Seeing the Unseen

Applications of Machine Learning with Ultrasound Data for Non-Destructive Testing

By

RAYMOND WONG

Department of Engineering Mathematics
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of MASTER OF SCIENCE in the Faculty of Engineering.

SEPTEMBER 2021

Abstract

Non-destructive testing (NDT) is a technique that involves evaluating materials and structures without damaging them. This is highly desired as NDT is capable of reducing costs and saving materials, where the most common application of NDT is for flaw detection. However, there is also a huge desire to look inside objects that are optically opaque for analysis; although this can be achieved with ultrasound tomography, current methods suffer from disadvantages such as long computation times and the incapability to provide real-time imaging, a desired feature in NDT. As both tasks likely consist of patterns, any feature recognition tasks are an ideal problem that can potentially be solved by machine learning (ML). Reduction in inspection times or potentially providing real-time inspection inside of a material may be possible by introducing ML. Thus, in this report, a flaw classification model is proposed using convolutional neural networks (CNN) with highly accurate predictions. However, the proposed method is susceptible to noisy data; thus, further improvements should be made but the idea of ML should not be dismissed. Furthermore, another CNN model has been employed to predict orientation maps within materials. The results show the proposed method is capable of achieving accurate results even under the influence of some noisy input data. However, the performance of the model significantly decreased as the complexity of the inside structure is increased. As real-life materials are typically more complex polycrystalline structures, the current method is incapable of reconstructing such complex materials to high accuracy; such that, the proposed model is incapable of such tasks. As the method is capable of reconstructing simpler material maps, this implies further improvements to the proposed method may increase the accuracy of more complex models. Thus, suggesting that by introducing ML, real-time inspections may become viable in the future upon further work.

Dedication and acknowledgements

I would like to express my thanks to Professor Anthony Mulholland for the constant supervision and support for both my thesis and career. Furthermore, I would also like to thank Dr. Luca Giuggioli and all the other staff for maintaining the high standards of the university for this programme regardless of the current situation. Finally, I want to thank Professor Alan Champneys for his motivational talks which have inspired me with new perspectives, allowing me to venture into my next career.

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: RAYMOND WONG DATE: 15/09/2021

Table of Contents

CHAPTER 1

INTRODUCTION

There is a huge desire to look inside objects that are optically opaque and to achieve this imaging techniques are often used. These techniques have a broad range of uses in areas such as the medical imaging field and seismology. Typical examples of applications consist of early stages of breast cancer diagnosis and seismic explorations for seeking hydrocarbon. This can be achieved by using waves, typically through the means of utilising the time between the transmitting and receiving of a signal from a transducer to create an image of the inside of a medium [27, 33]. This method of utilising waves for imaging is called computed tomography and was first invented and implemented by Godfrey Hounsfield and Allan McLeod Cormack when developing X-ray tomography in 1972, later awarded a Nobel prize in 1979 for this [25, 51]. However, the actual idea of reconstruction from data was invented by Johann Radon in 1917 [54].

A common wave that is used for imaging techniques is ultrasound. Ultrasound can be utilised for many applications such as looking inside the Earth, humans and even testing in engineering industries [22, 27, 47]. However, as the current quality of ultrasound tomographic reconstructions does not meet the accuracy that end-users require, there is a huge desire to progress in this area. In addition to the mentioned applications, ultrasound waves are also being increasingly applied in the field of engineering for non-destructive testing (NDT). Inspections within NDT typically consist of utilising the penetrating properties of waves and capturing the response of a material to the stimulated wave, where these inspections are typically capable of detecting defects [3, 16, 48]. This technique uses time of flight (ToF), where the principle is reliant on defects possessing a different acoustic impedance; such that, the ToF will differ in comparison to detecting a surface without defects [35, 54]. More recent research has been undertaken for tomography methods with ultrasound by image reconstruction, showing great potential for progression within this area [44]. Although these technologies are relatively mature for geophysics and medical purposes, the same technology cannot be applied within NDT due to the difference in the conditions in the environment; this could be the typical long processing times requires in seismology, where

NDT requires a short time for on-site inspections or the vast difference in materials in medical, where the material is typically soft, squidgy with a medium [22, 27, 47]. Data obtained from ToF may withhold much more information than it is currently possible to extract, the potential of tomography have attracted debates concerning whether this is possible in NDT.

1.1 Research Aims

Although current inspection methods for NDT is sufficient, the process can be strenuous if the data volume is vast; for such a pattern recognition task, this is an ideal task for machine learning (ML). ML is a part of artificial intelligence, focusing on the study of algorithms that improves through the experience of given data. Thus this report takes advantage of access to the online database USimgAIST which contains 7000 real ultrasonic inspection images to train a model and to carry out inspection tasks to evaluate the validity of ML for NDT [36]. The inspection tasks will consist of a more basic flaw detection, where the model tells the user the presence of a flaw in the inspection image. The second inspection task will consist of a more complex flaw detection, where the model will tell the user the presence and size of the flaw in the inspection image. Furthermore, although reconstruction can be achieved with ultrasound tomography, current methods have many disadvantages such as long computation times and the incapability to provide real-time imaging. Therefore, to tackle this, ML techniques will also be implemented for this task to predict reconstructions of a material map, where the validity of ML will be evaluated for this task. Thus, in summary, the objectives of this report is expressed in the following aims:

- Implement a model using ML techniques to predict the presence of a flaw and the size of the flaw from an ultrasonic inspection image
- Employ a model using ML techniques to reconstruct a material's orientation map from taking ToF input data
- Further evaluate the performance of the flaw prediction models and reconstruction models using noisy input data to imitate more realistic conditions
- Investigate the viability of using ML for flaw checks and reconstruction for real-time predictions of material maps

1.2 Outline

To tackle the mentioned problems, an overview of the concepts required to solve these problems such as tomography and machine learning will be explored within chapter Chapter 2. The followed methodology for network setup and data processing will be outlined in Chapter 3, where the evaluation of the results and the relevance of the results to the stakeholders will be discussed in Chapter 4. The novelty presented within the report, along with the drawn conclusions from the results and further work are presented in Chapter 5.

CHAPTER 2

BACKGROUND

This chapter will provide an introduction to the principles and applications of ultrasound and tomography in Section 2.1 and Section 2.2. In Section 2.3 the report will focus on how ultrasound can be modelled and how tomographic reconstructions can be produced from obtained data. Section 2.4 will introduce a brief overview of the core concepts in machine learning. The fundamental concepts that are heavily used within this methodology will be addressed; such as the various types of neural networks, layer types and methods to evaluate the model.

2.1 Ultrasound

Ultrasound is a high frequency sound wave and is commonly used for imaging techniques to look inside objects without damaging them. Benefits of using ultrasound as opposed to other more commonly used waves, such as X-ray, are that ultrasound is cheaper, more portable and produces no harmful ionising radiation; thus, there are very little to no safety concerns [54]. However, due to its ionising radiation, X-rays are at a significant disadvantage. Typical industries which utilises ultrasound lie within geophysics, medical imaging, medical therapies and in engineering [22, 27, 47]. Where the applications are for looking inside the Earth, humans and for non-destructive testing (NDT) respectively. However, more notably, there is growing popularity with this technology within the NDT community. The growth is perhaps due to its ability to detect flaws and the characterisation of materials by utilising ultrasound's penetrating properties [31, 35, 54]. Flaw detection is a common practice in NDT and can be achieved by utilising the wave's time of flight (ToF) data, where the principal is reliant on defects possessing a different acoustic impedance; such that, the ToF will differ in comparison to detecting an object without defects [35, 54]. However, a disadvantage of this technique is that it does not provide a full-field image, allowing the user to visually observe the area of interest. In addition, this technique is very limited to the size of detection as the data is obtained by an array of sensors [22]. More recent

research has been undertaken for tomography methods with ultrasound by image reconstruction, as a full-field imaging technique, this shows great potential for progression within this area [44].

2.2 Tomography

Tomography is an imaging technique capable of producing a 2D or 3D view of an internal structure, achieved by a transmitter sending penetrating waves through an object from multiple angles. A receiver then retrieves the attenuation of the waves from the data, where a computer then processes the information on how the waves are changed to produce the output image. An illustration of the principle behind the mentioned tomography scan is shown in Figure 2.1(a), where the tomography image output is shown in Figure 2.1(b), produced using X-ray computed tomography on a human brain. The procedure which the computer performs to obtain the images is typically based on tomographic reconstruction algorithms which produce a visual representation from the obtained data [1, 42, 49, 61]. As the output typically contains blurred backgrounds after reconstruction, filters are often used within the process of reconstruction to polish the image. Often achieved by Fourier transforms [49, 61].

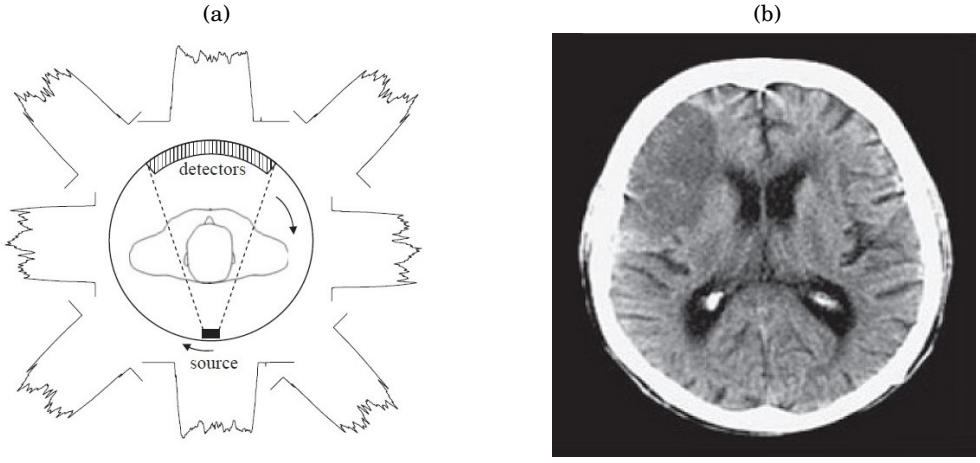

FIGURE 2.1. (a) An illustration of the physical principle behind tomography, where the transmitter (source) emits a penetrating wave through the object of interest (brain) and is received by the detectors. The signal waves outside the circle represents the attenuation data from the transmitter and receivers, which is used for reconstruction with algorithms. This process is repeated at multiple angles to record a series of planar projections. (b) A filtered tomography image produced by this process. [54]

Currently, the most common type of tomography is X-ray computed tomography, typically used in the medical industry with a vast amount of scans being performed daily [54]. Although ultrasound tomography is less commonly used, in addition to the benefits of using ultrasound, mentioned in Section 2.1; other benefits of tomography with ultrasound is the capability to producing real-time

images and the ability to probing deep within a material, without damaging them. Thus, this is much favoured within the NDT community as traditional tests to failure can be avoided [6]; resulting in less waste and money saved by not damaging the specimen. Currently, ultrasound for NDT is relatively limited; the algorithms used to create images from the data collected are very basic but it is clear that it possesses a lot more information in the data than is currently being utilised. Whereas, imaging algorithms for tomography within geophysics and medical industry is much more mature as the medium being observed is well known as opposed to the individual manufactured materials. The tomography images are obtained by processing the information obtained from the array of sensors by transmitting and receiving of signal across multiple elements simultaneously; thus, after processing the ToF data, capable of producing tomographic outputs [47, 51]; images capable of withholding information more than just visually [12, 22]. Thus, it can be observed that ultrasound tomography is relatively mature and developed in the medical industry; however, it is less so for applications in the NDT industry. As a result, the focus of this research will be based on NDT. Although existing work shows promising results of ultrasound tomography; the issue lies with the current techniques requiring long computing times which is not ideal for real-time inspections within the NDT industry, where there is an increasing demand for this [34]. However, with machine learning techniques, real-time inspections can be achieved with the capabilities of great accuracy upon sufficient training. Thus, machine learning shows great potential to solve reconstruction for tomography, which can be achieved with neural networks, discussed within Section 2.3.

