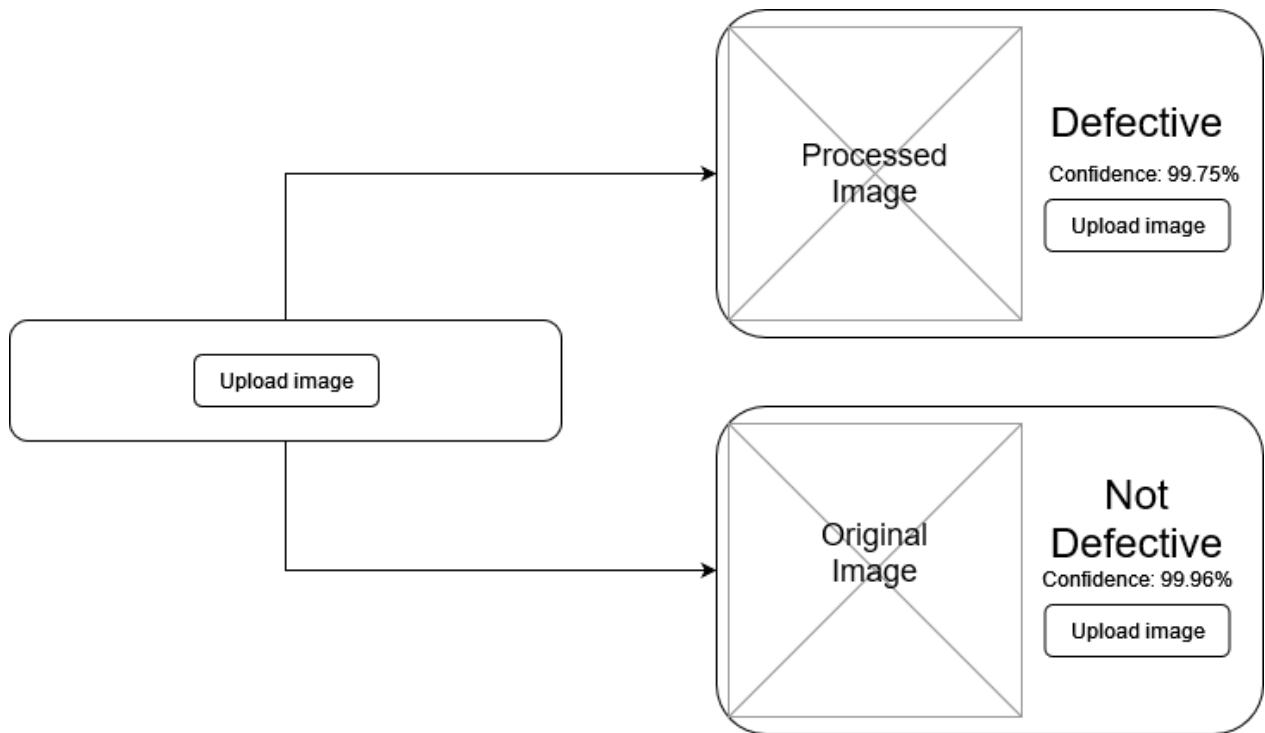


Figure 27 - GUI Wireframe

4.8.1 Implementation

The application is initialised using the `app.run()` function, as with any other Flask application. The user can then access the default endpoint (`/`) which is defined in the figure below.

```
23 @app.route('/', methods=['GET'])
24 def index():
25     return render_template('index.html')
```

Figure 28 - Implementation Default Endpoint

This endpoint returns a 'render template' which can utilize Jinja templating to change the user's view based on backend logic; however, this was not necessary since JQuery provides a more seamless experience.

For styling, Bootstrap along with some custom CSS was utilized, and their inclusion in the render template (`index.html`) can be observed in the subsequent figure.

```
7 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
8 <link rel="stylesheet" href="../static/css/style.css">
```

Figure 29 - Bootstrap and Custom CSS Link Tags

The template's body structure used the Bootstrap grid layout along with a custom 'flexbox' class (see figure below) to simplify the process of positioning and responsive resizing.

```
1 .flexbox {
2     display: flex;
3     align-items: center;
4     justify-content: center;
5 }
```

Figure 30 - Custom CSS Class

```

11 <body>
12   <div id="content p-5" class="flexbox">
13     <div class="card flexbox p-3 mt-5" style="min-width: 400px; max-width: 800px; max-height: 500px;">
14       <div class="card-body flexbox">
15         <div class="row">
16           <div class="col p-4">
17             <div id="image-box">
18               <!-- Image added here with heatmap/boxes when uploaded using js -->
19             </div>
20           </div>
21         </div>
22         <div class="row">
23           <div class="col">
24             <div id="loader" style="display:none;">
25               
26             </div>
27           </div>
28         </div>
29         <div class="row">
30           <div class="col">
31             <h2 id="defective-status"></h2>
32             <h5 id="confidence"></h5>
33           </div>
34           <div class="col">
35             <input type="file" accept="image/png, image/jpeg">
36           </div>
37         </div>
38       </div>
39     </div>
40   </div>

```

Figure 31 - Implementation Template Body

The grid layout can be observed in the usage of rows and columns (e.g., line 15 and 16). Additionally, components such as the image container div (line 17), the loading GIF (line 25), and h2 and h5 tags for the defective status and confidence, respectively (lines 31 to 32). It should also be noted that both png and jpeg images are permitted for uploading, as seen in the file select tag (line 35).

JQuery was responsible for dynamically modifying the page and is triggered when the file in the file select tag changes. This can be observed in the figure below.

```

$(document).ready(function (e) {
  $('input[type="file"]').change(function () {
    let file = this.files[0];
    let reader = new FileReader();
    reader.onloadend = function () {
      $('#image-box').html('<image id="img" src="' + reader.result + '" style="max-width: 100%;"/>');
    }
  }
}

```

Figure 32 - JQuery 'on file change' Logic

There is then a check to ensure a file was successfully added to the script (line 51). If successful, an AJAX POST request (line 55) is sent to the '/process-image' endpoint, which sends the raw image data after dynamically modifying the styling of the front-end (lines 61-66). On successful processing, the response is sanity-checked, and its data is used to update the HTML (line 67-82). If the backend encounters an error or there is a timeout, the site will display an error message (lines 84 to 87).

```
51     if (file) {
52         let formData = new FormData();
53         formData.append('raw_image', file);
54         reader.readAsDataURL(file);
55         $.ajax({
56             url: "/process-image",
57             type: 'POST',
58             data: formData,
59             processData: false,
60             contentType: false,
61             beforeSend: function(){
62                 $("#image-box").hide();
63                 $("#loader").show();
64                 $("#defective-status").html("");
65                 $("#confidence").html("");
66             },
67             success: function(response){
68                 $("#image-box").show();
69                 $("#loader").hide();
70                 if (response && response.is_defective !== undefined) {
71                     var confidence = response.confidence;
72                     $("#confidence").html("Confidence: " + confidence.toFixed(2) + "%");
73                     if (response.is_defective) {
74                         $("#defective-status").html("Defective");
75                         let base64data = response.image;
76                         $("#img").attr("src", "data:image/png;base64," + base64data);
77                     } else {
78                         $("#defective-status").html("Not Defective");
79                     }
80                 } else {
81                     alert("Invalid response from the server.");
82                 }
83             },
84             error: function(xhr, desc, err){
85                 $("#loader").hide();
86                 alert("Something went wrong! Please try again.");
87             }
88         });
89     } else {
90         alert("Image upload failed!");
91     }
```

Figure 33 - Implementation Template JQuery

The '/process-image' endpoint begins with a check to verify a file has been uploaded, as in the figure below.

```
27 @app.route('/process-image', methods=['POST'])
28 def upload():
29     if not request.files['raw_image']:
30         return 'No file uploaded'
31     else:
32         # Get the image from the request
33         image = request.files['raw_image']
```

Figure 34 - Implementation '/process-image' Check

Next, the file is saved to the 'tmp' folder (lines 37-39) with a UUID file name (line 36) to avoid collisions as in the figure below.

```
35 # Save the image to the tmp folder
36 file_name = str(uuid.uuid4()).hex
37 image_path = f"tmp/{file_name}.jpeg"
38 output_image_path = os.path.join('tmp', f'{file_name}-output.png')
39 image.save(image_path)
40
41 # Check if the image is defective + delete the image after 15 seconds
42 threading.Thread(target=garbage_collection, args=(image_path, 15), daemon=True).start()
43
44 defective, confidence = is_defective(image_path)
```

Figure 35 - Implementation '/process-image' File Saving and Processing

After saving, a new thread is created (line 42) to run the 'garbage_collection()' function on the file to delete it after 15 seconds, preventing a build-up of images (see figure below). The 'is_defective()' function is then called (line 44), which runs model inference on the input image and returns a boolean and the confidence values, as in the subsequent figure.

```
16 def garbage_collection(path: str, time_seconds: int):
17     time.sleep(time_seconds)
18     if os.path.exists(path):
19         os.remove(path)
20     exit(0)
```

Figure 36 - Implementation '/process-image' garbage_collection() Function

```

22 def is_defective(image_path):
23     with Image.open(image_path) as img:
24         img = img.resize((512, 512))
25         image = np.array(img)
26
27         # Normalize the image
28         image = image / 255.0
29
30         # Make a prediction on the image using the model
31         pred = model.predict(image[np.newaxis,:,:,:])
32         confidence_percentage = np.max(pred) * 100
33
34         # Check if the predicted class is 0 (defective)
35         if np.argmax(pred) == 0:
36             return True, confidence_percentage
37         else:
38             return False, confidence_percentage

```

Figure 37 - localization.py 'is_defective()' Function

The 'is_defective()' function first loads the image using the path passed to it (from the 'tmp' folder), seen on line 23. Next, it resizes the image (line 24) to allow the uploading of different image sizes, converts it to an array using numpy (line 25), and normalizes the image (line 28). After that, the prediction is made on the image array (line 31), and the confidence is calculated (line 32). Finally, the prediction class is validated (line 35). If it is predicted as defective, the function returns True, along with the confidence value (line 36). Otherwise, it returns False, also accompanied by the confidence value (line 38).

Going back to `/process-image` in the figure below, the function then checks the result of the prediction (line 46). If defective, the activation map is created and saved to the `'tmp'` folder using a modified version of the original image's filename (line 47). Then, a garbage thread is created for that file (line 49), it is saved (line 53) and encoded for transfer (line 57), added to a new response along with the status and confidence, and sent (lines 59-63). If not defective, the image does not get processed further than the initial prediction, so the response consists of the status and confidence.

```
46     if defective:
47         save_activation_map(image_path)
48         # Plot and save the output image to the tmp folder + delete the image after 60 seconds
49         threading.Thread(target=garbage_collection, args=(output_image_path, 60), daemon=True).start()
50
51         with Image.open(output_image_path) as img:
52             image_data = BytesIO()
53             img.save(image_data, format='PNG')
54             image_data.seek(0)
55
56             # Encode the image data as base64
57             base64_image = base64.b64encode(image_data.getvalue()).decode()
58
59             response = {'image': base64_image,
60                       'is_defective': defective,
61                       'confidence': confidence}
62
63             return jsonify(response)
64     else:
65         response = {'is_defective': defective,
66                   'confidence': confidence}
67
68         return jsonify(response)
```

Figure 38 - Implementation `/process-image` Response

Finally, as seen in the previous figure, the 'save_activation_map()' function is the implementation of class activation mapping (CAM) and can be seen in the subsequent figure. As with 'is_defective()', the image is loaded, resized, and converted to an array (lines 47 to 49). A prediction is then performed using the intermediate layer – the final convolutional layer (line 51), the output is processed, and the activation map is resized to overlay the image (lines 52 to 59). A plot is then created using matplotlib (lines 61), the activation map is overlaid over the image (lines 62 to 63), and the file is saved to be sent in the response (line 69).

```
41 weights = model.layers[-1].get_weights()[0]
42 class_weights = weights[:, 0]
43 intermediate = tf.keras.Model(model.input, model.get_layer("block5_conv3").output)
44
45 def save_activation_map(image_path):
46     # Load the image file and convert it to a NumPy array
47     with Image.open(image_path) as img:
48         img = img.resize((512, 512))
49         image = np.array(img)
50
51     conv_output = intermediate.predict(image[np.newaxis,:,:,:])
52     conv_output = np.squeeze(conv_output)
53
54     h = int(image.shape[0]/conv_output.shape[0])
55     w = int(image.shape[1]/conv_output.shape[1])
56
57     activation_maps = sp.ndimage.zoom(conv_output, (h, w, 1), order=1)
58     out = np.dot(activation_maps.reshape((image.shape[0]*image.shape[1], 512)), class_weights).reshape(
59         image.shape[0],image.shape[1])
60
61     fig, axs = plt.subplots(figsize=(6, 6))
62     axs.imshow(image)
63     axs.imshow(out, cmap='jet', alpha=0.35)
64     axs.axis('off')
65     plt.tight_layout()
66
67     # Save the figure to the tmp folder
68     file_name = os.path.splitext(os.path.basename(image_path))[0]
69     fig.savefig(os.path.join('tmp', f'{file_name}-output.png'))
70     plt.close(fig)
```

Figure 39 - localization.py 'save_activation_map()' Function

The full code used in the implementation along with a screenshot of the project directory structure can be found in the **Appendix**.

5. Results and Discussion

5.1 Performance Analysis of ML Models

This section provides a comprehensive experimental evaluation and comparison utilising relevant performance metrics.

5.1.1 Experimental Comparison

To ensure fair comparisons between models, the chosen approach for implementation involved utilizing transfer learning with the three architectures: VGG16, InceptionV3, and ResNet50. This approach entailed utilizing the pre-trained models with weights from the ImageNet dataset. To achieve this, the top classification layer was removed, and the weights of the base model were frozen, excluding the last five layers. Additionally, to define the outputs for binary classification, an extra pooling layer and a dense layer were added. By adopting this methodology, a consistent and standardized framework was established for evaluating the performance of the models, as discussed previously.

