Aerospace Engineering

Final Year Project Cover Sheet

Surname:	Edmunds	**Forenames:**	Ethan Luke
Registration No:	200327732		
Module Code:	AER389		
Module Name:	Aerospace Individual Investigative Project		
Assignment Title:	Machine Learning methods for the classification of microstructural features in materials for aerospace engineering applications		
Supervisor:	Dr. Meurig Thomas		
Submission Date:	2nd May 2024		

Signature:	[illegible]
Date:	2nd May 2024

University of Sheffield

Aerospace Engineering

Aerospace Individual Investigative Project

Machine Learning methods for the classification of microstructural features in materials for aerospace engineering applications

Ethan L. Edmunds

May 2024

Supervisor: Dr. Meurig Thomas

Dissertation submitted to the University of Sheffield in partial fulfilment of the requirements for the degree of Bachelor of Engineering

Abstract

Microstructural defects are inherently important in aerospace materials, due to their effects on mechanical properties, and therefore performance of components. This relationship warrants significant investigation into microstructural defects. Microstructural defect detection and classification is often laborious, costly, and yields sub-optimal results. With a focus on Convolutional Neural Networks (CNNs), this study investigates the effectiveness of deep learning techniques in automatically characterising microstructural defects in additively manufactured nickel-based superalloys. Utilising a variety of machine learning (ML) models, including K-nearest neighbours (KNN) and decision trees, this research explores the relationship between model performance and different data normalisation techniques. A significant portion of this research focuses on the design and optimisation of CNNs for automated classification of microstructural defects. This work proposes practical approaches for improving defect detection and classification processes, aiming to support the advancement of material engineering applications in the industry.

Key Words: Additive Manufacturing, Convolutional Neural Networks, Data Normalisation, Deep Learning, Machine Learning, Microstructural Defects, Nickel-based Superalloys

Abbreviations

KNN	K-nearest neighbours
CNN	Convolutional Neural Network
AM	Additive Manufacturing
SLM	Selective Laser Melting
ML	Machine Learning

Acknowledgements and Dedication

I extend my sincere gratitude to my supervisor, Dr. Meurig Thomas, for his invaluable guidance and support throughout this project. His encouragement to explore new avenues and delve deeper into this field has been truly appreciated. I am also grateful for his assistance in navigating post-graduate program applications, which has been immensely helpful to me.

I want to express my gratitude to my personal tutor, Dr. Patrick Smith, for his continuous support throughout my degree, making it a truly enjoyable experience. I am thankful for his assistance in applying to various programs and internships.

I would like to express my gratitude to my colleagues and friends. In particular, I want to acknowledge Kimberley Mainprize, whose advice has been invaluable in helping me learn how to build neural networks in MATLAB.

Lastly, I want to express my deepest gratitude to my parents and siblings, who have been my unwavering support throughout my entire life. Despite the challenges, my parents have always encouraged me to persevere and never give up. I am eternally grateful to them for their love and encouragement. This dissertation is dedicated to my parents.

Contents

1 Introduction

The selection of metallic alloys is crucial for ensuring optimal performance of aircraft components, necessitating a thorough assessment of properties such as strength, density, melting point, and resistance to corrosion and creep. The chosen processing route also significantly influences these characteristics, affecting the material's performance. This is particularly critical for nickel-based superalloys; a material class that is commonly used in gas turbines for aerospace and power generation [1].

The extreme conditions of gas turbines result in stringent design requirements for components. To meet these requirements, investigation into the properties of materials must be conducted to optimise the materials for specific applications. This warrants the detection and classification of defects, which significantly impact the mechanical properties of materials [2]. Understanding defects is crucial for assessing a material's suitability for applications, optimising manufacturing processes, and implementing effective maintenance strategies [3].

Traditionally, defect detection has relied on a combination of destructive and non-destructive techniques. However, these methods can be laborious, costly, and often yield suboptimal results. Therefore, there has been significant research into automated methodologies for defect detection and classification using advanced imaging techniques like radiography, optical imaging, tomography, and electron microscopy, which have advanced abilities in material characterisation with their high spatial resolution and precision.

The use of these imaging techniques often creates large amounts of data, necessitating the use of new data analysis techniques and technologies. The integration of machine learning (ML) has revolutionised data analysis, enabling the rapid and comprehensive analysis of datasets. A study by Aziz et al. [4] utilised supervised ML to classify defects in additively manufactured nickel-based superalloys, achieving approximately 90% accuracy with K-nearest neighbours (KNN) classification algorithms and decision trees. This study highlighted the potential of ML to significantly enhance efficiency in defect classification for materials research.

In this paper, the work of Aziz et al. [4] will be replicated to further explore the effectiveness of ML in classifying microstructural defects in nickel-based superalloys. This replication involves using KNN and decision tree algorithms on a dataset generated from metallographic examinations of alloys like CM247LC, LR8, and Alloy 713C. These alloys were processed by selective laser melting (SLM). The analysis will be complemented by an evaluation of the parameters influencing the accuracy of these machine learning methods and a comparative study of the algorithms used. Furthermore, this paper proposes the use of convolutional neural networks (CNN) for defect classification.

2 Literature Review

2.1 Additive Manufacturing

Additive manufacturing (AM) is a class of manufacturing processes that enables the creation of lighter, stronger, and more complex parts using a more efficient process. Traditional methods of manufacturing typically take a long time, result in material wastage due to their subtractive nature and are craftsmanship intensive [5]. AM instead fabricates components layer-by-layer from three-dimensional digital models, resulting in numerous advantages and applications [6], [7].

Additive manufacturing (AM) offers unique capabilities for producing designs that are challenging to achieve through traditional methods like milling or machining [8]. These capabilities include fabricating complex shapes derived from topological optimization. AM's versatility extends to handling a diverse range of materials, from metals and polymers to ceramics and biomaterials, allowing for the simultaneous use of multiple materials on a single platform [9]. This facilitates customization and optimization of component material compositions, enhancing functionality while reducing wastage. Compared to traditional manufacturing, AM simplifies the process, eliminating the need for precise planning of machining steps and tool selection. For example, in computer numerical control (CNC) machining, proper tools must be selected for specific materials, and the tool path must be carefully designed to prevent tool crash [10]. In contrast, AM methodologies offer a tool-free process and therefore can reduce both wear and machine setup times [11].

The advantages of AM have led to its increased adoption in rapid prototyping and low-scale mass production, driven by cost reductions and faster production cycles [7], [12]. In industries such as aerospace, where there is demand for complex geometries in components, AM proves useful. This is crucial for components like heat exchangers [13] or turbine blades which feature intricate internal cooling channels to maintain operational efficiencies [14], [15], [16].

Despite these benefits, AM faces significant challenges that hinder its wider adoption. A primary concern highlighted by researchers like Khanzadeh et al. [17] is the lack of understanding surrounding the 'process-structure-property' relationship in AM methods [17]. This gap in knowledge complicates the optimisation of AM processes and can lead to issues in component durability, as evidenced by failures in parts produced via techniques like Selective Laser Melting (SLM), despite superior mechanical properties [9]. Additionally, metallic components fabricated with AM are prone to microstructural defects, which pose risks in structural applications, particularly under cyclic loading conditions [18]. These challenges necessitate ongoing research efforts to enhance the reliability and application scope of AM technologies.

2.1.1 Selective Laser Melting

Selective Laser Melting (SLM) is one type of AM process. SLM operates by fusing fine powder layers using a high-power laser that follows paths defined by a computer-aided design (CAD) model to build three-dimensional structures layer by layer [19], [20]. The process starts with spreading a layer of metal powder onto a build plate. A laser then scans the powder surface, melting and solidifying the powder according to slices of the 3D model. After each layer is completed, the build plate lowers, and a new layer of powder is applied on top. This cycle repeats until the part is fully built. Each layer typically ranges from 20 to 100 microns thick, allowing for fine detail and complex geometries to be built [21]. A typical rig used for SLM is shown in Figure 1. The process in which SLM operates is very similar to other powder-bed type of AM processes, such as selective laser sintering (SLS) and electron beam melting (EBM) despite differences in details regarding each process. For example, parts produced by SLM are more susceptible to residual stresses and use much finer particle sizes than those produced by SLS and EBM [22].

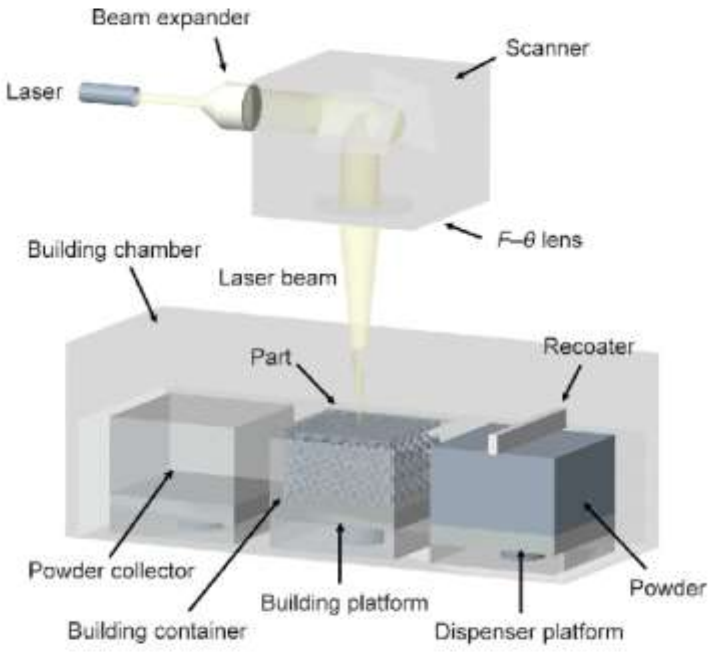


Figure 1: Schematic of SLM process [23]

SLM is effective with a variety of metals and alloys, including titanium, stainless steel, aluminium, cobalt chrome, and nickel-based superalloys [22]. The choice of material depends on the specific requirements of the application and compatibility with the manufacturing process.

SLM offers significant benefits over traditional manufacturing methods. It achieves high material utilisation, reducing waste, especially with expensive alloys. SLM also enables the creation of complex structures and internal features that traditional methods cannot achieve, allowing for innovative designs. Additionally, it allows for the optimisation of mechanical properties by strategically placing materials where needed, enhancing the performance of parts in critical applications [24]. Applications of additively manufactured components will be further discussed in later chapters.

Despite advantages, SLM faces notable challenges, preventing its wider adoption. The process can induce residual stresses and potential distortions due to high heat and rapid cooling, which may require sophisticated post-processing to rectify [25]. Parts produced often have a rougher surface finish, necessitating additional post-processing such as milling or polishing [26]. Moreover, the high costs associated with SLM technology and materials can be prohibitive, particularly for smaller enterprises.

2.2 Nickel-Based Superalloys

Superalloys are materials developed for high-temperature applications, typically featuring a nickel or cobalt base. These alloys are essential in environments where they face severe mechanical stress and demand high surface stability. The defining characteristics of superalloys include their ability to maintain mechanical integrity at temperatures near their melting points, resist mechanical degradation despite prolonged exposure to high temperatures, and tolerate harsh operating conditions [27]. This combination of properties makes superalloys particularly valuable in sectors such as aerospace and power generation, where reliability and performance at high temperatures are crucial. Nickel-based superalloys also typically exhibit exceptional resistance to oxidation and corrosion, making them suitable for a variety of applications [28].

Nickel-based superalloys are distinguished by their unique microstructural features, primarily the gamma (γ) and gamma prime (γ') phases, illustrated in Figure 2, which play a crucial role in defining their high-temperature capabilities.

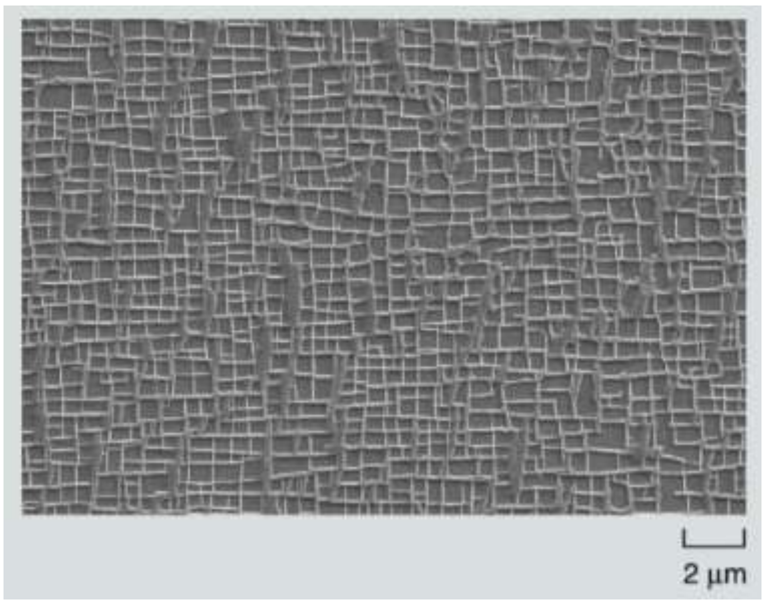


Figure 2: The microstructure of the single-crystalline nickel-based superalloy CMSX-44 after solution [29]

The γ phase, with its face-centred cubic (FCC) structure, forms the matrix of these superalloys [30]. This phase is inherently stable up to the melting point of the alloy, making it an excellent structural foundation due to its resistance to phase transformations at high temperatures. The phase is strengthened through solid solution strengthening, achieved by alloying nickel with elements such as Molybdenum, Tungsten, Cobalt, Chromium, and sometimes Rhenium and Ruthenium [1]. These elements hinder the movement of dislocations and slow down diffusion across the matrix, which significantly enhances the superalloy's mechanical properties at elevated temperatures.

Nickel-based superalloys derive their strength from the presence of the γ' phase, usually Ni_3Al, which precipitates within the γ matrix [31, p. 8]. This phase forms through an ordering reaction, maintaining a similar lattice structure but with an ordered distribution of nickel and aluminium atoms, effectively impeding dislocation movement and reinforcing the alloy's strength [1]. The γ' phase's volume fraction and distribution are crucial for optimizing high-temperature strength and creep resistance, typically targeted at around 70% [32]. This dual-phase microstructure, where the γ matrix provides ductility and toughness while the γ' phase contributes to strength and creep resistance, enables these alloys to excel in demanding conditions like turbine engines. Achieving this balance requires precision in manufacturing and processing.

Investment casting stands as a critical technique for producing nickel-alloy turbine blades, renowned for its capacity to handle the complex geometries required for high-volume production. This method integrates ceramic cores to precisely control internal configurations, enabling the introduction of advanced cooling schemes. Such schemes are pivotal, allowing engine components to operate beyond the melting point of the alloy, thus significantly boosting engine efficiency [33]. Additionally, investment casting facilitates the management of microstructural characteristics through directional solidification. This sophisticated process allows for the creation of single-crystal blades, which are celebrated for their superior high-temperature strength and enhanced oxidation resistance [34].

Despite its advantages, investment casting of nickel-alloys can be cost-prohibitive and time-intensive, especially when developing hard tooling for wax pattern moulding, making it less economical for low-volume production [35]. The process involves several meticulous steps: starting with the injection of wax or plastic into a metal die to form intricate patterns. These patterns are then assembled into a cluster and encased in a ceramic slurry to create a robust shell. Upon heating, the wax is melted away, leaving behind a hollow ceramic mould. Molten nickel alloy is poured into this mould, and upon solidification, the ceramic outer shell is removed, revealing the metal parts attached to a runner. The final phase includes detaching these parts from the runner and carrying out any necessary finishing touches. The process of investment casting for Nickel-alloys is illustrated in Figure 3.

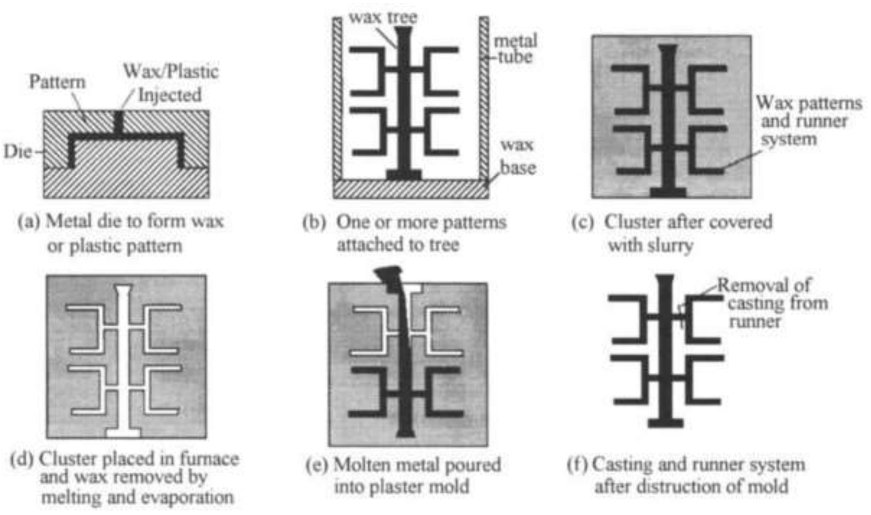

Pattern
Wax/Plastic Injected
Die
(a) Metal die to form wax or plastic pattern
wax tree
metal tube
wax base
(b) One or more patterns attached to tree
Wax patterns and runner system
(c) Cluster after covered with slurry
(d) Cluster placed in furnace and wax removed by melting and evaporation
(e) Molten metal poured into plaster mold
Removal of casting from runner
(f) Casting and runner system after distruction of mold

Figure 3: Overview of investment casting process [36, p. 6]

Powder metallurgy (P/M) methods are essential for making alloys which are crucial for aerospace parts. Compared to alloys made conventionally, these P/M alloys have more γ' forming elements [37]. P/M offers advantages like less elemental separation, finer grains, and fewer defects [32]. The process involves steps like melting metal into an ingot using vacuum induction (VIM), then turning it into fine powder through methods like gas atomization. This powder is sieved, packed, degassed, and consolidated using hot isostatic pressing (HIP). After that, thermomechanical processing (TMP) shapes the material, followed by heat treatment for optimal structure and distribution. An illustration of this process is shown in Figure 4.

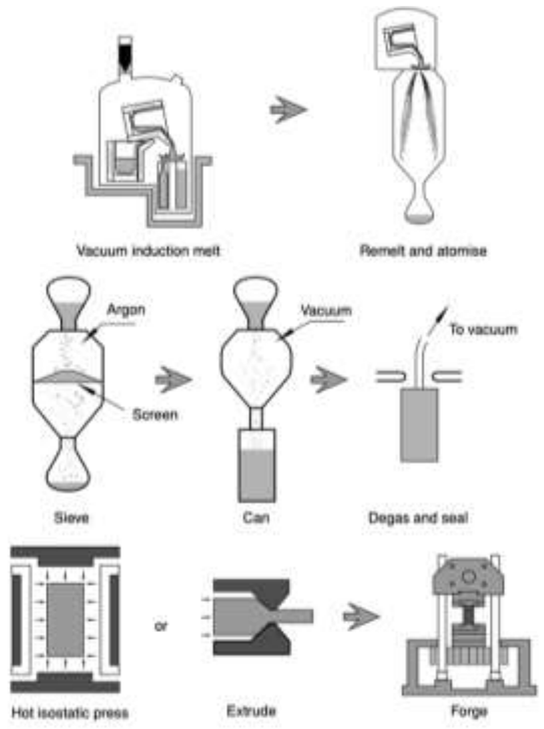


Figure 4: Powder Metallurgy Processing [1]

2.2.1 Examples of Nickel-based Superalloys

2.2.1.1 CM247LC

CM247LC is a cast nickel-base superalloy known for its low carbon content which enhances mechanical properties at elevated temperatures [38]. It is a derivative of MARM247, optimised for the demands of manufacturing directionally solidified turbine blades, improving creep and oxidation resistance [39]. Compared to other superalloys, it provides superior thermal fatigue resistance. Alloy technology advances have refined its microstructures, enhancing castability and reducing defects like segregation, thus improving manufacturing yield and performance. As research continues, CM247LC is increasingly favoured for its balanced properties of strength, durability, and cost-effectiveness, marking it as a key material in the future of high-temperature applications. The superalloy exhibits a high sensitivity to cracking, which restricts its broader use in the field of AM [40]. This limitation highlights the need for further research into defect formation to improve its performance.

2.2.1.2 Inconel 713C

Inconel 713C (IN713C) is a γ' precipitation-hardened cast nickel-based superalloy that is regularly used for high-temperature applications [41]. It is the most commonly used alloy for turbocharger turbines due to its high performance and cost-effectiveness [42], [43]. Renowned for its excellent mechanical properties, IN713C can withstand temperatures up to 800°C and is valued for its resistance to oxidation and thermal fatigue, alongside its good castability [43]. IN713C is an ideal choice for harsh operating environments, where its durability and resistance to oxidation and corrosion at elevated temperatures are critical [44].

2.2.1.3 LR8

LR8, a newly designed material for AM, is engineered to optimise performance and address typical challenges encountered in conventional nickel superalloys like IN713C [45]. Unfortunately, the literature describing its properties and characteristics is sparse. This is expected as it is a relatively new alloy.

2.2.2 Applications

Nickel-based superalloys have found extensive use in many industries because of their excellent thermomechanical properties and resistance to creep and corrosion. These properties have justified their use in components for gas turbines across the aerospace and power generation industries [46], [47], [48], [49]. Superalloys are also used in boilers for coal-based power plants, where higher steam temperatures and pressures (which in-turn create more efficient power plants) necessitate the use of superalloys for their superior performance characteristics [50], [51]. These alloys have also found some use in the nuclear industry. For example, they have been utilised in the steam generators of Pressurized Water

Reactors (PWRs) and within various subcomponents of reactor systems [52]. Many industries are heavily focused on enhancing the efficiency of gas turbines, by increasing the turbine entry temperature [53]. A primary challenge arises from the freestream temperature in gas turbines, which often surpasses the melting point of materials within the engine. One approach of addressing this is redesigning gas turbine components to incorporate intricate geometries and advanced cooling technologies, which help reduce blade temperatures, as depicted in Figure 5. Another strategy involves improving the materials used, particularly by increasing the melting point of the blade. However, despite their ability to withstand extreme temperatures and mechanical stresses, the performance of critical hot gas path components like vanes, blades, and combustors often restricts the overall durability of modern gas turbines [54]. Due to its numerous applications and that its characteristics often limit performance of gas turbine systems, improvement of these materials is vital, which can be achieved through further research.

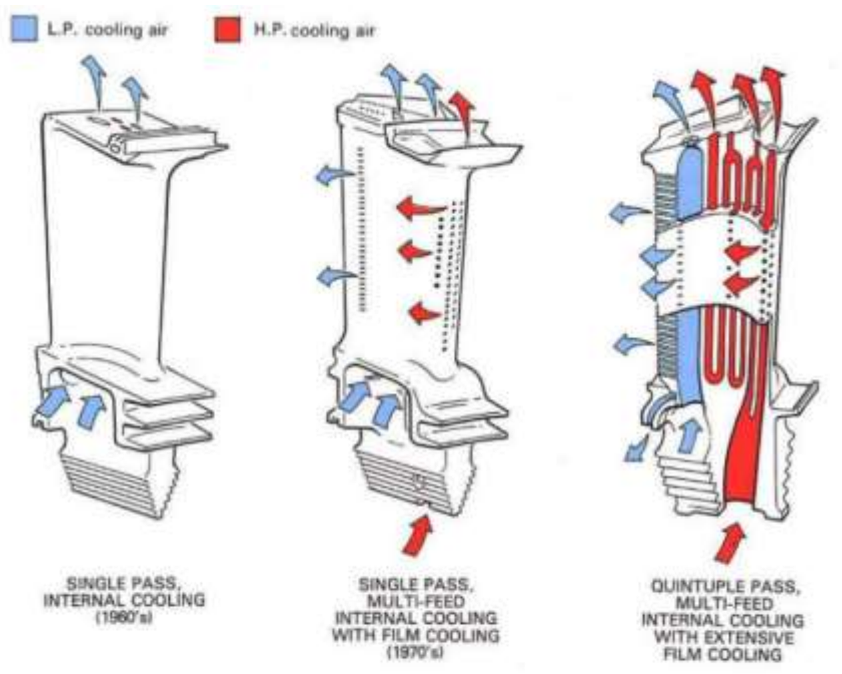

🟦 L.P. cooling air
🟥 H.P. cooling air
SINGLE PASS, INTERNAL COOLING (1960's)
SINGLE PASS, MULTI-FEED INTERNAL COOLING WITH FILM COOLING (1970's)
QUINTUPLE PASS, MULTI-FEED INTERNAL COOLING WITH EXTENSIVE FILM COOLING

Figure 5: Turbine blade with various cooling functionalities [55]

2.3 Microstructural Defects of Additively Manufactured Nickel-based superalloys

In AM of nickel-based superalloys, common defects such as cracks, pores, and lack of fusion have a profound impact on the mechanical properties of the final components.