2.3 Modelling Ultrasonic Travel-Time for NDT

2.3.1 Material Parameterising

To utilise machine learning for tomography, a mathematical model must be used as this will be fed into the neural network as synthetic data, unless a sufficient amount of real data can be obtained to train the network. The synthetic ToF data can be achieved by solving the forward model; whereas the material map can be generated with the ToF data via solving the inverse problem, where the model will be explained in Section 2.3.2. To generate either forward or inverse models, it is essential to first parameterise the material to study the physical system, such as the properties of the material in interest [37]. To achieve this, the most viable method is tessellation to model the structure of the material; two basic methods are Delaunay tessellation and Voronoi diagrams [15]. The latter is commonly used to parameterise material structures to create a geometry similar to a naturally occurring polycrystalline microstructure, a comparison is displayed in Figure 2.2. This is ideal for NDT as these structures are present in many materials such as metals and ceramics. From this, each region of the Voronoi diagram can then represent the distribution of a material's orientation, whilst maintaining its crystal structure. It is necessary to implement the anisotropic behaviour as the orientations affect the ToF [9].

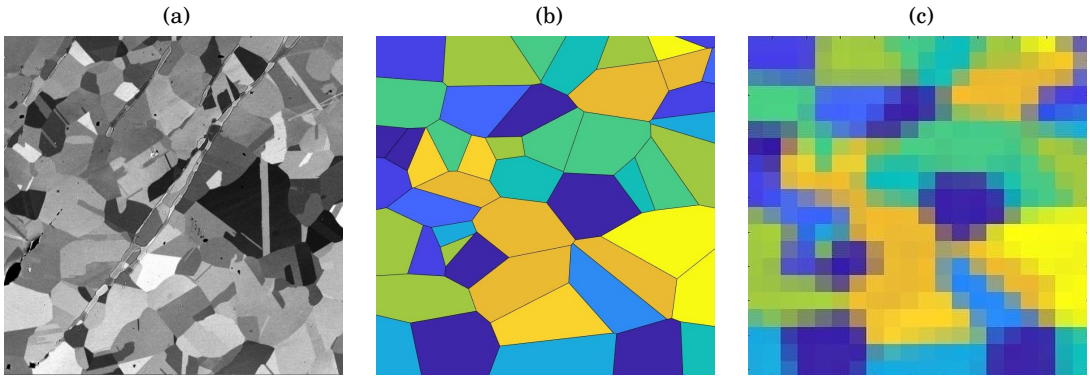

FIGURE 2.2. Comparison between an experimentally measured polycrystalline material structure of stainless steel, measured by a destructive method and electron backscatter diffraction (EBSD) shown in (a); against a Voronoi diagram (b) and a square grid representation of the Voronoi diagram shown in (c), both produced using MATLAB. [12, 21].

The Voronoi diagrams are generated by randomly distributing points within a given 2D or 3D space, where the grain size is dependent on the point density within the area. The points will then form into polygons, enclosing the areas to other polygons to the nearest point [26, 57]. Upon successfully material parameterising, it is then possible to proceed onto a forward or inverse model to obtain the results one needs; where this has been successfully demonstrated for material map reconstructions by Tant, et al. [55]. However, as inverse models are typically difficult to reconstruct with Voronoi diagrams, these are commonly solved by representing the Voronoi diagrams onto a square grid and solving the inverse problem for the new representation; this can be achieved via solving the optimisation problem. An example is shown in Figure 2.2(c) to illustrate the transformation from a Voronoi diagram to a square grid representation.

2.3.2 Mathematical Models

The two main types of mathematical models are forward models and inverse models. Forward models are typically mathematical models which attempt to calculate a state of a system that corresponds to some given model parameters in a theoretical manner. Thus, the forward problems typically consist of physics theories and has a unique solution corresponding to the parameters. An example of this is the implementation by Tant, et al. [55] with a fast marching method to solve the ToF matrix. As opposed to the forward model which solves the theory with given model parameters. Inverse models are the opposite, taking the effect of the current state to predict the model parameters as a whole. A successful implementation was proposed by Bourne, et al. [12], where this approach will be utilised to obtain synthetic data during the reconstruction with machine learning stage in Chapter 3.

Bourne, et al. [12] achieves reconstruction of the oriented Voronoi diagram by solving the optimi-

sation problem, defined as

$$F(X,\Theta) = \sum_{i=1}^{M} (T_i - T(a_i, b_i; X, \Theta))^2. \tag{2.1}$$

F represents the objective function which needs to be minimised, given the spatial location of the transmitting and receiving sensors a_i, b_i; where i represents the sensor's position which affects the simulated travel times T between M number of transmitter-receiver sensor pairs. To minimise **F**, optimisation techniques (least squares fit) are used to find the best fitted oriented Voronoi diagram (X,Θ) to reconstruct a similar Voronoi with minimal differences, for cell seeds X and the collection of orientations in the diagram Θ. Where the same slowness function,

$$S(\psi) = \frac{1}{c_1 + c_2 \cos(4\psi + \pi)}, \tag{2.2}$$

is used. For austenitic steel, $c_1 = 5621 ms^{-1}$ and $c_2 = 540 ms^{-1}$ are suitable constants for the slowness function, which will be used within this report, where ψ represents a particular orientation. The slowness function can be interpreted as the slowness of the travel time for a wave to pass through, with units of inverse velocity (sm^{-1}). Thus, the process considers the travel times of an ultrasound wave from a transmitter-receiver pair, passing through a polycrystalline structure (represented as a Voronoi) with a defined slowness curve. The sensors are indexed started from the bottom left corner and continuing clockwise, without considering any sensors lying on the same edge when determining the pairs of sensors; as ultrasound rays lying along edges contains no tomographic information as the wave has not travelled through the medium, thus, should be avoided [12]. Figure 2.3 illustrates an example of the first 3 steps of the mentioned process to obtain the travel time data. However, the results show that although a faithful reconstruction of a Voronoi diagram is possible, the method is very sensitive to noise within the ToF data. As noisy data is a common occurrence in ToF data, this method may not be practical. Thus, a ML approach will be undertaken.

2.4 Machine Learning

Machine learning (ML) is a part of artificial intelligence, focusing on the study of algorithms that allows computers to learn and to improve this learning through the experience of given data. The process consists of training a computer with data and applying algorithms to allow the machine to predict and learn. The notable branches of ML systems are supervised, unsupervised and reinforcement learning [19, 53]. However, as reinforcement learning is used to train intelligent agents via a reward based learning process; this is deemed less relevant to the problem of

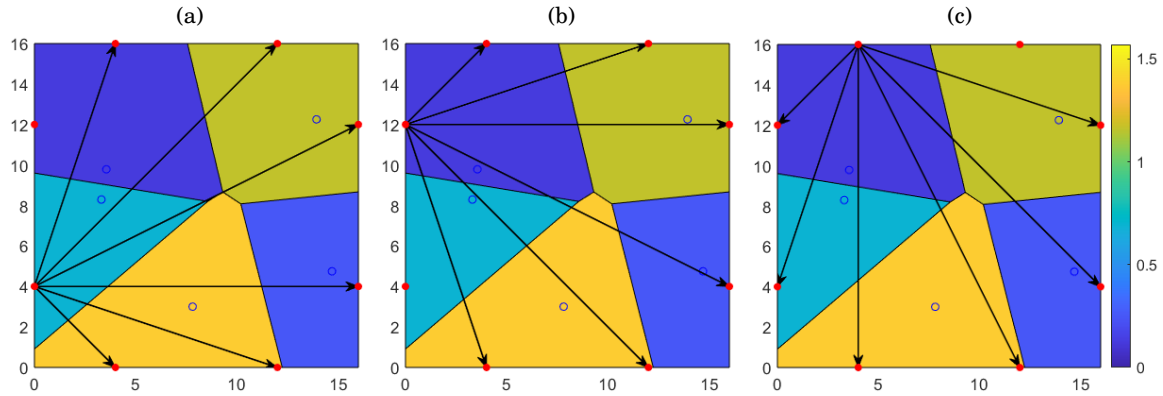

FIGURE 2.3. Illustration of the process to obtain travel time data with the approach Bourne, et al. [12] has performed. The blue circles are the Voronoi cell centres used to generate the boundaries, where the colour of the Voronoi cell represents the orientations of the material in radians. The transmitter-receiver pairs are shown as red dots lying around the edge, the arrows represents the ultrasound wave travel from the transmitter, considering the next sensor from the bottom left, continuing clockwise, whilst excluding sensors along the same edge. (a) Represents the first step of this process to recover the oriented Voronoi diagram; where (b) and (c) are steps 2, 3 and so forth.

interest, thus, this branch of ML will not be discussed within this report. In supervised learning, the machine relies on being trained by labelled data to typically formulates the data into a regression model and predicts a value based on this model. Another common supervised task is classification models, where the model formulates predictions based on categorising data into groups to predict classes. Whereas unsupervised learning is the opposite and attempts to learn with unlabelled data; thus, the machine attempts to discover information and draws conclusions using ML algorithms, without the influence of a human's input [19, 40]. Thus, ML proves to be a powerful tool in many situations, including for NDT [23]. Applications could be to approach this in a supervised learning manner, using a trained machine to predict the presence of flaws, with probability outputs of the likelihood of a flaw existing. Another application would be using a trained machine to generate an output of reconstruction estimates based on ToF inputs; where the time required will be reduced significantly compared to using traditional methods of inverse problems, as the trained model is only predicting from a simplified model as opposed to processing multiple phenomenological models.