Dataset Split

As mentioned earlier, the dataset consists of 1300 images of SPIs. The training-validation split for each task was set at 80% and 20% respectively. This allocation was chosen because it strikes a balance between providing an ample amount of data for training and reserving a sufficient portion for validation. By allocating 80% of the dataset for training, the model can learn from a diverse range of examples, enabling it to capture the underlying patterns and intricacies of the

SPIs. The remaining 20% set aside for validation ensures that the model's performance can be assessed on unseen data, allowing for a reliable evaluation of its generalization capabilities. This split aims to prevent overfitting, where the model memorizes the training data without truly understanding the underlying concepts, and instead promotes the development of a robust and effective model that can effectively generalize to new SPIs.

Input Image Dimensions

It was initially attempted to use the original 512x512 images for training; however, the GPU used lacked sufficient memory to handle the training of the proposed model. Consequently, a decision was made to downscale the images to 224x224. This resizing was deemed necessary because it allowed the GPU to accommodate the training process of the proposed model within its memory limitations.

By reducing the image dimensions from 512x512 to 224x224, the computational requirements were significantly reduced, enabling efficient training without compromising the overall quality of the dataset. This resolution is a commonly used standard in many computer vision tasks and has been shown to yield satisfactory results in various deep learning models [73]. Additionally, it aligns with the input size expectations of pre-trained models such as VGG [74], facilitating the utilization of pre-trained weights and transfer learning, which can greatly benefit the training process. The smaller image size not only allowed for successful model training but also contributed to faster computation during both the training and inference stages.

Results (224x224 Images)

The results obtained for the 224x224 images are presented in the following figures, which will be subsequently compiled into a table and discussed.

```
=====  
Custom Model  
Confusion Matrix:  
      Predicted Negative   Predicted Positive  
Actual Negative      143.0           13.0  
Actual Positive       0.0           103.0  
Accuracy: 0.950  
Precision: 0.888  
Recall: 1.000  
F1-score: 0.941  
=====
```

Figure 40 - Comparison Results Custom Model (224x224)

```
=====  
VGG16 (Transfer Learning)  
Confusion Matrix:  
      Predicted Negative   Predicted Positive  
Actual Negative      150.0           6.0  
Actual Positive       0.0           103.0  
Accuracy: 0.977  
Precision: 0.945  
Recall: 1.000  
F1-score: 0.972  
=====
```

Figure 41 - Comparison Results VGG16 (224x224)

```
=====  
InceptionV3 (Transfer Learning)  
Confusion Matrix:  
      Predicted Negative   Predicted Positive  
Actual Negative      143.0           13.0  
Actual Positive       9.0           94.0  
Accuracy: 0.915  
Precision: 0.879  
Recall: 0.913  
F1-score: 0.895  
=====
```

Figure 42 - Comparison Results InceptionV3 (224x224)

```

=====
ResNet50 (Transfer Learning)
Confusion Matrix:
      Predicted Negative   Predicted Positive
Actual Negative      149.0           7.0
Actual Positive      38.0           65.0
Accuracy: 0.826
Precision: 0.903
Recall: 0.631
F1-score: 0.743
=====

```

Figure 43 - Comparison Results ResNet50 (224x224)

Table 5 - Comparison Results Table (224x224)

Model	Training Accuracy (3sf)	Validation Accuracy (3sf)	Precision	Recall	F1-Score	TP	TN	FP	FN
Proposed Model	99.5%	95.0%	0.888	1.000	0.941	103	143	13	0
VGG16	99.9%	97.7%	0.945	1.000	0.972	103	150	6	0
InceptionV3	93.3%	91.5%	0.879	0.913	0.895	94	143	13	9
ResNet50	83.1%	82.6%	0.903	0.631	0.743	65	149	7	38

Discussion

According to the performance measures, VGG16 was the best-performing model, predicting non-defective cases with exceptional accuracy. It had the highest validation accuracy (97.7%), precision (0.945), recall (1), and F1-Score (0.972). These data indicate its outstanding overall performance in correctly categorising non-defective samples. The Custom Model likewise performed well, with a high accuracy of 95% and faultless recall. With a validation accuracy of 91.5%, precision of 0.879, recall of 0.913, and F1-Score of 0.895, InceptionV3 performed well. ResNet50, on the other hand, displayed lower accuracy and recall, with a validation accuracy of

82.6%, precision of 0.903, recall of 0.631, and F1-Score of 0.743. These measurements suggest that ResNet50 may have issues in recognising non-defective instances, perhaps leading to false positives.

Results (128x128 Images)

The results obtained for the 128x128 images are presented in the following figures, which will be subsequently compiled into a table and discussed.

```
=====  
Custom Model  
Confusion Matrix:  
      Predicted Negative   Predicted Positive  
Actual Negative      150.0           6.0  
Actual Positive       0.0          103.0  
Accuracy: 0.977  
Precision: 0.945  
Recall: 1.000  
F1-score: 0.972  
=====
```

Figure 44 - Comparison Results Custom Model (128x128)

```
=====  
VGG16 (Transfer Learning)  
Confusion Matrix:  
      Predicted Negative   Predicted Positive  
Actual Negative      154.0           2.0  
Actual Positive       2.0          101.0  
Accuracy: 0.985  
Precision: 0.981  
Recall: 0.981  
F1-score: 0.981  
=====
```

Figure 45 - Comparison Results VGG16 (128x128)

```

=====
InceptionV3 (Transfer Learning)
Confusion Matrix:
      Predicted Negative   Predicted Positive
Actual Negative      144.0         12.0
Actual Positive       6.0         97.0
Accuracy: 0.931
Precision: 0.890
Recall: 0.942
F1-score: 0.915
=====

```

Figure 46 - Comparison Results InceptionV3 (128x128)

```

=====
ResNet50 (Transfer Learning)
Confusion Matrix:
      Predicted Negative   Predicted Positive
Actual Negative      136.0         20.0
Actual Positive       8.0         95.0
Accuracy: 0.892
Precision: 0.826
Recall: 0.922
F1-score: 0.872
=====

```

Figure 47 - Comparison Results ResNet50 (128x128)

Table 6 - Comparison Results Table (128x128)

Model	Training Accuracy (3sf)	Validation Accuracy (3sf)	Precision	Recall	F1-Score	TP	TN	FP	FN
Proposed Model	98.2%	97.7%	0.945	1	0.972	103	150	6	0
VGG16	99.1%	98.5%	0.981	0.981	0.981	101	154	2	2
InceptionV3	93.0%	93.1%	0.890	0.942	0.915	97	144	12	6
ResNet50	89.9%	89.2%	0.826	0.922	0.872	95	136	20	8

Discussion

When the models were compared, VGG16 had the highest validation accuracy, suggesting greater overall performance. It also had the best precision and F1-score, signifying that it struck a fair balance between properly detecting positive samples and minimising false positives and false negatives. The custom model, on the other hand, had a somewhat greater recall, which means it accurately recognised all positive samples. InceptionV3 and ResNet50 demonstrated inferior validation accuracies, precision, recall, and F1-scores than VGG16 and the Custom Model. Based on these findings, VGG16 performed the best of the models, followed closely by the custom model.

*The overall validation accuracies increased when using 128x128 images, therefore these image dimensions were used for any further tasks. Also, because of its superior performance and ability to be trained on 512x512 images, a **transfer-learning trained VGG16** model was used for **back-end processing in the web application**.*

Model File Sizes

It should be noted that the custom model was considerably larger in file size than the other trained models, sitting at 408MB with 128x128 images compared to 86MB, 119MB, and 158MB for InceptionV3, ResNet50, and VGG16 respectively, as seen in the figure below.

Name	Size
trained-custom.h5	408,882 KB
trained-inceptionv3.h5	86,075 KB
trained-resnet50.h5	119,405 KB
trained-vgg16.h5	158,997 KB

Figure 48 - Model File Sizes with 128x128 Input

Additionally, only the file size of the proposed model notably increased, reaching 1.22GB, while the other models remained mostly unchanged. This can be attributed to most of their layers being frozen during transfer learning.

Name	Size
trained-custom.h5	1,219,890 KB
trained-inceptionv3.h5	86,074 KB
trained-resnet50.h5	119,405 KB
trained-vgg16.h5	158,997 KB

Figure 49 - Model File Sizes with 224x224 Input

5.2 Hyperparameter Tuning Results

Hyperparameter tuning was carried out using Bayesian optimization. The model was trained for 50 epochs for each hyperparameter combination. In total, there were 13 cycles of optimization, with the first three finding the best initial point using random search, and the remainder relying upon Bayesian optimization.

5.2.1 Selected Hyperparameters

The hyperparameters used for tuning were the learning rate and dropout rate. Although various hyperparameters could have been additionally utilized, it was decided to focus on the most crucial for training [75] [76]. Consequently, the Adam optimizer was selected as the optimization method.

Learning Rate

The learning rate is responsible for determining the step size during optimization; a rate that is too low would result in a search that takes a long time and can get stuck, whereas a rate that is too high is prone to overshooting the minimum [77].

Dropout Rate

Whilst the dropout rate controls what proportion of the data is randomly 'dropped out'; this has the benefit of helping to improve the model's overall generalization ability and prevent overfitting [78].

Adam Optimizer

Empirical findings show that Adam performs better in practice and performs favourably when compared to other stochastic optimisation techniques such as SGD, Adagrad, and Adadelata [79]. Due to its adaptive learning rates and moment estimates, it frequently converges more quickly than other optimizers, allowing it to move quickly toward the minimum for faster training [80].

Justification for Selected Hyperparameters

It provided for an efficient and effective approach to enhancing model performance by focusing on the two hyperparameters that have the most influence on overall performance and generalisation ability. Because the dataset is well-balanced, it was suitable to rely on validation accuracy to provide an overview of performance in hyperparameter tuning. Taking this into consideration, the model's performance is discussed using the performance metrics used in the prior section.

5.2.2 Tuning Attempt 1

The default learning rate for the Adam optimizer in Keras is 0.001, which serves as a benchmark for establishing an appropriate range. Thus, for the initial tuning attempt, a range of 0.001 to 0.01 was selected. Similarly, as dropout with a rate of 50% (0.5) is often used as a starting point [81], the initial range for dropout was chosen as 0% to 50%. This broader range facilitates more effective operation of Bayesian optimization, as restricting the range might necessitate additional tuning to uncover optimal values. The table and figure below provide a visual representation of these chosen ranges for learning rate and dropout values.

Table 7 - Initial Hyperparameters

Parameter	Lower Bound	Upper Bound
Learning Rate	0.001	0.01
Dropout Rate	0	0.5

```
78 # Define the hyperparameter search space
79 hyperparameter_space = {
80     'learning_rate': (0.001, 0.01),
81     'dropout_rate': (0.0, 0.5)
82 }
```

Figure 50 - Hyperparameter Tuning Ranges (Attempt 1)

The tuning output is summarized in the subsequent figure, showcasing the best validation accuracy achieved. The highest validation accuracy recorded was an impressive 98.85%. This notable accuracy was attained with a dropout rate of approximately 36% and a learning rate of around 0.0011. The obtained validation accuracy is indeed promising, and the chosen dropout rate falls within the acceptable range for regularization purposes. However, it is worth noting that the learning rate is positioned quite close to the lower bound, indicating the potential for further improvement by exploring a lower minimum bound for the learning rate.

```
| 3 | 0.6192 | 0.03576 | 0.003647 |
| 6 | 0.9692 | 0.3619 | 0.00205 |
| 7 | 0.9885 | 0.3626 | 0.001071 |
| 8 | 0.9654 | 0.364 | 0.001411 |
| 9 | 0.9731 | 0.3654 | 0.002673 |
| 10 | 0.9808 | 0.3655 | 0.001111 |
| 11 | 0.6192 | 0.3653 | 0.002629 |
| 12 | 0.6192 | 0.4977 | 0.00347 |
| 13 | 0.9731 | 0.3626 | 0.001182 |
=====
Best Hyperparameters:
{'dropout_rate': 0.3626120405125592, 'learning_rate': 0.0010706775798373842}
Best Validation Accuracy:
0.9884615540504456
```

Figure 51 - Hyperparameter Tuning Output (Attempt 1)

5.2.3 Tuning Attempt 2

Based on the findings of the previous tuning attempt, it became apparent that lowering the learning rate could potentially lead to improvements. Therefore, for this subsequent attempt, the range for the learning rate was narrowed down to 0.0001 to 0.001. This revised range offers enough flexibility to explore different values while avoiding excessive breadth that could hinder the search for the optimal learning rate.

```
78 # Define the hyperparameter search space
79 hyperparameter_space = {
80     'learning_rate': (0.0001, 0.001),
81     'dropout_rate': (0.0, 0.5)
82 }
```

Figure 52 - Hyperparameter Tuning Ranges (Attempt 2)

On the other hand, the range for dropout was maintained the same as in the previous attempt (0 to 0.5). This decision was based on the observation that the previous range did not appear to restrict improvements. By keeping the range unchanged, we allow for continued exploration of dropout values without imposing unnecessary constraints. A visual representation of these ranges for the learning rate and dropout values can be found in the previous figure.