Cracks are often related to thermal stresses induced by the rapid cooling and subsequent reheating cycles characteristic of additive manufacturing. These stresses can lead to the formation of cracks if the thermal contraction of the solidifying material generates tensions exceeding its fracture toughness. Such phenomena are particularly problematic in alloys designed for high-temperature applications, as the presence of cracks can severely degrade their mechanical performance under operational stresses. This aspect is detailed in the work focusing on the impact of defects on mechanical properties [56].

Pores in these superalloys are typically caused by entrapped gas or inadequate powder melting. The presence and distribution of pores, influenced by laser power, scanning speed, and other process parameters, are critical as they act as stress concentrators that can significantly undermine fatigue resistance. The study on the effects of defects explored through X-ray tomography provides extensive insights into how these pores form and their impact on the alloy's durability [2].

Lack of fusion defects are particularly critical as they occur when the laser fails to completely melt the powder, leading to weak spots in the material. These defects are typically found at the boundaries between scan tracks or layers, where they can initiate crack propagation under mechanical load. The research on numerical prediction of lack-of-fusion defects offers a detailed examination of these occurrences and their implications for structural integrity [57].

Optimising the additive manufacturing process parameters is crucial for reducing the incidence of these defects. In-situ monitoring techniques, as discussed in the study on melt pool monitoring for porosity prediction, can provide real-time adjustments to process parameters, helping to mitigate the formation of defects during manufacturing [17]. Furthermore, post-processing treatments such as hot isostatic pressing are also employed to reduce porosity and enhance the mechanical properties of the final product, as highlighted in the discussion on microstructural characteristics and mechanical properties of nickel-based superalloys [58].

Through a comprehensive understanding and management of these defects, the reliability and performance of nickel-based superalloys produced via AM can be significantly enhanced, particularly in critical applications such as aerospace and power generation.

2.4 Machine Learning

In recent decades, with the increasing digitisation of technology and various aspects of everyday life, we have gained unprecedented access to large datasets. A critical challenge is making this vast amount of data useful by extracting actionable insights. One pivotal tool that has emerged and evolved significantly is machine learning (ML), a subset of artificial intelligence.

ML involves programming computer algorithms to optimise performance criteria based on example data or past experiences [59], [60]. This technology has been instrumental in numerous modern advancements finding use in almost all aspects of computer science and software engineering, with significant applications in engineering and science [61]. Notable applications include the development of advanced natural language processing models like

ChatGPT, autonomous navigation systems for drones and self-driving cars, and effective data analysis tools available to the public.

Despite these successes, ML is not without its limitations. ML algorithms often struggle with flexibility, typically restricted to their specific parameters and lacking the ability to make common-sense inferences or logical deductions beyond their training data [60]. Addressing these shortcomings is a key focus of current research, aiming to enhance the adaptability and intelligence of ML systems.

A diverse array of ML algorithms exists, each designed to analyse data using unique strategies. These differences in the underlying architecture can significantly impact an algorithm's accuracy and utility. Considering this, experimental approaches will usually involve testing a variety of ML algorithms to determine their efficacy and applicability for a specific scenario. This paper will focus on two categories of machine learning models distinguished by their approach and characteristics: Supervised Machine Learning and Deep Learning.

2.4.1 Supervised Machine Learning

Supervised machine learning involves training a model with labelled data, where inputs (features) are paired with corresponding outputs (labels), allowing the model to learn a mapping function for predicting future outputs. A supervisor guides the training by providing correct answers during this process [62]. The goal is to develop a function, $f(x)$, that reliably predicts the true label, y, for new data based on known features, x [62]. This function ideally mirrors the underlying joint probability distribution, $P(x, y)$, ensuring consistent and accurate predictions across datasets.

Supervised learning includes classification, where data is categorized into discrete classes, and regression, which predicts continuous outcomes. Different machine learning models, or the parameters selected for their design, can exert a profound influence on the outcomes they yield. This emphasises the critical importance of thorough testing and careful design of machine learning models to achieve optimal performance and the intended results.

2.4.1.1 K-Nearest Neighbours

K-Nearest Neighbour (KNN) is a type of supervised ML algorithm that classifies data points based on the properties of their nearest neighbours in the feature space. This method is straightforward yet powerful, involving few assumptions about the structure of the data, making it versatile for a variety of classification tasks. KNN operates by calculating the distance between a query data point and all other points in the dataset, typically using a distance metric such as the Euclidean distance, defined by Equation 1.

$$dist(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

Equation 1: Euclidean distance formula used in KNN classification to calculate the distance between data points p and q in n-dimensional space [59]

In the KNN algorithm, the unknown point is then assigned to the class most found among its k nearest neighbours, where k is a predefined number chosen by the user. The neighbours are selected from a set of points that have already been classified, effectively using them as a reference group. The process of classifying a new data point using KNN is graphically represented in Figure 6, which illustrates how KNN evaluates the distances to determine the 'neighbourhood' of a point.

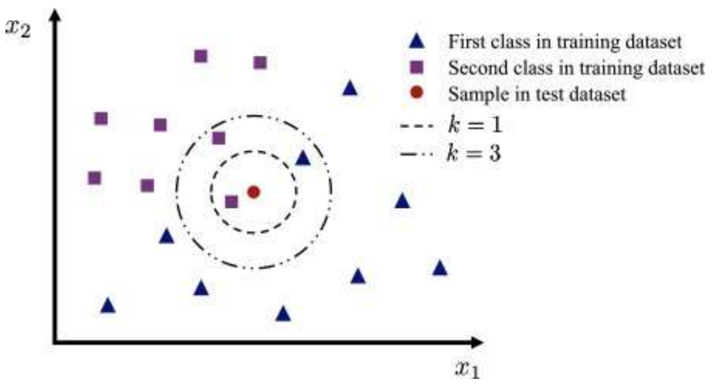

x_2
x_1
First class in training dataset
Second class in training dataset
Sample in test dataset
$k = 1$
$k = 3$

Figure 6: Principle of KNN classification algorithm [63]

KNN's performance relies heavily on parameters like k, the distance metric, and weights applied to features. Smaller k values make it sensitive to noise, while larger ones may blur class boundaries. Distance metrics like Euclidean, Manhattan, or Minkowski can be chosen based on dataset characteristics [63]. Weighting neighbour votes by distance can mitigate the influence of outliers [64].

Despite its simplicity, KNN is highly effective, known for its computational efficiency and ability to handle large, complex datasets with significant class homogeneity [59]. Careful parameter tuning and dataset understanding are essential for KNN's optimal performance. Selecting a suitable distance measure and setting k appropriately are vital for accurate grouping and reliable classification.

2.4.1.2 Decision Trees

Decision trees use a nonparametric, hierarchical structure with a divide-and-conquer strategy, making them highly effective for classification and regression tasks in machine learning [60]. Each internal node of a decision tree tests an attribute, each branch represents the outcome of the test, and each leaf node denotes a class label [59]. This setup is illustrated in Figure 6, where internal nodes are circles, branches are lines, and leaf nodes are squares. Its architecture allows decision trees to simplify complex decision-making processes by classifying data points based on their attributes and relationships to others.

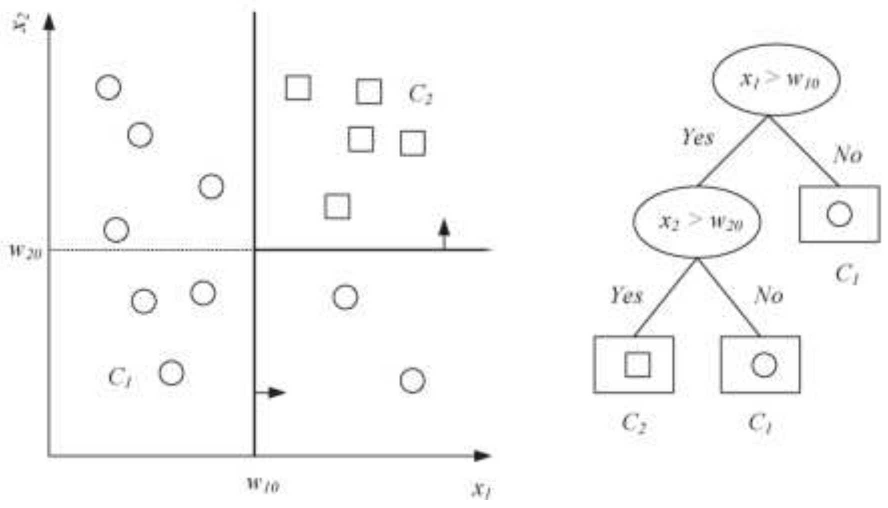


Figure 7: Example of dataset and the corresponding decision tree structure used for classification [60]

Building decision trees involves using a greedy algorithm to optimise criteria such as Gini impurity, entropy, or variance reduction, maximising information gain or minimise variance within subgroups [60], [65]. Despite their interpretability and capacity to handle non-linear data without feature scaling, decision trees can be prone to overfitting and instability. Overfitting, in decision trees, occurs when trees become too complex, learning noise as significant signals, which can negatively impact new data performance [66]. Instability means small data variations might change the structure of the tree. These issues can be mitigated by using ensemble methods like Random Forest, offering the possibility of inspecting the decision rules, investigating the relevance of each variable, and therefore stabilising predictions by averaging multiple trees [67], [68].

Despite potential drawbacks, decision trees are widely utilised across diverse fields due to their straightforward logic and clear decision-making processes [65]. They are especially valuable in scenarios where it's crucial to understand the rationale behind predictions.

2.4.2 Deep Learning

Traditional supervised ML models can struggle with complex, non-linear problems in high-dimensional datasets, necessitating the development of advanced ML algorithms like deep learning. Deep learning, using artificial neural networks to learn intricate data representations, addresses this limitation. They achieve this by discerning inherent patterns in observations, directly understanding the hierarchy of concepts from raw data [69], [70]. These algorithms have been pivotal in advancing the state-of-the-art across various domains, including speech and visual object recognition and object detection [70], [71]. Because of deep learning, as well as boosts in computational power, ML's ability to process high-dimensional data, such as transient data, images, and videos has evolved immensely [72]. The creation of a robust ecosystem of software and datasets, such as TensorFlow [73], [74] and ImageNet [75], have also supported these advancements. Deep learning models typically feature complex algorithmic architectures with a variety of components as summarised in Table 1. This structure is also illustrated in Figure 5.

Layer Type	Description
Input Layers	A layer that receives raw data inputs, initiating the data processing sequence.
Hidden Layers	Layers that perform computations on data received from the previous layer, by using weighted sums and biases, followed by non-linear activation functions, to learn complex patterns in the data [69].
Output Layers	A layer that produces the prediction or classification of results. This result is based on the patterns identified from the hidden layers and the raw data from the input layer.

Table 1: Summary of types of layers found in neural networks

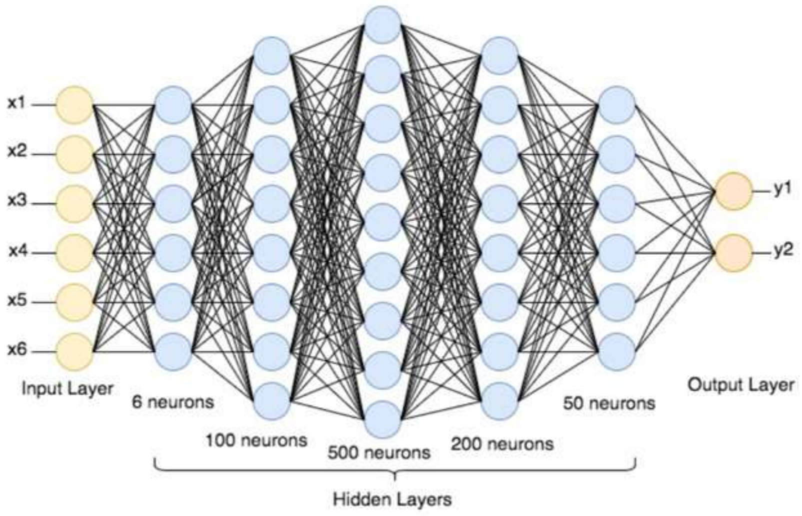


Figure 8: Example of a deep neural network architecture [76]

Within each of these layers exists a number of nodes. Each of these nodes, acts as a basic computational unit that processes data. They typically process output data of nodes in the preceding layer. The node's input data is weighted and aggregated, typically summed together with a bias term to form a linear combination. This result is then passed through a nonlinear activation function, which determines the node's output. The activation function is crucial as it introduces non-linearity to the model, allowing the network to recognise pattens in complex, non-linear datasets of high dimensionality. The perceptron is an example of a type of node and can be mathematically represented as seen in equation 2, with variables being identified and described in Table 2.

$$y(v;\ \theta) = f\left(\sum_{i=1}^{D} v_i w_i + w_0\right) = f(w^T v + w_0)$$

Equation 2: Perceptron [69]

Variable	Description
$\boldsymbol{y}$	Perceptron output
$\boldsymbol{v}$	Input vector, consisting of features and observations
$\boldsymbol{\theta}$	Parameter set, which includes all weights (w and w_0)
$\boldsymbol{w}$	Weight vector associated with the input vector, where each weight corresponds to the importance of each input feature
$\boldsymbol{w_0}$	Bias term, which adjusts the threshold level at which the activation function is triggers
$\boldsymbol{D}$	Dimensionality of the input vector

Table 2: Summary of variables found in mathematical equation of a perceptron.

The perceptron is the simplest form of a hidden layer. With the development of deep learning, the array of hidden layer types has grown vastly. Each of these types of hidden layers are designed to perform specialised functions based on their unique architecture. This specialisation of hidden layers, in conjunction with their assortment in a neural network architecture, allows deep learning algorithms to be optimised for specific tasks and use cases. Notable examples include Convolutional Neural Networks, which excel in processing visual data, and Generative Adversarial Networks, which are adept at generating new data that mimics authentic inputs.

2.4.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have revolutionised the field of deep learning, particularly for tasks involving image and sequential data analysis. CNNs effectively mimic the way humans process visual information [77], [78]. Their architecture is uniquely suited to handle images due to its translation invariance property, where the network recognises objects regardless of their shift in location within the image [72].

A typical CNN consists of multiple layers that each play a crucial role in learning from the data. The network usually begins with convolutional layers that apply various filters to the input to create feature maps. These feature maps highlight important visual features such as edges, textures, and patterns, mimicking early visual processing similar to the human eye [79]. As the information progresses through the network, subsequent layers compile these basic features to recognise more complex elements of the images, like parts of objects and eventually whole objects. Subsampling layers, often referred to as pooling layers, follow the convolutional layers and are used to reduce the spatial size of the representation, which decreases the number of parameters and computation in the network. This contributes to the robustness of the model by providing an abstracted form of the features being learned. This process is partly illustrated in Figure 8. Following the pattern recognition stages, CNNs may include fully-connected layers, similar to traditional neural networks, which take the high-level filtered information and translate it into final output, such as classification labels. CNNs are advantageous over fully connected networks as they require fewer parameters and are less prone to overfitting, making them more efficient for learning from large datasets [80].

Table 3 briefly summarises the function of each type of layer in a CNN:

Layer Type	Function
Convolutional	Applies learnable filters to input to create feature maps that highlight specific features in the data.
ReLU (Activation)	Introduces non-linearity into the model, allow it to learn complex patterns. Often follows convolutional layers.
Pooling	Reduces the dimensionality of each feature map but retains the most important information.
Fully-Connected	Takes the high-level filtered images and translates them into final output like classification labels.

Table 3: Summary of types of layers in typically CNN [81]

This structured approach allows CNNs to learn complex, high-dimensional, non-linear mappings from extensive collections of examples, making them highly effective for sophisticated tasks such as image recognition and speech recognition [65].

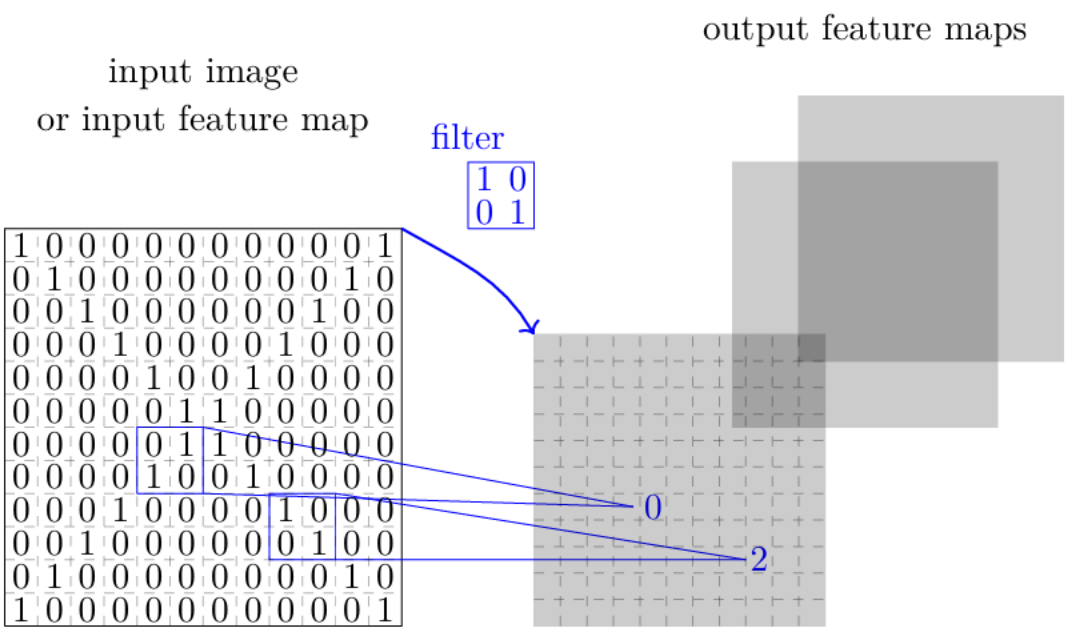


Figure 9: Illustration of convolutional layer with one input feature map that is convoluted by different filters to yield the output feature maps [72]

3 Methodology

3.1 Supervised Learning

A labelled dataset of commonly observed defect types in additively manufactured nickel-based alloys was created through metallographic examination of three alloys – CM247LC, LR8, and Inconel 713C – processed by SLM. These samples were produced as a part of various research projects at the University of Sheffield [45], [82]. The metallographs were analysed using ImageJ, a public domain image processing program. This software facilitated image thresholding and particle analysis, yielding a numerical dataset with quantitative information describing each defect. Each defect was then manually classified. An example of this classification is presented in Table 4. The dataset was subsequently imported into MATLAB for the development of various supervised ML models. The dataset originally consisted of 590 observations and 11 features. One of the features from the original dataset, 'ID', was removed as it did not provide any useful information for analysis. The dataset consists of 61 'Pore with Crack' instances, 132 'Pore' instances, 69 'Lack of Fusion' instances, and 328 'Crack' instances.

Defect Type	Area (μm^2)	Major Axis (μm)	Minor Axis (μm)	Angle	Circularity	Feret Diameter (μm)	Aspect Ratio	Roundness	Solidity
Crack	432.05	69.56	7.91	81.80	0.14	85.08	8.80	0.11	0.51
Pore	1787.06	51.64	44.06	34.75	0.78	55.12	1.17	0.85	0.94
Pwc	282.54	22.44	16.03	15.54	0.44	28.89	1.40	0.72	0.69
LoF	520.68	37.28	17.78	6.63	0.70	38.68	2.10	0.48	0.90

Table 4: Structure of the labelled dataset of defect types observed in additively manufactured Nickel alloys. With PwC referring to 'Pore with Crack' defects and LoF referring to 'Lack of Fusion' defects

3.2 Deep Learning

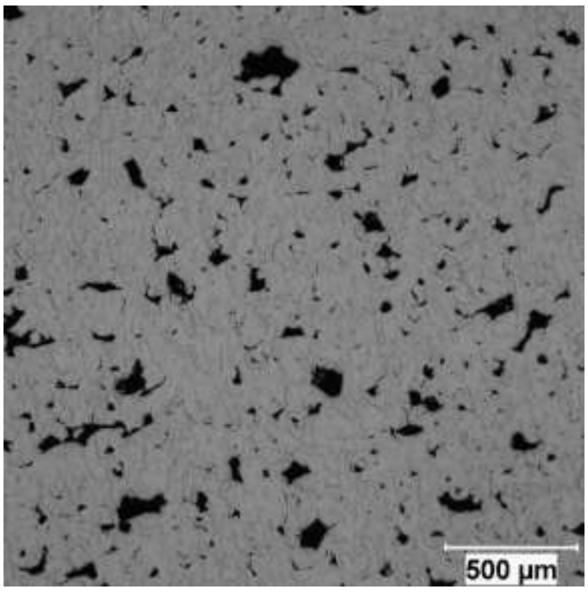


Figure 10: Micrograph of IN713C

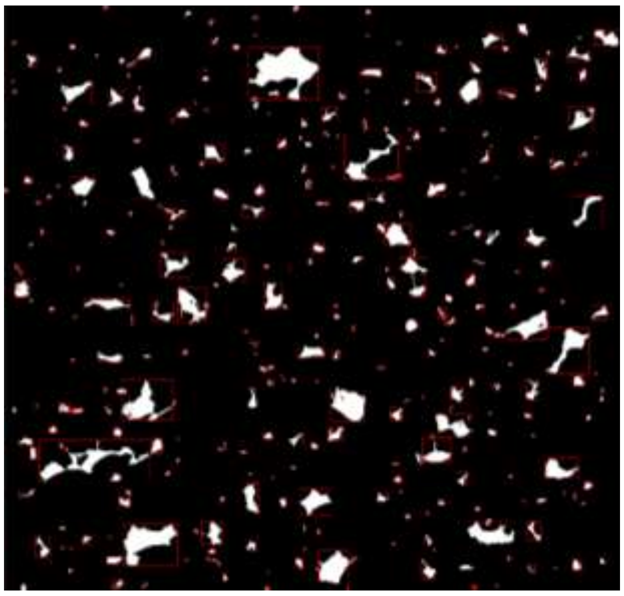


Figure 11: Binary Image of Micrograph after filtering

A collection of micrographs of IN713C, manufactured using SLM, were gathered and analysed. An example of these micrographs is shown in Figure 10. The micrographs were originally produced by Cong Lui, a doctoral student in the Department of Materials Science and Engineering at the University of Sheffield. Each micrograph was converted into a binary image using MATLAB. Each of these binary images was then enhanced to improve the quality of the dataset. During this enhancement, very small defects were removed to reduce noise in the ML algorithm, as these often represented minimal or misleading information. Defects located at the edges were also excluded, as they could misrepresent the actual shape of the defects. The enhanced binary images, an example of which is shown in Figure 11, were then segmented to isolate each individual defect. These segmented images, depicted in Figure 12, were manually classified to form the training and validation datasets for the development of the deep learning models. All code developed for this project are attached in the appendix below (Section 12.4).

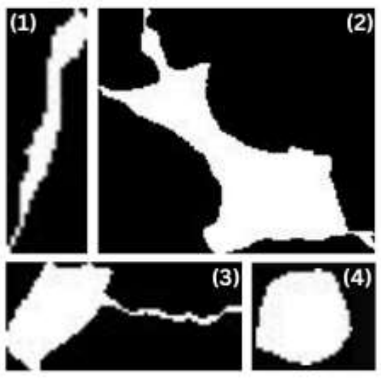


Figure 12: Example binary images of (1) cracks, (2) lack of fusion defect, (3) pore with crack and (4) pores

4 K-Nearest Neighbours

4.1 Data Processing and Analysis

The numerical dataset presented in Table 4 underwent analysis to discern patterns or correlations among various defect classes. To facilitate this, a minimum-maximum normalisation method was employed, and the resulting dataset was thoroughly examined. Figure 13 displays a 3D scatter plot delineating the distribution of Feret diameter, roundness, and circularity, with definitions of these parameters provided in Table 3.