2.4.1 Neural Networks

Neural networks are heavily based on the concept of how biological neural networks in the brain works. This includes the architecture and the principal of the phenomenological models used by neuroscientists for the brain [19, 40]. There are many variations of the network's

architectures; however, the most notable ones are artificial neural nets (ANN), deep neural nets (DNN), convolutional neural nets (CNN) and generative adversarial nets (GANs). The basic principle of an ANN, DNN and CNN comprises of 3 main layers, the input layer, hidden layer and output layer. The input layer takes an initial input, where it is processed by the hidden layers (can be multiple layers) to extract features. The output is then calculated depending on the weights and biases of each neuron (ranged between 0 and 1); depicting the strength of the connection and inactivity of the connection respectively. Where DNN are ANN but with many hidden layers, such that the network is "deep". Whereas one that possesses a structure with a small amount of hidden layers will be classified as an ANN, such that the network is "shallow". Alternatively, CNN are a specific type of DNN but specifically uses convolutional and pooling layers to extract features (explained in Section 2.4.3 thoroughly) [19, 43, 53]. Thus, for the case of a tomography reconstruction problem mentioned in Section 2.4, the input ToF data would be the input layer, where the processing will be undertaken within the hidden layers. Upon processing the input data, a reconstructed image would be the output layer. An illustration of both DNN and CNN are displayed in Figure 2.4(a) and Figure 2.4(b) respectively to demonstrate the typical architecture of the said notable networks.

Alternatively, GANs are another option to tackle reconstruction problems as they are capable of generating new data with similar characteristics as the training data [29]. This is achieved by utilising two neural networks, the generator and the discriminator. The generator generates fake data, where the discriminator learns to distinguish between the real samples and the fake samples with a binary classifier classifying the data as "real" or "fake". If the discriminator network misclassifies real data, the discriminator is penalised and will be trained and improved through loss functions. Similarly, if the generator fails to let the discriminator believe it is real data, the generator will be trained and improved through loss functions [8, 29, 41]. The typical architecture is shown in Figure 2.5.

2.4.2 Classification

To tackle the problem in a classification approach as mentioned in Section 2.3, a further understanding of classification systems is essential. Classification is one of the most common learning tasks used to predict classes [19, 39]. A classic example is training a model to output a prediction of a number from an input image, where the dataset used are a large set of images that are handwritten numbers; this is known as the MNIST dataset. Whereas an example relating to NDT would be to predict whether an ultrasound image is classed as "contains a flaw" or "does not contain a flaw". To achieve this with neural networks, a machine takes a dataset, typically a large amount of data is used for better performing models; for perspective, the MNIST dataset contains 70,000 images. This dataset is then typically split into 4 different subsets, X_train,X_test,Y_train and Y_test, where the X corresponds to the image data and Y are the labels/numbers corresponding to the image. The train and test datasets are often split into 9:1

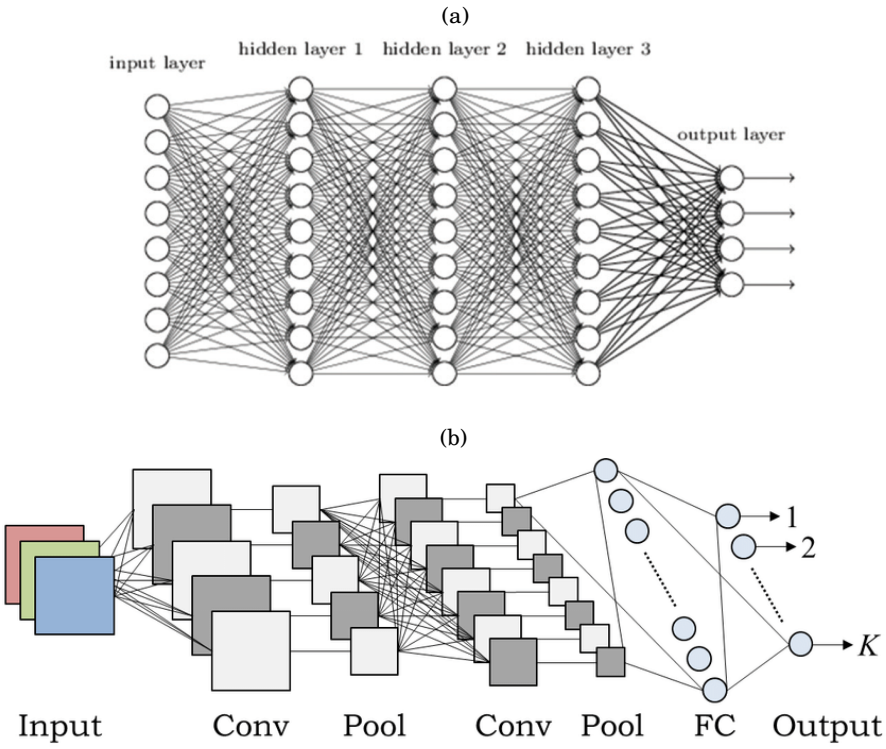

FIGURE 2.4. (a) Schematic diagram of a typical deep neural network (DNN) architecture with dense layers connecting each neuron from the previous layer to the neurons in the current layer [5]. (b) Schematic diagram of a typical convolutional neural network (CNN) architecture for a coloured image input; after processing with the hidden layers, the dense layer (shown as FC) produces a probability output representing the likelihood of the class, represented as 1, 2, ..., K [32].

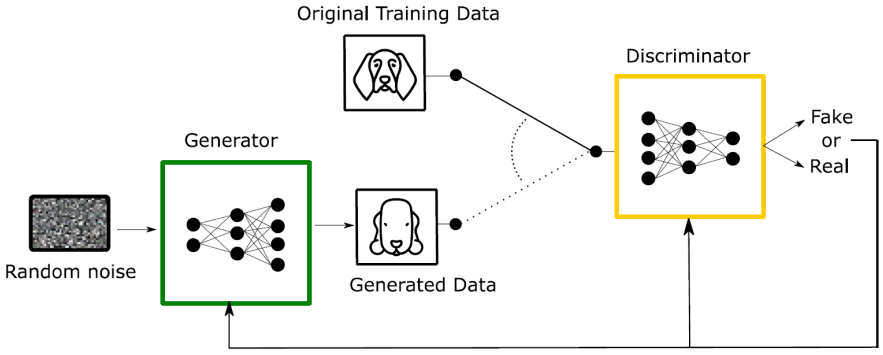

FIGURE 2.5. Schematic diagram of a typical generative adversarial network (GAN) architecture. [8]

or 8:2 ratios. The train datasets uses the images and labels to train an image, whereas the test datasets are used to evaluate the final performance of the model. In more advanced models, the

dataset can be split again to have an additional validation dataset, similar to the test dataset, it is used to evaluate the model; however, the evaluation is carried out during training rather than the final stage, where the machine will tune its parameters of the fitted model to attempt to optimise accuracy. Upon completion of training the machine, the model will be able to produce prediction outputs of any new input images [19, 39, 43, 53].

2.4.3 Neural Network Layers

To build an effective and suitable model, it is essential to understand the use of each layer. Network layers are essentially filters for the data, where the inputted data will be outputted in a form the user requires [17]. The most basic layer for neural nets is the dense layer. This layer takes all the inputs from the previous layer and connects them to all the neurons in the new layer; these connections can be seen in Figure 2.4(a). Dense layers are widely used as it provides the previous layer's combinations of features; thus, this layer is excellent at detecting patterns and repetition [17].

Convolutional layers excel in extracting features from the input; thus, they notoriously achieve great performance for images recognition. Much like the biological brain, low level features such as colour or edges are extracted within the earlier layers [19]; within the deeper layers, convolutional layers are capable of extracting more complicated features, a typical example of this is a face. This is achieved by passing a square filter called kernel (K) across the input data (typically images) until the entire matrix is traversed. The size of K and the amount K is shifted across the input data is defined by the user. The final output from this process is known as a feature map which is a series of dot products from the input and the passed filter [19, 39, 40]. An illustration to demonstrate this process is shown in Figure 2.6(a). Thus, upon completion, these features are combined within the neural net as a combination of simple and complex features to produce the output the user requires [19, 40].

Pooling layers are often used with convolutional layers to reduce computational power and the number of parameters to avoid the risk of overfitting [19]. The two types of pooling consist of max pooling and average pooling. The former retrieves the maximum value from a square patch, where the size is determined by the user, whereas the latter retrieves the average of all the values covered within that square patch. Thus, by retrieving values in such manner, pooling layers are capable of extracting more dominant features from the input data [19, 40]. This process is illustrated in Figure 2.6(b).

Activation functions are often used after convolutional layers or dense layers; these functions are used to introduce non-linearity into neural networks. Non-linearity is desired as layer outputs are usually linear functions; thus, if the whole deep net's layers are combined, it is equivalent to a single layer, one which is incapable of solving complex problems. Many activation functions

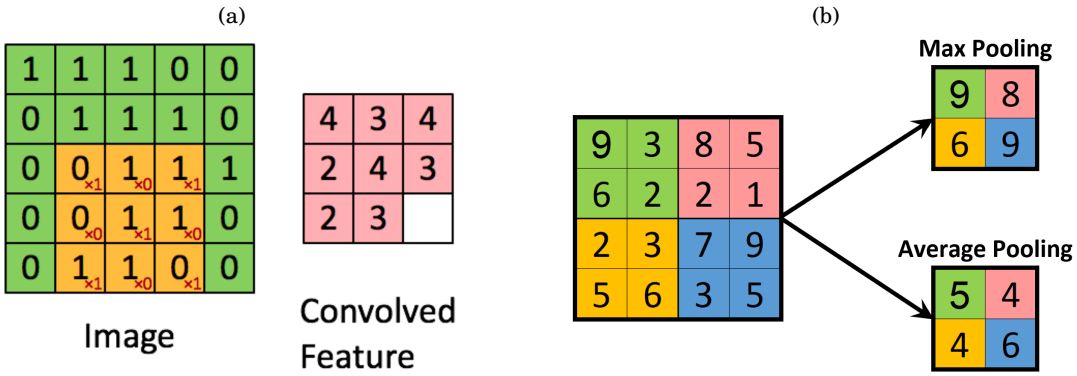

FIGURE 2.6. (a) Visualisation of the process of a convolutional layer. The kernel patch (yellow square) passes over the input data (green square) and is multiplied by the kernel (the red numbers in a cross within the yellow square); the sum is then calculated to the feature map, the final output (red square) [20]. (b) Visualisation of the two types of pooling with a filter of 2x2 passing over the input data. Figure adapted from [2].

exist for different purposes; however, the notable ones are sigmoid, softmax, rectified linear units (ReLU) and leaky ReLU. The sigmoid function is defined as

$$sigmoid(x) = \frac{1}{1+e^{-x}}, \tag{2.3}$$

for an input x [17]. The function is capable of introducing non-linearity with an output range between 0 and 1, ideal for probabilities. However, sigmoid functions are computationally expensive and the learning rate is easily saturated. The saturation of the learning is a common problem in machine learning, often caused by the function's gradient tending towards zero when using optimisation algorithms; thus, this is usually solved by choosing a suitable activation function. Thus, sigmoid is often used only for simpler problems, such as binary classifications and not commonly used over the past years [17, 19]. The softmax function is defined as

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}}, \quad \text{for } i = 1,\ldots,n \tag{2.4}$$

[17]. Although similar to sigmoid, softmax is also capable of producing an output of probabilities but is more susceptible to saturation. Thus, softmax layers are often used in practice for multi-class classification problems, commonly placed as the final layer for probability outputs [17, 19]. On the other hand, ReLU activation functions are commonly used following dense or convolutional layers in DNNs. The ReLU function is defined as

$$ReLU(x) = \max(0, x) \tag{2.5}$$

[17]. The function is often favoured due to its ability to not only add non-linearity, but to do so with low computational cost, whilst increasing training speed without being easily saturated like sigmoid. However, as the function essentially produces a $y = x$ graph for any input greater than zero, where zero is outputted for anything less; this results in any input less than zero dying off without learning, this is known as "dying ReLU" [17, 19]. Therefore, to solve this problem, the zero in the function can be replaced with another linear function with a hyperparameter to control the gradient. The gradient is typically close to zero to maintain ReLU's features without encountering the dying ReLU problem. This function is known as the leaky ReLU function which can be used in place of the ReLU. The leaky ReLU function is defined as

$$Le\ ReLU(x) = \max(\alpha x, x), \tag{2.6}$$

where α is the hyperparameter set to control the gradient, generally in magnitudes of approximately 0.01 to 0.1 [17, 19]. The graphical plots of the ReLU and leaky ReLU are shown in Figure 2.7(a) and the sigmoid function is shown in Figure 2.7(b). It should be noted that the softmax function can't be plotted in the same way due to the shift of the function depending on the input values; however, in general, the function should be similar to sigmoid with the middle point shifted depending on the input values [17, 19].