The result for this attempt yielded a higher validation accuracy of 99.23% at a dropout rate of 36.8% and a learning rate of 0.0008575, as can be ascertained from the output in the figure and table below. The table compares the values between attempts.


```

Found 1300 files belonging to 2 classes.
Using 260 files for validation.
| iter | target | dropou... | learni... |
-----
2023-05-25 15:30:10.393634: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384]
2023-05-25 15:30:11.285933: I tensorflow/stream_executor/cuda/cuda_blas.cc:161]
| 1 | 0.9846 | 0.3046 | 0.0002498 |
| 2 | 0.9846 | 0.1376 | 0.0005462 |
| 3 | 0.9923 | 0.3685 | 0.0008575 |
| 4 | 0.9846 | 0.3686 | 0.0008533 |
| 5 | 0.9885 | 0.3685 | 0.0008656 |
| 6 | 0.9923 | 0.2618 | 0.0002245 |
| 7 | 0.9808 | 0.2618 | 0.0001736 |
| 8 | 0.9885 | 0.2618 | 0.000212 |
| 9 | 0.9846 | 0.2619 | 0.0001987 |
| 10 | 0.9731 | 0.3685 | 0.0007478 |
| 11 | 0.9846 | 0.2618 | 0.0003753 |
| 12 | 0.9923 | 0.2619 | 0.0002824 |
| 13 | 0.9692 | 0.2618 | 0.0002899 |
=====
Best Hyperparameters:
{'dropout_rate': 0.36851035221745243, 'learning_rate': 0.0008575236394668898}
Best Validation Accuracy:
0.9923076629638672

```

Figure 53 - Hyperparameter Tuning Output (Attempt 2)

Table 8 - Comparison between Tuning Attempts 1 and 2

Attempt #	Best Validation Accuracy (2dp)	Best Learning Rate (3sf)	Best Dropout Rate (3sf)	Learning Rate Change Since Previous Attempt	Dropout Rate Change Since Previous Attempt
1	98.84%	0.001070	0.363	N/A	N/A
2	99.23%	0.000858	0.369	-0.000212	+0.006

This table shows a lower learning rate and a higher dropout rate have a positive impact on accuracy.

5.2.4 Tuning Attempt 3 (Final)

```
79 # Define the hyperparameter search space
80 hyperparameter_space = {
81     'learning_rate': (0.0005, 0.001),
82     'dropout_rate': (0.2, 0.5)
83 }
```

Figure 54 - Hyperparameter Tuning Ranges (Attempt 3)

Building upon the notable improvement achieved in the previous attempt, it became evident that further gains in validation accuracy could be obtained through additional tuning. To focus the search and explore more promising regions of the hyperparameter space, the range for the learning rate was narrowed down to 0.0005 to 0.001, as can be seen in the figure above. Additionally, the range for dropout was adjusted to span from 20% to 50%. By restricting the range within this narrower interval, the tuning process can concentrate on dropout values that have shown potential for enhanced generalized performance. This constrained range enables a more focused exploration, aiming to uncover the most effective dropout rate for improving validation accuracy.

In the latest tuning attempt, a higher validation accuracy of 100% was achieved. This accuracy was obtained with a dropout rate of 46.39% and a learning rate of approximately 0.0006245. The corresponding output can be observed in the figure below, confirming these optimal values. This is again accompanied by a table comparing the results between tuning attempts.

```

Found 1300 files belonging to 2 classes.
Using 260 files for validation.
| iter | target | dropou... | learni... |
-----
2023-05-27 20:28:40.015709: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384
2023-05-27 20:28:41.982714: I tensorflow/stream_executor/cuda/cuda_blas.cc:16
| 1 | 0.9923 | 0.4639 | 0.0006679 |
| 2 | 0.9846 | 0.4908 | 0.0009262 |
| 3 | 0.9962 | 0.3486 | 0.0006912 |
| 4 | 0.9808 | 0.3486 | 0.0007028 |
| 5 | 0.9846 | 0.3486 | 0.0006756 |
| 6 | 0.9731 | 0.4638 | 0.0006868 |
| 7 | 0.9808 | 0.4639 | 0.000716 |
| 8 | 0.9885 | 0.464 | 0.0006524 |
| 9 | 0.9923 | 0.4639 | 0.0006791 |
| 10 | 0.9846 | 0.464 | 0.000576 |
| 11 | 1.0 | 0.4639 | 0.0006425 |
| 12 | 0.9885 | 0.464 | 0.0006764 |
| 13 | 0.9923 | 0.4639 | 0.0006501 |
=====
Best Hyperparameters:
{'dropout_rate': 0.46386941186999164, 'learning_rate': 0.0006425108307104302}
Best Validation Accuracy:
1.0

```

Figure 55 - Hyperparameter Tuning Output (Attempt 3)

Table 9 - Comparison between Tuning Attempts 1, 2, and 3

Attempt #	Best Validation Accuracy (2dp)	Best Learning Rate (3sf)	Best Dropout Rate (3sf)	Learning Rate Change Since Previous Attempt	Dropout Rate Change Since Previous Attempt
1	98.84%	0.001070	0.363	N/A	N/A
2	99.23%	0.000858	0.369	-0.000212	+0.006
3	100.00%	0.000643	0.464	-0.000215	+0.095

As with the previous comparison, this table shows that a lower learning rate and higher dropout rate benefitted the model's classification ability.

Result Verification

The model was retrained using these parameters for verification, and the retraining process yielded a validation accuracy of 98.85%. It is worth noting that previous epochs achieved even higher accuracies, reaching 99.23%, as shown in the figure below.

```
Epoch 46/50
33/33 [=====] - 2s 62ms/step - loss: 2.6133e-05 - accuracy: 1.0000 - val_loss: 0.0595 - val_accuracy: 0.9923
Epoch 47/50
33/33 [=====] - 2s 62ms/step - loss: 2.4428e-05 - accuracy: 1.0000 - val_loss: 0.0607 - val_accuracy: 0.9923
Epoch 48/50
33/33 [=====] - 2s 62ms/step - loss: 2.1877e-05 - accuracy: 1.0000 - val_loss: 0.0616 - val_accuracy: 0.9885
Epoch 49/50
33/33 [=====] - 2s 62ms/step - loss: 2.0421e-05 - accuracy: 1.0000 - val_loss: 0.0635 - val_accuracy: 0.9846
Epoch 50/50
33/33 [=====] - 2s 62ms/step - loss: 1.7777e-05 - accuracy: 1.0000 - val_loss: 0.0642 - val_accuracy: 0.9885
```

Figure 56 - Final Training Output

The training metrics, which consist of training and validation accuracy, as well as training and validation loss, were visualized using the 'pyplot' module from the matplotlib library. Two graphs were generated to represent these metrics. Please refer to the figures on the following pages to view the graphs. The analysis of the results depicted in the graphs will be discussed subsequently.

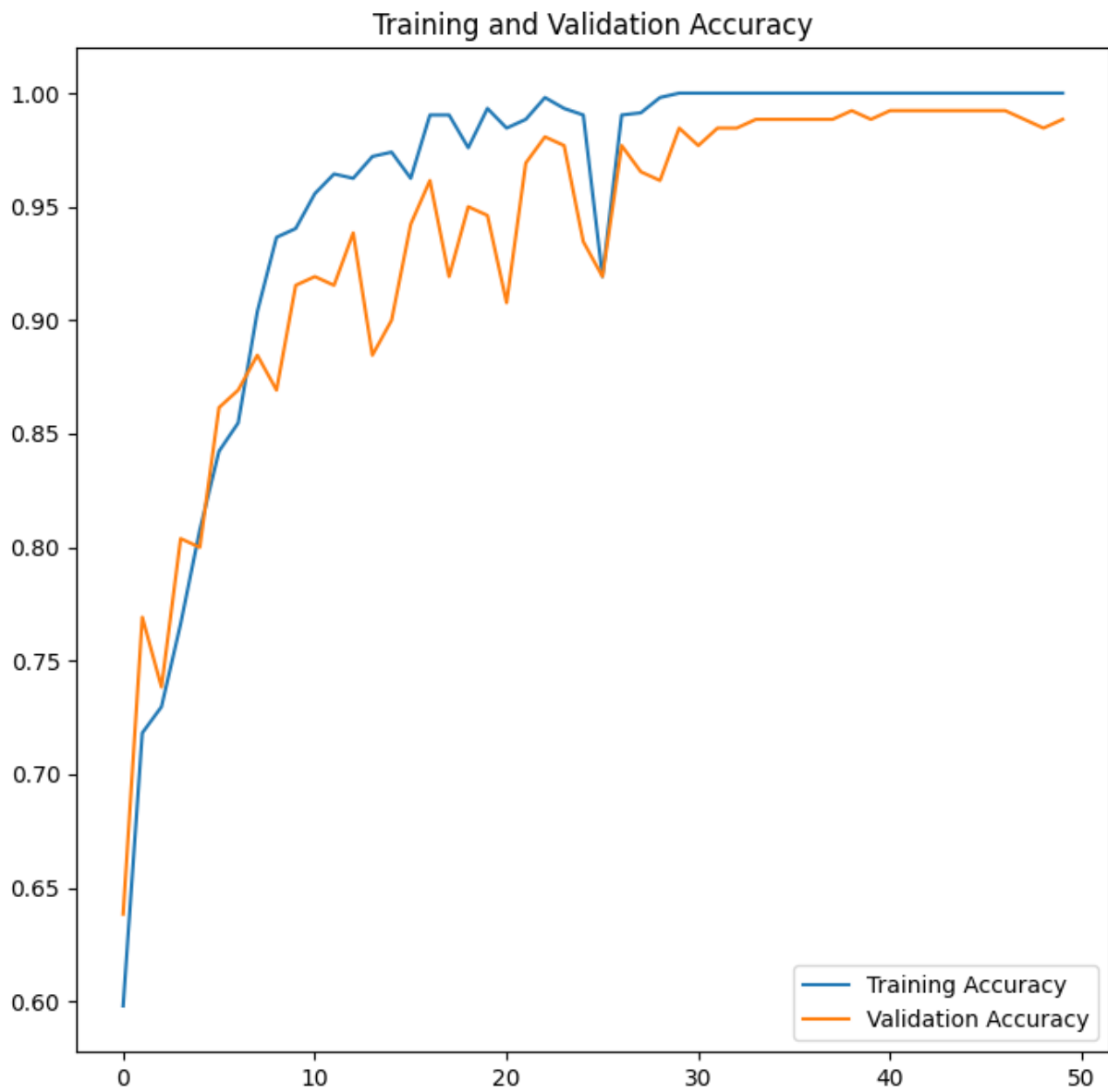


Figure 57 - Training and Validation Accuracy Plot

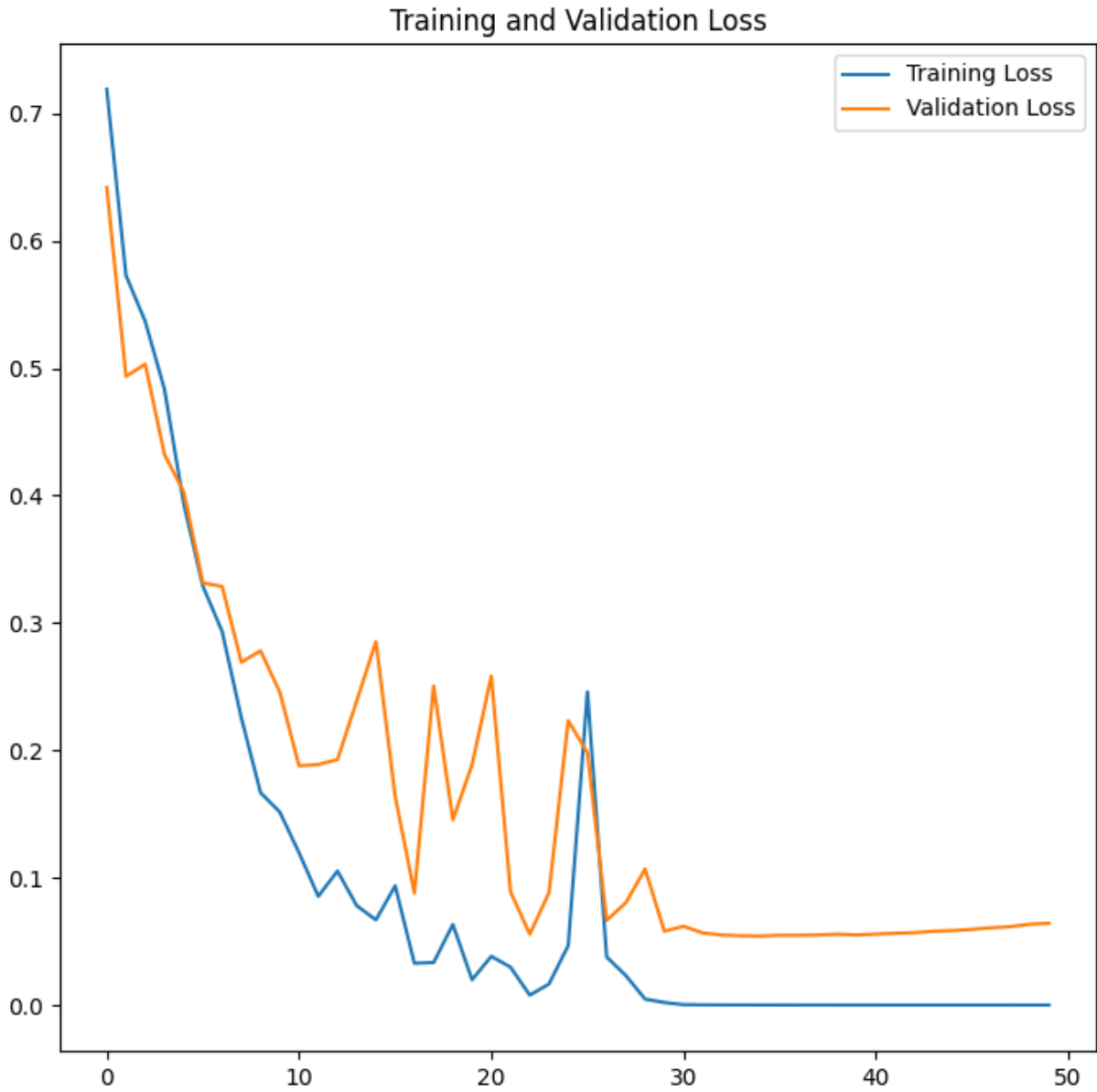


Figure 58 - Training and Validation Loss Plot

Graph Results Discussion

The model demonstrated a promising initial accuracy of 60%, which experienced a rapid increase within the first 10 epochs, surpassing the 90% threshold. However, the subsequent improvement rate gradually slowed down. Compared to the models examined in the earlier comparison, it became apparent that the model in this study required a relatively larger number of epochs to achieve its optimal accuracy. Specifically, it took approximately 40 to 50 epochs for the model to reach its peak performance.