Parameter	**Description**
Feret Diameter	The Feret diameter is the distance between two parallel lines tangent to an object's boundary and perpendicular to a specified direction. It measures the object’s width at a particular angle
Roundness	Roundness quantifies how closely the shape of an object resembles a circle, calculated as the ratio of the radius of the smallest enclosing circle to the radius of the maximum inscribed circle.
Circularity	Circularity is a dimensionless measure that evaluates how circular a shape is, where 1 indicates a perfect circle and lower values denote more irregular shapes.

Table 5: Definitions of Feret diameter, roundness, and circularity [83]

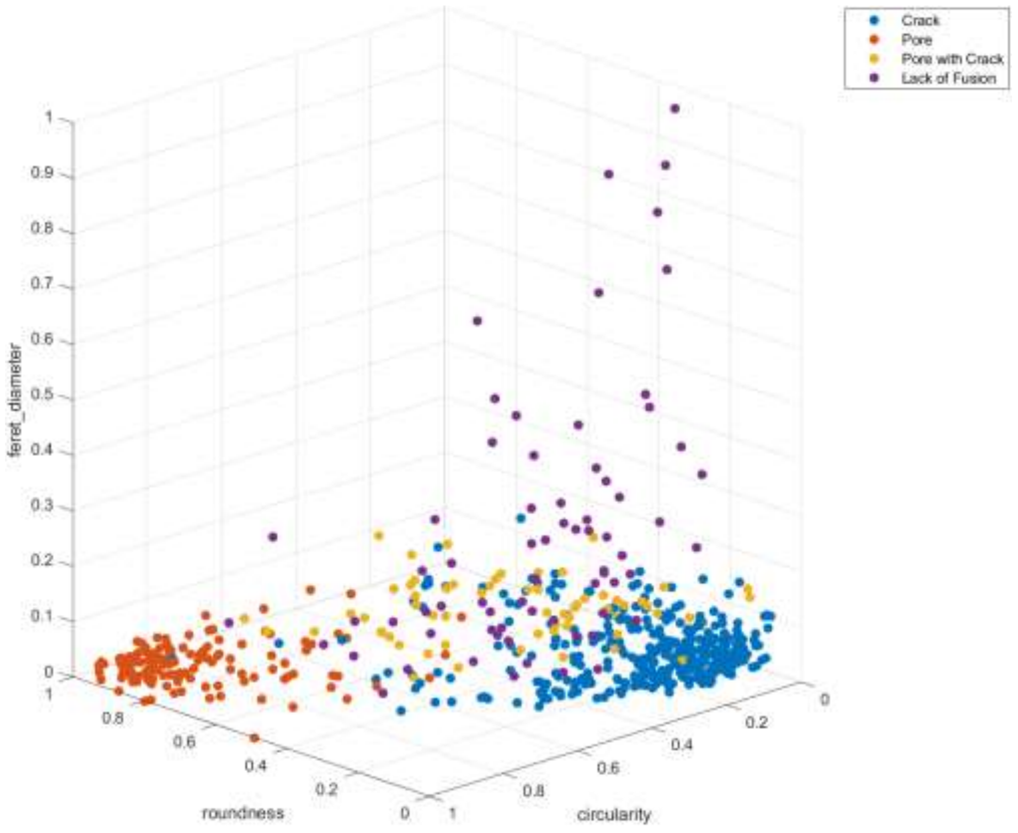


Figure 13: 3D Scatter Plot showing Feret diameter, roundness, and circularity for datapoints

Figure 13 illustrates that ‘Lack of Fusion’ defects exhibit a wider range of Feret diameter compared to other defect classes, potentially enabling ML models to distinguish them effectively. Moreover, there's a notable difference between ‘Pore’ and ‘Crack’ defect types: ‘Pore’ defects typically show high roundness and circularity, while ‘Crack’ defects usually have lower values. This distinction offers another pattern for ML algorithms to leverage during classification. However, ‘Pore with Crack’ defects may present classification challenges, as they often share similar Feret diameters with both ‘Pore’ and ‘Crack’ but lack clear alignment with any specific class regarding roundness and circularity.

Before training the ML models, the original dataset underwent preprocessing. Preprocessing is essential in ML because datasets can often be incomplete, noisy, or inconsistent, which may disrupt the training process of certain machine learning models [84]. To address these issues, normalisation techniques are typically employed, as they scale attributes with reference to the rest of the dataset. In addition to minimum maximum (min max) normalisation, z-score normalisation was also executed using equations 3 and 4 respectively. Consequently, three distinct datasets were created: one without normalisation, one with z-score normalisation, and one with min-max normalisation. Each dataset was then used to train the supervised ML models. By comparing ML models trained on these respective datasets, the effect of the normalisation method on the performance of ML models for microstructural defect classification can be evaluated.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Equation 3: Min-max normalisation [84]

$$x' = \frac{x - \mu}{\sigma}$$

Equation 4: Z-score normalisation [84]

4.2 Importance of K in Model Performance

For KNN in particular, k is another parameter which can significantly affect the performance of the ML model. To evaluate this, a baseline training and testing dataset was created, with different normalisation methods applied. The KNN algorithm was trained using the training dataset. Finally, the test dataset was evaluated using a relatively straightforward approach to measuring accuracy. This simplified approach is shown in equation 5.

$$Model\ Efficiency = \frac{a}{b} * 100$$

Equation 5: Simplified approach to measure ML model accuracy

For equation 5, a is the number of correctly predicted elements that match the test dataset, and b is the total number of elements in the test dataset. Figure 14 illustrates the impact of varying k values on the model accuracy for each normalisation method, using this simplified accuracy metric, for the baseline training and test dataset.

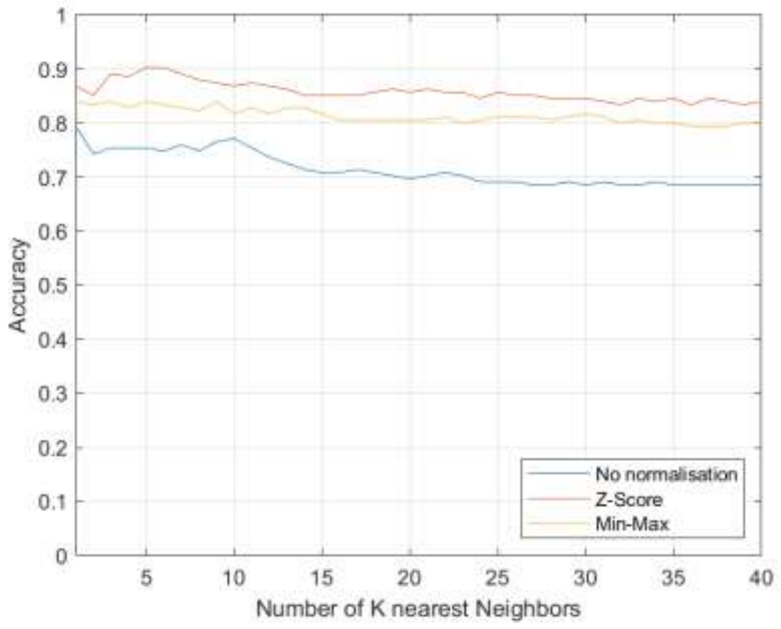


Figure 14: Variation of model accuracy with k for various normalisation methods for baseline training and testing dataset

Figure 14 emphasises the crucial role of k in KNN classification. It highlights that data without normalisation leads to inferior KNN model performance, underlining the importance of data preprocessing. Z-score normalisation typically yields optimal accuracy, standardising feature ranges by transforming data points based on standard deviation and mean. This mitigates scale discrepancies among variables, enhancing model accuracy and stability. Optimal KNN accuracy for baseline training and testing datasets was achieved when $k = 5$.

The training dataset contained 70% of the original dataset and the test dataset contained the rest of the datapoints. To ensure robust model generation, a stratified sampling technique was used to maintain consistent class distributions across the training and test datasets. This ensures that the ratio of instances of each class relative to each other is the same in both the training and test datasets. This is an important factor in effectively training ML models, so enough information about each class can be supplied for training.

4.3 Variation in KNN Classification Performance with Randomised Training and Test Datasets

Performance of ML models can vary significantly depending on the size and contents of the training and test datasets. To show this, training, and testing datasets, with 70% and 30% of datapoints respectively, were randomly created 500 times, and normalised. These ML models were then evaluated using the simplified approach to measuring accuracy as shown in equation 5. The resulting performance data is illustrated in Figure 15, with numerical data being supplemented in Table 6, showing the average performance of the KNN classification algorithm with varying training and test datasets.

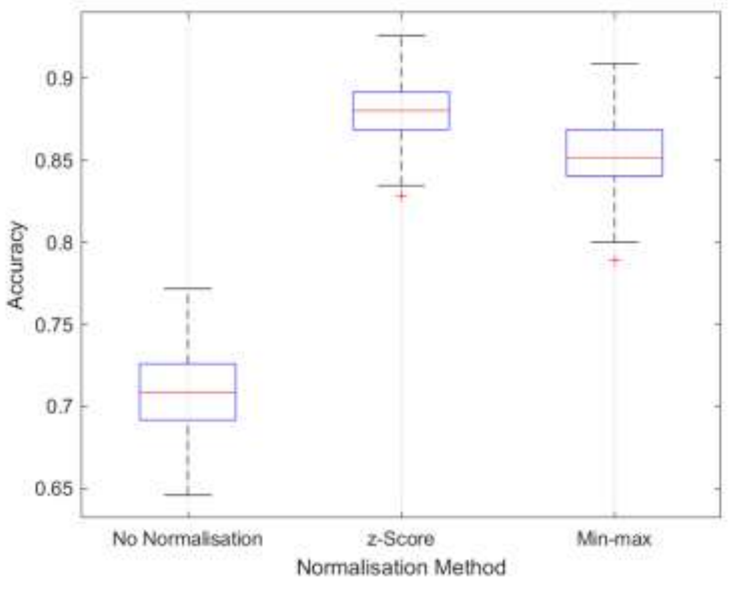


Figure 15: Box plot summarising performance of ML models for various training and testing datasets

Normalisation Method	Mean Accuracy	Standard Deviation of Accuracies
No Normalisation	0.7098	0.0241
z-Score	0.8804	0.0169
Min max	0.8536	0.0200

Table 6: Statistical data summarising performance of ML models for various training and testing datasets

The data demonstrates that KNN models constructed without normalisation consistently displayed the lowest accuracy, emphasising the pivotal role of normalisation in improving model performance. Models utilising datasets normalised via z-score transformation exhibited the highest mean accuracy. Additionally, lower standard deviation in accuracies, particularly observed with z-score normalisation, signifies greater consistency and reliability in model outcomes. This reliability is vital in real-world applications where predictability and consistency of model performance are paramount. In contrast, datasets without normalisation showed greater variation in classification algorithm performance.

4.4 KNN Model Performance using Baseline Datasets with z-Score Normalisation

Based on the analysis conducted, z-score normalisation, for the baseline training and testing datasets, showed the best performance to be used for the KNN classification algorithm. The baseline iteration, the accuracies of which are shown in Figure 12, suggests that $k = 5$ yields the optimum accuracy for the KNN classification algorithm.

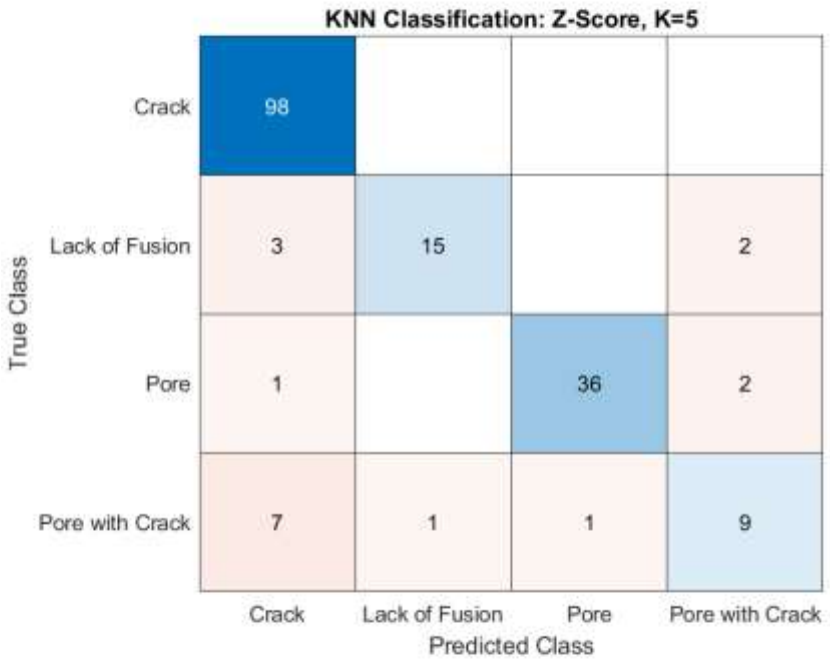


Figure 16: Optimal KNN classification model performance

The confusion matrix in Figure 16 highlights the KNN classification algorithm's performance, revealing its struggle with accurately classifying 'Pore with Crack' defects. Among 18 instances, 5 were misclassified as 'Crack' and 2 as 'Pore', resulting in a mere 38.9% accuracy for 'Pore with Crack' classification. This observation aligns with Figure 13, where 'Pore with Crack' defects lacked distinct features from 'Pore' and 'Crack' classes. Despite these challenges, the KNN model achieved an impressive overall accuracy of 90.3%, mainly due to its flawless classification of 'Crack' defects. This overall accuracy, calculated using equation 6, quantifies the model's ability to predict across various class types accurately.

$$Accuracy = \frac{Number\ of\ true\ positives}{Size\ of\ testing\ dataset} = \frac{158}{175} = 0.903$$

Equation 6: Simplified accuracy calculation for KNN classification algorithm

Confusion matrices evaluating the performance of the baseline training and datasets with no normalisation and min max normalisation were also created and are attached in the appendix (Section 12.1.1) for the readers viewing.

5 Decision Trees

The evaluation of decision trees as a ML model for the classification of microstructural defects is fairly like that of the KNN classification model evaluation. It considers the effect of normalisation methods on the performance of decision trees, as well as the variation of performance with a variety of training and testing datasets.

5.1 Variation in Decision Tree Performance with Various Training and Testing Datasets

Similarly to KNN classification algorithms, decision tree ML algorithms' performance can vary based on the training and test datasets employed. To illustrate this, a process akin to the KNN classification evaluation was conducted. Training and test datasets were randomly generated, and a decision tree model was trained on each pair. This process was iterated 500 times, evaluating the decision tree algorithm's performance in each iteration. The experiment revealed significant variability in decision tree models' performance across datasets with no normalisation, z-score normalisation, and min-max normalisation, as demonstrated in Figure 17 and Table 7.

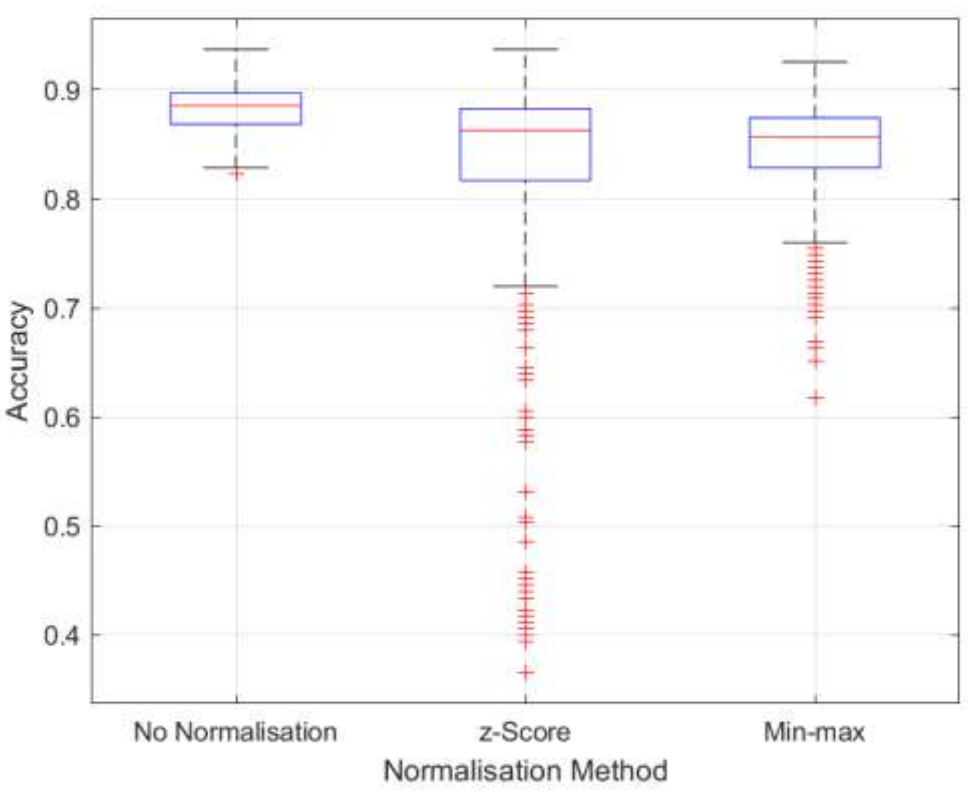


Figure 17: Statistical information regarding accuracy of decision trees ML algorithm with various normalisation methods

Normalisation Method	Mean Accuracy	Standard Deviation of Accuracies
No Normalisation	0.8842	0.0216
z-Score	0.8194	0.1195
Min max	0.8467	0.0472

Table 7: Statistical data regarding decision tree performance for various training and testing datasets

Figure 17 illustrates considerable performance variability in the decision tree algorithm, particularly noticeable in datasets subjected to z-score and min-max normalisation, displaying numerous outliers. Unlike KNN classification algorithms, decision tree models exhibited optimal performance with datasets that lacked normalisation. This superior performance could be attributed to the inherent nature of decision trees, which are less sensitive to data scaling. This happens because decision trees split nodes based on data order rather than absolute values, diminishing the need for normalisation and occasionally making it unnecessary to maintain or enhance model accuracy [85].

5.2 Decision Tree Performance using Baseline Dataset with no Normalisation

The baseline training and testing datasets, without normalisation, were used for the training of a decision tree, yielding the confusion matrix shown in Figure 17.

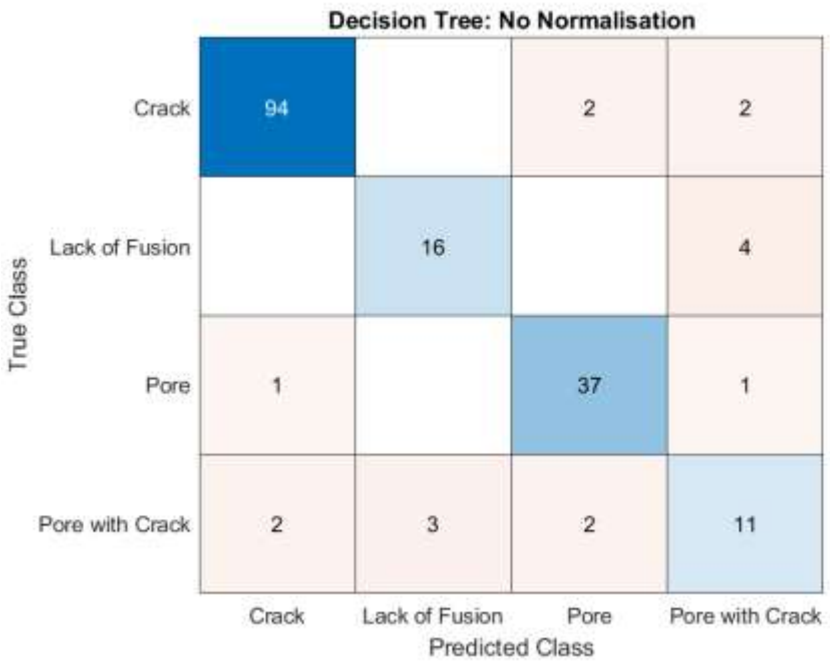


Figure 18: confusion matrix of decision tree without normalisation

The decision tree algorithm achieved an overall accuracy like the KNN classification algorithm at 90.3%. However, it encountered challenges in accurately classifying 'Pore with Crack' defects, achieving an accuracy of 61.1%. Despite this, it outperformed the KNN algorithm in classifying 'Pore with Crack' defects. While the KNN algorithm mostly misclassified them as 'Crack' defects, the decision tree algorithm incorrectly classified them as 'Crack', 'Lack of Fusion', and 'Pore' defects. Nonetheless, the decision tree model demonstrated high accuracy in predicting 'Crack' and 'Pore' defects.

Confusion matrices evaluating the performance of the baseline training and datasets with no normalisation and min max normalisation were also created and are attached in the appendix (Section 12.1.2) for the readers viewing.

6 Deep Convoluted Neural Networks

6.1 Data Processing

To build and train CNNs, images are utilised as input rather than numerical data. Similarly to supervised ML methods, input data needs to undergo preprocessing. In the case of CNN's, binary images of defects from the original micrographs were created, as shown in Figure 11, and resized to the same format: [100 x 100 x 1]. In this format, each image of the defect will be 100 pixels in width and height, without any colour scaling. This standardisation is crucial because CNNs, especially those implemented in MATLAB, require consistent input dimensions to ensure compatibility across the network's layers.

However, resizing images presents significant challenges. Firstly, enlarging or shrinking images can lead to distortions of their characteristics and quality. For instance, enlarging small defects can degrade the image quality. Conversely, reducing larger original images can result in a loss of critical details. Current research efforts are aiming to overcome this limitation by addressing these challenges, seeking to maintain image integrity not only in still images but also in video formats. This endeavour is intended to improve the accuracy and effectiveness of CNN applications [86], [87], [88]. One method of reducing the effect of resizing images is to increase their resolution.

Another concern was that resizing of the images to the same size might distort the original images properties, such as aspect ratio. This is an especially big concern, considering 'Crack' defects typically exhibit a higher aspect ratio (ratio of width to height), whereas 'Pore' defects typically exhibit more roundness. Fortunately, MATLAB's image processing tools preserve the aspect ratio of each image during resizing, ensuring no modification to its proportions.

After resizing the images, the dataset consisted of 2030 images, categorised into their respective classes: 689 images of 'Crack', 250 images of 'Lack of Fusion', 904 images of 'Pore', and 187 images of 'Pore with Crack'. Following this, the dataset was divided into training and testing sets, with 70% allocated for training and 30% for validation. To ensure a balanced representation of each class, a stratified sampling method was employed, maintaining the same proportion of each defect type in both the training and validation datasets. This stratification methodology mirrors the approach used in supervised ML to enhance the models' generalisation ability by preserving the original distribution of classes across datasets.

6.2 Baseline Neural Network

6.2.1 Baseline Neural Network Training

Initially, a baseline convolutional neural network (CNN) was established with three layers and ten filters. The network was trained in 564 iterations. The architecture of this neural network is illustrated in the appendix (Section 12.2.1) for reference. Each layer of the neural network consisted of a convolutional layer, a batch normalization layer, a 'ReLU' layer, and a pooling layer, adhering to the standard configuration for CNNs. This setup serves as a foundational model for subsequent exploration and optimisation endeavours.