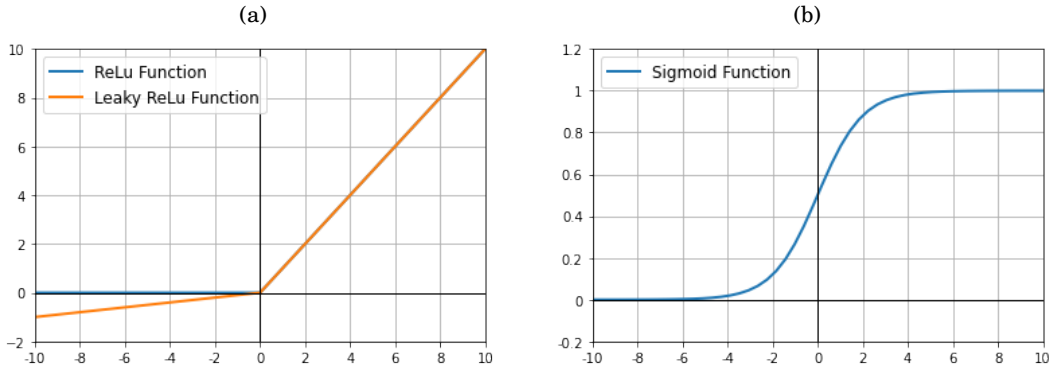

FIGURE 2.7. (a) Graph function to show the difference between the ReLU function (Equation 2.5), plotted in blue and the leaky ReLU function (Equation 2.6), plotted in orange; where a parameter of $\alpha = 0.1$ has been used for leaky ReLU. (b) Shows the graph function of the sigmoid function (Equation 2.3).

2.4.4 Network Training

To train a network it is necessary for the model to interpret whether the output is deemed "good" or "bad". In machine learning, this is often implemented with loss function, where this is typically a function which we are aiming to minimise (maximise in some instances). In the case of tomography, an example would be to minimise discrepancies to reconstruct an accurate tomographic image. Many loss functions exist but for different purposes; the most common loss function is the mean absolute error (MAE), given by

$$MAE(x) = \frac{\sum_{i=1}^{n} |x_i^{true} - x_i^{pred}|}{n}. \tag{2.7}$$

Where x_i^{pred} denotes the predicted output and x_i^{true} denotes the true output. Another similar loss function that is commonly used is the mean squared error (MSE), defined as

$$MSE(x) = \frac{\sum_{i=1}^{n} |x_i^{true} - x_i^{pred}|^2}{n}. \tag{2.8}$$

Both functions are measures of prediction errors between the actual and predicted values; thus, these are versatile functions which can be applied to a wide range of problems. However, more specific loss functions exist for more specific purposes, such as the binary cross entropy function (BCE), given by

$$BCE(x) = -\frac{1}{n}\sum_{i=1}^{n} x_i^{true} \log(x_i^{pred}) + (1 - x_i^{true}) \log(1 - x_i^{pred}). \tag{2.9}$$

This function is often used in binary classification problems, where the output of the model produces either zero or one. Whereas multi-class cross entropy function (MCCE) is often used for multi-class classification problems, defined as

$$MMCE(x) = -\frac{1}{n}\sum_{i=1}^{n} x_i^{true} \log(x_i^{pred}) \tag{2.10}$$

[7, 17, 19, 62]. To actually minimise these loss functions, optimisation techniques are often required. Many algorithms exist to achieve this but typically, the objective is for the model to search for a global optimum for a given function, representative of the discrepancies between simulated and observed data. A common approach used to minimise the loss functions are descent methods, where the most popular method is the gradient descent method, a relatively

straightforward approach to the optimisation problem. The technique calculates the magnitude of the steps with direction corresponding to the current gradient and iterates until it reaches a minimum, shown in Figure 2.8(a). However, a common issue with this simple and fast technique is that it may never fully converge or it is capable of only finding a local optimum, as illustrated in Figure 2.8(b) [13].

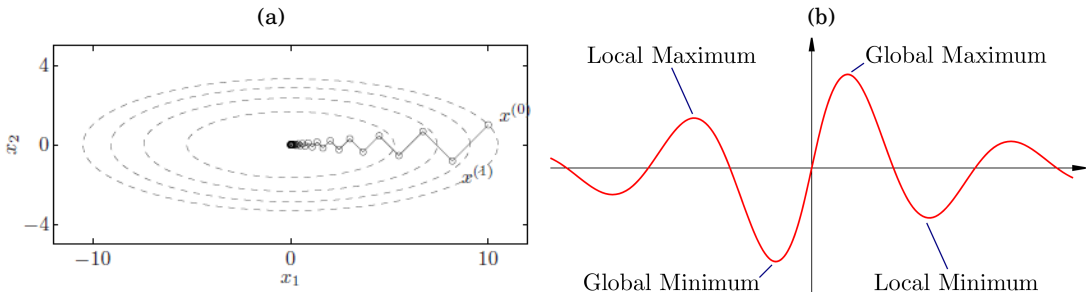

FIGURE 2.8. (a) Iterations of the gradient descent method converging towards the minimum, with an initial starting point at $x^{(0)} = (10, 1)$ [13]. (b) Illustration shows how descent methods may only search for a local optimum, rather than a global optimum [4].

Extensions of the gradient descent can be made to tackle the non-converging issue, such as stochastic gradient descent or mini-batch gradient descent, but each coming with its own limitations. More recently, further extensions of gradient descent methods have been implemented, such as the adaptive moment estimation (Adam) [38]. Adam optimisers utilises adaptive gradients to allow different parameters to have different learning rates; as a result, speeding the training process, as opposed to the traditional methods [17, 19, 38]. However, to fully utilise the potential of optimisers for optimum performance, tuning an optimiser's hyperparameters are necessary. The most fundamental parameter to be tuned is the learning rate as a poorly selected rate may result in a model to not learning. An ideal value for the learning rate does not exist due to the nature of convergence with functions. Thus, a learning rate too little may increase the computational times significantly as many steps are required to reach the optimum point; whereas a learning rate too large may result in jumps that never lead to the optimum point, depicted in Figure 2.9. Therefore, methods such as random search or Bayesian optimisation can be used to tune the hyperparameter to rge find values that performs best [17, 19, 38].

2.4.5 Model Evaluation Methods

Performance of trained models can vary largely depending on the architecture or how the hyperparameters are tuned; therefore it is essential to evaluate the performance of a particularly trained model with viable performance metrics [17, 19, 50]. For classification problems, this is typically evaluated with the accuracy and precision of the predictions and can be presented in

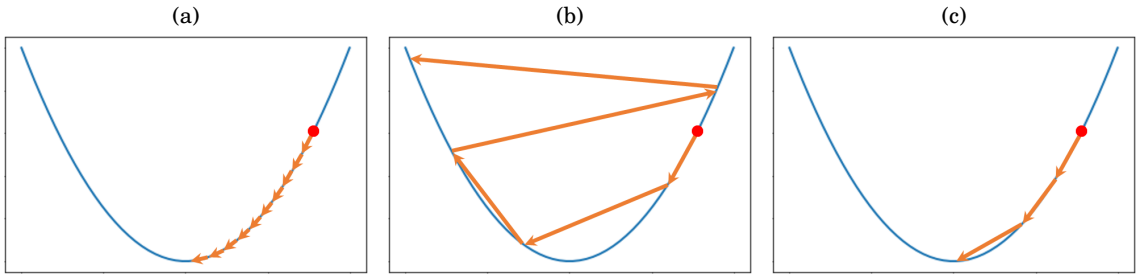

FIGURE 2.9. (a) Denotes the case where a learning rate that is too small is used, resulting in longer computing times. (b) Represents the case when the learning rate is too large which may result in steps too large for the model to converge. (c) Illustrates the balance between the two cases, where the most suitable learning rate can typically be found by hyperparameter tuning. Figure adapted from [18].

a confusion matrix. An example is shown in Figure 2.10(a) for the binary flaw classification example. Within the confusion matrix, accuracy is expressed as

$$\zeta = \frac{TP + TN}{TP + TN + FP + FN}, \tag{2.11}$$

where ζ is called the accuracy; whereas precision of the classification's predictions is expressed as

$$\nu = \frac{TP}{TP + FP}, \tag{2.12}$$

where ν denotes the precision. True positive and true negatives are denoted as TP and TN respectively, likewise, false positive and false negative is denoted as FP and FN respectively. The confusion matrix can also be utilised for receiver operator characteristic (ROC) curves as another performance metric [24]. ROC curves can be utilised to optimise the threshold balance between TP rates and FP rates; an ideal value would consist of TP rate of 1 and FP rate of 0 [24, 50]. However, realistically this is unlikely due to many factors; thus, the threshold shifted to optimise both TP and FP rates as demonstrated within Figure 2.10(b). This balance between the two can be captured by maximising the area under the curve (AUC) where the perfect classifier AUC is 1 and a random classifier AUC is 0.5 [24, 50]. The ROC curve can be drawn from plotting the TP rate against the FP rate, where each point is obtained from shifting the said threshold. An example of a ROC curve is shown in Figure 2.10(c).