The accuracy and loss graphs exhibit an inverse relationship, indicating that higher accuracy corresponds to lower loss. Although minor discrepancies exist, the overall trend suggests that as the model improves its performance, the loss decreases, resulting in higher accuracy. During the training process, there were two noticeable periods, epochs 10 to 20 and 30 to 50, where slight overfitting may have occurred.

To address these observations, future improvements could focus on mitigating the observed overfitting during specific epochs. Implementing additional regularization techniques, such as weight decay, or adjusting the model architecture may help enhance the model's generalization capabilities [82]. Furthermore, exploring alternative optimization algorithms or refining hyperparameter tuning strategies could potentially optimize the model's performance and reduce the number of epochs required to achieve peak accuracy.

5.3 GUI and User Experience

The frontend, as discussed earlier, is a proof-of-concept to demonstrate the technology's applicability. As previously mentioned, the GUI used a card structure, as will be shown in the following figures.

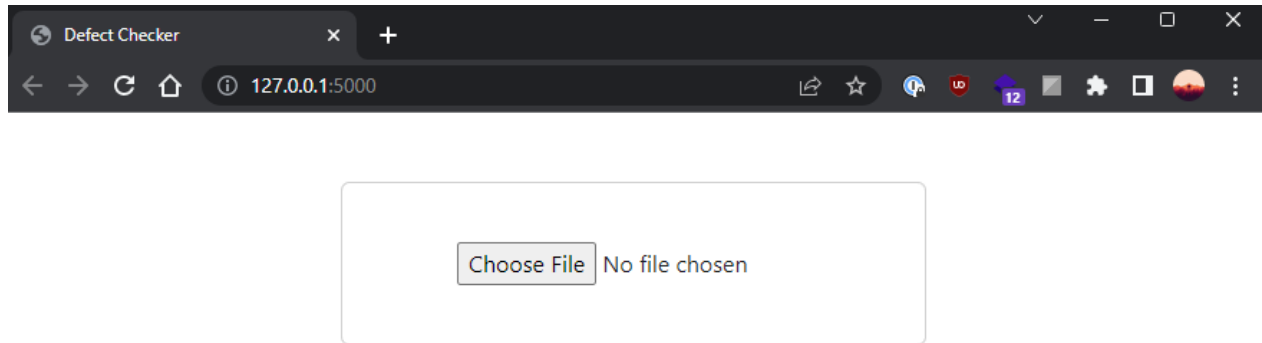


Figure 59 - Implementation Landing Page

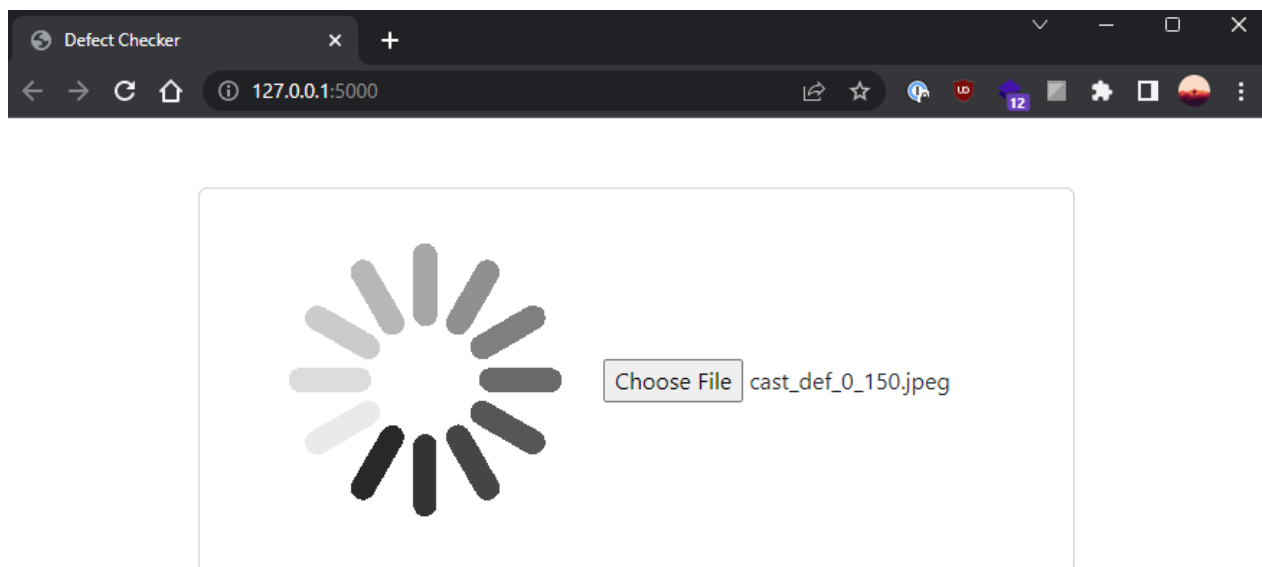


Figure 60 - Implementation GUI Loading GIF

The first figure above illustrates the landing page, featuring a card that includes a file selection button; this design allows users to upload files easily.

The second figure showcases the page after an image has been uploaded, depicting a waiting state as the server processes the request. This is indicated by an animated loading icon, providing visual feedback to the user.

In the event of a back-end error or request timeout, an error message is displayed, as shown in the figure below. This prompt alerts the user to the issue encountered during the processing of their request.

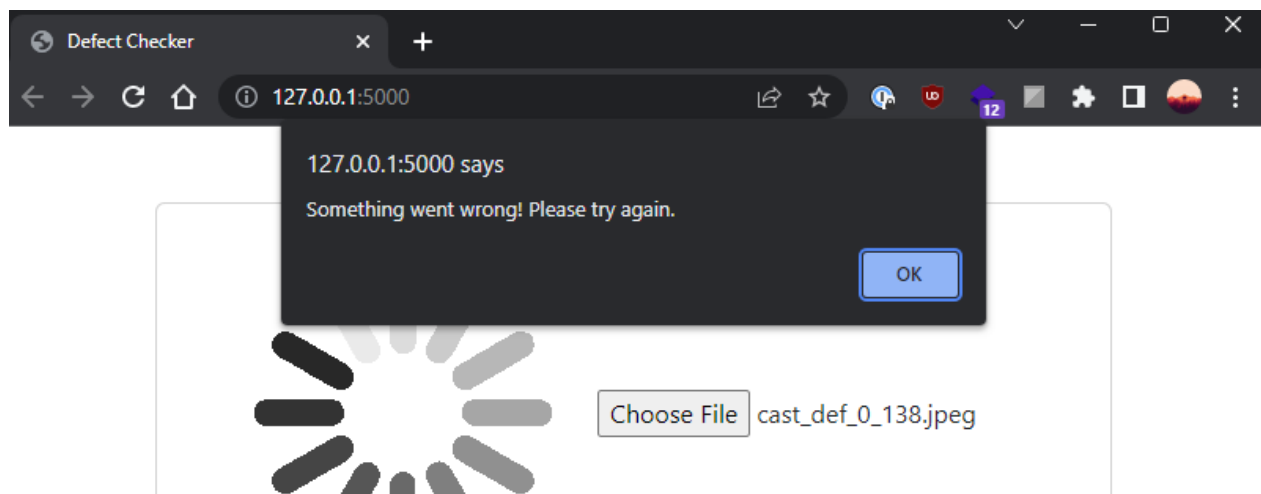


Figure 61 - Implementation GUI Error Message

The figure below presents the output obtained when an image of a non-defective casting is uploaded. The confidence percentage accompanying the classification indicates the level of certainty expressed by the model in its classification decision. Given the high accuracy of the model utilized for this application – specifically, a transfer learning trained VGG16 model – the predictions are often made with 100% confidence. This indicates the model's strong level of certainty in its classifications.

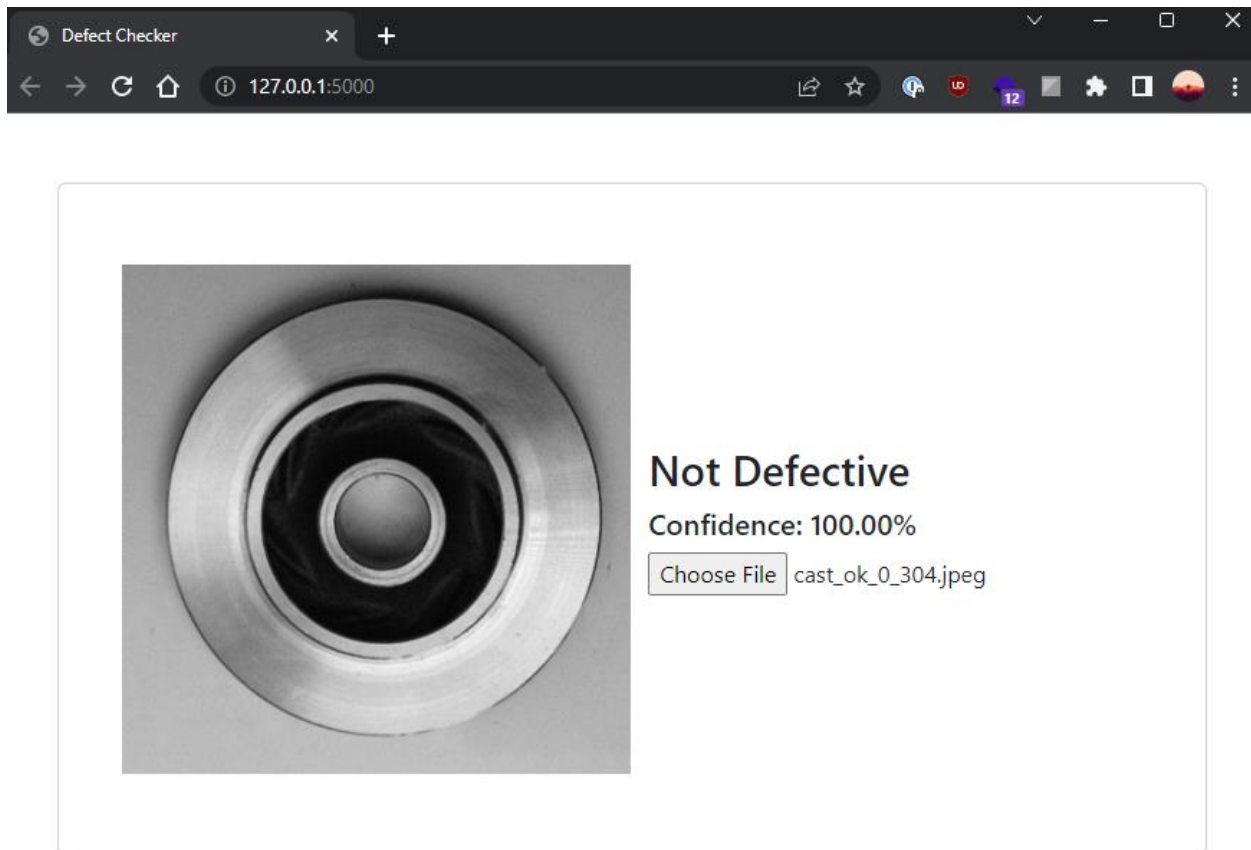


Figure 62 - Implementation GUI Result (Not Defective)

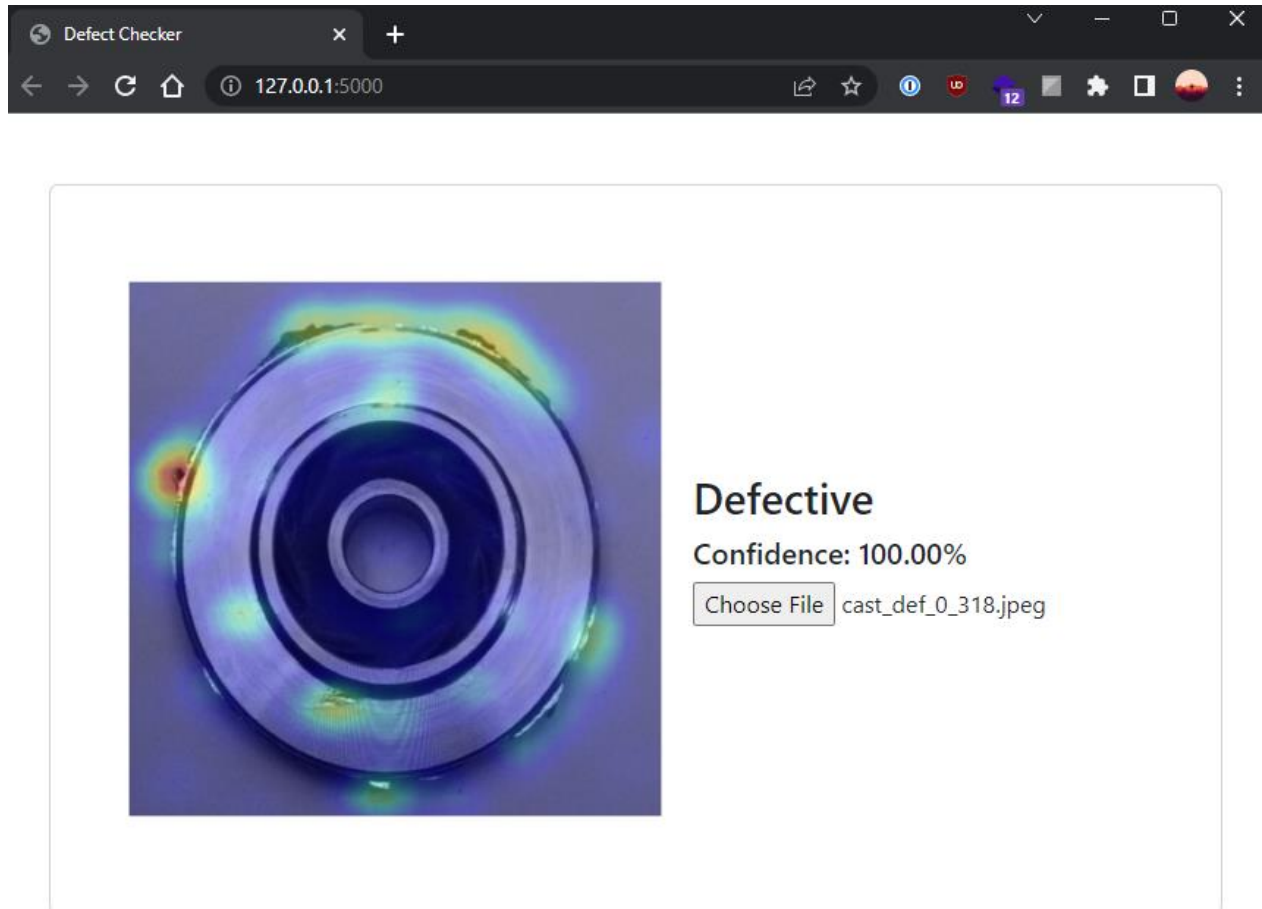


Figure 63 - Implementation GUI Result (Defective)

The activation map in the figure above highlights how techniques such as CAM can be used to localize defects and provide insights into model decision-making. It also illustrates the ability of activation maps to provide error-finding capabilities. In the figure, a minor inconsistency in the background towards the top right portion resulted in some erroneous weighting being assigned. This observation emphasizes the significance of incorporating variations and diverse samples within the dataset and its implications on the network's decision-making.