6.2.2 Baseline Neural Network Evaluation

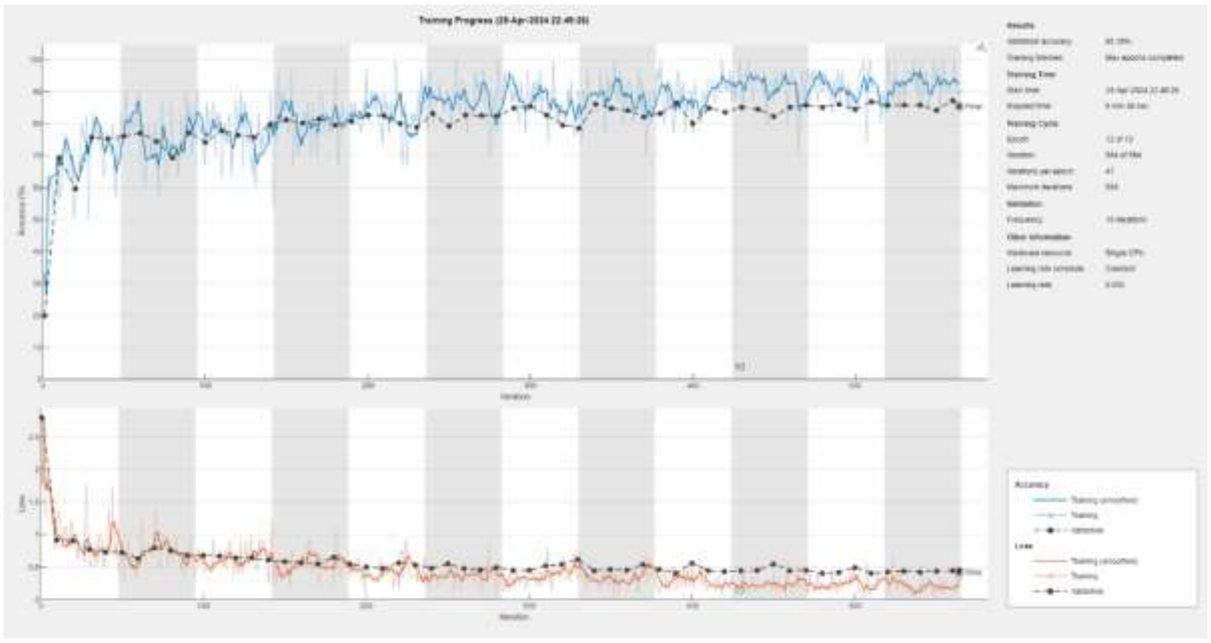
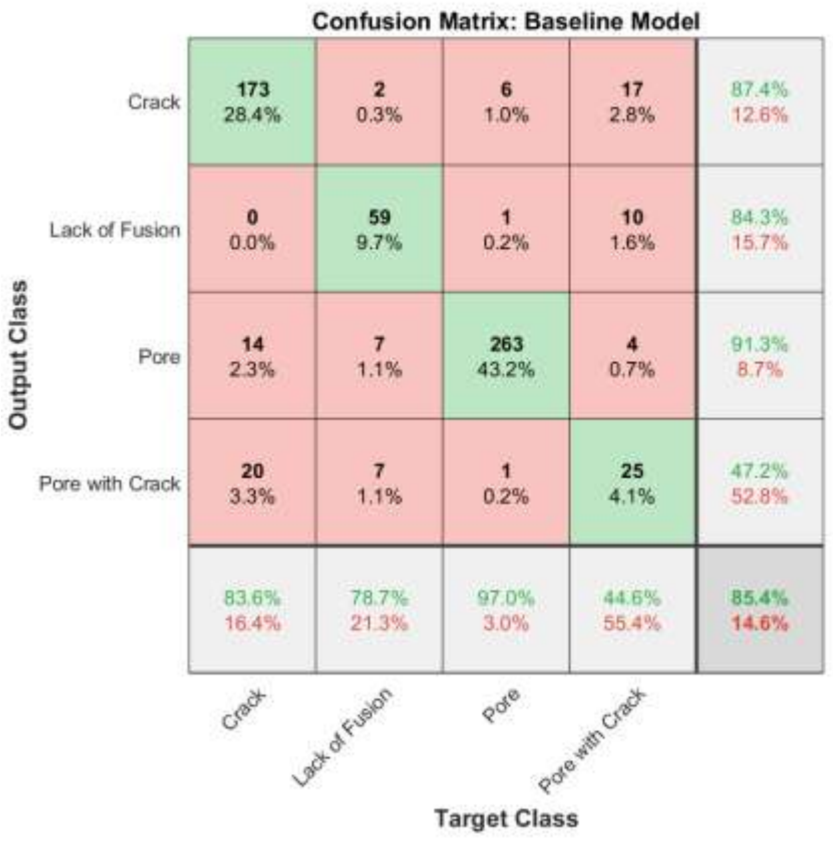

Results
Validation accuracy: [illegible]
Training finished: Max epochs completed
Training Time
Start time: [illegible]
Elapsed time: [illegible]
Training Cycle
Epoch: 12 of 12
Iteration: 564 of 564
Iterations per epoch: 47
Maximum iterations: 564
Validation
Frequency: 10 iterations
Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Learning rate: 0.001

Figure 19: Training information for baseline CNN

Figure 19 illustrates the training progress of a baseline neural network, showing the evolution of accuracy and loss over iterations for both training and validation phases. Loss represents the discrepancy between predicted and actual values in a model, often considered as the inverse of accuracy. Initially, both accuracy and loss show steep variations, with accuracy rapidly increasing and loss decreasing sharply, typical of early training dynamics. As training progresses, the accuracy stabilizes between 80-90. Overfitting is evident as the model performs well on training data but not as effectively on validation data, suggesting it has learned the noise and specific details of the training set at the expense of generalisation [89]. This visualisation aids in assessing the model's learning behaviour and highlights areas for potential improvement in robustness and generalisation to new data. Supplementary information regarding the training of the baseline neural network is supplied in the appendix (Section 12.2.2).

After training, the performance of the baseline neural network was assessed using a confusion matrix. The baseline neural network achieved an overall accuracy of 85.4% and demonstrated high precision in classifying ‘Pore’ defects. However, like the KNN and decision tree algorithms, it struggled with the classification of ‘Pore with Crack’ defects, achieving only a 44.6% accuracy rate. Future iterations of the CNN will investigate how various hyperparameters, such as number of layers and filters in the neural network, impact performance. It should be noted that the orientation of the axes for true and predicted values in the confusion matrices differs from those used for the KNN and decision tree models, due to the configuration settings in MATLAB.



Confusion Matrix: Baseline Model

Output Class \ Target Class	Crack	Lack of Fusion	Pore	Pore with Crack	
Crack	173 28.4%	2 0.3%	6 1.0%	17 2.8%	87.4% 12.6%
Lack of Fusion	0 0.0%	59 9.7%	1 0.2%	10 1.6%	84.3% 15.7%
Pore	14 2.3%	7 1.1%	263 43.2%	4 0.7%	91.3% 8.7%
Pore with Crack	20 3.3%	7 1.1%	1 0.2%	25 4.1%	47.2% 52.8%
	83.6% 16.4%	78.7% 21.3%	97.0% 3.0%	44.6% 55.4%	**85.4%** **14.6%**

Figure 20: Confusion matrix of baseline CNN model

6.3 Grid Layer Search

While the baseline neural network showed promising performance, many key parameters, such as the number of layers and filters per layer, remain unexplored. To investigate how different neural network architectures affect CNN performance, a grid layer search was conducted, involving varied configurations of layers and filters. Due to computational constraints, each network was trained with 40% of the dataset and a maximum of 266 iterations. Although these results may not fully represent the models' true performance, grid search provides valuable insights into architecture variations efficiently. The optimal network identified will undergo further training with the full dataset, enabling a systematic comparison with the baseline model. Data summarising the results of grid layer search are presented in Table 8 and Figure 21.

	Number of Filters					
Layers	**10**	**15**	**20**	**25**	**30**	**Mean**
3	80.624	78.9819	83.5796	82.4302	79.9672	81.11658
4	84.0722	82.9228	85.8785	85.3859	83.908	84.43348
5	84.0722	87.3563	83.4154	86.5353	85.5501	85.38586
6	83.7438	86.0427	78.6535	83.5796	85.3859	83.4811
Mean	83.12805	83.82593	82.88175	84.48275	83.7028	-

Table 8: Validation accuracies for CNNs with various network architectures

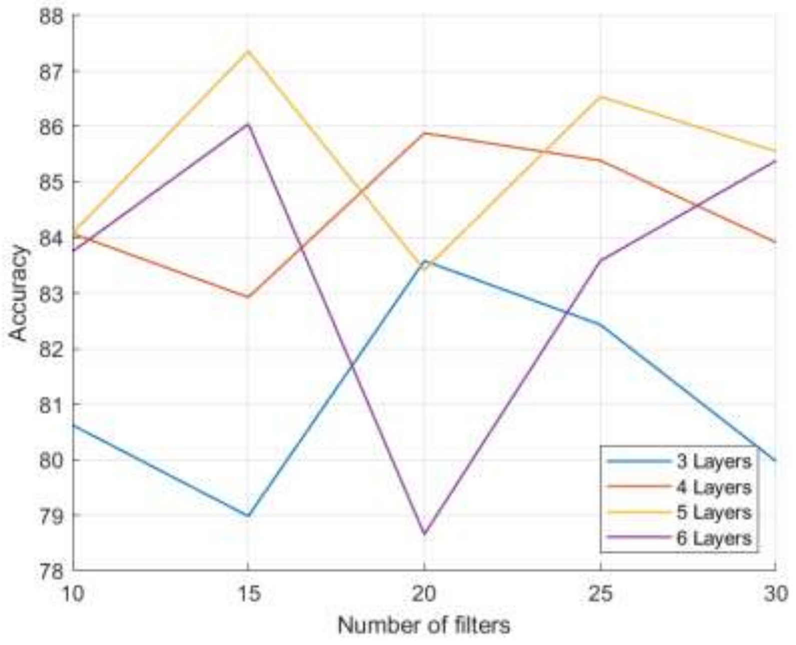


Figure 21: Variation of accuracy with number of layers and number of filters for CNN

The analysis indicates no evident correlation between neural network architecture and classification effectiveness for microstructural defects. However, Table 8 highlights that networks with 5 layers and 25 filters per layer generally performed well. Through grid layer search, optimal parameters were identified as 5 layers and 15 filters per layer. Subsequently, a new neural network was trained using these parameters and the full dataset, maintaining the iteration count consistent with the baseline model.

6.3.1 Grid Layer Search Optimised Neural Network Evaluation

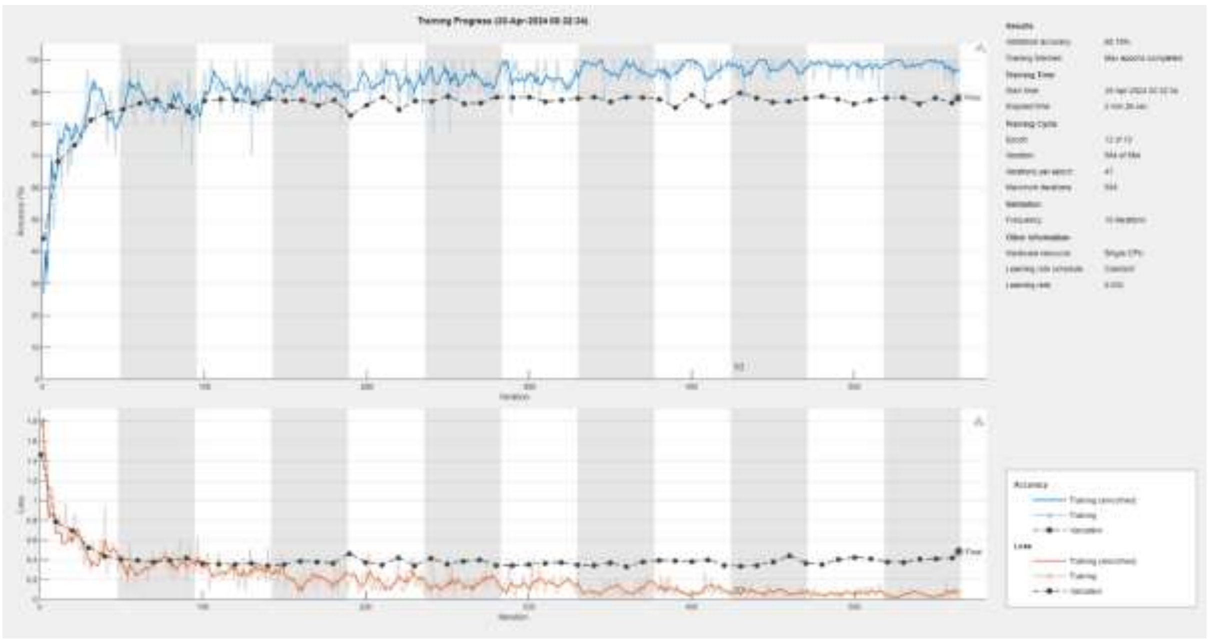


Figure 22: Training information for CNN optimised by grid layer search

Figure 22 shows the training progress for the neural network with parameters determined by grid layer search. Similarly to the baseline, there was a steep increase in model accuracy in initial iterations, but soon converged to roughly 95% accuracy for the rest of the training. It should be noted that there was still a significant difference in the training accuracy and validation accuracy suggesting overfitting.

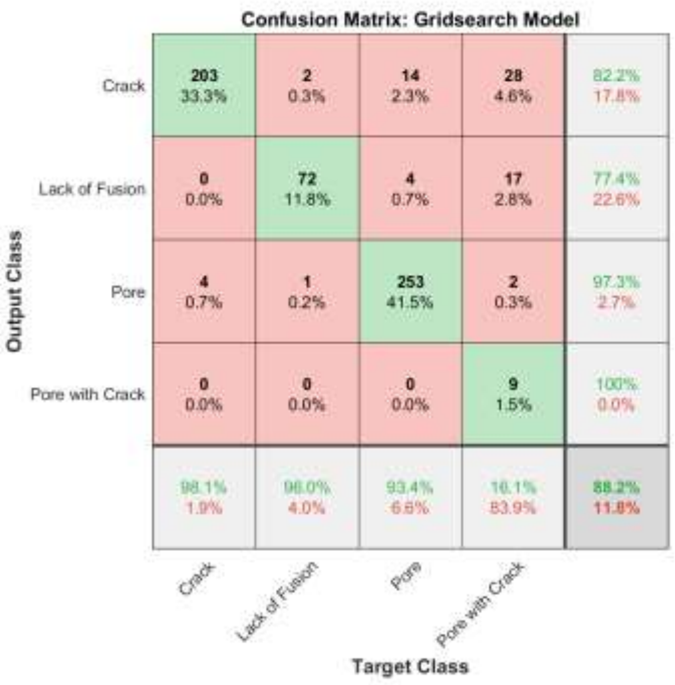


Figure 23: Confusion matrix of grid layer search optimised CNN

Compared to the baseline neural network, the model configured with optimal hyperparameters exhibited improved accuracy, achieving an overall rate of 88.2%. This optimally tuned model showed significant enhancements in accurately classifying ‘Crack’, ‘Lack of Fusion’ and ‘Pore’ defects. However, its performance in classifying ‘Pore with Crack’ defects diminished to merely 16.1%. Despite these improvements, the classification of ‘Pore with Crack’ defects continued to be suboptimal, with only 16.1% being correctly identified. Supplementary information regarding the training of the grid layer search optimised neural network is available in the appendix (Section 12.2.3) for the readers viewing.

6.4 Bayesian Optimisation

Bayesian optimisation is a technique used to efficiently find the best input settings for a function that is costly and slow to evaluate. It uses Gaussian process regression to create a substitute model for the objective function, which is a mathematical expression that optimisation algorithms aim to minimise or maximise, enabling the measurement of uncertainty and employing an acquisition function to guide the selection of sampling points [90]. This method can minimise various types of objective functions within specific limits, working with different types of inputs, and managing functions that may produce varied outcomes for identical input values [91]. Bayesian optimisation, particularly advantageous for tuning hyperparameters in CNNs, adeptly manages the trade-off between exploring new parameter combinations and exploiting known effective ones to optimise performance with minimal evaluations. This iterative approach strategically enhances prediction accuracy regarding function performance, proving invaluable for resource-intensive tasks such as CNN training. By aiming to minimise the objective function value, which signifies the error in classifying microstructural defects of the original dataset, Bayesian optimisation ensures optimal outcomes with fewer trials and reduced computational expense.

Figures 24 and 25 display the application of Bayesian optimisation for fine-tuning the hyperparameters of a neural network, specifically examining the number of layers and filters in the CNN. Figure 24, the objective function model, showcases a 3D surface plot that maps the optimisation landscape. Here, the x-axis and y-axis denote the number of filters and layers, while the z-axis indicates the estimated objective function value. This plot features observed evaluation points, the model's mean prediction, the next proposed point for evaluation, and the model's estimate of the optimal point. Figure 25 monitors the progress of the optimisation across multiple evaluations, depicting both the minimum observed objective values (blue line) and the model’s estimated minimum (green line). There is an initial sharp drop in objective values, signalling rapid enhancements, which then levels off, suggesting that the optimisation is nearing an optimal hyperparameter configuration. These visualisations effectively highlight how Bayesian optimization skilfully explores and refines the search space within the hyperparameter landscape, thereby optimising the neural network’s architecture for enhanced performance.

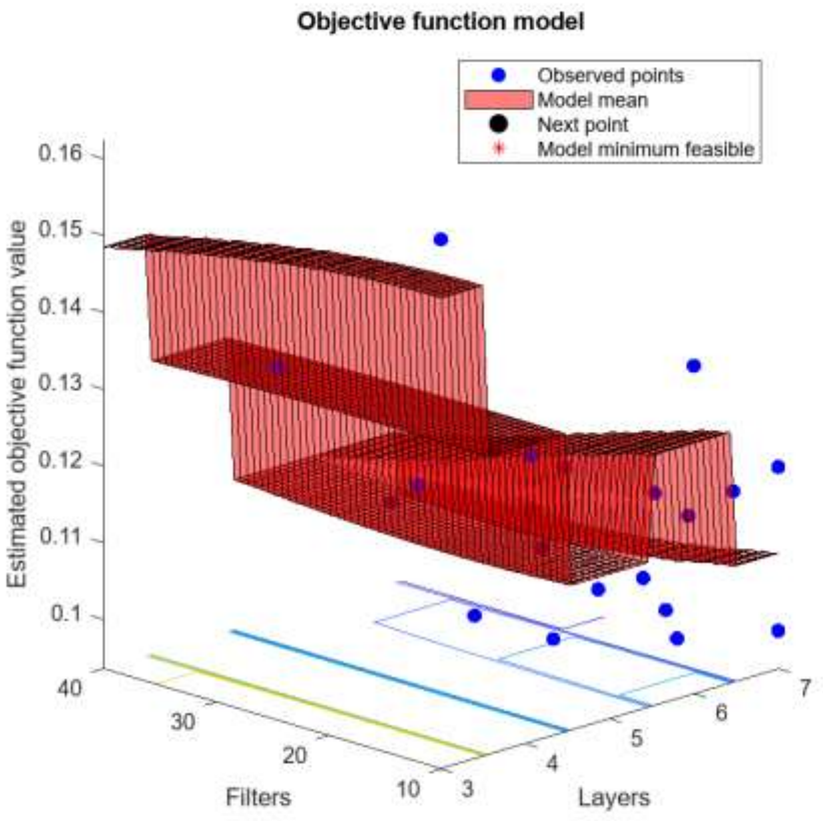


Figure 24: Variation of estimated objective function value with number of layers and number of filters used in neural network design

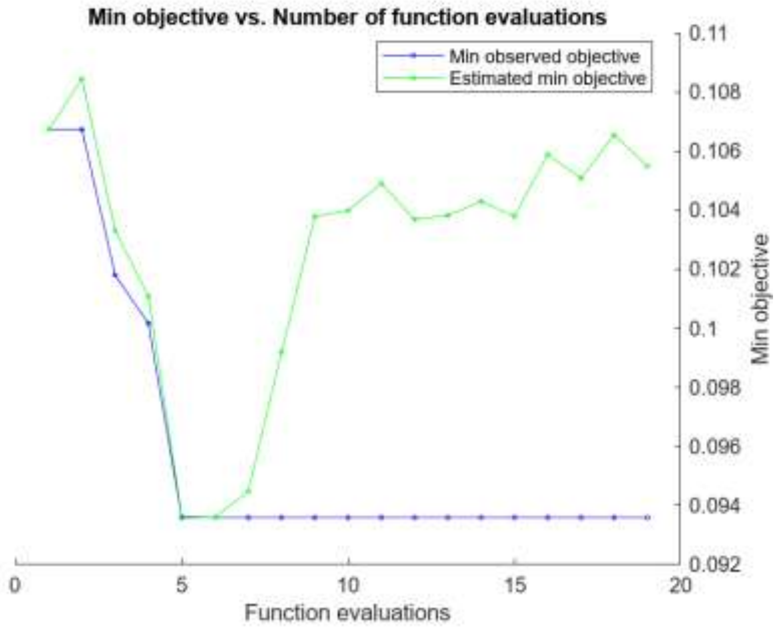


Figure 25: Variation of Objective Function with iterations of Bayesian Optimisation

6.4.1 Bayesian Optimised Neural Network Evaluation

Due to MATLAB's Bayesian optimisation implementation, specific hyperparameters for the optimal configuration couldn't be retrieved. However, based on observations from Figure 24, the optimal CNN architecture likely comprises 6 layers with 20 filters per layer. Despite this limitation, the optimal model was saved and evaluated. Figure 26 presents the confusion matrix from cross-validation of the Bayesian-optimised neural network, offering a detailed analysis of its classification performance. Cross-validation ensures robust evaluation by using multiple data subsets for training and testing [92]. High accuracies are observed for 'Crack' (92.8%), 'Lack of Fusion' (85.3%), and 'Pore' (98.9%), indicating effective defect identification with minimal misclassifications. However, 'Pore with Crack' classification remains challenging, with 50% accuracy and notable misclassifications into other categories, consistent with previous model evaluations. The optimised algorithm achieved a validation accuracy of 90.6%. Despite the confusion matrix's results, training data (which can be found in Section 12.2.4) suggested that the model was 97.4% accurate in classifying data.

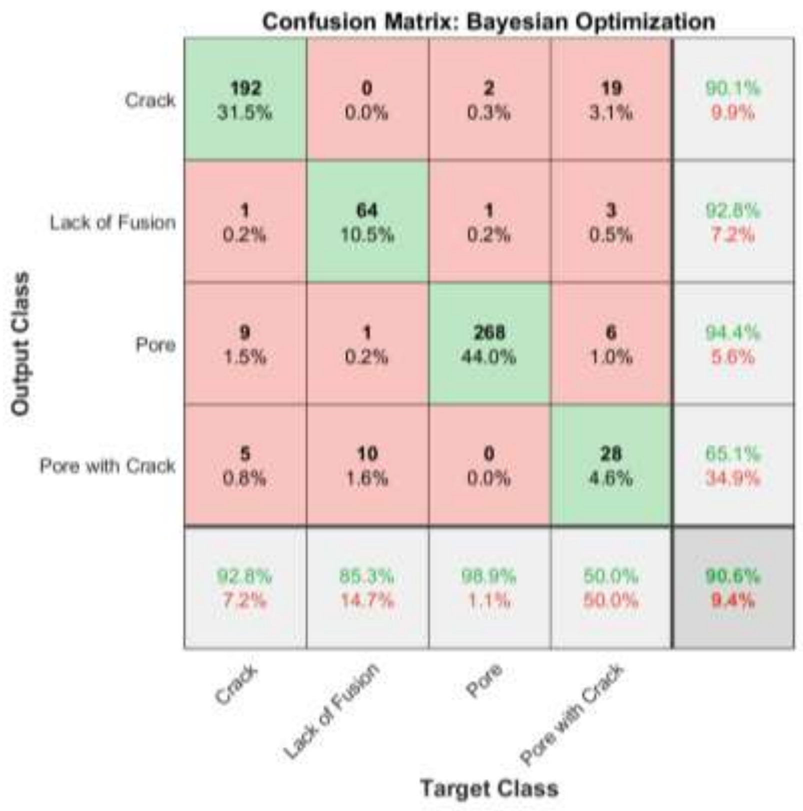


Figure 26: Confusion matrix of Bayesian optimised CNN

7 Summary of Results

Supervised machine learning algorithms exhibited varied performance based on the normalisation method. KNN with z-score normalisation led to the highest mean accuracy at 88.04%, surpassing other methods. Decision Trees performed optimally without normalisation, achieving a mean accuracy of 88.42%, albeit with higher variability when zscore normalisation was applied. Neural networks showed differing performance; the baseline model achieved 85.4% accuracy, improved to 88.2% with grid search, and further to 90.6% with Bayesian optimisation, highlighting the efficacy of advanced optimisation methods in boosting accuracy.