(a)

		Predicted	
		Flaw	No Flaw
Actual	Flaw	True Positive TP	False Negative FN
	No Flaw	False Positive FP	True Negative TN

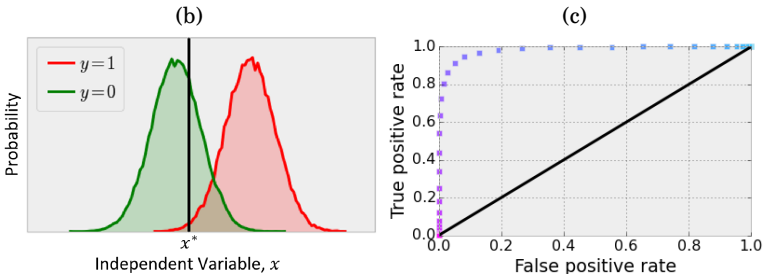

FIGURE 2.10. (a) Example of a confusion matrix for a binary flaw classification problem, where typically each cell would be filled with the number of predictions. (b) Illustration of how tuning the threshold x^* may affect the TP and FP rates, where $y = 1$ may represent a flaw present and $y = 0$ represents a flaw not being present. The example demonstrates that the current threshold would induce a larger FP rate; thus, the threshold should ideally shift to the right to reduce the FP rate, whilst increasing the TP rate, serving as a balance between the two sides. Figure adapted from [52]. (c) Displays an example of a ROC curve obtained from plotting the TP rate against the FP rate at each distinct threshold values [52].

Another metric to evaluate performance is validation accuracy. This can be obtained by using the trained model to evaluate the validation dataset, where the returned accuracy denotes how well the model predicts the inputs. A common issue during this process is underfitting or overfitting a trained model. The former denotes a model which hasn't been fully trained to recognise relevant patterns in the training data, leading to poor performance. Whereas the latter denotes a model which learns patterns specific to the training data, therefore leading to generalising all inputs. Typically this can be interpreted by monitoring the validation loss and training loss. Where the loss is the sum of the proportion of errors in the predictions made for the validation or the training data sets respectively. If validation loss is greater than training loss, this implies overfitting, whereas if the validation loss is less, this implies underfitting. However, to avoid overfitting, dropout layers are typically introduced to allow variation within the pattern recognition. Dropout layers allow a defined percentage of the trained neurons to be deleted. Another method is to introduce early stopping; the model is trained until the validation loss does not improve by a specified percentage [17, 19, 50].

CHAPTER 3

METHODOLOGY

In order to explore the possibilities of ML for NDT, the approach to tackle this objective is to undertake the problem from two different perspectives. The first being a classification approach, where the trained model takes an input of an image containing ultrasound data of a stainless steel plate; the model then outputs a prediction regarding whether the input image contains a flaw. The second problem being the reconstruction problem which takes ToF data and attempts to predict a reconstruction of the material map from the ToF. Section 3.1 will discuss the method undertaken to tackle the classification problem, whereas the methodology used to approach the reconstruction problem is presented in Section 3.2.

3.1 Flaw Classification Problem

3.1.1 Datasets

The classification approach takes advantage of access to the extensive USimgAIST dataset presented by Ye, et al. [36]. The dataset contains 7000 real ultrasonic inspection images produced by B-scans, a technique that utilises ultrasonic waves to produce images of the reflected wave travel [36, 54]. The dataset contains images of non-defective and defectives cases, where the images are obtained from stainless steel plates with flaw sizes varying from no flaw, 1mm, 3mm, 5mm and 10mm. Example images of the dataset is displayed in Figure 3.1. The approach taken by Ye, et al. [36] mainly focuses on detection accuracy, neural net complexity, memory usage and speed; thus, this report will aim to investigate the potential of neural nets focused for more practical use in NDT. This includes classification of not only flaw detection but also size; furthermore, evaluation of the performance with the addition of noise to imitate more realistic data in practice will be undertaken.

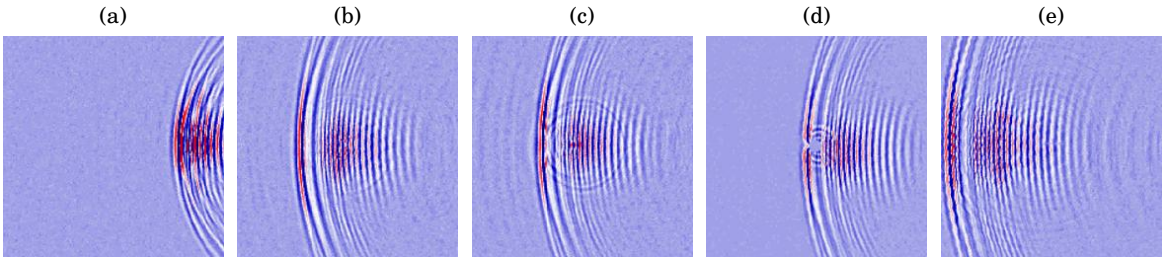

FIGURE 3.1. Examples of the ultrasonic inspection images from USimgAIST dataset, where (a) to (e) are images of no flaw, 1mm flaw, 3mm flaw, 5mm flaw and 10mm flaw respectively [36].

3.1.2 Implementation

The entire process including image pre-processing, hyperparameter tuning, training and data evaluation is implemented in Python using *Google Colab* [10]. However, it should be noted that initially, the original implementation had been undertaken using MATLAB due to the familiarity of the language. However, due to limitations such as available online packages and more importantly speed, a transition to Python had to be made [45, 60]. Although inherently, MATLAB does not have issues with speed, the limitation arises when the available access to computational power relies on the user's computer itself. Whereas *Google Colab* provides a powerful computing resources for online access with an Nvidia K80 graphics processing unit (GPU) with 12GB available random access memory (RAM) [10]; thus, with *Google Colab's* available computing resource, processing time was significantly reduced. A comparison of the training times between the two is displayed in Table 3.1. An Adam optimiser has been used to train both models for the adaptive ability of the optimiser, both with a default learning rate of 0.001. Although both model's architectures are inherently similar, the Python approach consists of hyperparameter tuning; thus, the number of layers and input shape varied. The Python implementation uses *Tensorflow* and *Keras* packages to train the neural network, where hyperparameter tuning was processed using *KerasTuner*; whereas the MATLAB implementation has been undertaken with the *Deep Network Designer* app to train the network.

Language	Epochs	Models Trained	Time Elapsed	Time Taken per Model
MATLAB	20	1	88 Minutes	88 Minutes
Python (Colab)	20	10	272 Minutes	27 Minutes

TABLE 3.1. Training time comparison between MATLAB and Python with *Google Colab's* GPU. The execution on Python was capable of training 10 models during hyperparameter tuning; whereas the execution on MATLAB was only capable of training 1 model without hyperparameter tuning over a longer period of time.

3.1.3 Model Training and Evaluation

To tackle the classification problem, a CNN will be employed with the principal reasoning that these types of neural nets typically excel at extracting features [19]. Thus, the built CNN should be able to classify the presence of flaws within an image, where the first CNN will be used to evaluate the presence of flaws, whereas the second CNN will be used to evaluate the presence and size of the flaws. To produce an effective model, the input shape of layers and number of layers has been tuned to produce the highest validation accuracy with an MMCE loss function (see Equation 2.10), where the default 0.001 learning rate of Adam optimiser due to its performance without tuning. The MMCE loss function has been selected due to the nature of the classification problem; arguably the BCE loss function (see Equation 2.9) can also be used for the first problem but the nature of the MMCE function itself is equal to the BCE loss function when it is a binary problem [7, 17, 19, 62]; thus, the selection of MMCE has been chosen for consistency of both models. To reduce interference time of the tuning phase, a random search of the specified parameter ranges will be processed for 10 iterations with 20 epochs per execution, trained with a batch size of 64. To visualise the performance and to ensure the model was converging, *TensorBoard* callback was used to retrieve the accuracy and loss per epoch graphs, with an objective to minimise the validation accuracy. The final tuned architecture of both models consisted of 5 hidden layers consisting of 2D convolutional layers with leaky ReLU activation to introduce non-linearity, where the leaky version of ReLU has been selected to avoid the "dying ReLU" problem as mentioned in Section 2.4.3; these layers are followed by 2D max pooling layers to improve extraction. A dropout layer has been introduced at the end of each hidden layers except the first; this has been implemented to allow the model to initially learn more before allowing the model to perform better with variation in data and to avoid overfitting [17, 19, 50]. The last hidden layer consists of the addition of a dense layer of 2 for the binary model, whereas a dense layer of 5 has been used for the multi-class model with a softmax activation; this has been implemented to "compile" the features and allow the model to output the relevant number of classifications. A visualisation of the binary flaw classification network's architecture is displayed in Figure 3.2, where the multi-class flaw size classification network's architecture is illustrated in Figure 3.3.

To evaluate the trained models, *Numpy* and *Scikit* were used for image processing, adding noise to images and producing graphs to evaluate the trained models [30, 56]. Noise has been added to the images during the evaluation stage to evaluate the performance of the models when noise exists in the data, which is a typical occurrence in practice [28]. Both multiplicative noise and additive noise has been utilised to evaluate both scenarios, where multiplicative noise is a random process that involves a function adding noise to the image to create a noise mask [11]. Adding multiplicative noise in the data has been achieved by using *Numpy's randint* to generate a random number from a range of values [30]. The range would vary depending on the scale, where 100% noise would allow a range from 0 to 255, the maximum colour shade value. Additive noise is where the image is interfered by constants, where this has been implemented by using

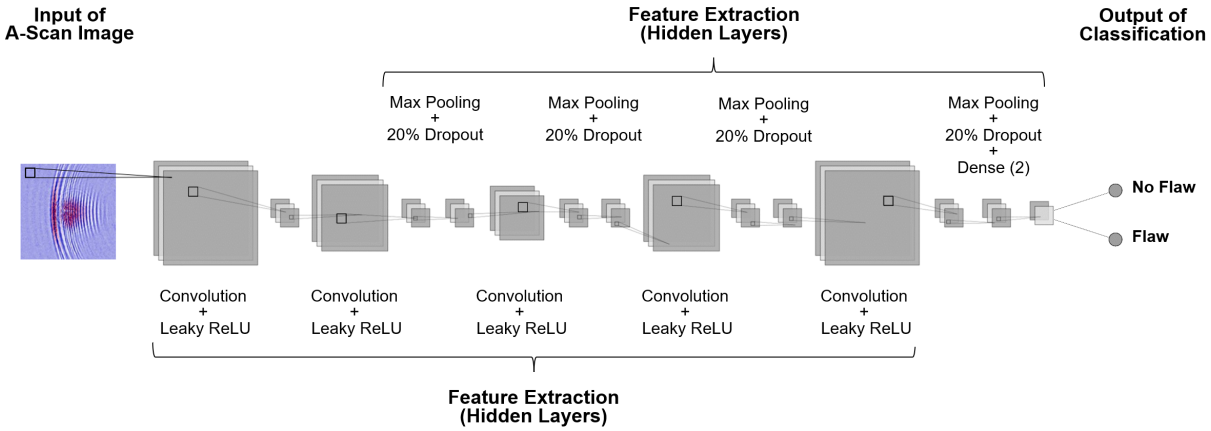

FIGURE 3.2. Illustration of the network architecture for the binary flaw classification model. The model takes an input of a B-scan inspection image and outputs a prediction of the presence of a flaw.