5.4 Discussion

This discussion centres around the fulfilment of both functional and non-functional requirements that were successfully met (identifiable by the requirement ID in italics). Discussion regarding the unmet requirements can be found in Section 6.2, which specifically addresses the limitations encountered during the project.

The application and model's speed can be evaluated by measuring the time required for model inference. In the figure below, the first log line depicts a 20-millisecond interval, where the endpoint initiates a thread to make a prediction using the entire model, followed by a POST response. The subsequent three lines demonstrate inference times of 22 milliseconds, 18 milliseconds, and another POST response. This demonstration confirms the fulfilment of requirements *M-NFR1*, *M-NFR3*, and *F-NFR2*; as fast model inference (*M-NFR1*) leads to batch processing capability (*M-NFR3*), resulting in a smooth and responsive user experience (*F-NFR2*).

```
1/1 [=====] - 0s 20ms/step
127.0.0.1 - - [25/May/2023 20:45:19] "POST /process-image HTTP/1.1" 204 -
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 18ms/step
127.0.0.1 - - [25/May/2023 20:45:30] "POST /process-image HTTP/1.1" 200 -
```

Figure 64 - Implementation Performance Logs

The initial POST involved analysing a non-defective image, requiring only one inference. In contrast, the subsequent POST involved a defective image that necessitated additional processing. The second image required two inferences to be made. Firstly, the application checked if the image was defective and received a positive result. Secondly, the activation

mapping function extracted weightings from the final convolutional layer of the model, leading to a separate prediction.

While the prediction performance of this model is indeed dependent on the server's graphics processing power, being a web application with server-side processing means that the client can seamlessly use a low-powered device without experiencing any significant decrease in speed, increasing its interoperability. Additionally, the application's modern frameworks ensure compatibility with a wide range of browsers (*F-NFR3*). The user interface, designed with cards, is intuitive and straightforward (*F-NFR1*) and it is responsive to different screen sizes (*F-NFR4*).

After uploading an image through the file upload button (*F-FR1*), the system generates an output image that includes an activation map overlay (*F-FR4*) and displays the classification results (*F-FR2*). This image can be downloaded by the user for further analysis (*F-FR5*).

The proposed model can classify surface defects (*M-FR1*) to an accuracy of 99.23% (*M-FR4*), accepting any RGB image as input (*M-FR3*), and could provide insights into decision-making through feature activation map extraction (*M-NFR4*).

*In conclusion, all requirements were met other than **M-FR2**, **M-NFR2**, and **F-FR3**, which are discussed in **Limitations** (section 6.2).*

6. Conclusion and Future Work

6.1 Verification of the Study

The study followed established best practices and guidelines in the field of deep learning and model evaluation. A sufficiently sized, high-quality, and balanced dataset was selected for training and validation. This dataset selection process aimed to provide representative samples and mitigate biases that could impact the model's performance.

During the training process, the Adam optimizer, a widely recognized industry standard [82], was employed to optimize the model's parameters. The Adam optimizer is known for its effectiveness in yielding successful and reliable results [83]. Furthermore, the implementation of the dropout regularization technique was utilized to minimize overfitting, promoting better generalization of the model.

To assess the models' performance, various evaluation metrics, including accuracy, precision, recall, and F-1 score, were employed. These metrics are widely adopted in machine learning tasks due to their ability to provide a comprehensive assessment of the model's classification ability [84]. Accuracy measures the overall correctness of predictions, while precision and recall focus on the model's ability to minimize false positives and false negatives, respectively. The F-1 score combines both precision and recall, offering a balanced evaluation of the model's performance.

In conclusion, the experimental setup, along with evaluation metrics, statistical analyses, and comparative assessments with existing architectures contribute to the credibility and reliability of the study's findings, allowing for meaningful conclusions to be drawn and providing insights into the performance of the CNN architecture in comparison to other established models.

6.2 Limitations

Throughout the project, a few limitations have been identified that warrant discussion. These limitations can be categorized into two groups: unmet requirements and additional limitations beyond the specified project requirements.

6.2.1 Unmet Requirements

Table 10 - Unmet Requirements

Requirement ID	Description	Justification/Explanation
M-FR2	The system shall support multiple classes of surface defects.	Annotating the images accurately proved to be highly challenging due to the wide range of distinct casting defects and the lack of domain knowledge. The complexity and diversity of the defects made it impractical to annotate the images accurately.
M-NFR2	The model shall be able to handle variations in lighting conditions and camera angles.	The dataset consisted of photos taken from the same general angle in mostly uniform lighting conditions. There were no other similar datasets, therefore the model could not be trained as such.
F-FR3	The frontend shall support real-time streaming of images from a connected camera.	As the frontend is a proof-of-concept system, it instead used manual file upload functionality. Switching to a camera would be relatively simple in implementation, however, the author did not have access to the SPIs.

6.2.2 Additional Limitations

The main limitation encountered other than those already listed was limited GPU memory, which prevented training using higher dimension images such as 512x512. This was the initial reasoning for downscaling the images to lower dimensions for model comparison, however, it turned out that classification performance consequently marginally improved. Additionally, as 512x512 training was not possible using the proposed model, it was decided to use a fine-tuned VGG16 model instead for the frontend, to retain image quality for demonstration purposes.

6.3 Future Work

Although the project achieved overall success, there are still areas that warrant improvement and further development. Firstly, annotating the dataset to provide labelled bounding boxes and the ability to distinguish between different types of defects (*M-FR2*) would greatly enhance the system's effectiveness. By incorporating this additional information into the training process, the model can be trained to accurately identify and classify various defect categories, resulting in more comprehensive defect detection and data aggregation.

Secondly, further efforts should be dedicated to redesigning and developing the model with the goal of achieving a smaller size and higher memory efficiency. This optimization would not only contribute to faster inference times and reduced computational requirements but also make the system more scalable and accessible for real-world deployment. By optimizing the model's efficiency, the system would be able to handle larger datasets and be deployed on resource-constrained devices more effectively.

In addition to model improvements, the integration of live camera functionality into the front-end application (*F-FR3*) would greatly enhance the system's usability and practicality. This feature would allow users to capture real-time images or videos for defect detection, providing immediate feedback and facilitating prompt decision-making through more seamless interactivity.

Addressing these aspects would result in a more accurate, efficient, and user-friendly system, elevating its overall performance and impact.

6.4 Conclusion

The primary finding of this research underscores the effectiveness of a novel deep neural network in surface defect classification tasks. However, it also highlights that employing transfer learning techniques can yield equal or potentially superior results in terms of effectiveness and efficiency. This discovery emphasizes the significance of leveraging pre-trained models and existing knowledge in the field of deep learning to enhance the performance and productivity of defect classification systems. By leveraging transfer learning, researchers and practitioners can achieve robust and accurate defect detection while optimizing computational resources and reducing training time. This insight contributes to the broader understanding of the practical implementation and optimization of DNN models for surface defect classification, paving the way for more efficient and effective applications in industrial and quality control domains.

7. References

- [1] Carrier Vibrating, "Reducing The Rate Of Casting Defects In the Foundry Line," 1 June 2018. [Online]. Available: <https://www.carriervibrating.com/resources/blog/reducing-rate-of-casting-defects-foundry-line/>. [Accessed 5 November 2022].
- [2] F. Kreier, "Metal-lifespan analysis shows scale of waste," 2022. [Online]. Available: <https://www.nature.com/articles/d41586-022-01467-8>. [Accessed 6 November 2022].
- [3] X. Tao, D. Zhang, W. Ma, X. Liu and D. Xu, "Automatic Metallic Surface Defect Detection and Recognition with Convolutional Neural Networks," *Special Issue Advanced Intelligent Imaging Technology*, vol. 8, no. 9, pp. 1575-1600, 2018.
- [4] Editorial, "Deep Learning Vs. Traditional Image Processing – A Comparison," 2021. [Online]. Available: <https://roboticsbiz.com/deep-learning-vs-traditional-image-processing-a-comparison/>. [Accessed 12 November 2022].
- [5] J. Howarth, "57+ Amazing Artificial Intelligence Statistics (2022)," 2022. [Online]. Available: <https://explodingtopics.com/blog/ai-statistics>. [Accessed 6 November 2022].
- [6] C. M. Ryan, A. Parnell and C. Mahoney, *Real-Time Anomaly Detection for Advanced Manufacturing: Improving on Twitter's State of the Art*, 1911.05376: arXiv, 2019.
- [7] L. M. G. Fonesca, L. M. Namikawa and E. F. Castejon, "'Digital Image Processing in Remote Sensing", 2009 Tutorials of the XXII Brazilian Symposium on Computer Graphics and Image Processing," 2009.
- [8] M. Sood, "Digital Image Processing," 2019. [Online]. Available: <https://cevgroup.org/digital-image-processing/>. [Accessed 20 November 2022].
- [9] M. Azimi, "Digital Image Processing Lectures 1 & 2," [Online]. Available: https://www.engr.colostate.edu/ECE513/SP09/lectures/lectures1_2.pdf. [Accessed 20 November 2022].
- [10] R. Kundu, "Image Processing: Techniques, Types, & Applications [2022]," 2022. [Online]. Available: <https://www.v7labs.com/blog/image-processing-guide>. [Accessed 24 November 2022].
- [11] Y. Shin, M. Kim, K.-W. Pak and D. Kim, "Practical Methods of Image Data Preprocessing for Enhancing the Performance of Deep Learning Based Road Crack Detection," *ICIC Express Letters Part B: Applications*, vol. 11, no. 4, pp. 373-379, 2020.
- [12] D. G. Ranganathan, "A Study to Find Facts Behind Preprocessing on," *Journal of Innovative Image Processing (JIIP)*, vol. 03, no. 01, pp. 66-74, 2021.

-
- [13] A. Fredrick, "Getting Started with Image Preprocessing in Python," 2021. [Online]. Available: <https://www.section.io/engineering-education/image-preprocessing-in-python/>. [Accessed 26 November 2022].
- [14] G. Kumar and P. K. Bhatia, "A Detailed Review of Feature Extraction in Image Processing Systems," 2014.
- [15] B. J. Lei, E. A. Hendriks and M. J. T. Reinders, "On Feature Extraction from Images," 1999.
- [16] A. Humeau-Heurtier, "Texture Feature Extraction Methods: A Survey," *IEEE Access*, vol. 7, pp. 8975-9000, 2019.
- [17] MathWorks, "Feature extraction for machine learning and deep learning," [Online]. Available: <https://uk.mathworks.com/discovery/feature-extraction.html>. [Accessed 25 November 2022].
- [18] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan and J. Walsh, "Deep Learning vs. Traditional Computer Vision," [Online]. Available: <https://arxiv.org/ftp/arxiv/papers/1910/1910.13796.pdf>. [Accessed 25 November 2022].
- [19] N. Neogi, D. K. Mohanta and P. K. Dutta, "Review of vision-based steel surface inspection systems," *EURASIP Journal on Image and Video Processing*, no. Article number: 50, 2014.
- [20] X. Sun, J. Gu, S. Tang and J. Li, "Research Progress of Visual Inspection Technology of Steel Products—A Review," *Applied Sciences*, vol. 8, no. 11, 2018.
- [21] X. Zheng, S. Zheng, Y. Kong and J. Chen, "Recent advances in surface defect inspection of industrial products using deep learning techniques," *The International Journal of Advanced Manufacturing Technology*, vol. 113, pp. 35-58, 2021.
- [22] T. Wang, "Spectral Methods and Computational Trade-offs in," Cambridge, 2016.
- [23] H. Jia, Y. Murphey, J. Shi and T.-S. Chang, "An intelligent real-time vision system for surface defect detection," 2004.
- [24] J. Yang, S. Li, Z. Wang and G. Yang, "Real-Time Tiny Part Defect Detection System in Manufacturing Using Deep Learning," *IEEE Access*, vol. 7, pp. 89278-89291, 2019.
- [25] ITiger, "understanding basic difference between a CNN and RNN," 18 January 2018. [Online]. Available: <https://stackoverflow.com/questions/48318448/understanding-basic-difference-between-a-cnn-and-rnn#:~:text=So%2C%20to%20answer%20your%20question,usually%20used%20for%20image%20classification..> [Accessed 27 November 2022].
- [26] M. K. Gurucharan, "Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network," 28 July 2022. [Online]. Available:

-
- <https://www.upgrad.com/blog/basic-cnn-architecture/#:~:text=other%20advanced%20tasks,-,What%20is%20the%20architecture%20of%20CNN%3F,the%20main%20responsibility%20for%20computation..> [Accessed 27 November 2022].
- [27] Cortex Robotics, "Automated Inspection Systems vs Human Visual Inspection," [Online]. Available: <https://cortexrobotics.my/automated-inspection-systems-vs-human-visual-inspection/#:~:text=Automated%20optical%20inspection%20systems%20play,procedures%20in%20most%20production%20cycles..> [Accessed 28 November 2022].
- [28] S. Ravikumar, K. I. Ramachandran and V. Sugamaram, "Machine learning approach for automated visual inspection of machine components," *Expert Systems with Applications*, vol. 38, no. 4, pp. 3260-3266, 2011.
- [29] D. Theckedath and R. R. Sedamkar, "Detecting Affect States Using VGG16, ResNet50 and SE-ResNet50 Networks," *SN Computer Science*, vol. 1, no. 2, p. 79, 2020.
- [30] A. Shazia, T. Z. Xuan, J. H. Chuah, J. Usman, P. Qian and K. W. Lai, "A comparative study of multiple neural network for detection of COVID-19 on chest X-ray," *EURASIP Journal on Advances in Signal Processing*, vol. 2021, no. 1, p. 50, 2021.
- [31] R. Ren, T. Hung and K. C. Tan, "A Generic Deep-Learning-Based Approach for Automated Surface Inspection," *IEEE Transactions on Cybernetics*, vol. 48, no. 3, pp. 929-940, 2018.
- [32] techopedia, "Data Acquisition," techopedia, 9 February 2018. [Online]. Available: <https://www.techopedia.com/definition/30000/data-acquisition>. [Accessed 28 November 2022].
- [33] Wikipedia, "Analog-to-digital converter," [Online]. Available: https://en.wikipedia.org/wiki/Analog-to-digital_converter. [Accessed 26 May 2023].
- [34] J. Valvano and R. Yerraballi, "Chapter 14: Analog to Digital Conversion, Data Acquisition and Control," [Online]. Available: https://users.ece.utexas.edu/~valvano/Volume1/E-Book/C14_ADCdataAcquisition.htm. [Accessed 26 May 2023].
- [35] M. Javaid, A. Haleem, S. Rab, R. P. Singh and R. Suman, "Sensors for daily life: A review," *Sensors International*, vol. 2, no. 2666-3511, pp. 100-121, 2021.
- [36] Britannica, "transducer," [Online]. Available: <https://www.britannica.com/technology/transducer-electronics>. [Accessed 26 May 2023].

-
- [37] DATAQ Instruments, “WINDAQ Acquisition and Playback Software,” [Online]. Available: <https://www.fieldworks.nl/media/files/WinDaq.pdf>. [Accessed 26 May 2023].
- [38] Tektronix, “Why is ActiveX? technology important to my data acquisition applications?,” [Online]. Available: <https://www.tek.com/en/support/faqs/why-activex-technology-important-my-data-acquisition-applications>. [Accessed 26 May 2023].
- [39] The R Foundation, “What is R?,” [Online]. Available: <https://www.r-project.org/about.html>. [Accessed 28 November 2022].
- [40] V. Kumar, “Python Vs R: What’s Best for Machine Learning,” Towards Data Science, 11 September 2019. [Online]. Available: <https://towardsdatascience.com/python-vs-r-whats-best-for-machine-learning-93432084b480>. [Accessed 29 November 2022].
- [41] Data Flair, “Keras vs OpenCV – Differences Between OpenCv and Keras,” [Online]. Available: <https://data-flair.training/blogs/keras-vs-opencv/>. [Accessed 28 November 2022].
- [42] K. Dubovikov, “PyTorch vs TensorFlow — spotting the difference,” Towards Data Science, 20 June 2017. [Online]. Available: <https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b>. [Accessed 29 November 2022].
- [43] W3schools, “NumPy Introduction,” W3schools, [Online]. Available: https://www.w3schools.com/python/numpy/numpy_intro.asp. [Accessed 30 November 2022].
- [44] lukewood, “Writing Keras Models With TensorFlow NumPy,” 28 August 2021. [Online]. Available: https://keras.io/examples/keras_recipes/tensorflow_numpy_models/. [Accessed 30 November 2022].
- [45] InterviewBit, “Pandas Vs NumPy: What’s The Difference? [2022],” InterviewBit, 1 September 2022. [Online]. Available: <https://www.interviewbit.com/blog/pandas-vs-numpy/#:~:text=Pandas%20is%20mostly%20used%20for,easy%20to%20apply%20mathematical%20functions.&text=Pandas%20library%20works%20well%20for,heterogeneous%20types%20of%20data%20simultaneously..> [Accessed 30 November 2022].
- [46] MathWorks, “Programming with MATLAB,” MathWorks, [Online]. Available: <https://uk.mathworks.com/products/matlab/programming-with-matlab.html>. [Accessed 1 December 2022].
- [47] LabelBox, “Building vs. buying a training data platform,” LabelBox, [Online]. Available: <https://labelbox.com/learn/build-vs-buy/>. [Accessed 1 December 2022].

-
- [48] Turing, "Python FastAPI vs Flask: A Detailed Comparison," Turing, [Online]. Available: <https://www.turing.com/kb/fastapi-vs-flask-a-detailed-comparison>. [Accessed 1 December 2022].
- [49] R. Dabhi, "casting product image data for quality inspection," [Online]. Available: <https://www.kaggle.com/datasets/ravirajsinh45/real-life-industrial-dataset-of-casting-product>. [Accessed 20 May 2023].
- [50] Atlassian, "What is Jira used for?," Atlassian, [Online]. Available: <https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for#jira-for-software-development-teams>. [Accessed 28 November 2022].
- [51] Keras, "Keras Applications," [Online]. Available: <https://keras.io/api/applications/>. [Accessed 16 May 2023].
- [52] DeepAI, "What is an Inception Module?," [Online]. Available: <https://deepai.org/machine-learning-glossary-and-terms/inception-module>. [Accessed 17 May 2023].
- [53] A Name Not Yet Taken AB, "ResNet50 Image Classification in Python," 27 May 2020. [Online]. Available: <https://www.annytab.com/resnet50-image-classification-in-python/>. [Accessed 19 May 2023].
- [54] X. Feng, X. Gao and L. Luo, "A ResNet50-Based Method for Classifying Surface Defects in Hot-Rolled Strip Steel," *Mathematics*, vol. 9, no. 19, p. 1, 2021.
- [55] AnalyticsVidhya, "10 Techniques to Solve Imbalanced Classes in Machine Learning (Updated 2023)," 26 April 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/>. [Accessed 16 May 2023].
- [56] S. Narkhede, "Understanding Confusion Matrix," 9 May 2018. [Online]. Available: <https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62>. [Accessed 16 May 2023].
- [57] J. Korstanje, "The F1 score," 31 August 2021. [Online]. Available: <https://towardsdatascience.com/the-f1-score-bec2bbc38aa6>. [Accessed 16 May 2023].
- [58] S. Kumar, "Parameter vs Hyperparameters In Machine Learning," LinkedIn, 17 January 2023. [Online]. Available: <https://www.linkedin.com/pulse/parameter-vs-hyperparameters-machine-learning-sanjay-kumar-mba-ms-phd/>. [Accessed 26 May 2023].
- [59] E. Elgeldawi, A. Sayed, A. R. Galal and A. M. Zaki, "Hyperparameter Tuning for Machine Learning Algorithms Used for Arabic Sentiment Analysis," *Informatics*, vol. 8, no. 2227-9709, 2021.

-
- [60] K. Nyuytiybiy, "Parameters and Hyperparameters in Machine Learning and Deep Learning," 30 December 2020. [Online]. Available: <https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac>. [Accessed 19 May 2023].
- [61] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei and S.-H. Deng, "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1674-862X, pp. 26-40, 2019.
- [62] B. Li, "Random Search Plus: A more effective random search for machine learning hyperparameters optimization," Master's Thesis, University of Tennessee, 2020.
- [63] J. Snoek, H. Larochelle and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951-2959.
- [64] P. I. Frazier, W. B. Powell and S. Dayanik, "Tutorial on Bayesian optimization," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 1-52, 2017.
- [65] D. J. Murray-Smith, "Sensitivity Analysis for Model Evaluation," in *Testing and Validation of Computer Simulation Models: Principles, Methods and Applications*, Springer International Publishing, 2015, pp. 49-60.
- [66] J. D. Saliccioli, Y. Crutain, M. Komorowski and D. C. Marshall, "Sensitivity Analysis and Model Validation," in *Secondary Analysis of Electronic Health Records*, Springer International Publishing, 2016, pp. 263-271.
- [67] J. B. Maverick, J. R. Brown and P. Rathburn, "How Is Sensitivity Analysis Used?," Investopedia, 28 March 2022. [Online]. Available: <https://www.investopedia.com/ask/answers/052115/what-are-some-examples-ways-sensitivity-analysis-can-be-used.asp>. [Accessed 26 May 2023].
- [68] G. Teodoro, T. M. Kurç, L. F. R. Taveira, A. C. M. A. Melo, Y. Gao, J. Kong and J. Saltz, "Algorithm sensitivity analysis and parameter tuning for tissue image segmentation," *Bioinformatics.*, vol. 7, no. 33, pp. 1064-1072, 2017.
- [69] M. He, B. Li and S. Sun, "A Survey of Class Activation Mapping for the Interpretability of Convolution Neural Networks," in *International Conference On Signal And Information Processing, Networking And Computers*, San Antonio, TX, 2023.
- [70] Y.-h. Sheu, "Illuminating the Black Box: Interpreting Deep Neural Network Models for Psychiatric Research," *Frontiers in Psychiatry*, vol. 11, no. 1664-0640, 2020.
- [71] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh and D. Batra, *Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization*, 1610.02391, 2019.

-
- [72] R. L. Draelos and L. Carin, *Use HiResCAM instead of Grad-CAM for faithful explanations of convolutional neural networks*, 2011.08891, 2021.
- [73] H. Talebi and P. Milanfar, *Learning to Resize Images for Computer Vision Tasks*, 2103.09950: arXiv, 2021.
- [74] Keras, "VGG16 and VGG19," [Online]. Available: <https://keras.io/api/applications/vgg/>. [Accessed 26 May 2023].
- [75] H. Lin, W. Zeng, X. Ding, Y. Huang, C. Huang and J. Paisley, *Learning Rate Dropout*, 1912.00144: arXiv, 2019.
- [76] L. Zeng, H. Zhang, Y. Li, M. Li and S. Wang, "Supervision dropout: guidance learning in deep neural network," *Multimedia Tools and Applications*, vol. 18850, no. 12, p. 18831, 2023.
- [77] E. Zvornicanin, "Relation Between Learning Rate and Batch Size," Baeldung, 16 March 2023. [Online]. Available: <https://www.baeldung.com/cs/learning-rate-batch-size>. [Accessed 21 May 2023].
- [78] J. Brownlee, "A Gentle Introduction to Dropout for Regularizing Deep Neural Networks," *Machine Learning Mastery*, 3 December 2018. [Online]. Available: <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>. [Accessed 27 May 2023].
- [79] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, 1412.6980: arXiv, 2017.
- [80] Sakshii, "Adam optimizer: A Quick Introduction," AskPython, 27 February 2023. [Online]. Available: <https://www.askpython.com/python/examples/adam-optimizer>. [Accessed 21 May 2023].
- [81] StackOverflow, "Dropout rate guidance for hidden layers in a convolution neural network," 24 December 2017. [Online]. Available: <https://stackoverflow.com/questions/47892505/dropout-rate-guidance-for-hidden-layers-in-a-convolution-neural-network>. [Accessed 21 May 2023].
- [82] K. Ajitesh, "Weight Decay in Machine Learning: Concepts," 7 June 2022. [Online]. Available: <https://vitalflux.com/weight-decay-in-machine-learning-concepts/>. [Accessed 27 May 2023].
- [83] S. Bock, J. Goppold and M. Weiß, *An improvement of the convergence proof of the ADAM-Optimizer*, 1804.10587: arXiv, 2018.
- [84] N. B. Harikrishnan, "Confusion Matrix, Accuracy, Precision, Recall, F1 Score," *Analytics Vidhya*, 10 December 2019. [Online]. Available: <https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd>. [Accessed 28 May 2023].

-
- [85] C. Helbig, "Losses and lifetimes of metals in the economy," 2022. [Online]. Available: <https://www.nature.com/articles/s41893-022-00895-8>. [Accessed 6 November 2022].
- [86] D. Tabernik, S. Šela, J. Skvarč and D. Skočaj, "Segmentation-based deep-learning approach for surface-defect detection," *Journal of Intelligent Manufacturing*, vol. 31, pp. 759-776, 2020.
- [87] C. P. Maria Petrou, *Image Processing: The Fundamentals*, 2nd ed., Chichester: John Wiley and Sons, Ltd., 2010.
- [88] L. M. N. E. F. C. L. M. G. Fonseca, "'Digital Image Processing in Remote Sensing", 2009 *Tutorials of the XXII Brazilian Symposium on Computer Graphics and Image Processing*," 2009.
- [89] MathWorks, "Enhancement methods in image processing," [Online]. Available: <https://uk.mathworks.com/discovery/image-enhancement.html>. [Accessed 24 November 2022].
- [90] O. Patel, Y. P. S. Maravi and S. Sharma, "A Comparative Study of Histogram Equalization Based Image Enhancement Techniques for Brightness Preservation and Contrast Enhancement," *Signal & Image Processing : An International Journal*, vol. 4, no. 5, pp. 11-25, 2013.
- [91] M. Elgendy, "The Computer Vision Pipeline, Part 4: feature extraction," 2020. [Online]. Available: <https://freecontent.manning.com/the-computer-vision-pipeline-part-4-feature-extraction/>. [Accessed 25 November 2022].
- [92] S. Balaji, "Binary Image classifier CNN using TensorFlow," 29 August 2020. [Online]. Available: <https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697>. [Accessed 27 November 2022].
- [93] P. Damacharla, A. R. M. V., J. Ringenberg and A. Y. Javaid, "TLU-Net: A Deep Learning Approach for Automatic Steel Surface Defect Detection," in *2021 International Conference on Applied Artificial Intelligence (ICAPAI)*, 2021.
- [94] P. Huilgol, "Top 4 Pre-Trained Models for Image Classification with Python Code," *AnalyticsVidhya*, 12 May 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/>. [Accessed 16 May 2023].
- [95] Z. Wharton, A. Behera and A. Bera, *An attention-driven hierarchical multi-scale representation for visual recognition*, 2110.12178, 2021.
- [96] G. Rohini, "Everything you need to know about VGG16," 23 September 2021. [Online]. Available: <https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918>. [Accessed 17 May 2023].