Supervised Algorithm	**Normalisation Method**	**Mean Value of Highest Accuracy**	**Standard Deviation of Highest Accuracy**
KNN	No Normalisation	71.24%	02.28%
KNN	z-Score normalisation	88.04%	01.54%
KNN	Min-max normalisation	85.30%	01.92%
Decision Trees	No Normalisation	88.42%	02.16%
Decision Trees	z-Score normalisation	81.94%	11.95%
Decision Trees	Min-max normalisation	84.67%	04.72%

Table 9: Summary of results for supervised machine learning algorithms

Deep Learning Model	**Accuracy**
Baseline	85.4%
Grid search Optimised Model	88.2%
Bayesian Optimised Model	90.6%

Table 10: Summary of accuracies for different strategies employed with neural network design

8 Future Work

Many of the deep learning models trained exhibited overfitting, as indicated by consistently higher accuracy on the training dataset compared to the validation dataset. To mitigate overfitting, several strategies can be explored. Increasing the size of the training data or employing data augmentation techniques can provide more diverse examples for the model, reducing its tendency to memorise noise. Simplifying the model architecture by applying regularisation methods such as L1 and L2 regularisation can help control overfitting by penalising large weights [93]. Dropout, particularly effective in neural networks, randomly deactivates neurons during training to prevent co-adaptation [94]. Early stopping, which halts training when validation performance declines, and cross-validation, ensuring model effectiveness across different data splits, are also effective techniques. Ensemble methods like bagging and boosting, as well as pruning unnecessary model components, can further reduce variance [95]. Additionally, feature selection techniques can streamline the model by removing redundant or irrelevant features, leading to improved generalization performance. Many of these techniques could be employed to reduce overfitting on the model.

To validate the effectiveness of the methodologies proposed in this paper, their application could be extended to a diverse range of materials, considering that microstructural defects are prevalent across various materials. Moreover, the potential applications of this novel deep learning-based approach seem broader compared to traditional supervised machine learning methods. Future research could explore the utilisation of this methodology in analysing videos to detect and classify defects as the alloy solidifies, aligning with ongoing efforts in this area [96]. This extension could offer valuable insights into real-time defect detection and classification during the manufacturing process, enhancing quality control measures in materials production.

Recognising the potential limitations of the data used in this study, it's important to improve the dataset for better reliability and applicability. This can be done by adding more diverse data points and exploring additional data sources. Methods like data augmentation can help in gathering more data [97]. Additionally, considering the use of generative adversarial networks (GANs) to create realistic synthetic data could be beneficial. GANs have shown promise in generating high-quality synthetic data, which can enhance the training of machine learning models alongside existing datasets [98].

9 Project Discussion and Reflection

9.1 Aims, Objectives & Project Management

The original aims and objectives, as depicted in Table 11, underwent substantial changes at the onset of the second semester due to various reasons. Challenges in acquiring microstructural images for abradable coatings analysis prompted a shift in focus. Inspired by the potential of utilising deep learning and binary images for defect classification, I proposed this approach. Following a discussion with my supervisor, Dr. Thomas, we collectively decided to pursue this new direction. The updated aims and objectives can be found in Figure 12, all of which have been completed according to the Gantt chart illustrated in Section 12.3.2. The original aims and objectives have also been supplied in Section 12.3.1. It is important to note, that for a period of time during December 2023 and January 2024, little to no work was completed. This is because I was reevaluating the project scope, aims and objectives, while completing examinations.

ID	Objective
1	Understand the practical applications of machine learning in classifying defects in aerospace materials.
2	Replicate the work of Aziz et al. [4] by implementing a supervised KNN classification ML model to classify defects in additively manufactured Nickel-Based Superalloy components.
3	Replicate the work of Aziz et al. [4] by implementing a supervised decision tree ML model to classify defects in additively manufactured Nickel-Based Superalloy components.
4	Develop an image processing algorithm to generate individual images of each defect present in micrographs of IN713C produced by SLM.

5	Evaluate the performance of a baseline CNN in classifying binary images of defects.
6	Conduct a grid layer search to identify an optimal CNN architecture and evaluate its performance in classifying binary images of defects in IN713C.
7	Utilize Bayesian optimization to determine an optimal CNN architecture and assess its performance in classifying binary images of defects in IN713C.
8	Contrast and compare the performance of various ML models in classifying different defect types in additively manufactured Nickel-Based Superalloys.

Table 11: Revised Aims and Objectives

9.2 Self-review

This project has been a great experience for me, pushing the boundaries of my technical skills beyond my initial focus in computational mechanics. While I had a solid foundation in materials science and previous software engineering experience, my knowledge of advanced machine learning methods was limited. Initially aided by Lantz' book [59], the first objectives of the project were relatively straightforward. However, learning image processing and designing CNNs in MATLAB was more difficult. The process of writing the literature review exposed me to many topics beyond the project scope, broadening my understanding of machine learning and its applications in materials science. Overall, this project has deepened my knowledge of machine learning and materials science but has helped me find a new passion in these fields.

I regret not having the opportunity to advance further with the analysis and development of ML models. If I had formulated this plan earlier, ideally in the first semester, I believe I could have achieved more and explored a wider range of possibilities. Throughout the year, managing various aspects of my final year, including responsibilities as Lead Analysis Engineer on Project Sunride, consumed significant time. Furthermore, applying for graduate schemes and post-graduate programs, along with managing other modules, proved challenging. Despite these obstacles, I am satisfied with the results of this project.

I have thoroughly enjoyed working on this project. Despite the steep learning curve and complexities of analysing ML models, I view this experience as invaluable. It has not only developed my analytical skills but also nurtured creativity and a willingness to explore novel ideas. This project has broadened my horizons and sparked an interest in exploring these fields in the future. I am hopeful for the opportunity to further delve into this topic through future study, continuing my journey in this exciting field.

10 Conclusion

Numerical data describing microstructural defects in additively manufactured nickel-alloys underwent classification using KNN algorithms and decision trees. Model accuracy varied across different normalisation methods and randomised training/testing datasets. KNN achieved the highest accuracy with z-score normalisation, while decision trees performed best without any normalisation. Optimal models for both methods achieved an overall accuracy of 90.3%.

A workflow was implemented to develop and refine a neural network for classifying microstructural defects in IN713C alloys. Initially, a baseline neural network was trained, followed by the application of grid layer search and Bayesian optimisation to enhance its architecture. The optimal CNN achieved an overall accuracy of 90.6%, demonstrating slight improvement compared to the discussed supervised methods.

The supervised methods proved notably simpler to train, demanding less time, effort, and computational resources. Nevertheless, given CNNs' inherent capabilities, particularly with larger and more intricate datasets featuring numerous classes and dimensions, they are anticipated to outperform other methods in defect classification tasks with more data.

Future work entails several key aspects, notably the acquisition of additional data to enhance model robustness, exploration of techniques to mitigate overfitting in neural networks, and further investigation into diverse applications of CNNs to broaden their utility in characterising microstructural features.

11 Bibliography

[1] R. C. Reed, *The Superalloys: Fundamentals and Applications*, 1st edition. Cambridge University Press, 2006.

[2] A. du Plessis, I. Yadroitsava, and I. Yadroitsev, “Effects of defects on mechanical properties in metal additive manufacturing: A review focusing on X-ray tomography insights,” *Materials & Design*, vol. 187, p. 108385, 2020, doi: https://doi.org/10.1016/j.matdes.2019.108385.

[3] R. R. D. W. J. Sames F. A. List, S. Pannala and S. S. Babu, “The metallurgy and processing science of metal additive manufacturing,” *International Materials Reviews*, vol. 61, no. 5, pp. 315–360, 2016, doi: 10.1080/09506608.2015.1116649.

[4] U. Aziz, A. Bradshaw, J. Lim, and M. Thomas, “Classification of defects in additively manufactured nickel alloys using supervised machine learning,” *Materials Science and Technology*, vol. 39, no. 16, pp. 2464–2468, 2023, doi: 10.1080/02670836.2023.2207337.

[5] C. K. Chua, C. H. Wong, and W. Y. Yeong, *Standards, Quality Control, and Measurement Sciences in 3D Printing and Additive Manufacturing*, 1st edition. San Diego: Elsevier Science & Technology, 2017. [Online]. Available: http://ebookcentral.proquest.com/lib/sheffield/detail.action?docID=4871409

[6] A. Mumith, M. Thomas, Z. Shah, M. Coathup, and G. Blunn, “Additive manufacturing current concepts, future trends,” *The bone & joint journal*, vol. 100B, no. 4, pp. 455–460, 2018, doi: 10.1302/0301-620X.100B4.BJJ-2017-0662.R2.

[7] S. Brischetto, P. Maggiore, and C. G. Ferro, “Special Issue on ‘Additive Manufacturing Technologies and Applications,’” *Technologies*, vol. 5, no. 3, 2017, doi: 10.3390/technologies5030058.

[8] J. ZHU, H. ZHOU, C. WANG, L. ZHOU, S. YUAN, and W. ZHANG, “A review of topology optimization for additive manufacturing: Status and challenges,” *Chinese Journal of Aeronautics*, vol. 34, no. 1, pp. 91–110, 2021, doi: https://doi.org/10.1016/j.cja.2020.09.020.

[9] K. G. Prashanth, “Selective Laser Melting: Materials and Applications,” *Journal of Manufacturing and Materials Processing*, vol. 4, no. 1, 2020, doi: 10.3390/jmmp4010013.

[10] Y. Zhang, X. Xu, and Y. Liu, “Numerical control machining simulation: a comprehensive survey,” *International Journal of Computer Integrated Manufacturing*, vol. 24, no. 7, pp. 593–609, Jul. 2011, doi: 10.1080/0951192X.2011.566283.

[11] X. Zhang and F. Liou, “Introduction to additive manufacturing,” in *Additive Manufacturing*, San Diego: Elsevier, 2021. [Online]. Available: http://ebookcentral.proquest.com/lib/sheffield/detail.action?docID=6630989

[12] J. P. Kruth, B. Vandenbroucke, J. V. Vaerenbergh, and P. Mercelis, “Benchmarking of different SLS/SLM processes as Rapid Manufacturing techniques,” 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:18667440

[13] E. S. Statnik, K. V. Nyaza, A. I. Salimon, D. Ryabov, and A. M. Korsunsky, “In Situ SEM Study of the Micro-Mechanical Behaviour of 3D-Printed Aluminium Alloy,” *Technologies*, vol. 9, no. 1, 2021, doi: 10.3390/technologies9010021.

[14] L. Magerramova, B. Vasilyev, and V. Kinzburskiy, *Novel Designs of Turbine Blades for Additive Manufacturing*, vol. 5C. in Turbo Expo: Power for Land, Sea, and Air, vol. 5C. South Korea: The American Society of Mechanical Engineers, 2016. doi: 10.1115/GT2016-56084.

[15] G. Huang, Z. Min, L. Yang, P.-X. Jiang, and M. Chyu, “Transpiration cooling for additive manufactured porous plates with partition walls,” *International Journal of Heat and Mass Transfer*, vol. 124, pp. 1076–1087, 2018, doi: https://doi.org/10.1016/j.ijheatmasstransfer.2018.03.110.

[16] C. K. Stimpson, J. C. Snyder, K. A. Thole, and D. Mongillo, “Effectiveness Measurements of Additively Manufactured Film Cooling Holes,” *Journal of Turbomachinery*, vol. 140, no. 1, p. 011009, Oct. 2017, doi: 10.1115/1.4038182.

[17] M. M. Mojtaba Khanzadeh Sudipta Chowdhury, Mark A. Tschopp, Haley R. Doude and L. Bian, “In-situ monitoring of melt pool images for porosity prediction in directed energy deposition processes,” *IISE Transactions*, vol. 51, no. 5, pp. 437–455, 2019, doi: 10.1080/24725854.2017.1417656.

[18] M. Gorelik, “Additive manufacturing in the context of structural integrity,” *International Journal of Fatigue*, vol. 94, pp. 168–177, 2017, doi: https://doi.org/10.1016/j.ijfatigue.2016.07.005.

[19] K. G. Prashanth *et al.*, “Microstructure and mechanical properties of Al–12Si produced by selective laser melting: Effect of heat treatment,” *Materials Science and Engineering: A*, vol. 590, pp. 153–160, 2014, doi: https://doi.org/10.1016/j.msea.2013.10.023.

[20] N. Singh, P. Hameed, R. Ummethala, G. Manivasagam, K. G. Prashanth, and J. Eckert, “Selective laser manufacturing of Ti-based alloys and composites: impact of process parameters, application trends, and future prospects,” *Materials Today Advances*, vol. 8, p. 100097, 2020, doi: https://doi.org/10.1016/j.mtadv.2020.100097.

[21] X. Song, W. Zhai, R. Huang, J. Fu, M. W. Fu, and F. Li, “Metal-Based 3D-Printed Micro Parts & Structures,” in *Encyclopedia of Materials: Metals and Alloys*, F. G. Caballero, Ed., Oxford: Elsevier, 2022, pp. 448–461. doi: https://doi.org/10.1016/B978-0-12-819726-4.00009-0.

[22] E. Yasa, “Selective laser melting: principles and surface quality,” in *Additive Manufacturing*, San Diego: Elsevier, 2021. [Online]. Available: http://ebookcentral.proquest.com/lib/sheffield/detail.action?docID=6630989

[23] B. Nagarajan, Z. Hu, X. Song, W. Zhai, and J. Wei, “Development of Micro Selective Laser Melting: The State of the Art and Future Perspectives,” *Engineering*, vol. 5, no. 4, pp. 702–720, 2019, doi: https://doi.org/10.1016/j.eng.2019.07.002.

[24] T. D. Ngo, A. Kashani, G. Imbalzano, K. T. Q. Nguyen, and D. Hui, “Additive manufacturing (3D printing): A review of materials, methods, applications and challenges,” *Composites Part B: Engineering*, vol. 143, pp. 172–196, 2018, doi: https://doi.org/10.1016/j.compositesb.2018.02.012.

[25] R. Acevedo, P. Sedlak, R. Kolman, and M. Fredel, “Residual stress analysis of additive manufacturing of metallic parts using ultrasonic waves: State of the art review,” *Journal of Materials Research and Technology*, vol. 9, no. 4, pp. 9457–9477, 2020, doi: https://doi.org/10.1016/j.jmrt.2020.05.092.

[26] T. T. Opoz, A. Burgess, J. I. Ahuir-Torres, H. R. Kotadia, and S. Tammas-Williams, “The effect of surface finish and post-processing on mechanical properties of 17-4 PH stainless steel produced by the atomic diffusion additive manufacturing process (ADAM),” *The International Journal of Advanced Manufacturing Technology*, vol. 130, no. 7, pp. 4053–4066, Feb. 2024, doi: 10.1007/s00170-024-12949-6.

[27] C. T. Sims, N. S. Stoloff, and W. C. Hagel, *Superalloys II: High-Temperature Materials for Aerospace and Industrial Power*. New York: John Wiley & Sons, 1987.

[28] T. M. Pollock and S. Tin, “Nickel-Based Superalloys for Advanced Turbine Engines: Chemistry, Microstructure and Properties,” *Journal of Propulsion and Power*, vol. 22, no. 2, pp. 361–374, 2006, doi: 10.2514/1.18239.

[29] J. Rösler, O. Näth, S. Jäger, F. Schmitz, and D. Mukherji, “Nanoporous Ni-based superalloy membranes by selective phase dissolution,” *JOM: the journal of the Minerals, Metals & Materials Society*, vol. 57, pp. 52–55, Mar. 2005, doi: 10.1007/s11837-005-0234-7.

[30] R. Sowa, S. Arabasz, and M. Parlinska-Wojtan, “Classification and microstructural stability of high generation single crystal Nickel-based superalloys,” *Zastita materijala*, vol. 57, pp. 274–281, Jan. 2016, doi: 10.5937/ZasMat1602274S.

[31] H. Farhat, “Chapter 8 - Lifetime extension: Assessment and considerations,” in *Operation, Maintenance, and Repair of Land-Based Gas Turbines*, H. Farhat, Ed., Elsevier, 2021, pp. 175–196. doi: https://doi.org/10.1016/B978-0-12-821834-1.00003-4.

[32] K. Christofidou, “The Superalloys.” The University of Sheffield, 2023.
[33] J. E. Kanyo, S. Schafföner, R. S. Uwanyuze, and K. S. Leary, “An overview of ceramic molds for investment casting of nickel superalloys,” *Journal of the European Ceramic Society*, vol. 40, no. 15, pp. 4955–4973, 2020, doi: https://doi.org/10.1016/j.jeurceramsoc.2020.07.013.
[34] F. Wang, D. Ma, J. Zhang, S. Bogner, and A. Bührig-Polaczek, “Solidification behavior of a Ni-based single crystal CMSX-4 superalloy solidified by downward directional solidification process,” *Materials Characterization*, vol. 101, pp. 20–25, 2015, doi: https://doi.org/10.1016/j.matchar.2015.01.003.
[35] C. M. Cheah, C. K. Chua, C. W. Lee, C. Feng, and K. Totong, “Rapid prototyping and tooling techniques: A review of applications for rapid investment casting,” *Int. J. Adv. Manuf. Technol.*, vol. 25, pp. 308–320, Feb. 2005, doi: 10.1007/s00170-003-1840-6.
[36] C. Poli, “Chapter 6 - Metal Casting Processes,” in *Design for Manufacturing*, C. Poli, Ed., Burlington: Butterworth-Heinemann, 2001, pp. 115–126. doi: https://doi.org/10.1016/B978-075067341-9.50010-0.
[37] J. S. Benjamin and J. M. Larson, “Powder metallurgy techniques applied to superalloys,” *Journal of Aircraft*, vol. 14, no. 7, pp. 613–623, 1977.
[38] H.-E. Huang and C.-H. Koo, “Characteristics and Mechanical Properties of Polycrystalline CM 247 LC Superalloy Casting,” *MATERIALS TRANSACTIONS*, vol. 45, no. 2, pp. 562–568, 2004, doi: 10.2320/matertrans.45.562.
[39] A. Alhuzaim, S. Imbrogno, and M. M. Attallah, “Direct laser deposition of crack-free CM247LC thin walls: Mechanical properties and microstructural effects of heat treatment,” *Materials & Design*, vol. 211, p. 110123, 2021, doi: https://doi.org/10.1016/j.matdes.2021.110123.
[40] Y. Lv *et al.*, “Cracking inhibition behavior and the strengthening effect of TiC particles on the CM247LC superalloy prepared by selective laser melting,” *Materials Science and Engineering: A*, vol. 858, p. 144119, 2022, doi: https://doi.org/10.1016/j.msea.2022.144119.
[41] G. Liu, J. S. Cantó, S. Winwood, K. Rhodes, and S. Birosca, “The effects of microstructure and microtexture generated during solidification on deformation micromechanism in IN713C nickel-based superalloy,” *Acta Materialia*, vol. 148, pp. 391–406, 2018, doi: https://doi.org/10.1016/j.actamat.2018.01.062.
[42] Q. Li *et al.*, “Solidification characteristics and high temperature tensile properties of Ni-based superalloy IN713C,” *Journal of Alloys and Compounds*, vol. 923, p. 166390, 2022, doi: https://doi.org/10.1016/j.jallcom.2022.166390.
[43] M. Coleman, H. Alshehri, R. Banik, W. Harrison, and S. Birosca, “Deformation mechanisms of IN713C nickel based superalloy during Small Punch Testing,” *Materials Science and Engineering: A*, vol. 650, pp. 422–431, 2016, doi: https://doi.org/10.1016/j.msea.2015.10.056.
[44] G. Liu, J. S. Cantó, S. Birosca, S. Wang, and Y. Zhao, “A study of high cycle fatigue life and its correlation with microstructural parameters in IN713C nickel-based superalloy,” *Materials Science and Engineering: A*, vol. 877, p. 145161, 2023, doi: https://doi.org/10.1016/j.msea.2023.145161.
[45] C. Liu, “Selective laser melting of nickel superalloys for aerospace applications: defect analysis and material property optimisation,” Doctoral Thesis, University of Sheffield, Sheffield, 2021. [Online]. Available: https://etheses.whiterose.ac.uk/30067/
[46] H. Fecht and D. Furrer, “Processing of Nickel-Base Superalloys for Turbine Engine Disc Applications,” *Advanced Engineering Materials*, vol. 2, no. 12, pp. 777–787, 2000, doi: https://doi.org/10.1002/1527-2648(200012)2:12<777::AID-ADEM777>3.0.CO;2-R.
[47] R. C. Reed, T. Tao, and N. Warnken, “Alloys-By-Design: Application to nickel-based single crystal superalloys,” *Acta Materialia*, vol. 57, no. 19, pp. 5898–5913, 2009, doi: https://doi.org/10.1016/j.actamat.2009.08.018.
[48] M. H. M. B. Henderson D. Arrell, R. Larsson and G. Marchant, “Nickel based superalloy welding practices for industrial gas turbine applications,” *Science and Technology of Welding and Joining*, vol. 9, no. 1, pp. 13–21, 2004, doi: 10.1179/136217104225017099.

[49] G. Gudivada and A. K. Pandey, “Recent developments in nickel-based superalloys for gas turbine applications: Review,” *Journal of Alloys and Compounds*, vol. 963, p. 171128, 2023, doi: https://doi.org/10.1016/j.jallcom.2023.171128.

[50] N. R. Muktinutalapati, A. Natarajan, and M. Arivarasu, “Hot Corrosion of Superalloys in Boilers for Ultra-Supercritical Power Plants,” in *Superalloys for Industry Applications*, S. Cevik, Ed., Rijeka: IntechOpen, 2018. doi: 10.5772/intechopen.76083.

[51] X. Xie, Y. Wu, C. Chi, and M. Zhang, “Superalloys for Advanced Ultra-Super-Critical Fossil Power Plant Application,” in *Superalloys*, M. Aliofkhazraei, Ed., Rijeka: IntechOpen, 2015. doi: 10.5772/61139.

[52] P. Millet, “Power Industry: Corrosion,” in *Encyclopedia of Materials: Science and Technology*, K. H. J. Buschow, R. W. Cahn, M. C. Flemings, B. Ilschner, E. J. Kramer, S. Mahajan, and P. Veyssière, Eds., Oxford: Elsevier, 2001, pp. 7811–7813. doi: https://doi.org/10.1016/B0-08-043152-6/01403-0.

[53] M. Courtis, C. Skamniotis, A. Cocks, and P. Ireland, “Coupled aerothermal-mechanical analysis in single crystal double wall transpiration cooled gas turbine blades with a large film hole density,” *Applied Thermal Engineering*, vol. 219, p. 119329, 2023, doi: https://doi.org/10.1016/j.applthermaleng.2022.119329.

[54] Moverare, Johan J. and Reed, Roger C., “Thermomechanical fatigue in single crystal superalloys,” *MATEC Web of Conferences*, vol. 14, p. 06001, 2014, doi: 10.1051/matecconf/20141406001.

[55] A. S. Shaikh, “Development of a γ’ Precipitation Hardening Ni-Base Superalloy for Additive Manufacturing,” Master of Science Thesis, Chalmers University of Technology, Sweden, 2018. doi: 10.13140/RG.2.2.11472.81921.

[56] R. A. Aridi *et al.*, “Characterization of defects in additively manufactured materials from mechanical properties,” *Materials Science and Engineering: A*, vol. 898, p. 146390, 2024, doi: https://doi.org/10.1016/j.msea.2024.146390.

[57] N. C. Alexandra Morvayová and G. Casalino, “Numerical prediction of lack-of-fusion defects in selective laser melted AlSi10Mg alloy,” *Materials Science and Technology*, vol. 39, no. 16, pp. 2334–2340, 2023, doi: 10.1080/02670836.2023.2198889.

[58] K. Ma and J. Wang, “Microstructural Characteristics and Mechanical Properties of an Additively Manufactured Nickel-Based Superalloy,” *Crystals*, vol. 12, no. 10, 2022, doi: 10.3390/cryst12101358.