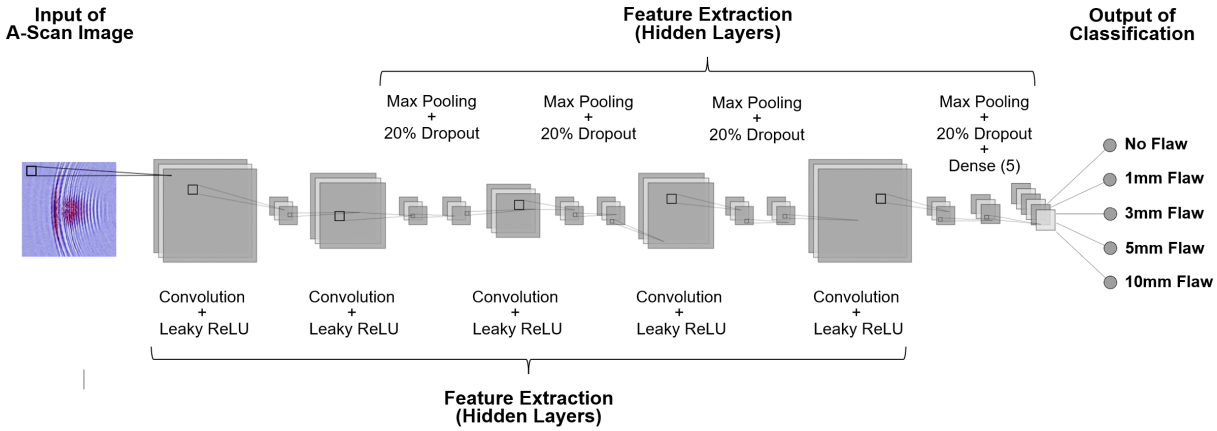

FIGURE 3.3. Illustration of the network architecture for the flaw size classification model. The model takes an input of a B-scan inspection image and outputs a prediction of the presence of a flaw and the size of the flaw.

Scikit's random_noise with "salt and pepper" [11]; such that values of 0 or 255 are masked across the red, green, blue data randomly, where the amount added is set by the user ranging from 0 to 1 [56]. An example of the noise masked images are shown in Figure 3.4. For full details of the method, refer to Section 4.3 for the implemented Python scripts to reproduce the processing of noisy data and the scripts for tuning and training the models.

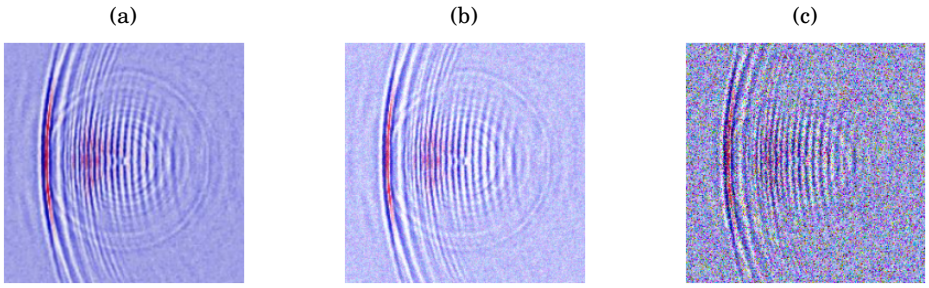

FIGURE 3.4. (a) Represents the original image without the addition of noise, where (b) shows the same image with the addition of multiplicative noise at a scale of 25%, generated using *randint*. (c) Displays additive noise added to the original image using *random_noise* with salt and pepper, with the amount set to 25%.

3.2 Reconstruction Problem

3.2.1 Datasets

The approach performed by Bourne, et al. [11] had underlying issues such as time and susceptibility to noise [12]; thus, the approach taken within this report will aim to investigate the viability of using ML to predict the material map. To tackle the reconstruction problem, the approach demonstrated by Bourne, et al. [12] will be utilised to obtain synthetic data to train the neural network to predict the material map from ToF data, where the theory of the approach is explained in Section 2.3. The input data fed into the network is the travel times obtained from indexing the transmitter-receiver sensor pairs as demonstrated in Figure 2.3. The output data of the network is generated from obtaining the underlying orientations of the material map used to produce the ToF, where the parameters of the slowness function used are for austenitic steel, as explained in Section 2.3.2. To ensure the model learns and produce accurate results, a sufficient amount of data is required [17, 19, 40]; thus, a synthetic dataset of 10,000 input and output data has been generated. The material map will be parameterised within a 16x16 square grid, where two transducer elements are used on each edge as illustrated previously within Figure 2.3; thus, the inputs will consist of travel times from the 24 transmitter-receiver pair combinations. Although a larger grid and more transducers may be used for higher definition, the selected values have been chosen as the definition is deemed sufficient to evaluate the viability of the problem in interest. If a larger square grid or more transducers are used, this will significantly increase interference time due to the significant increase in data; the difference is illustrated in Table 3.2.

3.2.2 Implementation

From the knowledge gained during the classification problem in Section 3.1.2. The entire process will be implemented in Python using *Google Colab* [10] except the generation of the synthetic

Transducers per Side	Transmitter-Receiver Pairs	Input Data Increase
2	24	-
3	54	225%
4	96	400%
5	150	625%

TABLE 3.2. Table illustrating the increase in amount of travel time input data per extra transducer, resulting in an increase in computing time.

input and output data which has been generated using MATLAB. The same hardware available with *Google Colab* has been utilised, such as the Nvidia K80 and 12GB RAM to reduce computing times [17]. Although 10,000 input and output data has been generated, a reduction in the data used to train the models has been implemented due to *Google Colab's* restrictions with runtime disconnections past 12 hours [10]. As training a model with 10,000 input and output data with 24 transmitter-receiver pairs requires over 16 hours; a reduction to 7500 dataset has been utilised to train the models to avoid runtime disconnections.

3.2.3 Model Training and Evaluation

A CNN will also be used to tackle the reconstruction problem rather than a DNN as convolutional layers typically excel over dense layers for extracting features from images and especially signal data [17, 19, 40]. Thus the principal reason behind using a CNN is the belief that travel times from ultrasound sensors may be interpreted as signal data, like sound and telecommunication [17, 19, 40]. Thus the built CNN should intake travel times and output a prediction of a single pixel of the cell orientation per trained network. Although reconstructing the orientations of the full material map has been initially attempted, this will cause issues by creating classification outputs of certain reconstructions. As the problem at interest theoretically has an infinite amount of material map outcomes, this approach is not viable; thus, a network outputting predictions of a pixel per cell orientation has been implemented. A similar network architecture with multiple hidden layers consisting of convolutional, leaky ReLU, dropout layers with a dense layer of 1 at the end has been used to train the network without the softmax activation, where the architecture reasoning is explained thoroughly in Section 3.2.3. However, as opposed to 2D convolutional and 2D max pooling layers, these have been replaced with the 1D versions due to the shape of the travel times (24×1 array). An illustration of the network's architecture is shown in Figure 3.5 for clarity.

The number of layers, filters, kernel size and learning rate has been selected by using *KerasTuner* with random search to find the best performing parameters by hyperparameter tuning the model. Whereas *Tensorflow* and *Keras* has been used to train the neural network. A *Tensorbaord* callback function was used to ensure the model was converging by evaluating the accuracy and loss per

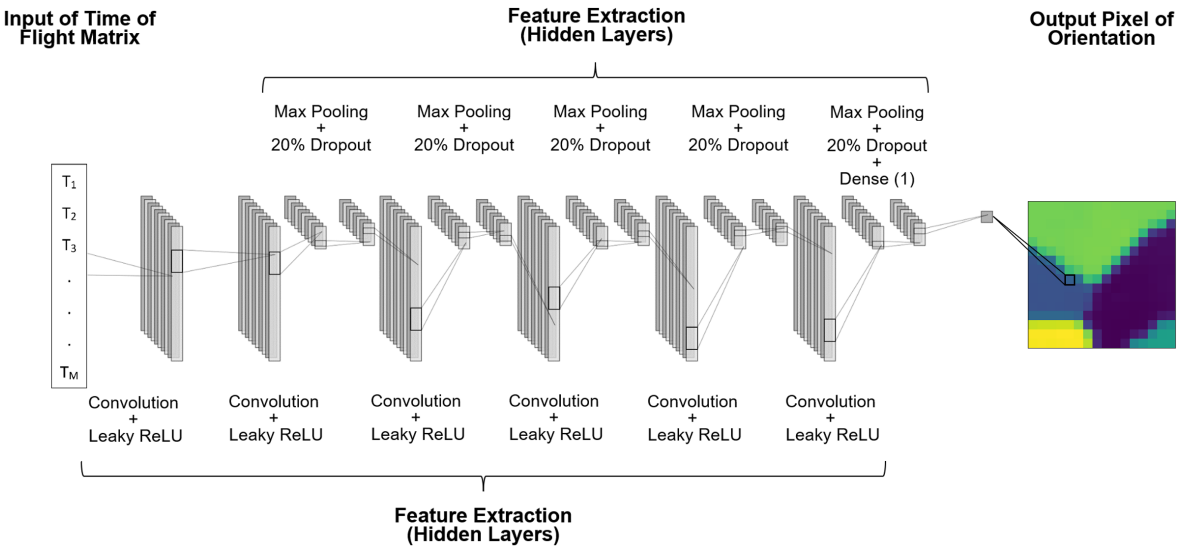

FIGURE 3.5. Illustration of the network architecture for the reconstruction model, where T represents the arrival times and M represents the transmitter-receiver pairs; as two transducers per side have been used, this value would be 24. The model takes an ToF data as the input and outputs a prediction of a cell orientation pixel per trained network; thus for a parameterised 16x16 square grid map, this would consist of 256 networks.

epoch graphs, where the loss function used was MAE (see Equation 2.7) with an Adam optimiser. An MAE loss function has been selected over the previous MMCE as the objective is to minimise the differences between the actual and predicted outcome, such that ideally, the difference is zero [7, 17, 19, 62]. As opposed to the classification approach, it was not possible to reduce interference time during tuning by using smaller iterations and epochs due to the complexity of the problem; such that the model required longer training times to process accurate performances represented by the hyperparameters. Thus, 20 iterations with 64 epochs per execution, trained with a batch size of 64 has been used during the tuning stage.

An initial implementation of 10 cell centre seeds has been undertaken; however, the quality of the results was unsatisfactory (see Appendix A.3.2 for examples of results). Thus, to ensure the validity of the code and network architecture, a test run has been executed by setting a 4 cell centre seed material map, such that the orientations of the material map are set out as 4 squares. By setting non-random cell centre seeds allowed the problem to be simpler in hopes that the model would learn quickly to solve this problem and verify the validity of the written code; thus, a smaller sample size of 2500 data has been generated. An example of the non-random 4 seed material map is shown in Figure 3.6(a). Upon completing the working code with the 4 seed model, the next stage was to train more complex models related; thus, a 2 seed material map and a 5 seed material map model were trained, where the dataset used consisted of Voronoi cell centres

set to be random. Examples of the 2 seed material map is illustrated are Figure 3.6(b) and the 5 seed material map displayed in Figure 3.6(c).