-
- [97] M. Shaha and M. Pawar, "Transfer Learning for Image Classification," in *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2018, pp. 656-660.
- [98] Datagen, "Understanding VGG16: Concepts, Architecture, and Performance," [Online]. Available: <https://datagen.tech/guides/computer-vision/vgg16/>. [Accessed 17 May 2023].
- [99] Google Cloud, "Advanced Guide to Inception v3," [Online]. Available: <https://cloud.google.com/tpu/docs/inception-v3-advanced>. [Accessed 17 May 2023].
- [100] S. Mukherjee, "The Annotated ResNet-50," 18 August 2022. [Online]. Available: <https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758>. [Accessed 19 May 2023].
- [101] MathWorks, "Investigate Network Predictions Using Class Activation Mapping," MathWorks, [Online]. Available: <https://www.mathworks.com/help/deeplearning/ug/investigate-network-predictions-using-class-activation-mapping.html>. [Accessed 19 May 2023].
- [102] A. Makris, I. Kontopoulos and K. Tserpes, "COVID-19 detection from chest X-Ray images using Deep Learning and Convolutional Neural Networks," *medRxiv*, no. 10.1101/2020.05.22.20110817, 2020.

Appendix



Miscellaneous

Ethics Form

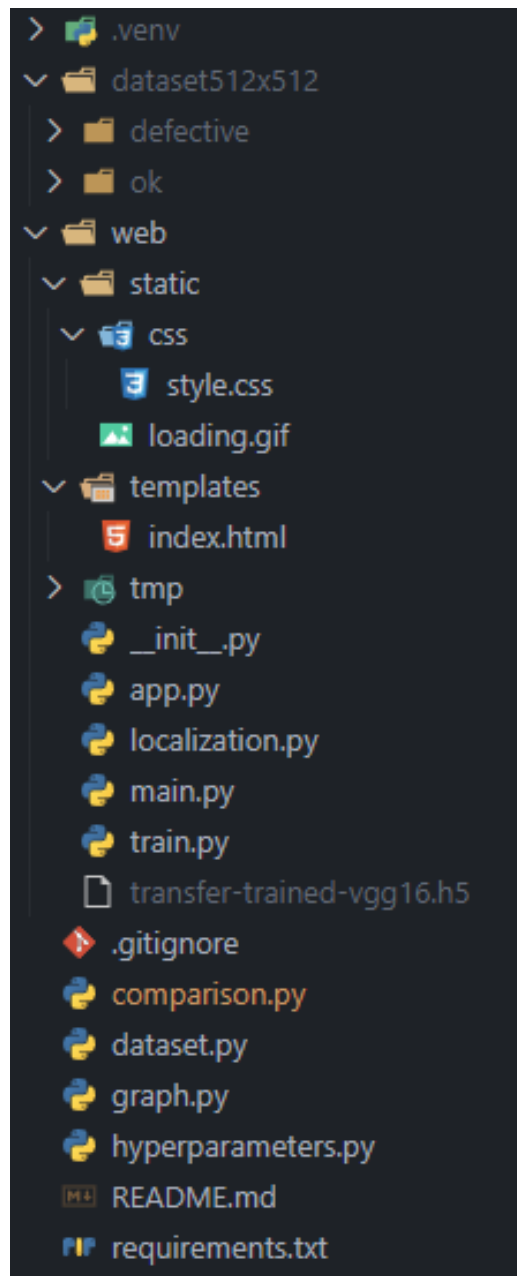


School of
Engineering

**BSc FINAL YEAR PROJECT
ETHICAL CONSIDERATIONS
2022-23**

YOUR DETAILS	
Name of student: <u>Theodore Smith Scott</u>	
Supervisor: <u>Bugra Alkan</u>	
Project title: <u>Investigating Deep Neural Networks in the Context of Surface Defect Detection in Metal Casting</u>	
Main aim of project: <u>To create a novel deep neural network, compare and evaluate different existing deep learning architectures, and create a proof-of-concept defect detection system for submersible pump impellers.</u>	
CONTACT WITH OTHERS	
Will your project bring you into contact with other people (e.g. via an online survey)?	
Yes <input type="checkbox"/> No <input checked="" type="checkbox"/>	
If you answered "No", sign the section below and submit this page only to your supervisor for countersigning, otherwise complete the whole form prior to submission. Also, if you answered "No" then only this page needs to be included as an appendix to your dissertation.	
YOUR SIGNATURE	
Signature: <u></u>	Date: <u>29/05/2023</u>
ETHICAL APPROVAL (To be completed by your supervisor)	
I have checked the above for accuracy and I am satisfied that the information provided is an accurate reflection of the intended study.	
There are no ethical issues causing my concern <input checked="" type="checkbox"/>	
Signature: <u></u>	Date: <u>29/05/2023</u>
Name (please print): <u>Bugra Alkan</u>	

Project Directory Structure



requirements.txt

```
absl-py==1.3.0
asttokens==2.2.1
astunparse==1.6.3
async-timeout==4.0.2
backcall==0.2.0
bayesian-optimization==1.4.3
cachelib==0.10.2
cachetools==5.2.0
certifi==2022.9.24
charset-normalizer==2.1.1
click==8.1.3
colorama==0.4.6
contourpy==1.0.7
cyclier==0.11.0
decorator==5.1.1
executing==1.2.0
Flask==2.2.3
Flask-Session==0.4.0
flatbuffers==22.11.23
fonttools==4.38.0
gast==0.4.0
google-auth==2.14.1
google-auth-oauthlib==0.4.6
google-pasta==0.2.0
grpcio==1.51.1
h5py==3.7.0
idna==3.4
importlib-metadata==6.0.0
importlib-resources==5.12.0
imutils==0.5.4
ipython==8.11.0
itsdangerous==2.1.2
jedi==0.18.2
Jinja2==3.1.2
joblib==1.2.0
keras==2.10.0
Keras-Preprocessing==1.1.2
kiwisolver==1.4.4
libclang==14.0.6
Markdown==3.4.1
MarkupSafe==2.1.1
matplotlib==3.7.0
matplotlib-inline==0.1.6
```

```
numpy==1.23.5
oauthlib==3.2.2
opencv-python==4.6.0.66
opt-einsum==3.3.0
packaging==21.3
pandas==1.5.3
parso==0.8.3
pickleshare==0.7.5
Pillow==9.4.0
prompt-toolkit==3.0.38
protobuf==3.19.6
pure-eval==0.2.2
pyasn1==0.4.8
pyasn1-modules==0.2.8
Pygments==2.14.0
pyparsing==3.0.9
python-dateutil==2.8.2
pytz==2022.7.1
requests==2.28.1
requests-oauthlib==1.3.1
rsa==4.9
scikit-learn==1.2.1
scipy==1.10.1
six==1.16.0
stack-data==0.6.2
tensorboard==2.10.1
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow==2.10.1
tensorflow-estimator==2.10.0
tensorflow-io-gcs-filesystem==0.28.0
termcolor==2.1.1
threadpoolctl==3.1.0
traitlets==5.9.0
typing_extensions==4.4.0
urllib3==1.26.13
wcwidth==0.2.6
Werkzeug==2.2.2
wrapt==1.14.1
zipp==3.14.0
```

Application Code

app.py

```
import os
import threading
import time
import uuid
from io import BytesIO
import base64

from flask import Flask, render_template, request, jsonify
from localization import is_defective, save_activation_map
from PIL import Image

app = Flask(__name__)
app.config.from_object(__name__)

def garbage_collection(path: str, time_seconds: int):
    time.sleep(time_seconds)
    if os.path.exists(path):
        os.remove(path)
    exit(0)

@app.route('/', methods=['GET'])
def index():
    return render_template('index.html')

@app.route('/process-image', methods=['POST'])
def upload():
    if not request.files['raw_image']:
        return 'No file uploaded'
    else:
        # Get the image from the request
        image = request.files['raw_image']

        # Save the image to the tmp folder
        file_name = str(uuid.uuid4().hex)
        image_path = f"tmp/{file_name}.jpeg"
        output_image_path = os.path.join('tmp', f'{file_name}-output.png')
        image.save(image_path)

        # Check if the image is defective + delete the image after 15 seconds
```

```

        threading.Thread(target=garbage_collection, args=(image_path, 15),
daemon=True).start()

    defective, confidence = is_defective(image_path)

    if defective:
        save_activation_map(image_path)
        # Plot and save the output image to the tmp folder + delete the image
after 60 seconds
        threading.Thread(target=garbage_collection, args=(output_image_path,
60), daemon=True).start()

        with Image.open(output_image_path) as img:
            image_data = BytesIO()
            img.save(image_data, format='PNG')
            image_data.seek(0)

            # Encode the image data as base64
            base64_image = base64.b64encode(image_data.getvalue()).decode()

            response = {'image': base64_image,
                        'is_defective': defective,
                        'confidence': confidence}

            return jsonify(response)
    else:
        response = {'is_defective': defective,
                    'confidence': confidence}

        return jsonify(response)

```

localization.py

```
import os

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import tensorflow as tf
from PIL import Image

from train import train_model
from tensorflow.keras.models import load_model

matplotlib.use('Agg')

if not os.path.exists('transfer-trained-vgg16.h5'):
    print("Model not found!\nTraining model...")
    model = train_model()
print("Loading trained model...")
model = load_model('transfer-trained-vgg16.h5')

def is_defective(image_path):
    with Image.open(image_path) as img:
        img = img.resize((512, 512))
        image = np.array(img)

        # Normalize the image
        image = image / 255.0

        # Make a prediction on the image using the model
        pred = model.predict(image[np.newaxis,:,:,:])
        confidence_percentage = np.max(pred) * 100

        # Check if the predicted class is 0 (defective)
        if np.argmax(pred) == 0:
            return True, confidence_percentage
        else:
            return False, confidence_percentage

weights = model.layers[-1].get_weights()[0]
class_weights = weights[:, 0]
```

```

intermediate = tf.keras.Model(model.input,
model.get_layer("block5_conv3").output)

def save_activation_map(image_path):
    # Load the image file and convert it to a NumPy array
    with Image.open(image_path) as img:
        img = img.resize((512, 512))
        image = np.array(img)

    conv_output = intermediate.predict(image[np.newaxis, :, :, :])
    conv_output = np.squeeze(conv_output)

    h = int(image.shape[0]/conv_output.shape[0])
    w = int(image.shape[1]/conv_output.shape[1])

    activation_maps = sp.ndimage.zoom(conv_output, (h, w, 1), order=1)
    out = np.dot(activation_maps.reshape((image.shape[0]*image.shape[1], 512)),
class_weights).reshape(
        image.shape[0],image.shape[1])

    fig, axs = plt.subplots(figsize=(6, 6))
    axs.imshow(image)
    axs.imshow(out, cmap='jet', alpha=0.35)
    axs.axis('off')
    plt.tight_layout()

    # Save the figure to the tmp folder
    file_name = os.path.splitext(os.path.basename(image_path))[0]
    fig.savefig(os.path.join('tmp', f'{file_name}-output.png'))
    plt.close(fig)

```

train.py

```
import os
import random

import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def set_seed(seed):
    tf.random.set_seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    random.seed(seed)

def get_model(SHAPE: tuple):
    set_seed(33)

    vgg = VGG16(weights='imagenet', include_top=False, input_shape = SHAPE)

    for layer in vgg.layers[:-8]:
        layer.trainable = False

    x = vgg.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(2, activation="softmax")(x)

    model = tf.keras.Model(vgg.input, x)
    model.compile(loss = "categorical_crossentropy",
                  optimizer = Adam(learning_rate=0.0001),
                  metrics=["accuracy"])

    return model

def train_model():
    # Define variables
    SHUFFLE = True
    SHAPE = (512, 512, 3)
    batch_size = 8
```

```

# Load and preprocess the datasets
root_dir = 'C:/Programming/FinalYearProject/dataset512x512'
train_datagen = ImageDataGenerator(
    validation_split=0.2, rescale=1./255
)
validation_datagen = ImageDataGenerator(
    validation_split=0.2, rescale=1./255
)
train_generator = train_datagen.flow_from_directory(
    root_dir,
    target_size = (SHAPE[0], SHAPE[1]),
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle = SHUFFLE,
    subset = 'training',
    seed = 33
)
validation_generator = validation_datagen.flow_from_directory(
    root_dir,
    target_size = (SHAPE[0], SHAPE[1]),
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle = SHUFFLE,
    subset = 'validation',
    seed = 33
)
model = get_model(SHAPE)
model.fit(train_generator,
          validation_data=validation_generator,
          epochs=50)
model.save('transfer-trained-vgg16.h5')

if __name__ == "__main__":
    train_model()