[59] B. Lantz, *Machine Learning with R*, 1st edition. Birmingham, UNITED KINGDOM: Packt Publishing, Limited, 2013. [Online]. Available: http://ebookcentral.proquest.com/lib/sheffield/detail.action?docID=1343653

[60] E. Alpaydin, *Introduction to machine learning*, 3rd edition. in Adaptive computation and machine learning. Cambridge, Massachusetts: MIT Press, 2014.

[61] Z. Chen and B. Liu, *Lifelong Machine Learning*, 1st edition. London: Springer Cham, 2022.

[62] X. Zhu and A. B. Goldberg, “Introduction to Semi-Supervised Learning,” in *Introduction to Semi-Supervised Machine Learning*, 1st edition., in Synthesis Lectures on Artificial Intelligence and Machine Learning, no. 1939–4608. , Switzerland: Springer Cham, 2022. [Online]. Available: https://link-springer-com.sheffield.idm.oclc.org/book/10.1007/978-3-031-01548-9

[63] K. Chomboon, P. Chujai, P. Teerarassamee, K. Kerdprasop, and N. Kerdprasop, “An empirical study of distance metrics for k-nearest neighbor algorithm,” in *Proceedings of the 3rd international conference on industrial application engineering*, 2015, p. 4.

[64] J. Gou, L. Du, Y. Zhang, and T. Xiong, “A new distance-weighted k-nearest neighbor classifier,” *J. Inf. Comput. Sci*, vol. 9, no. 6, pp. 1429–1436, 2012.

[65] Lior Rokach and Oded Z Maimon, *Data Mining With Decision Trees: Theory And Applications*. Singapore, SINGAPORE: World Scientific Publishing Company, 2007. [Online]. Available: http://ebookcentral.proquest.com/lib/sheffield/detail.action?docID=1679477

[66] M. Bramer, “Avoiding Overfitting of Decision Trees,” in *Principles of Data Mining*, London: Springer London, 2007, pp. 119–134. doi: 10.1007/978-1-84628-766-4_8.

[67] J. Ali, R. Khan, N. Ahmad, and I. Maqsood, “Random Forests and Decision Trees,” *International Journal of Computer Science Issues*, vol. 9, no. 5, pp. 272–278, 2012.

[68] M. Fratello and R. Tagliaferri, “Decision Trees and Random Forests,” in *Encyclopedia of Bioinformatics and Computational Biology*, S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach, Eds., Oxford: Academic Press, 2019, pp. 374–383. doi: https://doi.org/10.1016/B978-0-12-809633-8.20337-3.

[69] S. K. Zhou, H. Greenspan, and D. Shen, *Deep Learning for Medical Image Analysis*. San Diego, UNITED STATES: Elsevier Science & Technology, 2017. [Online]. Available: http://ebookcentral.proquest.com/lib/sheffield/detail.action?docID=4789490

[70] N. Ketkar, *Deep Learning with Python : A Hands-On Introduction*. Berkeley, CA, UNITED STATES: Apress L. P., 2017. [Online]. Available: http://ebookcentral.proquest.com/lib/sheffield/detail.action?docID=6363117

[71] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[72] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, “An Introduction to Deep Reinforcement Learning,” *Foundations and Trends® in Machine Learning*, vol. 11, no. 3–4, pp. 219–354, 2018, doi: 10.1561/2200000071.

[73] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” *arXiv.org*, 2016, [Online]. Available: https://www.proquest.com/working-papers/tensorflow-large-scale-machine-learning-on/docview/2078019164/se-2?accountid=13828

[74] R. Kumar, “A Large-Scale Machine Learning System, Tensor flow: A Review,” *NeuroQuantology*, vol. 19, no. 11, pp. 543–548, 2021, doi: 10.48047/nq.2021.19.11.NQ21258.

[75] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec. 2015, doi: 10.1007/s11263-015-0816-y.

[76] M. Bahi and M. Batouche, “Deep Learning for Ligand-Based Virtual Screening in Drug Discovery,” Oct. 2018, pp. 1–5. doi: 10.1109/PAIS.2018.8598488.

[77] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, Apr. 1980, doi: 10.1007/BF00344251.

[78] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[79] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, “Visualizing Higher-Layer Features of a Deep Network,” *Technical Report, Univeristé de Montréal*, Jan. 2009.

[80] J. Antony, K. McGuinness, N. E. O. Connor, and K. Moran, “Quantifying Radiographic Knee Osteoarthritis Severity using Deep Convolutional Neural Networks,” *arXiv.org*, 2016, [Online]. Available: https://www.proquest.com/working-papers/quantifying-radiographic-knee-osteoarthritis/docview/2080579441/se-2?accountid=13828

[81] “Learn About Convolutional Neural Networks,” MathWorks. Accessed: May 01, 2024. [Online]. Available: https://uk.mathworks.com/help/deeplearning/ug/introduction-to-convolutional-neural-networks.html

[82] C. Boig, “The Application of Additive Manufacturing to Nickel-Base Superalloys for Turbocharger Applications,” Doctoral Thesis, University of Sheffield, Sheffield, 2019. [Online]. Available: https://etheses.whiterose.ac.uk/23416/

[83] “Particle Shape Analysis,” SympaTEC. Accessed: May 01, 2024. [Online]. Available: https://www.sympatec.com/en/particle-measurement/glossary/particle-shape/

[84] J. Han, J. Pei, and M. Kamber, *Data Mining, Southeast Asia Edition*, 1st edition. San Diego: Elsevier Science & Technology, 2006. [Online]. Available: http://ebookcentral.proquest.com/lib/sheffield/detail.action?docID=291712

[85] B. de Ville, “Decision trees,” *WIREs Computational Statistics*, vol. 5, no. 6, pp. 448–455, 2013, doi: https://doi.org/10.1002/wics.1278.

[86] S.-H. Nam *et al.*, "Content-Aware Image Resizing Detection Using Deep Neural Network," in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 106–110. doi: 10.1109/ICIP.2019.8802946.

[87] J. Kim, Y. Park, K. P. Choi, J. Lee, S. Jeon, and J. Park, "Dynamic frame resizing with convolutional neural network for efficient video compression," in *Applications of Digital Image Processing XL*, A. G. Tescher, Ed., SPIE, 2017, p. 103961R. doi: 10.1117/12.2270737.

[88] M. Arabboev *et al.*, "Development of a Novel Method for Image Resizing Using Artificial Neural Network," in *Intelligent Human Computer Interaction*, H. Zaynidinov, M. Singh, U. S. Tiwary, and D. Singh, Eds., Cham: Springer Nature Switzerland, 2023, pp. 527–539.

[89] D. M. Hawkins, "The Problem of Overfitting," *J. Chem. Inf. Comput. Sci.*, vol. 44, no. 1, pp. 1–12, Jan. 2004, doi: 10.1021/ci0342472.

[90] P. I. Frazier, "A Tutorial on Bayesian Optimization." 2018. [Online]. Available: https://arxiv.org/abs/1807.02811

[91] "Bayesian Optimization Algorithm," MathWorks. Accessed: May 01, 2024. [Online]. Available: https://uk.mathworks.com/help/stats/bayesian-optimization-algorithm.html

[92] M. W. Browne, "Cross-Validation Methods," *Journal of Mathematical Psychology*, vol. 44, no. 1, pp. 108–132, Mar. 2000, doi: 10.1006/jmps.1999.1279.

[93] A. Y. Ng, "Feature selection, L1 vs. L2 regularization, and rotational invariance," in *Proceedings of the Twenty-First International Conference on Machine Learning*, in ICML '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 78. doi: 10.1145/1015330.1015435.

[94] S. Cai, Y. Shu, G. Chen, B. C. Ooi, W. Wang, and M. Zhang, "Effective and Efficient Dropout for Deep Convolutional Neural Networks." 2020.

[95] G. Seni and J. F. Elder, "Ensembles Discovered," in *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*, Cham: Springer International Publishing, 2010, pp. 1–13. doi: 10.1007/978-3-031-01899-2_1.

[96] I. Han, S. Feng, A. Lui, F. Wilde, P. S. Grant, and E. Liotti, "Tracking the evolution of hot tears in aluminium alloys using high-speed X-ray imaging," in *16th International Conference on Modelling of Casting, Welding and Advanced Solidification Processes*, Banff: IOP Publishing, 2023, p. 012065. doi: 10.1088/1757-899x/1281/1/012065.

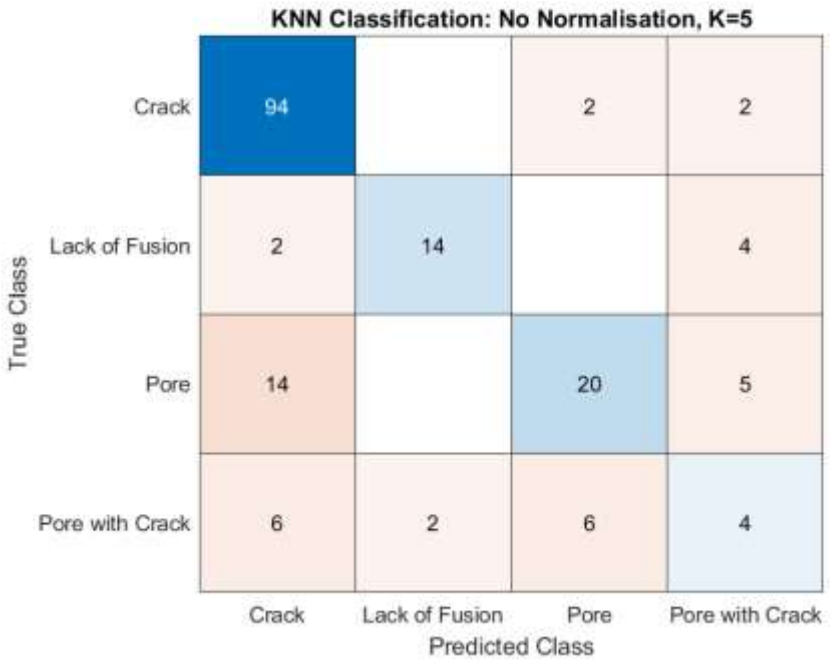
[97] "Augment Images for Deep Learning Workflows," MathWorks. Accessed: May 01, 2024. [Online]. Available: https://uk.mathworks.com/help/deeplearning/ug/image-augmentation-using-image-processing-toolbox.html

[98] A. Figueira and B. Vaz, "Survey on Synthetic Data Generation, Evaluation Methods and GANs," *Mathematics*, vol. 10, no. 15, 2022, doi: 10.3390/math10152733.

12 Appendix

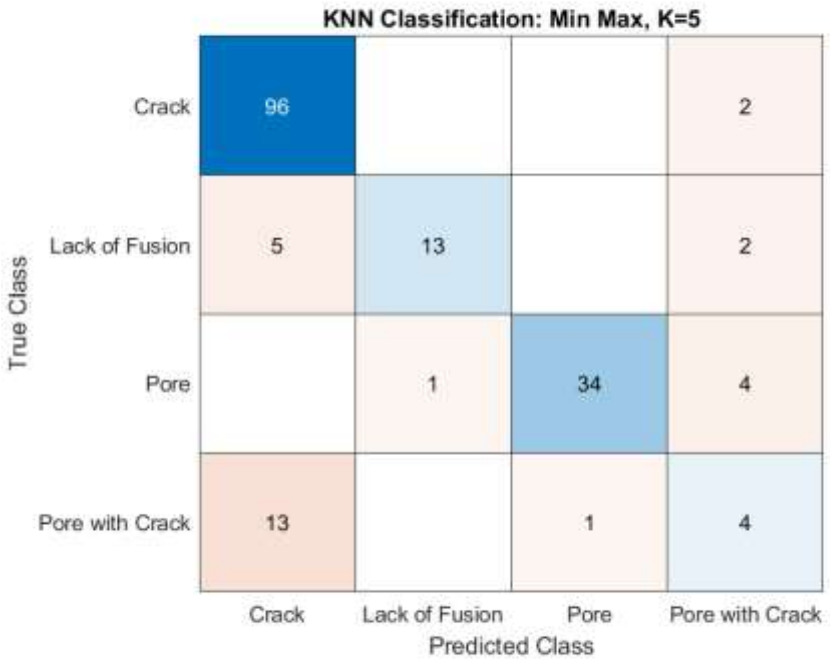
12.1 Supervised Machine Learning Methods

12.1.1 KNN Classification Confusion Matrices: No Normalisation & Min-Max Normalisation



KNN Classification: No Normalisation, K=5

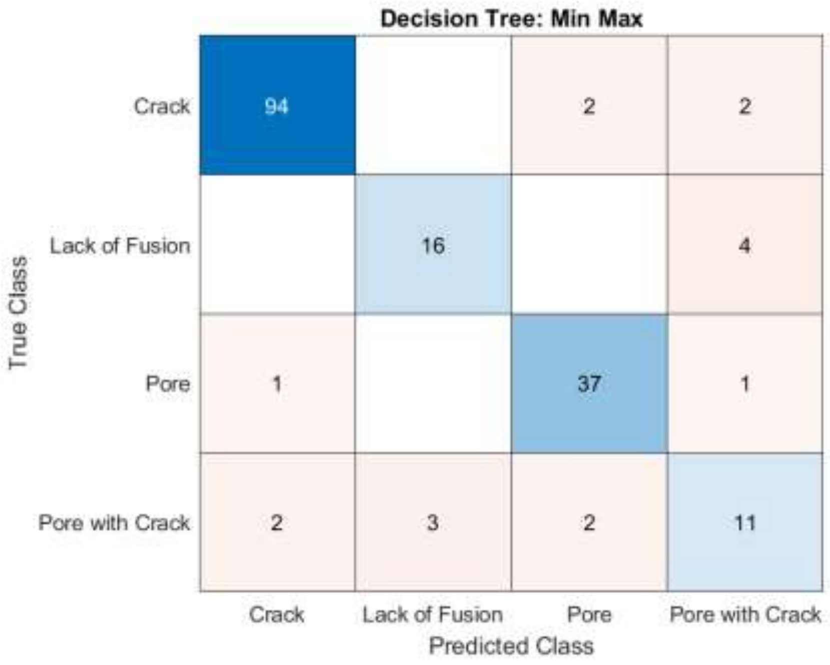
True Class \ Predicted Class	Crack	Lack of Fusion	Pore	Pore with Crack
Crack	94		2	2
Lack of Fusion	2	14		4
Pore	14		20	5
Pore with Crack	6	2	6	4

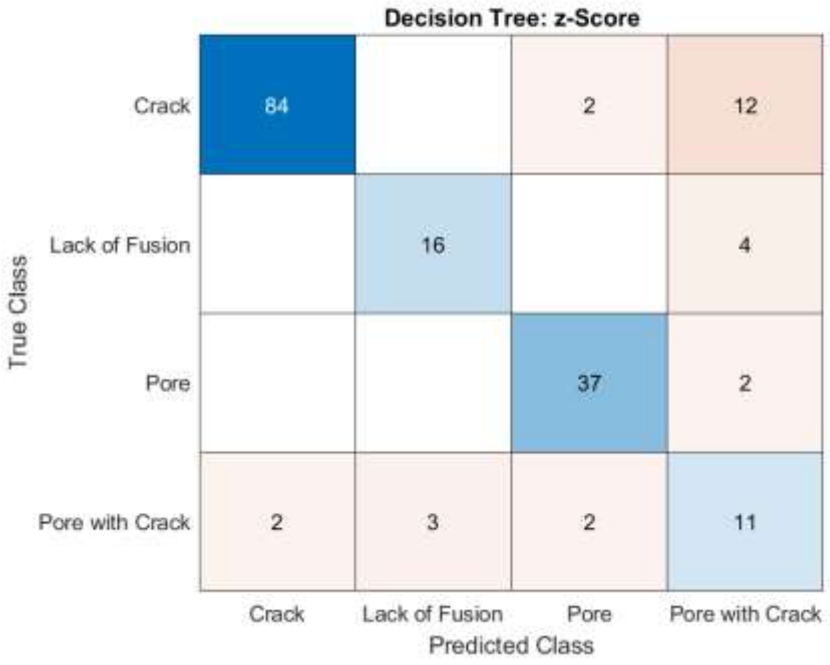


KNN Classification: Min Max, K=5

True Class \ Predicted Class	Crack	Lack of Fusion	Pore	Pore with Crack
Crack	96			2
Lack of Fusion	5	13		2
Pore		1	34	4
Pore with Crack	13		1	4

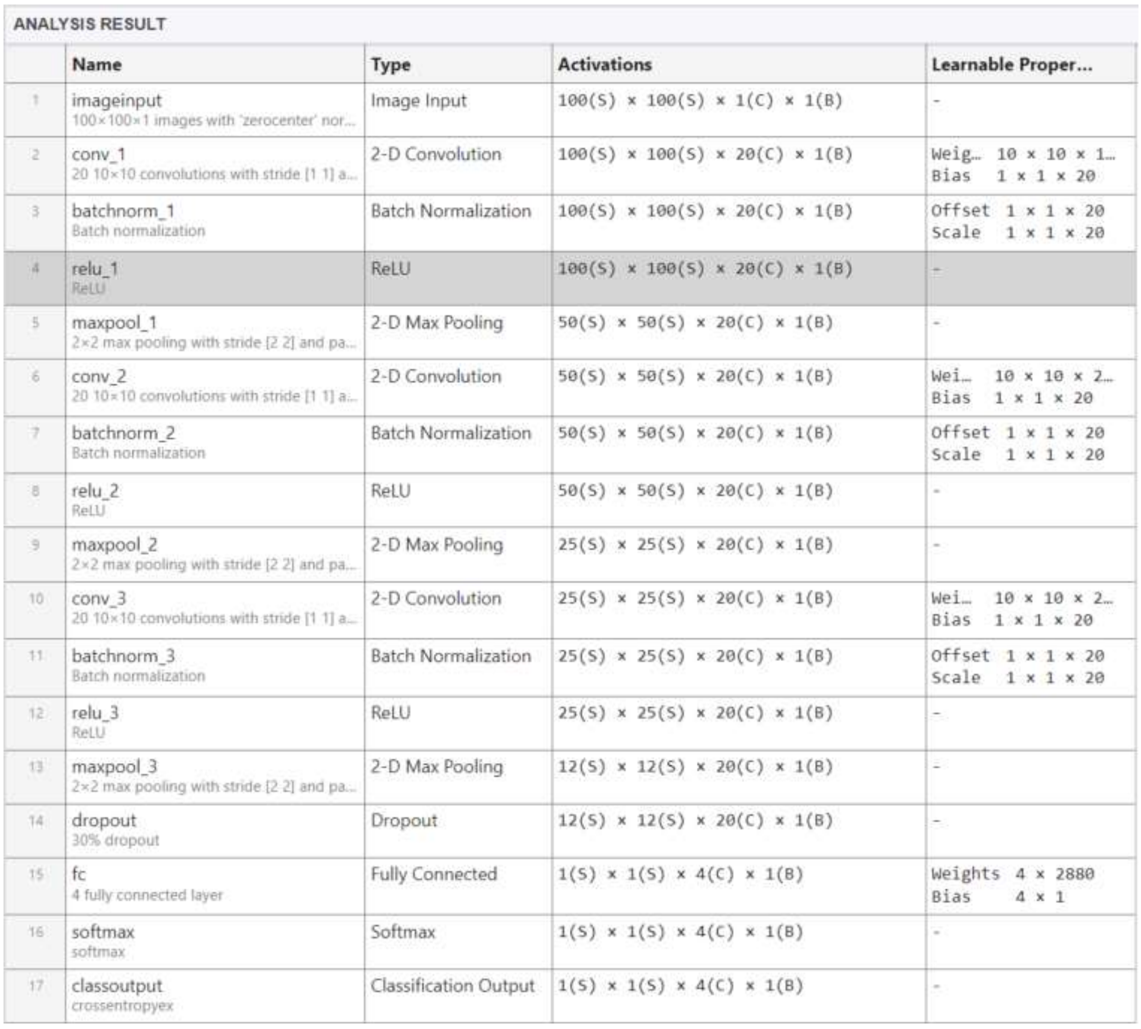
12.1.2 Decision Tree Confusion Matrices: Min-Max Normalisation & Z-Score Normalisation





12.2 Deep Learning

12.2.1 Baseline Model Architecture



ANALYSIS RESULT

	Name	Type	Activations	Learnable Proper...
1	imageinput 100×100×1 images with 'zerocenter' nor...	Image Input	100(S) × 100(S) × 1(C) × 1(B)	-
2	conv_1 20 10×10 convolutions with stride [1 1] a...	2-D Convolution	100(S) × 100(S) × 20(C) × 1(B)	Weig... 10 × 10 × 1... Bias 1 × 1 × 20
3	batchnorm_1 Batch normalization	Batch Normalization	100(S) × 100(S) × 20(C) × 1(B)	Offset 1 × 1 × 20 Scale 1 × 1 × 20
4	relu_1 ReLU	ReLU	100(S) × 100(S) × 20(C) × 1(B)	-
5	maxpool_1 2×2 max pooling with stride [2 2] and pa...	2-D Max Pooling	50(S) × 50(S) × 20(C) × 1(B)	-
6	conv_2 20 10×10 convolutions with stride [1 1] a...	2-D Convolution	50(S) × 50(S) × 20(C) × 1(B)	Wei... 10 × 10 × 2... Bias 1 × 1 × 20
7	batchnorm_2 Batch normalization	Batch Normalization	50(S) × 50(S) × 20(C) × 1(B)	Offset 1 × 1 × 20 Scale 1 × 1 × 20
8	relu_2 ReLU	ReLU	50(S) × 50(S) × 20(C) × 1(B)	-
9	maxpool_2 2×2 max pooling with stride [2 2] and pa...	2-D Max Pooling	25(S) × 25(S) × 20(C) × 1(B)	-
10	conv_3 20 10×10 convolutions with stride [1 1] a...	2-D Convolution	25(S) × 25(S) × 20(C) × 1(B)	Wei... 10 × 10 × 2... Bias 1 × 1 × 20
11	batchnorm_3 Batch normalization	Batch Normalization	25(S) × 25(S) × 20(C) × 1(B)	Offset 1 × 1 × 20 Scale 1 × 1 × 20
12	relu_3 ReLU	ReLU	25(S) × 25(S) × 20(C) × 1(B)	-
13	maxpool_3 2×2 max pooling with stride [2 2] and pa...	2-D Max Pooling	12(S) × 12(S) × 20(C) × 1(B)	-
14	dropout 30% dropout	Dropout	12(S) × 12(S) × 20(C) × 1(B)	-
15	fc 4 fully connected layer	Fully Connected	1(S) × 1(S) × 4(C) × 1(B)	Weights 4 × 2880 Bias 4 × 1
16	softmax softmax	Softmax	1(S) × 1(S) × 4(C) × 1(B)	-
17	classoutput crossentropyex	Classification Output	1(S) × 1(S) × 4(C) × 1(B)	-

12.2.2 Baseline Model Training Information

Epoch	Iteration	Time Elapsed	Mini-batch	Validation	Mini-batch	Validation	Base Learning
		(hh:mm:ss)	Accuracy	Accuracy	Loss	Loss	Rate
1	1	00:00:03	33.33%	19.87%	1.7674	2.7994	0.002
1	10	00:00:09	56.67%	69.29%	1.145	0.9132	0.002
1	20	00:00:14	56.67%	59.61%	1.0934	0.9095	0.002
1	30	00:00:20	73.33%	75.70%	0.6862	0.7689	0.002
1	40	00:00:26	76.67%	75.21%	0.7855	0.7312	0.002
2	50	00:00:32	76.67%	76.03%	0.9379	0.7293	0.002
2	60	00:00:37	70.00%	77.01%	0.6379	0.6312	0.002
2	70	00:00:43	70.00%	74.38%	1.1144	0.7972	0.002
2	80	00:00:48	70.00%	69.29%	0.8518	0.7555	0.002
2	90	00:00:54	86.67%	77.18%	0.4933	0.6794	0.002
3	100	00:01:00	66.67%	74.06%	0.6971	0.683	0.002