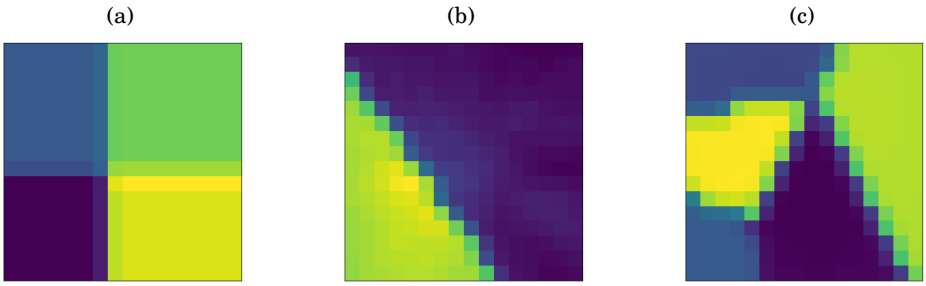

FIGURE 3.6. (a) Displays the non-random 4 seed material map, split into 4 different orientations to test the functionality of the code. Whereas (b) and (c) shows the randomly generated 2 seed and 5 seed material map respectively, where these will be used to evaluate the validity of this approach to the reconstruction problem.

Hyperparameter tuning had been carried out for the initial model with 4 seeds, where the same parameters were used to train the 2 seed and 5 seed models. However, this produced unsatisfactory results due to the lack of learning for the 2 and 5 seed models. Thus, it became apparent that the three models possess different relations; therefore, the tuning stage has been processed separately for each existing model to ensure the parameters used for the model had been fully optimised. The parameters found and used to train the specified models over 500 epochs are shown in Table 3.3.

Model	**Hidden Layers**	**Learning Rate**	**Filters**	**Kernel Size**
4 Seed	5	3.5×10^{-3}	12	2
2 Seed	5	5.0×10^{-4}	10	4
5 Seed	5	4.5×10^{-4}	10	4

TABLE 3.3. Best performing parameters used to train the specified models, where the hyperparameters were searched by using *KerasTuner*.

Following the approach used during the classification models, *Numpy* and *Scikit* were used to evaluate the trained models by adding multiplicative and additive noise. This approach to evaluating the models follows the same principle, such noise is added to imitate data obtained in real-life practice within NDT [28]. Multiplicative noise has been implemented by using *Numpy's randint* with the mentioned scale approach; however, as the reconstruction approach does not use hex values, this approach will use values of -π to π representing the orientation range, as opposed to the previous 0 to 255 range [12]. Additive noise has been achieved by using *Scikit's random_noise* with "salt and pepper" [11]; however, a value range of -π to π has been set as opposed to the previous 0 to 255 range approach for the aforementioned reason. For full details of the method, refer to Section 4.3 for the implemented Python scripts to reproduce the processing of noisy data and the scripts for tuning and training the reconstruction models.

CHAPTER 4

RESULTS

Following the training and image processing procedures discussed in Section 3, the results obtained from the evaluated flaw classification problem is presented in Section 4.1, where the evaluated results for the reconstruction problem is discussed within Section 4.2. The stakeholder relevance of the results is outlined in Section 4.3, where the constructed code used for the experiments is shown in Section 4.4.

4.1 Flaw Classification Problem

To quantify the validity of using ML to classify flaws, ROC curves (for explanation, see Section 2.4.5) have been formulated to represent the success of the method. Although ROC curves are typically used to evaluate the probability of an occurrence, featuring a tuned threshold [55]; the approach presented does not possess such threshold. Thus, ROC curves have been generated by using the *roc_curve* function within the *Sklearn* package to evaluate the success. As opposed to the typical threshold method ROC curve requires, the function utilises distinct classification scores to generate each step within the ROC curve, where the classification score is a metric produced using *OneVsRestClassifier* to fit the data [46]; thus, this approach allows the evaluation to be viable without such threshold by imitating a similar procedure. The generated ROC curve is shown in Figure 4.1(a) and Figure 4.1(b) for the flaw classification and flaw size classification respectively. The corresponding AUC scores produced is displayed on the graph representing the performance of the trained model. An AUC score of 1 depicts 100% prediction accuracy, whereas a score of 0 depicts 0% prediction accuracy [24, 50].

From the results displayed in Figure 4.1, it can be deduced that the most accurate trained model is the binary model as it produced the highest AUC score of 0.97, where the model predicts the presence of a flaw alone. Conversely, the model which also predicts the flaw size, shown in Figure 4.1(b), performed worse, although a difference of 0.01 was obtained from the "no flaw" AUC

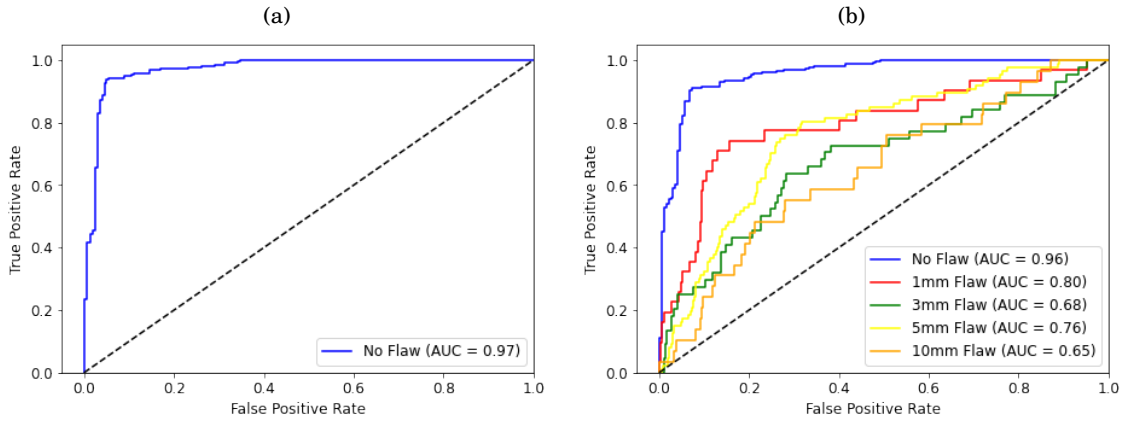

FIGURE 4.1. (a) Displays the ROC curve generated for the trained CNN model (see schematic in Figure 3.2) to detect the presence of a flaw; due to the binary nature of the problem, only one curve has been drawn. (b) Shows the ROC curve obtained from the trained CNN model (schematic illustrated in Figure 3.3) to detect the presence and size of the flaw, where both models are trained and given an input from the USimgAIST dataset [36]. The straight dotted black line represents a random classifier model, such that any curve above this line represents a model which performs better than a random classification model [24, 50].

score. This suggests that the model is capable of accurately predicting the existence of flaws but underperformed when the size of the flaw is considered. Whereas, the model which only considers the presence of flaws is capable of accurately predicting either. However, as the AUC scores of the size predictions are not approximately 0.5, this suggests that the predictions are still satisfactory and not completely random. Therefore, this suggests that both models are effective at predicting the presence of flaws but is still incapable of fully predicting the size of the flaw to a satisfactory accuracy; this is likely due to high similarity of the underlying features between each flaw size datasets. However, as the AUC scores suggest that the flaw size predictions are not random, it may be possible that a change in network architecture or additional data can be used to improve the performance. Although the USimgAIST dataset contains over 7000 images used to train the models, it may be possible that this is still not sufficient for the network to fully learn the features as the 7000 images are split into many more flaw size categories; thus, significantly reducing the dataset size. This can be supported by the accuracy and loss graphs produced during training; the accuracy and loss graphs for the flaw detection is depicted in Figure 4.2(a) and Figure 4.2(b) respectively. The graphs show that sufficient training has been applied from the convergence of both accuracy and loss graphs; thus, potentially resulting in superior results. Whereas the accuracy and loss graphs for the flaw size classification is depicted in Figure 4.2(c) and Figure 4.2(d) respectively. The graphs suggests that although the model is learning with high accuracy, the accuracy graph has not fully converged even though the loss graph appears to be converged; this implies that either more data is necessary or the learning rate needs to be altered [17, 19].

However, due to time constraints, further tuning of the learning rate has not been undertaken.

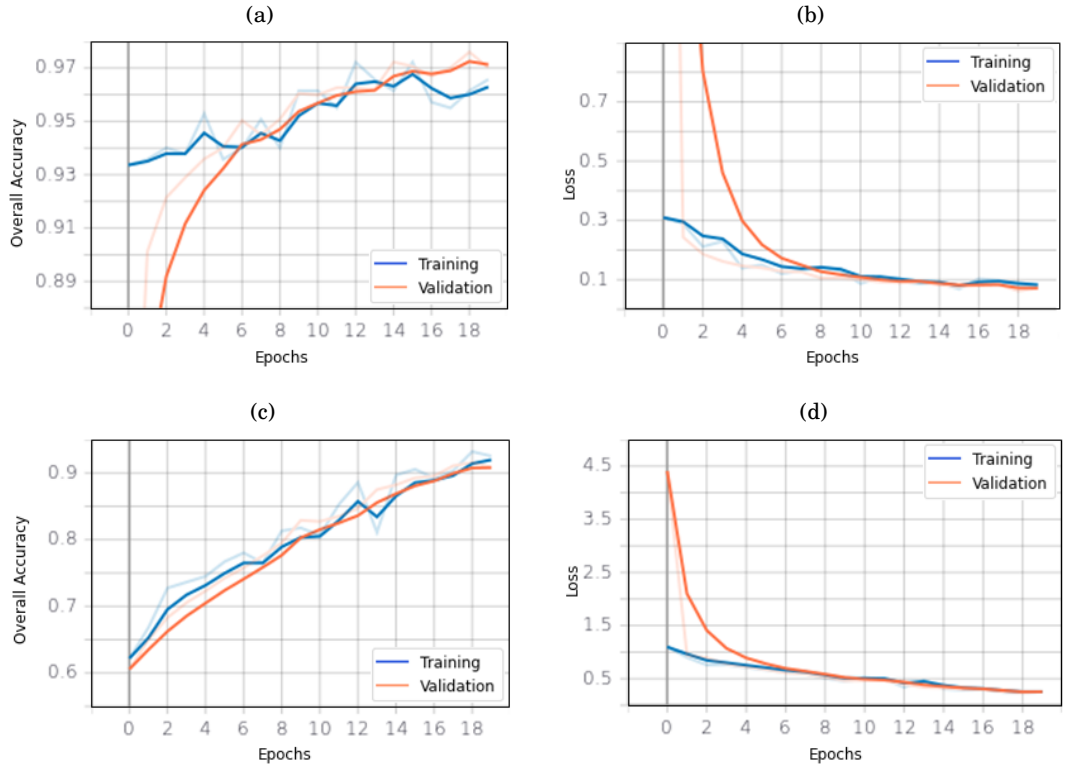

FIGURE 4.2. (a) and (b) depicts the accuracy and loss graphs generated during training the flaw detection CNN model (network architecture shown in Figure 3.2) respectively. Whereas (c) and (d) shows the same graphs produced but for the flaw size classification CNN model (see Figure 3.3 for network architecture). The faint lines on the graph depict the original curve before the smoothing of the curves produced by *Tensorflow*.