```

main.py

```
from app import app

if __name__ == '__main__':
    app.run(debug=True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-
alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
GLh1TQ8iRABdZL1603oVMWSktQOp6b7In1Z13/Jr59b6EGGoI1aFkw7cmDA6j6gD"
crossorigin="anonymous">
  <link rel="stylesheet" href="../static/css/style.css">
  <title>Defect Checker</title>
</head>
<body>
  <div id="content p-5" class="flexbox">
    <div class="card flexbox p-3 mt-5" style="min-width: 400px; max-width:
800px; max-height: 500px;">
      <div class="card-body flexbox">
        <div class="row">
          <div class="col p-4">
            <div id="image-box">
              <!-- Image added here with heatmap/boxes when
uploaded using js -->
            </div>
          </div>
        </div>
        <div class="row">
          <div class="col">
            <div id="loader" style="display:none;">
              
            </div>
          </div>
        </div>
        <div class="row">
          <div class="col">
            <h2 id="defective-status"></h2>
            <h5 id="confidence"></h5>
          </div>
          <div class="col">
            <input type="file" accept="image/png, image/jpeg">
          </div>
        </div>
      </div>
    </div>
  </div>
</body>
</html>
```

```

        </div>
    </div>
</div>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-w76AqPfdkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CXvN" crossorigin="anonymous"></script>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.3/jquery.min.js"></script>
<script>
    $(document).ready(function (e) {
        $('#input[type="file"]').change(function () {
            let file = this.files[0];
            let reader = new FileReader();
            reader.onloadend = function () {
                $('#image-box').html('');
            }
            if (file) {
                let formData = new FormData();
                formData.append('raw_image', file);
                reader.readAsDataURL(file);
                $.ajax({
                    url: "/process-image",
                    type: 'POST',
                    data: formData,
                    processData: false,
                    contentType: false,
                    beforeSend: function(){
                        $("#image-box").hide();
                        $("#loader").show();
                        $("#defective-status").html("");
                        $("#confidence").html("");
                    },
                    success: function(response){
                        $("#image-box").show();
                        $("#loader").hide();
                        if (response && response.is_defective !== undefined)
{
                            var confidence = response.confidence;
                            $("#confidence").html("Confidence: " +
confidence.toFixed(2) + "%");
                            if (response.is_defective) {
                                $("#defective-status").html("Defective");
                                let base64data = response.image;

```

```
        $("#img").attr("src",
"data:image/png;base64," + base64data);
    } else {
        $("#defective-status").html("Not Defective");
    }
    } else {
        alert("Invalid response from the server.");
    }
    },
    error: function(xhr, desc, err){
        $("#loader").hide();
        alert("Something went wrong! Please try again.");
    }
    });
} else {
    alert("Image upload failed!");
}
});
});
</script>
</body>
</html>
```

style.css

```
.flexbox {  
  display: flex;  
  align-items: center;  
  justify-content: center;  
}
```

Experiment Code

comparison.py

```
import os
import random

import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import VGG16, InceptionV3, ResNet50
from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Prevent GPU memory allocation issues
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)

# Define variables
SHUFFLE = True
SHAPE = (128, 128, 3)
batch_size = 8

# Load and preprocess the datasets
root_dir = 'C:/Programming/FinalYearProject/dataset512x512'
train_datagen = ImageDataGenerator(
    validation_split=0.2, rescale=1./255
)
test_datagen = ImageDataGenerator(
    validation_split=0.2, rescale=1./255
)
train_generator = train_datagen.flow_from_directory(
    root_dir,
    target_size = (SHAPE[0], SHAPE[1]),
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle = SHUFFLE,
    subset = 'training',
```

```

        seed = 33
    )
validation_generator = test_datagen.flow_from_directory(
    root_dir,
    target_size = (SHAPE[0], SHAPE[1]),
    batch_size = batch_size,
    class_mode = 'categorical',
    shuffle = SHUFFLE,
    subset = 'validation',
    seed = 33
)

def set_seed(seed):
    tf.random.set_seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    random.seed(seed)

def build_custom_model(SHAPE: tuple):
    model = Sequential([
        Conv2D(32, 3, padding='same', activation='relu', input_shape=SHAPE),
        MaxPooling2D(),
        Conv2D(64, 3, padding='same', activation='relu'),
        Conv2D(64, 3, padding='same', activation='relu'),
        MaxPooling2D(),
        Conv2D(128, 3, padding='same', activation='relu'),
        Conv2D(128, 3, padding='same', activation='relu'),
        MaxPooling2D(),
        Flatten(),
        Dense(1024, activation='relu'),
        Dense(1024, activation='relu'),
        Dense(2, activation='sigmoid')
    ])
    model.compile(optimizer=Adam(learning_rate=0.0001),
                  loss="categorical_crossentropy",
                  metrics=['accuracy'])

    return model

def build_model(name: str, SHAPE: tuple):
    set_seed(33)

    if name == "custom":

```

```

        return build_custom_model(SHAPE)

    if name == "vgg16":
        base_model = VGG16(weights='imagenet', include_top=False,
input_shape=SHAPE)
        base_model.summary()
    elif name == "inceptionv3":
        base_model = InceptionV3(weights='imagenet', include_top=False,
input_shape=SHAPE)
        base_model.summary()
    elif name == "resnet50":
        base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=SHAPE)
        base_model.summary()
    else:
        raise Exception("Invalid model name")

    for layer in base_model.layers[:-5]:
        layer.trainable = False

    x = base_model.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(2, activation="softmax")(x)

    model = tf.keras.Model(base_model.input, x)
    model.compile(optimizer=Adam(learning_rate=0.0001),
                  loss="categorical_crossentropy",
                  metrics=['accuracy'])
    return model

def train_model(name: str, model):

    history = model.fit(train_generator,
                        validation_data=validation_generator,
                        steps_per_epoch=train_generator.samples/train_generator.batch_size,
                        epochs=10)
    model.save(f'trained-{name}.h5')
    print(f"Saved model: trained-{name}.h5")

    return history

# Define a function to calculate TP, TN, FP, FN rates
def calculate_rates(y_true_labels, y_pred_labels):

```

```

# Calculate the confusion matrix metrics
tn = tf.keras.metrics.TrueNegatives()
tn.update_state(y_true_labels, y_pred_labels)
fp = tf.keras.metrics.FalsePositives()
fp.update_state(y_true_labels, y_pred_labels)
fn = tf.keras.metrics.FalseNegatives()
fn.update_state(y_true_labels, y_pred_labels)
tp = tf.keras.metrics.TruePositives()
tp.update_state(y_true_labels, y_pred_labels)
# Get the TP, TN, FP, FN rates
tp_rate = tp.result().numpy()
tn_rate = tn.result().numpy()
fp_rate = fp.result().numpy()
fn_rate = fn.result().numpy()

return tp_rate, tn_rate, fp_rate, fn_rate

def calculate_metrics(y_true, y_pred, name: str):
    tp, tn, fp, fn = calculate_rates(y_true, y_pred)

    # Calculate the accuracy, precision, recall, and F1-score
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    f1_score = 2 * precision * recall / (precision + recall)

    print("\n=====")
    print("{}".format(name))
    # Print the confusion matrix
    print("Confusion Matrix:")
    print("          Predicted Negative   Predicted Positive")
    print("Actual Negative      {}                {}".format(tn, fp))
    print("Actual Positive      {}                {}".format(fn, tp))
    # Print the performance metrics
    print("Accuracy: {:.3f}".format(accuracy))
    print("Precision: {:.3f}".format(precision))
    print("Recall: {:.3f}".format(recall))
    print("F1-score: {:.3f}".format(f1_score))
    print("=====\n")

def evaluate_model(model, y_true, name: str):
    y_pred = model.predict(validation_generator, verbose=0)
    y_pred = np.argmax(y_pred, axis=1)

```

```

calculate_metrics(y_true, y_pred, name)

def train_models():
    """ TRAIN MODELS """
    custom_model = build_model("custom", SHAPE)
    vgg16 = build_model("vgg16", SHAPE)
    inceptionv3 = build_model("inceptionv3", SHAPE)
    resnet50 = build_model("resnet50", SHAPE)
    trained_custom_model = train_model("custom", custom_model)
    trained_vgg16 = train_model("vgg16", vgg16)
    trained_inceptionv3 = train_model("inceptionv3", inceptionv3)
    trained_resnet50 = train_model("resnet50", resnet50)

    # Get the training accuracy of each model
    custom_model_training_accuracy = trained_custom_model.history['accuracy'][-1]
    vgg16_training_accuracy = trained_vgg16.history['accuracy'][-1]
    inceptionv3_training_accuracy = trained_inceptionv3.history['accuracy'][-1]
    resnet50_training_accuracy = trained_resnet50.history['accuracy'][-1]
    # Output the training accuracy of each model
    print("Custom Model Training Accuracy: ", custom_model_training_accuracy)
    print("VGG16 Training Accuracy: ", vgg16_training_accuracy)
    print("InceptionV3 Training Accuracy: ", inceptionv3_training_accuracy)
    print("ResNet50 Training Accuracy: ", resnet50_training_accuracy)

    return trained_custom_model, trained_vgg16, trained_inceptionv3,
    trained_resnet50

def load_trained_models():
    """ LOAD TRAINED MODELS """
    custom = tf.keras.models.load_model('trained-custom.h5')
    vgg16 = tf.keras.models.load_model('trained-vgg16.h5')
    inceptionv3 = tf.keras.models.load_model('trained-inceptionv3.h5')
    resnet50 = tf.keras.models.load_model('trained-resnet50.h5')

    return custom, vgg16, inceptionv3, resnet50

def evaluate_models(custom, vgg16, inceptionv3, resnet50):
    """ EVALUATE MODELS - DISABLE SHUFFLE ON DATASETS """
    y_true = validation_generator.classes
    evaluate_model(custom, y_true, "Custom Model")
    evaluate_model(vgg16, y_true, "VGG16 (Transfer Learning)")
    evaluate_model(inceptionv3, y_true, "InceptionV3 (Transfer Learning)")
    evaluate_model(resnet50, y_true, "ResNet50 (Transfer Learning)")

```

```
### UNCOMMENT FOR TRAINING + CHANGE SHUFFLE TO TRUE ###
train_models()

### UNCOMMENT FOR EVALUATION + CHANGE SHUFFLE TO FALSE ###
# custom, vgg16, inceptionv3, resnet50 = load_trained_models()
# evaluate_models(custom, vgg16, inceptionv3, resnet50)
```

hyperparameters.py

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers.experimental.preprocessing import Rescaling,
Resizing
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from bayes_opt import BayesianOptimization

root_dir = 'C:/Programming/FinalYearProject/dataset512x512'

# Define your train and validation datasets
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    root_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(512, 512),
    batch_size=32,
    labels="inferred",
    label_mode="binary",
    color_mode="rgb"
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    root_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(512, 512),
    batch_size=32,
    labels="inferred",
    label_mode="binary",
    color_mode="rgb"
)

def build_model(learning_rate, dropout_rate):
    model = Sequential([
        Rescaling(1./255, input_shape=(512, 512, 3)),
        Resizing(128, 128),
        Conv2D(32, 3, padding='same', activation='relu'),
        MaxPooling2D(),
        Conv2D(64, 3, padding='same', activation='relu'),
        Conv2D(64, 3, padding='same', activation='relu'),
        MaxPooling2D(),
```



```

    Conv2D(128, 3, padding='same', activation='relu'),
    Conv2D(128, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(1024, activation='relu'),
    Dense(1024, activation='relu'),
    Dense(2, activation='sigmoid')
])

# Compile the model with the specified learning rate and dropout rate
optimizer = Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer,
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])

# Set the dropout rate for applicable layers
for layer in model.layers:
    if isinstance(layer, tf.keras.layers.Dropout):
        layer.rate = dropout_rate

return model

# Define the function to optimize
def optimize_model(learning_rate, dropout_rate):
    # Build the model
    model = build_model(learning_rate, dropout_rate)

    # Train the model
    history = model.fit(train_ds, validation_data=val_ds, epochs=50, verbose=0)

    # Get the best validation accuracy
    best_val_accuracy = max(history.history['val_accuracy'])

    return best_val_accuracy

if __name__ == '__main__':
    # Define the hyperparameter search space
    hyperparameter_space = {
        'learning_rate': (0.0005, 0.001),
        'dropout_rate': (0.2, 0.5)
    }

    # Perform Bayesian optimization

```

```
optimizer = BayesianOptimization(f=optimize_model,
pbounds=hyperparameter_space, verbose=2)
optimizer.maximize(init_points=3, n_iter=10)

# Get the best hyperparameters and the corresponding validation accuracy
best_hyperparameters = optimizer.max['params']
best_val_accuracy = optimizer.max['target']

print("Best Hyperparameters:")
print(best_hyperparameters)
print("Best Validation Accuracy:")
print(best_val_accuracy)
```

graph.py

```
from matplotlib import pyplot as plt

from hyperparameters import build_model, train_ds, val_ds

EPOCHS = 50
LEARNING_RATE = 0.0006425108307104302
DROPOUT_RATE = 0.46386941186999164

model = build_model(LEARNING_RATE, DROPOUT_RATE)
history = model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS, verbose=1)
model.save('C:/Programming/FinalYearProject/tuned-trained-model.h5')

## Assigning the history of the model to variables
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(EPOCHS)

## Plotting the training and validation accuracy and loss
plt.figure(figsize=(18, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

dataset.py

```
import tensorflow as tf
from matplotlib import pyplot as plt

root_dir = 'C:/Programming/FinalYearProject/dataset512x512'

# Define your train and validation datasets
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    root_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(512, 512),
    batch_size=32,
    labels="inferred",
    label_mode="binary",
    color_mode="rgb"
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    root_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(512, 512),
    batch_size=32,
    labels="inferred",
    label_mode="binary",
    color_mode="rgb"
)

# Print the class names
class_names = train_ds.class_names
print(class_names)

# Visualize the dataset
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i].numpy()[0].astype(int)])
        plt.axis("off")
plt.show()
```