3	110	00:01:06	63.33%	77.83%	0.7729	0.6721	0.002
3	120	00:01:11	76.67%	76.35%	0.6538	0.6438	0.002
3	130	00:01:18	73.33%	75.70%	0.5868	0.6416	0.002
3	140	00:01:24	90.00%	79.64%	0.2351	0.6094	0.002
4	150	00:01:30	80.00%	81.12%	0.6379	0.5825	0.002
4	160	00:01:36	83.33%	80.13%	0.6229	0.5668	0.002
4	170	00:01:42	86.67%	81.44%	0.4648	0.5462	0.002
4	180	00:01:49	96.67%	79.47%	0.1895	0.6518	0.002
5	190	00:01:55	86.67%	80.62%	0.3462	0.5437	0.002
5	200	00:02:02	80.00%	82.59%	0.4278	0.5041	0.002
5	210	00:02:08	90.00%	82.43%	0.3329	0.4777	0.002
5	220	00:02:13	90.00%	79.97%	0.1813	0.5662	0.002
5	230	00:02:19	80.00%	78.82%	0.3831	0.535	0.002
6	240	00:02:24	80.00%	83.09%	0.5161	0.4853	0.002
6	250	00:02:29	86.67%	79.15%	0.4505	0.5505	0.002
6	260	00:02:35	83.33%	82.59%	0.5967	0.4759	0.002
6	270	00:02:40	96.67%	82.43%	0.1399	0.4587	0.002
6	280	00:02:45	100.00%	82.43%	0.0573	0.4962	0.002
7	290	00:02:51	86.67%	84.89%	0.3328	0.4481	0.002
7	300	00:02:57	86.67%	85.39%	0.3181	0.4476	0.002
7	310	00:03:03	90.00%	82.59%	0.2931	0.5215	0.002
7	320	00:03:09	83.33%	79.47%	0.4706	0.5391	0.002
8	330	00:03:15	90.00%	78.49%	0.3133	0.6179	0.002
8	340	00:03:20	83.33%	86.04%	0.4819	0.4447	0.002
8	350	00:03:26	90.00%	84.73%	0.4724	0.4633	0.002
8	360	00:03:32	93.33%	84.07%	0.1773	0.4521	0.002
8	370	00:03:38	83.33%	82.10%	0.4148	0.5383	0.002
9	380	00:03:44	100.00%	83.09%	0.1079	0.462	0.002
9	390	00:03:49	93.33%	86.37%	0.1898	0.408	0.002
9	400	00:03:55	76.67%	79.97%	0.5288	0.5583	0.002
9	410	00:04:01	93.33%	84.89%	0.1918	0.4462	0.002
9	420	00:04:07	90.00%	83.58%	0.3327	0.4333	0.002
10	430	00:04:13	90.00%	85.06%	0.2392	0.4424	0.002
10	440	00:04:19	93.33%	84.40%	0.1716	0.449	0.002
10	450	00:04:26	86.67%	82.27%	0.4116	0.5486	0.002
10	460	00:04:32	93.33%	85.06%	0.1315	0.4448	0.002
10	470	00:04:39	96.67%	85.71%	0.0814	0.4501	0.002
11	480	00:04:46	90.00%	85.06%	0.2154	0.4028	0.002
11	490	00:04:52	93.33%	86.04%	0.2591	0.4152	0.002
11	500	00:04:58	83.33%	84.40%	0.276	0.496	0.002
11	510	00:05:04	90.00%	86.86%	0.2419	0.4073	0.002
12	520	00:05:10	100.00%	85.71%	0.06	0.4244	0.002
12	530	00:05:16	86.67%	85.71%	0.2617	0.4391	0.002
12	540	00:05:21	100.00%	85.71%	0.0776	0.4242	0.002
12	550	00:05:27	96.67%	84.24%	0.1689	0.4401	0.002
12	560	00:05:32	96.67%	87.19%	0.1054	0.4447	0.002

12	564	00:05:36	90.00%	85.39%	0.3027	0.4529	0.002

12.2.3 Grid layer Search, Optimal Model Training Information

Epoch	Iteration	Time Elapsed	Mini-batch	Validation	Mini-batch	Validation	Base Learning
		(hh:mm:ss)	Accuracy	Accuracy	Loss	Loss	Rate
1	1	00:00:02	26.67%	44.01%	1.7337	1.4662	0.002
1	10	00:00:04	80.00%	68.14%	0.6685	0.784	0.002
1	20	00:00:07	80.00%	73.23%	0.5535	0.6972	0.002
1	30	00:00:09	93.33%	81.12%	0.4208	0.5261	0.002
1	40	00:00:11	73.33%	83.42%	0.9515	0.4392	0.002
2	50	00:00:14	93.33%	84.56%	0.329	0.4072	0.002
2	60	00:00:16	93.33%	86.37%	0.3023	0.3944	0.002
2	70	00:00:18	90.00%	87.52%	0.3121	0.3773	0.002
2	80	00:00:21	90.00%	85.39%	0.2379	0.3975	0.002
2	90	00:00:23	96.67%	83.74%	0.1665	0.4207	0.002
3	100	00:00:25	96.67%	87.19%	0.2593	0.3655	0.002
3	110	00:00:28	83.33%	87.68%	0.458	0.3577	0.002
3	120	00:00:30	90.00%	87.36%	0.3833	0.3546	0.002
3	130	00:00:33	93.33%	86.54%	0.2101	0.3639	0.002
3	140	00:00:35	86.67%	87.85%	0.5336	0.342	0.002
4	150	00:00:37	96.67%	87.19%	0.1099	0.3545	0.002
4	160	00:00:40	90.00%	87.36%	0.2392	0.3875	0.002
4	170	00:00:42	100.00%	85.71%	0.1183	0.3772	0.002
4	180	00:00:45	96.67%	87.36%	0.0728	0.3674	0.002
5	190	00:00:47	93.33%	82.59%	0.1824	0.465	0.002
5	200	00:00:49	90.00%	85.88%	0.2021	0.3747	0.002
5	210	00:00:52	90.00%	88.34%	0.1381	0.35	0.002
5	220	00:00:54	83.33%	84.40%	0.3717	0.4189	0.002
5	230	00:00:57	96.67%	87.19%	0.1261	0.3404	0.002
6	240	00:00:59	96.67%	87.03%	0.0462	0.4173	0.002
6	250	00:01:02	96.67%	88.51%	0.0976	0.3581	0.002
6	260	00:01:04	93.33%	86.21%	0.1426	0.3862	0.002
6	270	00:01:06	93.33%	86.54%	0.1412	0.3995	0.002
6	280	00:01:09	100.00%	88.34%	0.1149	0.3465	0.002
7	290	00:01:11	100.00%	88.18%	0.0672	0.3461	0.002
7	300	00:01:14	96.67%	88.34%	0.068	0.3522	0.002
7	310	00:01:16	100.00%	86.86%	0.0641	0.3673	0.002
7	320	00:01:19	90.00%	87.36%	0.2415	0.3746	0.002
8	330	00:01:21	86.67%	87.85%	0.2843	0.3475	0.002
8	340	00:01:24	100.00%	88.34%	0.0203	0.3444	0.002
8	350	00:01:26	93.33%	86.86%	0.1887	0.3687	0.002
8	360	00:01:29	100.00%	88.34%	0.0717	0.3319	0.002
8	370	00:01:31	100.00%	88.18%	0.0363	0.3787	0.002

9	380	00:01:33	96.67%	87.68%	0.0658	0.3956	0.002
9	390	00:01:36	100.00%	85.06%	0.0526	0.3922	0.002
9	400	00:01:39	100.00%	89.00%	0.0476	0.3831	0.002
9	410	00:01:41	96.67%	85.55%	0.071	0.4001	0.002
9	420	00:01:44	96.67%	86.86%	0.0918	0.3462	0.002
10	430	00:01:46	100.00%	89.66%	0.0199	0.3345	0.002
10	440	00:01:49	100.00%	88.01%	0.0696	0.3443	0.002
10	450	00:01:52	96.67%	86.70%	0.0731	0.379	0.002
10	460	00:01:54	100.00%	87.03%	0.0466	0.4407	0.002
10	470	00:01:57	100.00%	87.85%	0.0761	0.368	0.002
11	480	00:02:00	100.00%	88.51%	0.0348	0.3524	0.002
11	490	00:02:03	100.00%	87.68%	0.0204	0.4052	0.002
11	500	00:02:05	100.00%	86.21%	0.0089	0.4245	0.002
11	510	00:02:08	100.00%	87.36%	0.0729	0.4121	0.002
12	520	00:02:11	100.00%	88.01%	0.0206	0.3786	0.002
12	530	00:02:14	100.00%	88.18%	0.0188	0.3763	0.002
12	540	00:02:17	96.67%	86.21%	0.0786	0.4077	0.002
12	550	00:02:20	100.00%	88.01%	0.0298	0.4127	0.002
12	560	00:02:23	100.00%	86.54%	0.0386	0.4193	0.002
12	564	00:02:25	96.67%	87.85%	0.0845	0.4913	0.002

12.2.4 Bayesian Optimisation, Optimal Model Training Information

Epoch	Iteration	Time Elapsed	Mini-batch	Validation	Mini-batch	Validation	Base Learning
		(hh:mm:ss)	Accuracy	Accuracy	Loss	Loss	Rate
1	1	00:00:03	13.33%	43.51%	1.955	1.3692	0.002
1	10	00:00:08	93.33%	78.00%	0.6033	0.6724	0.002
1	20	00:00:12	93.33%	83.42%	0.4165	0.5218	0.002
1	30	00:00:15	80.00%	84.24%	0.4693	0.467	0.002
1	40	00:00:19	93.33%	82.27%	0.2365	0.4576	0.002
2	50	00:00:23	90.00%	84.24%	0.3086	0.461	0.002
2	60	00:00:27	83.33%	81.77%	0.3195	0.5111	0.002
2	70	00:00:30	86.67%	88.01%	0.4444	0.3762	0.002
2	80	00:00:34	86.67%	89.33%	0.3898	0.3367	0.002
2	90	00:00:38	93.33%	88.18%	0.2165	0.3147	0.002
3	100	00:00:42	90.00%	89.16%	0.2701	0.3312	0.002
3	110	00:00:47	96.67%	88.67%	0.2326	0.3279	0.002
3	120	00:00:51	76.67%	90.15%	0.5134	0.2843	0.002
3	130	00:00:55	90.00%	90.97%	0.1841	0.2606	0.002
3	140	00:00:59	90.00%	90.97%	0.3701	0.2563	0.002
4	150	00:01:03	86.67%	90.80%	0.2485	0.2632	0.002
4	160	00:01:07	90.00%	89.98%	0.3041	0.2731	0.002
4	170	00:01:11	93.33%	91.13%	0.2242	0.2449	0.002
4	180	00:01:15	90.00%	90.97%	0.155	0.2476	0.002

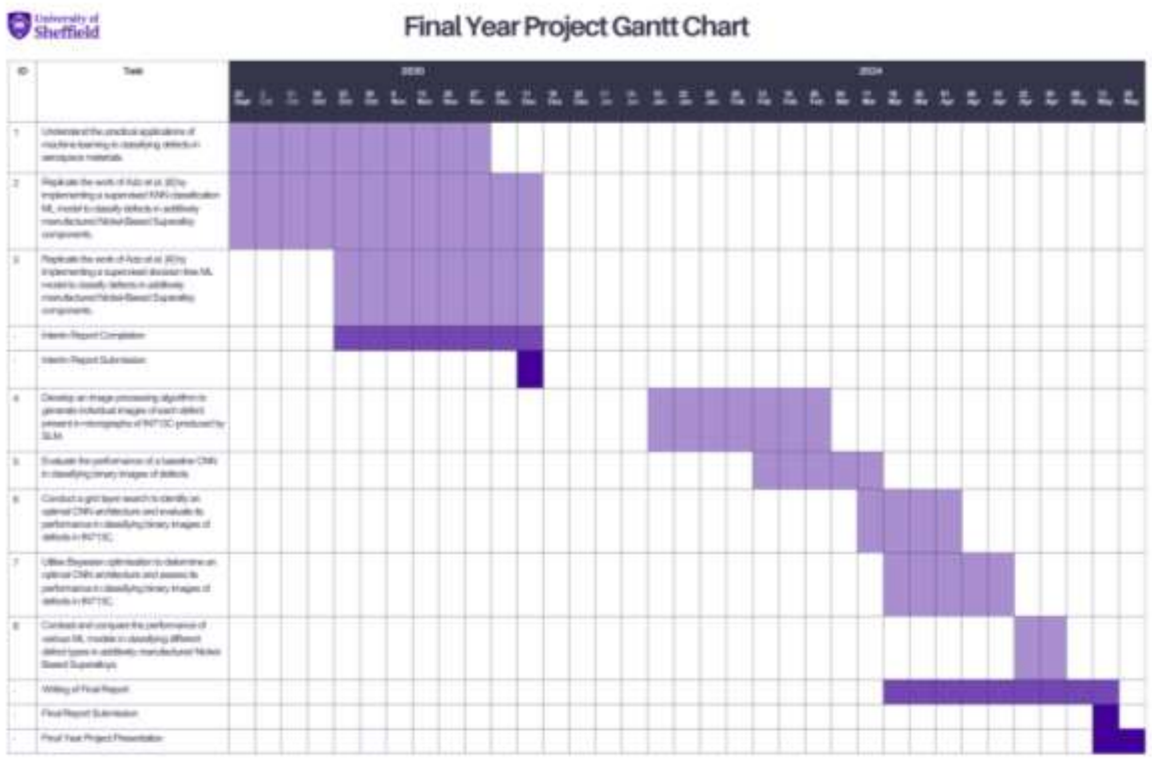
5	190	00:01:18	96.67%	92.94%	0.1493	0.2364	0.002
5	200	00:01:22	83.33%	92.28%	0.3951	0.2255	0.002
5	210	00:01:25	86.67%	93.92%	0.2603	0.2027	0.002
5	220	00:01:28	83.33%	91.79%	0.5367	0.2556	0.002
5	230	00:01:32	90.00%	92.61%	0.2199	0.2172	0.002
6	240	00:01:35	100.00%	92.28%	0.0574	0.2173	0.002
6	250	00:01:38	83.33%	92.28%	0.335	0.2162	0.002
6	260	00:01:41	86.67%	92.28%	0.205	0.2342	0.002
6	270	00:01:44	90.00%	93.10%	0.2014	0.2187	0.002
6	280	00:01:47	93.33%	93.92%	0.1505	0.1923	0.002
7	290	00:01:50	86.67%	93.10%	0.3554	0.2029	0.002
7	300	00:01:53	86.67%	90.80%	0.2766	0.2386	0.002
7	310	00:01:57	93.33%	93.92%	0.139	0.204	0.002
7	320	00:02:00	96.67%	93.60%	0.0604	0.1893	0.002
8	330	00:02:04	93.33%	94.25%	0.2078	0.1717	0.002
8	340	00:02:07	96.67%	94.25%	0.0939	0.1612	0.002
8	350	00:02:10	96.67%	95.73%	0.1071	0.1479	0.002
8	360	00:02:14	96.67%	94.75%	0.1213	0.1613	0.002
8	370	00:02:18	96.67%	94.25%	0.0856	0.1574	0.002
9	380	00:02:21	100.00%	93.27%	0.0781	0.195	0.002
9	390	00:02:25	96.67%	94.91%	0.0575	0.179	0.002
9	400	00:02:28	100.00%	93.76%	0.055	0.1807	0.002
9	410	00:02:32	96.67%	94.42%	0.1712	0.1797	0.002
9	420	00:02:35	93.33%	94.75%	0.1947	0.1564	0.002
10	430	00:02:39	93.33%	95.40%	0.1549	0.1454	0.002
10	440	00:02:43	93.33%	95.40%	0.1166	0.1447	0.002
10	450	00:02:46	96.67%	95.40%	0.1329	0.1295	0.002
10	460	00:02:50	93.33%	96.06%	0.1093	0.1426	0.002
10	470	00:02:54	100.00%	95.73%	0.0193	0.1156	0.002
11	480	00:02:58	100.00%	95.07%	0.011	0.1506	0.002
11	490	00:03:02	100.00%	95.07%	0.0633	0.1516	0.002
11	500	00:03:06	100.00%	94.75%	0.0097	0.1742	0.002
11	510	00:03:10	96.67%	93.92%	0.0398	0.196	0.002
12	520	00:03:14	93.33%	94.75%	0.0832	0.1476	0.002
12	530	00:03:19	100.00%	94.25%	0.0111	0.1659	0.002
12	540	00:03:23	93.33%	95.57%	0.1229	0.1482	0.002
12	550	00:03:28	96.67%	95.73%	0.0895	0.1409	0.002
12	560	00:03:32	100.00%	95.07%	0.0183	0.138	0.002
12	564	00:03:35	90.00%	97.04%	0.2358	0.1034	0.002

12.3 Project Management

12.3.1 Original Aims and Objectives

ID	Objective
1	Understand the practical applications of machine learning in classifying defects in aerospace materials.
2	Replicate the work of Aziz et al. [4] by implementing a supervised KNN classification ML model to classify defects in additively manufactured Nickel-Based Superalloy components.
3	Replicate the work of Aziz et al. [4] by implementing a supervised decision tree ML model to classify defects in additively manufactured Nickel-Based Superalloy components.
4	Understand the structure of abradable coatings and understand what defects are present
5	Procure data for defects present in abradable coatings to be used in a machine learning algorithm
6	Use a supervised and unsupervised machine learning model to classify the defects present in abradable coatings
7	Evaluate the accuracy of both models at classifying defects present in abradable coatings
8	Evaluate the precision of machine learning techniques in defect classification against presently employed methods

12.3.2 Updated Gantt Chart



12.4 Code

12.4.1 Supervised Machine Learning Methods

```
close all;
```

Data Collection & Formatting

```
ds = readtable("defect_master_data.csv"); % Get dataset

ds = removevars(ds,{'id'}); % Remove the ID column as this isn't necessary

ds % Show dataset
```

Data Analysis

Show 3D plots of the data

```
ds_minMax = minMaxNorm(ds);

ds_minMax_crack = ds_minMax(ds_minMax.defect_type == "Crack", :);

ds_minMax_pore = ds_minMax(ds_minMax.defect_type == "Pore", :);

ds_minMax_pwc = ds_minMax(ds_minMax.defect_type == "Pore with Crack", :);

ds_minMax_lof = ds_minMax(ds_minMax.defect_type == "Lack of Fusion", :);



figure
scatter3(ds_minMax_crack, "roundness", "circularity", "feret_diameter",
"filled");

hold on

scatter3(ds_minMax_pore, "roundness", "circularity", "feret_diameter",
"filled");

scatter3(ds_minMax_pwc, "roundness", "circularity", "feret_diameter",
"filled");

scatter3(ds_minMax_lof, "roundness", "circularity", "feret_diameter",
"filled");



legend(["Crack", "Pore", "Pore with Crack", "Lack of Fusion"])
```

Data Preparation

Creating training and test datasets

```
% Create training data for zScore and minmax
```

```
[ds_train, ds_test] = createData(ds);

ds_zScore_train = zScoreNorm(ds_train);

ds_minMax_train = minMaxNorm(ds_train);

ds_zScore_test = zScoreNorm(ds_test);

ds_minMax_test = minMaxNorm(ds_test);
```

Finding Optimal K Value

Finding the optimal K value for all datasets (zScore, Min Max and no normalisation)

```
maxK = 40; % Define maximum n value

knnAccuracy = zeros(3,maxK); % Preallocate memory to hold accuracy readings
for each



for i = 1:maxK


    knnModel = fitcknn(ds_train{:, 1:end-1}, ds_train{:, end},
"NumNeighbors", i);
    knnModel_zScore = fitcknn(ds_zScore_train{:, 1:end-1},
ds_zScore_train{:, end}, "NumNeighbors", i);

    knnModel_minMax = fitcknn(ds_minMax_train{:, 1:end-1},
ds_minMax_train{:, end}, "NumNeighbors", i);



    predicted_labels = predict(knnModel, ds_test{:, 1:end-1});

    predicted_labels_zScore = predict(knnModel_zScore, ds_zScore_test{:,
1:end-1});

    predicted_labels_minMax = predict(knnModel_minMax, ds_minMax_test{:,
1:end-1});



    knnAccuracy(1, i) = sum(strcmp(predicted_labels,
ds_test{:,end}))/numel(ds_test{:,end});
    knnAccuracy(2, i) = sum(strcmp(predicted_labels_zScore,
ds_test{:,end}))/numel(ds_test{:,end});

    knnAccuracy(3, i) = sum(strcmp(predicted_labels_minMax,
ds_test{:,end}))/numel(ds_test{:,end});



end
```

```
avgNoNorm = mean(knnAccuracy(1,:));
avgZScore = mean(knnAccuracy(2,:));

avgMinMax = mean(knnAccuracy(3,:));



fprintf("The average accuracy for the model without normalisation is " +
avgNoNorm*100)
fprintf("The average accuracy for the model with zScore is " +
avgZScore*100)

fprintf("The average accuracy for the model with minMax is " +
avgMinMax*100)

fprintf("The maximum accuracy for the model without normalisation is " +
max(knnAccuracy(1,:))*100)

fprintf("The maximum accuracy for the model with zScore is " +
max(knnAccuracy(2,:))*100)

fprintf("The maximum accuracy for the model with minMax is " +
max(knnAccuracy(3,:))*100)

% Plot the variation of accuracy with K nearest neighbor for each

% normalised data set

figure

plot(knnAccuracy(1,:));

hold on

plot(knnAccuracy(2,:));

plot(knnAccuracy(3,:));

hold off

xlabel("Number of K nearest Neighbors");

ylabel("Accuracy");

legend(["No normalisation", "Z-Score", "Min-Max"], Location="southeast");

ylim([0 1])

grid on

xlim([1 maxK])

saveas(gcf,'K-variation.png')

[maxAccuracyKNN, maximumIndexKNN] = max(knnAccuracy(2,:));

disp("The maximum accuracy observed is " + maxAccuracyKNN + " for Z score
normalisation when K is equal to " + maximumIndexKNN);
```

Statistical Information

```
% Number of iterations to run

%

i_max = 500;


kNN_accuracyData = zeros(i_max, 3); % Preallocating memory space to hold
accuracy for kNN Classification algorithm
for i = 1:i_max


    % Randomly create train and test datasets for no normalisation, z-score
    % and min-max

    [ds_train_iter, ds_test_iter] = createData(ds);

    ds_zScore_train_iter = zScoreNorm(ds_train_iter);

    ds_minMax_train_iter = minMaxNorm(ds_train_iter);

    ds_zScore_test_iter = zScoreNorm(ds_test_iter);

    ds_minMax_test_iter = minMaxNorm(ds_test_iter);


    maxK = 20; % Define maximum k value
    knnAccuracy = zeros(3,maxK); % Preallocate memory to hold accuracy
readings for each


    for j = 1:maxK


        % Train kNN classification model given k value
        knnModel_iter = fitcknn(ds_train_iter{:, 1:end-1},
ds_train_iter{:, end}, "NumNeighbors", j);

        knnModel_zScore_iter = fitcknn(ds_zScore_train_iter{:, 1:end-1},
ds_zScore_train_iter{:, end}, "NumNeighbors", j);

        knnModel_minMax_iter = fitcknn(ds_minMax_train_iter{:, 1:end-1},
ds_minMax_train_iter{:, end}, "NumNeighbors", j);


        % Predict the labels of test dataset
        predicted_labels_iter = predict(knnModel_iter, ds_test_iter{:,
1:end-1});

        predicted_labels_zScore_iter = predict(knnModel_zScore_iter,
ds_zScore_test_iter{:, 1:end-1});
```

```
        predicted_labels_minMax_iter = predict(knnModel_minMax_iter,
ds_minMax_test_iter{:, 1:end-1});


        knnAccuracy(1, i) = sum(strcmp(predicted_labels_iter,
ds_test_iter{:,end}))/numel(ds_test_iter{:,end});
        knnAccuracy(2, i) = sum(strcmp(predicted_labels_zScore_iter,
ds_test_iter{:,end}))/numel(ds_test_iter{:,end});

        knnAccuracy(3, i) = sum(strcmp(predicted_labels_minMax_iter,
ds_test_iter{:,end}))/numel(ds_test_iter{:,end});


    end


    % Save accuracy data to matrix
    kNN_accuracyData(i, :) = [max(knnAccuracy(1,:)) max(knnAccuracy(2,:))
max(knnAccuracy(3,:))];

    fprintf(kNN_accuracyData(i, 1) + " " + kNN_accuracyData(i, 2) + " " +
kNN_accuracyData(i, 3) + " ... Iteration " + i + " complete...\n")


end
% Reveal the mean and standard deviation of accuracies for different

% normalisation methods

accuracyStats = [mean(kNN_accuracyData(:,1)) std(kNN_accuracyData(:,1));
mean(kNN_accuracyData(:,2)) std(kNN_accuracyData(:,2));
mean(kNN_accuracyData(:,3)) std(kNN_accuracyData(:,3))]

figure

boxplot([kNN_accuracyData(:,1) kNN_accuracyData(:,2)
kNN_accuracyData(:,3)],'Labels',{'No Normalisation','z-Score', 'Min-
max'});

xlabel('Normalisation Method')

ylabel('Accuracy')

set(gca, 'XGrid','on')

saveas(gcf,'KNNBoxPlot.png')

%}
```

kNN Classification Algorithms

```
% Develop confusion matrix for no normalisation, using KNN

knnModel_noNorm_optimal = fitcknn(ds_train{:, 1:end-1}, ds_train{:, end},
"NumNeighbors", maximumIndexKNN);

knnPredlabels_noNorm = predict(knnModel_noNorm_optimal, ds_test{:, 1:end-
1});

% Plot confusion matrix

figure;

confusion_KNNoptimal_noNorm = confusionmat(ds_test{:, end},
knnPredlabels_noNorm);

confusionchart(ds_test{:, end}, knnPredlabels_noNorm);

title("KNN Classification: No Normalisation, K=" + maximumIndexKNN)

saveas(gcf,'KNNClassificationNoNorm.png')


% Develop confusion matrix for zScore normalisation, using KNN
knnModel_zScore_optimal = fitcknn(ds_zScore_train{:, 1:end-1},
ds_zScore_train{:, end}, "NumNeighbors", maximumIndexKNN);

knnPredlabels_zScore = predict(knnModel_zScore_optimal, ds_zScore_test{:,
1:end-1});

% Plot confusion matrix

figure;

confusion_KNNoptimal = confusionmat(ds_test{:, end},
knnPredlabels_zScore);

confusionchart(ds_test{:, end}, knnPredlabels_zScore);

title("KNN Classification: Z-Score, K=" + maximumIndexKNN)

saveas(gcf,'KNNClassificationZScore.png')

% Develop confusion matrix for minMax normalisation, using KNN when

knnModel_minMax_optimal = fitcknn(ds_minMax_train{:, 1:end-1},
ds_minMax_train{:, end}, "NumNeighbors", maximumIndexKNN);

knnPredlabels_minMax = predict(knnModel_minMax_optimal, ds_minMax_test{:,
1:end-1});

% Plot confusion matrix

figure;

confusion_KNNoptimal_minMax = confusionmat(ds_test{:, end},
knnPredlabels_minMax);

confusionchart(ds_test{:, end}, knnPredlabels_minMax);

title("KNN Classification: Min Max, K=" + maximumIndexKNN)
```

```
saveas(gcf,'KNNClassificationMinMax.png')
```

Decision Trees

Comparing the accuracy of decision trees for various normalisation methods.

```
% Training the model for each

dTreeModel = fitctree(ds_train{:, 1:end-1}, ds_train{:, end});

dTreeModel_zScore = fitctree(ds_zScore_train{:, 1:end-1},
ds_zScore_train{:, end});

dTreeModel_minMax = fitctree(ds_minMax_train{:, 1:end-1},
ds_minMax_train{:, end});



% Predicting what each defect is
predicted_labels = predict(dTreeModel, ds_test{:, 1:end-1});

predicted_labels_zScore = predict(dTreeModel_zScore, ds_zScore_test{:,
1:end-1});

predicted_labels_minMax = predict(dTreeModel_minMax, ds_minMax_test{:,
1:end-1});



% Create a confusion matrix for each normalisation method
confusionMat_dTree_noNormal = confusionmat(ds_test{:, end},
predicted_labels);

confusionMat_dTree_zScore = confusionmat(ds_zScore_test{:, end},
predicted_labels_zScore);

confusionMat_dTree_minMax = confusionmat(ds_minMax_test{:, end},
predicted_labels);



figure
confusionchart(ds_test{:, end}, predicted_labels)

title("Decision Tree: No Normalisation")

saveas(gcf,'DecisionTreeNoNorm.png')

figure

confusionchart(ds_zScore_test{:, end}, predicted_labels_zScore)

similarity = sum(strcmp(ds_zScore_test{:, end},
predicted_labels_zScore))/numel(predicted_labels_zScore)

title("Decision Tree: z-Score")
```

```
saveas(gcf,'DecisionTreeZScore.png')

figure

confusionchart(ds_minMax_test{:, end}, predicted_labels)

title("Decision Tree: Min Max")

saveas(gcf,'DecisionTreeMinMax.png')
```

Decision Tree Variability Study

```
dTreeAccuracy = zeros(i_max, 3);



for i = 1:i_max


    [ds_train_iter, ds_test_iter] = createData(ds);
    ds_zScore_train_iter = zScoreNorm(ds_train_iter);

    ds_minMax_train_iter = minMaxNorm(ds_train_iter);

    ds_zScore_test_iter = zScoreNorm(ds_test_iter);

    ds_minMax_test_iter = minMaxNorm(ds_test_iter);



    % Training the model for each
    dTreeModel_iter = fitctree(ds_train_iter{:, 1:end-1}, ds_train_iter{:,
end});

    dTreeModel_zScore_iter = fitctree(ds_zScore_train_iter{:, 1:end-1},
ds_zScore_train_iter{:, end});

    dTreeModel_minMax_iter = fitctree(ds_minMax_train_iter{:, 1:end-1},
ds_minMax_train_iter{:, end});



    % Predicting what each defect is
    predicted_labels_iter = predict(dTreeModel_iter, ds_test_iter{:,
1:end-1});

    predicted_labels_zScore_iter = predict(dTreeModel_zScore_iter,
ds_zScore_test_iter{:, 1:end-1});

    predicted_labels_minMax_iter = predict(dTreeModel_minMax_iter,
ds_minMax_test_iter{:, 1:end-1});



    dTreeAccuracy(i, 1) = sum(strcmp(predicted_labels_iter,
ds_test_iter{:,end}))/numel(ds_test_iter{:,end});
```

```
    dTreeAccuracy(i, 2) = sum(strcmp(predicted_labels_zScore_iter,
ds_test_iter{:,end}))/numel(ds_test_iter{:,end});

    dTreeAccuracy(i, 3) = sum(strcmp(predicted_labels_minMax_iter,
ds_test_iter{:,end}))/numel(ds_test_iter{:,end});



    fprintf(dTreeAccuracy(i, 1) + " " + dTreeAccuracy(i, 2) + " " +
dTreeAccuracy(i, 3) + " ... Iteration " + i + " complete...\n")
end

accuracyStatsDTree = [mean(dTreeAccuracy(:,1)) std(dTreeAccuracy(:,1));
mean(dTreeAccuracy(:,2)) std(dTreeAccuracy(:,2)); mean(dTreeAccuracy(:,3))
std(dTreeAccuracy(:,3))]

figure

boxplot([dTreeAccuracy(:,1) dTreeAccuracy(:,2)
dTreeAccuracy(:,3)],'Labels',{'No Normalisation','z-Score', 'Min-max'});

grid on

xlabel('Normalisation Method')

ylabel('Accuracy')

saveas(gcf,'DTreeBoxPlot.png')
```

Functions

Function that splits a dataset into training data and test data (with even distribution in labels)

```
function [trainData, testData] = createData(table)



    trainData = array2table(zeros(0,10));
    trainData.Properties.VariableNames = table.Properties.VariableNames;

    testData = array2table(zeros(0,10));

    testData.Properties.VariableNames = table.Properties.VariableNames;



    labels = unique(table.defect_type);


    for i = 1:length(labels)


        data = table(ismember(table.defect_type, labels(i)), :);

        n = length(data.defect_type);

        cv = cvpartition(n, "HoldOut", 0.3);
        idxTrain = training(cv);
        tblTrain = data(idxTrain, :);
        
        idxTest = test(cv);
        
        tblTest = data(idxTest, :);
        
        
        trainData = [trainData; tblTrain];
        testData = [testData; tblTest];
        
        
    end
    
    
end
```

Function that normalises the data via z-Score transform

```
function ds_zScore = zScoreNorm(ds)


    [row, col] = size(ds); % Get size of numerical dataset
    ds_zScore = ds;
    
    
    % Transform the original data in zScored data
    for i = 1:col-1
    
        avg = mean(ds{:,i});
        
        sd = std(ds{:,i});
        
        for j = 1:row
        
            ds_zScore{j,i} = (ds{j,i}-avg)/sd;
            
        end
        
    end
    
    
end
```

Function

```
function ds_minMax = minMaxNorm(ds)

    % Transform the original data into Min Max data
    
    
    [row, col] = size(ds); % Get size of numerical dataset
    
    ds_minMax = ds;
    for i = 1:col-1
        minCol = min(ds{:,i});

        maxCol = max(ds{:,i});


        for j = 1:row
            ds_minMax{j,i} = (ds{j,i}-minCol)/(maxCol-minCol);

        end

    end


end
```

12.4.2 Image Segmentation Algorithm

```
function m = processImage(image_no)

    image_name = "image_";
    image_format = ".tif";
    directory = "Original Inconel 713C Images\\";


    foldername = image_name + image_no;


    % Create full name for image file to be downloaded
    full_name = directory + image_name + image_no + image_format;

    % Download and show the image
    img = imread(full_name);
    figure;
    imshow(img);


    img_gs = rgb2gray(img); % Grayscale the image
    img_gs = img_gs < 70; % Using thresholding to identify pixels with a
brightness less than that of the threshold


    [rows, cols] = size(img_gs); % Find the size of the image


    img_gs(round(rows*0.95):end , : ) = []; % Crop out the bottom 5% of the
image to remove the scale


    img_gs = imclearborder(img_gs); % Get rid of all images on the border
    img_gs = bwareafilt(img_gs, [20 inf]); % Filter out small defects with an
area of less than 20


    def_ds = regionprops(img_gs, "Image", "BoundingBox", "Area"); % Creates a
datastore containing information and an image of each defect

    % Show each image
    figure;
    imshow(img_gs);
    hold on
    % Draw a box around each defect in the filtered image
    for i = 1:length(def_ds)
        rectangle(Position=def_ds(i).BoundingBox, EdgeColor='r', LineWidth =
1)
    end
    hold off


    % Sort the datastore based on area
    def_ds_table = struct2table(def_ds)
    def_ds_table = sortrows(def_ds_table, "Area", "ascend");
    def_ds = table2struct(def_ds_table);


    fprintf("The number of defects in the image after filtering is: " +
length(def_ds) + "\n\n");


    % Iterate through each defect in the image and download them
    figure
    for i = 1:length(def_ds)

        % Show each individual defect in the micrograph
        target_def = def_ds(i).Image;
        subplot(1,1,1), imshow(target_def, "Border", "loose")


        % Create the defect image name
        defectImage_name = "unsorted_images\\" + image_name + image_no + "_" +
string(i) + ".tif";


        % Write the image to a file
        imwrite(target_def, defectImage_name);


    end


    m = 0;
    fprintf("Image " + image_no + " has been processed...\n\n")
```

```
end
```

12.4.3 Deep Learning

Load Image Data

```
% Define image folder name
imageFolder = "defect_dataset";


% Collect data from image folder
imds = imageDatastore(imageFolder, "IncludeSubfolders", true, "LabelSource",
"foldernames");


% Show the number of each type of class
tbl_count = countEachLabel(imds)

totalImgCount = sum(tbl_count{:,"Count"})


% Preview one of each class
crack_preview = find(imds.Labels == 'Crack', 1);

pore_preview = find(imds.Labels == 'Pore', 1);

poreWithCrack_preview = find(imds.Labels == 'Pore with Crack', 1);

lackOfFusion_preview = find(imds.Labels == 'Lack of Fusion', 1);


figure
subplot(2,2,1), imshow(readimage(imds,crack_preview))

subplot(2,2,2), imshow(readimage(imds,pore_preview))

subplot(2,2,3), imshow(readimage(imds,poreWithCrack_preview))

subplot(2,2,4), imshow(readimage(imds,lackOfFusion_preview))
```

Training, Validation and Test dataset creation

```
% Create training, validation, and test datasets
[imdsTrain, imdsVal] = splitEachLabel(imds, 0.7, 0.3, "randomized");


tbl_count = countEachLabel(imdsTrain)
tbl_count = countEachLabel(imdsVal)
```

Image Augmentation

```
% Specifies preprocessing options for image augmentation, such as resizing,

% rotation, translation and reflection.


% Randomly translate the images up to three pixels horizontally and
% vertically, rotate the images with an angle up to 20 degrees

% imageAugmenter = imageDataAugmenter( ...

%     RandXReflection=true, ...

%     RandYReflection=true, ...

%     RandRotation=[-180 180]);


% Define image size
imageSize = [100 100];

imdsTrain_aug = augmentedImageDatastore(imageSize, imdsTrain);

imdsVal_aug = augmentedImageDatastore(imageSize, imdsVal);


% Create array of labels for model evaluation
YVal = categorical(imdsVal.Labels);


minibatch = read(imdsTrain_aug);
imshow(imtile(minibatch.input))
```

Design Deep Neural Network (Baseline)

```
% Define constant parameters

num_classes = 4;

num_filters = 19;

filter_size = 10;


% Auxiliary parameters
aux_params{1} = num_classes;

aux_params{2} = imageSize;


% Define network layers
```

```
baseline_layers = [

    imageInputLayer([imageSize 1])


    % block 1 %
    convolution2dLayer(filter_size,num_filters,"Padding","same");

    batchNormalizationLayer;

    reluLayer;

    maxPooling2dLayer(2,'Stride',2);


    % block 2 %
    convolution2dLayer(filter_size,num_filters,"Padding","same");

    batchNormalizationLayer;

    reluLayer;

    maxPooling2dLayer(2,'Stride',2);


    % block 3 %
    convolution2dLayer(filter_size,num_filters,"Padding","same");

    batchNormalizationLayer;

    reluLayer;

    maxPooling2dLayer(2,'Stride',2);


    % block 4 %
    convolution2dLayer(filter_size,num_filters,"Padding","same");

    batchNormalizationLayer;

    reluLayer;

    maxPooling2dLayer(2,'Stride',2);


    % block 5 %
    convolution2dLayer(filter_size,num_filters,"Padding","same");

    batchNormalizationLayer;

    reluLayer;

    maxPooling2dLayer(2,'Stride',2);
    % block 6 %
    convolution2dLayer(filter_size,num_filters,"Padding","same");

    batchNormalizationLayer;

    reluLayer;

    maxPooling2dLayer(2,'Stride',2);


    % block 7 %
    convolution2dLayer(filter_size,num_filters,"Padding","same");

    batchNormalizationLayer;

    reluLayer;

    maxPooling2dLayer(2,'Stride',2);


    dropoutLayer(0.3)


    fullyConnectedLayer(num_classes)
    softmaxLayer

    classificationLayer];


% Display the network design
% analyzeNetwork(baseline_layers);


% Disable pop-up window while training
net.trainParam.showWindow = 0;


% Training options
baseline_options = trainingOptions('adam', ...

    "MiniBatchSize",30, ...

    'InitialLearnRate',0.002, ...

    'MaxEpochs',12, ...

    'Shuffle','every-epoch', ...

    'ValidationData',imdsVal_aug, ...

    'ValidationFrequency',10, ...

    'Verbose',true, ...

    'Plots','training-progress');
```

Neural Network Training

```
tic; % Start timer

disp ("Training Neural Network...")

[baseline_model, baseline_model_info] = trainNetwork(imdsTrain_aug,
baseline_layers, baseline_options)

baseline_training_time = toc % Stop timing after training

% saveas(gcf,'baselineModel_training.png')

clear tic;
```

Evaluation of baseline neural network

```
% classify the validation output using the trained network

[baseline_YPred, ~] = classify(baseline_model, imdsVal_aug);


% Plot confusion matrixa
figure;

plotconfusion(YVal, baseline_YPred);

title("Confusion Matrix: Baseline Model")

saveas(gcf,'baselineModel_eval.png')
```

Implement and evaluate effects of using 'Grid Search'

```
n = size(imdsTrain.Files, 1);

ind = randsample(1:n, totalImgCount*0.3);

subsetTrain = subset(imdsTrain, ind);


countEachLabel(subsetTrain)
subsetTrain_aug = augmentedImageDatastore(imageSize, subsetTrain);


% Define the grid of hyperparameters
num_layers = [3, 4, 5, 6]; % number of layers

num_filters = [10, 15, 20, 25, 30]; % number of filters per layer

gridsearch_best_accuracy = 0;
```

```
gridsearch_accuracy = zeros(length(num_layers), length(num_filters)); %
preallocating memory


gridsearch_options = trainingOptions('adam', ...
    "MiniBatchSize", 16, ...

    'InitialLearnRate',0.002, ...

    'MaxEpochs', 7, ...

    'Shuffle','every-epoch', ...

    'ValidationData', imdsVal_aug, ...

    'ValidationFrequency',8, ...

    'Verbose',true);
```

Design deep neural network

```
% Loop over the grid of hyperparameters

tic

for i = 1:length(num_layers)

    for j = 1:length(num_filters)

    % Current hyperparameters

    hyper_params = [num_layers(i), num_filters(j)];


    % Create and train model with current hyperparams
    gridsearch_layers = create_model(hyper_params, aux_params);


    % Train model
    str = "Number filters: " + num_filters(j) + " | " + "Number layers: " +
num_layers(i);

    disp(str);

    [model,info] = trainNetwork(subsetTrain_aug, gridsearch_layers,
gridsearch_options);

    close(findall(groot, 'Tag', 'NNET_CNN_TAININGPLOT_UIFIGURE'));


    % Extract validation accuracy for current model
    gridsearch_accuracy(i,j) = info.ValidationAccuracy(end);


    % Store parameters if they are better than previous
    if gridsearch_accuracy(i,j) > gridsearch_best_accuracy

        gridsearch_best_accuracy = gridsearch_accuracy(i, j);

        gridsearch_best_params = hyper_params;

    end

    end

end

gridsearch_training_time = toc
```

```
gridsearch_accuracy
```

Show graphically variation of accuracy with filters and layers

```
% Show best parameters

gridsearch_best_params


% Create array for strings for label in graph
gridsearch_legend = string(num_layers);


% Plot the variation of accuracy with number of filters and number of
% layers

figure;

hold on

for i = 1:length(num_layers)

    plot(num_filters, gridsearch_accuracy(i,:), "LineWidth", 1);

end

xlabel('Number of filters')

ylabel('Accuracy')

xlim([min(num_filters) max(num_filters)])

% ylim([0 max(max(gridsearch_accuracy))*1.1])

gridsearchLegend = ["3 Layers" "4 Layers" "5 Layers" "6 Layers" "7 Layers"];

legend(gridsearchLegend, Location="southeast")

grid on

hold off

saveas(gcf, "GridlayerSearchFindings.png");
```

Use best hyperparameters from performing 'Grid Search' to train the network

```
gridsearch_layers = create_model(gridsearch_best_params, aux_params); % Create
model from best hyperparameters (grid search)

tic; % Start timing before training

disp("Using the best parameters from performing grid search to train the final
grid search network.")

disp("Number of layers: " + gridsearch_best_params(1) + " | " + "Number of
filters: " + gridsearch_best_params(2));

[gridsearch_model, gridsearch_model_info] = trainNetwork(imdsTrain_aug,
gridsearch_layers, baseline_options); % Train model

gridsearch_training_time = toc; % Stop timer after training

clear tic;
```

Evaluate the best hyper parameters from performing 'Grid Search'

```
% Classify the validation output using the trained network

[gridsearch_YPred, ~] = classify(gridsearch_model, imdsVal_aug);


% Plot confusion matrix
figure;

plotconfusion(YVal, gridsearch_YPred);

title("Confusion Matrix: Gridsearch Model")

% saveas(gcf,['GridLayerSearch_OptimalTraining_6Layers_30Filters_Eval.png'])
```

Hyperparameter optimisation using Bayesian optimisation

```
% select variables to optimise

optVars = [

    optimizableVariable('Layers',[3 7],'Type','integer')

    optimizableVariable('Filters',[10 40],'Type','integer')

    ];


% create the objective function, it trains a neural network and returns the
% classification error on the validation set
```

```
ObjFcn =
makeObjFcn(imdsTrain_aug,imdsVal_aug,imdsVal.Labels,baseline_options,aux_param
s);



% Start timer
tic


% perform bayesian optimisation by minimising the classification error on
% the validation set

BayesObject = bayesopt(ObjFcn,optVars, ...

    'MaxTime',14*60*60, ...

    'IsObjectiveDeterministic',false, ...

    'UseParallel',false, ...

    'MaxObjectiveEvaluations',15);

bayesian_training_time = toc;

clear toc;
```

```
% load best network found in the optimisation
bestIdx = BayesObject.IndexOfMinimumTrace(end);

fileName = BayesObject.UserDataTrace{bestIdx}

fileName = "0.093596.mat"

bayesStruct = load(fileName)
```

Evaluate best hyperparameters from Bayesian Optimisation

```
% classify the validation output using the trained network

[bayes_YPred,~] = classify(bayesStruct.trainedNet,imdsVal_aug);


% plot confusion matrix
figure;

plotconfusion(YVal,bayes_YPred)

title("Confusion Matrix: Bayesian Optimization")
```

```
% accuracy in percent
bayes_accuracy = 100*sum(bayes_YPred == YVal)/numel(YVal);

disp("Basyesian optimization network accuracy is: " + num2str(bayes_accuracy)
+ "%");

saveas(gcf, "BayesianOptimisation_Eval.png");
```

Functions

```
function layers = create_model(hyper_params, aux_params)

    % Unpack hyperparameter values under test

    num_layers = hyper_params(1);

    num_filters = hyper_params(2);


    % Unpack auxiliary parameters needed to build network
    num_classes = aux_params{1};

    imageSize = aux_params{2};


    % create input layer
    layers = [

        imageInputLayer([imageSize 1])];


    % create blocks of conv -> batch norm -> relu -> max pool layers
    for i = 1:(num_layers-1)

        layers = [layers

            convolution2dLayer(3, num_filters, "Padding", "same")

            batchNormalizationLayer

            reluLayer

            maxPooling2dLayer(2,"Stride",2,"Padding","same")

        ];

    end


    % output layers
    layers = [layers


    convolution2dLayer(3, num_filters, "Padding", "same")

    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2, "Stride", 1, "Padding", "same")


    fullyConnectedLayer(num_classes)

    softmaxLayer

    classificationLayer];

end
```

Bayesian Optimisation Function

```
function ObjFcn = makeObjFcn(dsTrain,dsVal,YVal_Pred,options,aux_params)

%OBJFCN objective function used in Bayesian Optimisation

ObjFcn = @valErrorFun;

    function [valError,cons,fileName] = valErrorFun(optVars)

        hyper_params = [optVars.Layers optVars.Filters];


        % create layers using the bayesian optimised hyperparameters
        layers = create_model(hyper_params,aux_params);


        tic;    % start the timer before training

        % train network

        trainedNet = trainNetwork(dsTrain,layers,options);

        close(findall(groot,'Tag','NNET_CNN_TRAININGPLOT_UIFIGURE'))

        bayes_training_time = toc;    % Stop the timer after training

        clear tic;


        % classify the validation output using the trained network
        [YPredicted,~] = classify(trainedNet,dsVal);


        % calculate the validation error
        valError = 1 - mean(YPredicted == YVal_Pred);


        % save the trained network as a '.mat' file using the validation

        % error as the filename

        fileName = num2str(valError) + ".mat";
        save(fileName,'trainedNet','valError','options',...
            'bayes_training_time');
        cons=[];
    end
end
```