To further assess the performance of the models, the validation accuracy has been evaluated with a validation split of 20% on the dataset; a sample of 20 results with the validation accuracy of each set is shown in Appendix A.1 and A.2 for the flaw detection and flaw size classification models respectively. The results shows that the former model produces high accuracy for both flaw and no flaw classifications; however, the latter model is only capable of producing high accuracy for flaw detection but fails to perform greatly when classifying the size, supporting the ROC curve evaluation. Interestingly, the model is capable of classifying 1mm flaws significantly better than the other flaw sizes, this is likely be due to overfitting of 1mm flaws. To avoid this, the amount of data in each category within the dataset should be equal, which is not the case in the USimgAIST dataset [36]. Another method to avoid this could be early stop functions; however, due to Figure 4.2(c) and 4.2(d) indicating more data should be needed, the former method should be followed. Following the approach discussed in Section 3.1.3, validation accuracy has been further assessed

with the addition of multiplicative and additive noise. The evaluated validation accuracy of the full dataset results from the noise evaluation of the flaw detection model are outlined in Figure 4.3(a), where the results from the addition of additive noise (σ) and multiplicative noise (μ) on the flaw size detection model are shown in Figure 4.3(b) and Figure 4.3(c) respectively. From Figure 4.3, it can be deduced that although excellent accuracy can be produced without noise, the addition of either type of noise will significantly reduce the validation accuracy for both models. Although a decay in accuracy is expected as the noise level increases, the results illustrate that an instantaneous drop in accuracy occurs as soon as noise is added. Therefore, suggesting that although the ML method proposed outputs fast predictions (requiring less than a minute to predict entire test dataset), the method may only be suitable for controlled environments without noise. Alternatively further training should be introduced with the addition of noisy training data to experiment if the performance will be improved under the influence of noisy inputs.

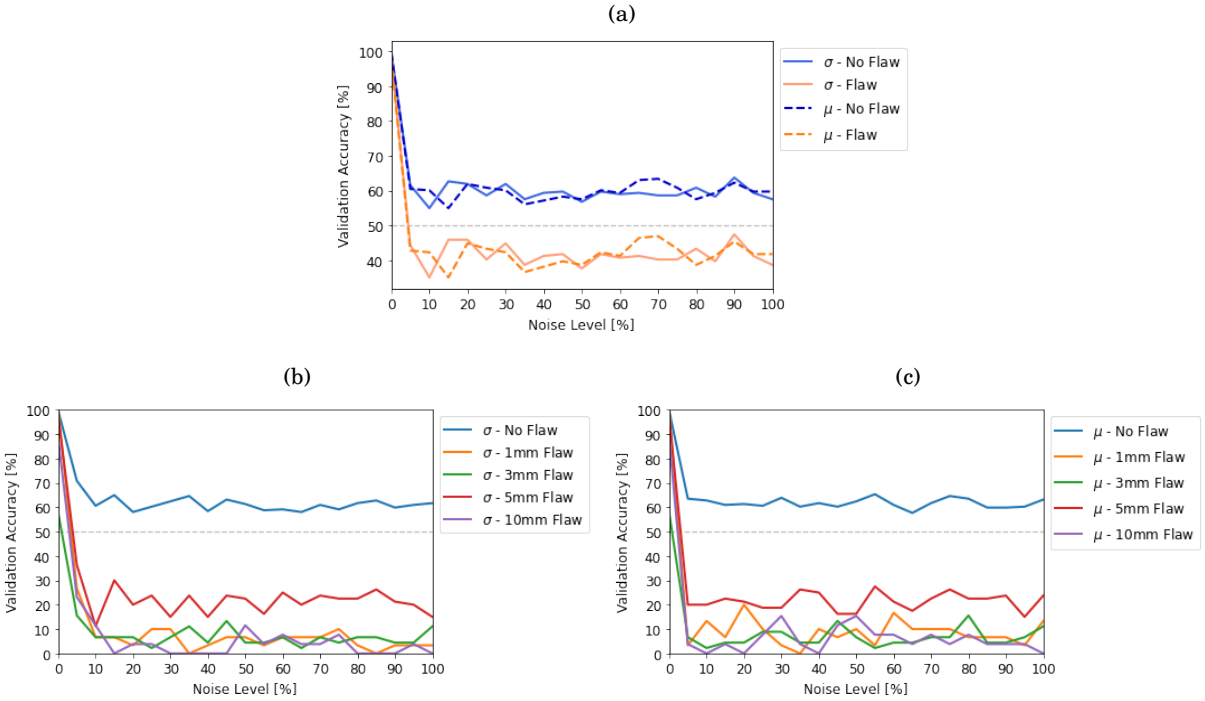

FIGURE 4.3. (a) Noise evaluation for the flaw detection model (schematic shown in Figure 3.2), where σ represents additive noise and μ depicts multiplicative noise. (b) Displays the additive noise evaluation for the flaw size classification model (see schematic in Figure 3.3), where the evaluation for multiplicative noise is shown in (c). The grey dashed line depicts the draw off from 50% for clarity, where below or at 50% the predictions are considered random.

4.2 Reconstruction Problem

Following the training procedure stated in Section 3.2.3, the epoch accuracy and loss graphs have been evaluated as a metric to verify the trained model has learnt the features [17, 19, 40]. However, as the condition of the accuracy metric differs from the flaw classification problem, the accuracy graph wasn't viable as there was a zero accuracy output. The accuracy metric requires the pixel orientation to predict the exact true value to generate a non zero accuracy, this proves to be a difficult task in predicting the exact value; whereas, in the flaw classification problem, the classification of "correct" and "incorrect" answers is significantly easier than an exact value. Thus, only the loss graph will be taken into account to consider the learning of the trained models. The generated graphs for the 2 seed and 5 seed models are depicted in Figure 4.4(a) and Figure 4.4(b) respectively. The graph for the 4 seed model is shown in Appendix A.3.1 to reduce cluttering as the model was used to verify the validity of the code and is formed on the basis of a simple model; thus, an in-depth evaluation will not be undertaken, where the focus will be on both 2 seed and 5 seed models.

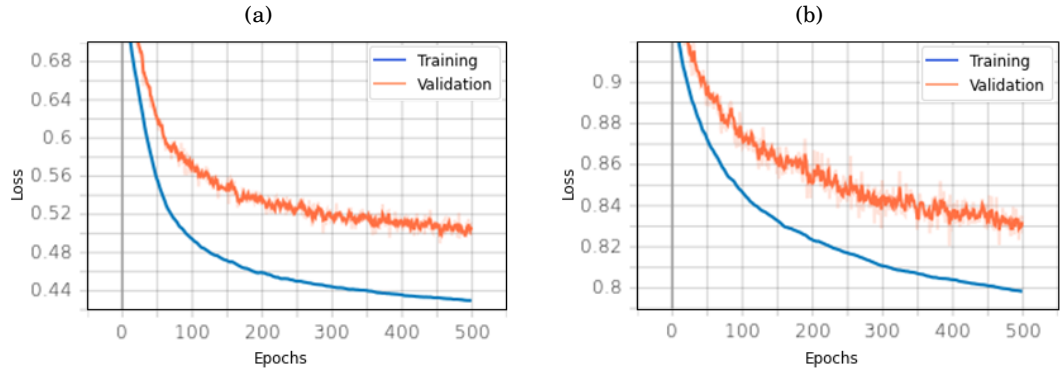

FIGURE 4.4. (a) Depicts the epoch loss graph for the 2 seed cell centre model, where the graph was produced by *Tensorflow* during training. (b) Shows the epoch loss graph for the 5 seed cell centre model. Both graphs show convergence towards a lower loss; thus, illustrating that the reconstruction models are learning with a CNN architecture (for schematic see Figure 3.5).

From the graphs, it can be deduced that the models are in fact learning as both models are converging towards a lower loss value. Although the loss curve shows that full convergence has yet to be reached, the results depict the viability of the method stated. This is likely to be resolved if a larger dataset can be used; however, due to computational restrictions, a larger dataset has been avoided (see Section 3.2.2).

Following the training of the CNN, the material maps have been predicted using the travel time inputs with the trained models. As the number of predicted outputs per ToF input from a 1D convolutional layer depends on the kernel size, an initial evaluation with a summed output

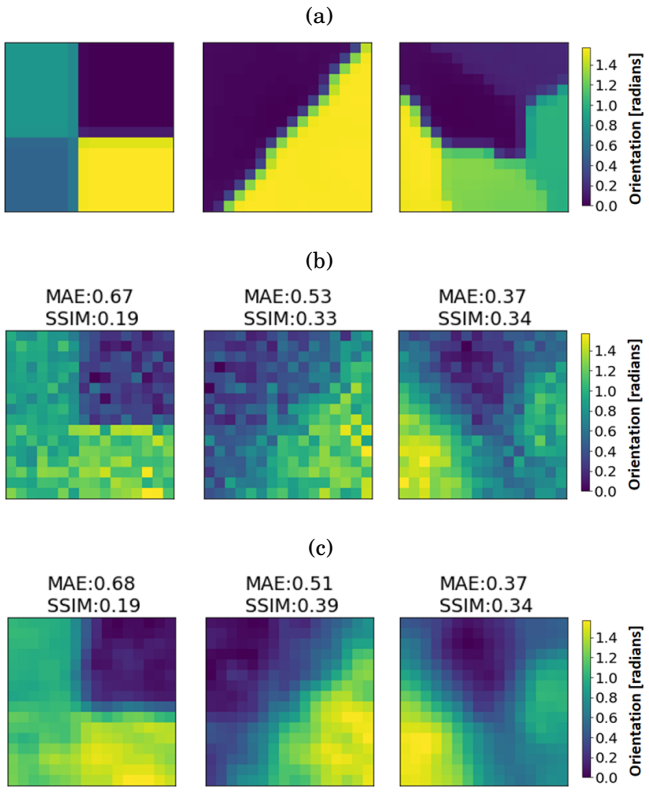

FIGURE 4.5. (a) Illustrates examples of the true orientation maps for the 4 seed, 2 seed and 5 seed models going from left to right (network architecture shown in Figure 3.5, the hyperparameters used to train the model are depicted in Table 3.3), where the ToF data will be generated from this map accordingly. (b) Depicts the corresponding predicted orientation maps using a CNN, where (c) shows the output predictions after Guassian smoothing.

feature image of the total kernel outputs and an averaged output image has been evaluated [17, 19]; however, as both approaches generated poor results, it has been concluded the most sensible approach would be to obtain the image with the lowest MAE from the predicted kernel outputs by indexing. The lowest MAE image from the predicted outputs per ToF input approach has been undertaken as this was the loss function used to train the model, such that this is deemed the most appropriate metric. Following the lowest MAE search for each ToF input, the predicted outputs will now show the best predictions from the multiple kernel outputs. Examples of the outputs of all the models are shown in Figure 4.5. Figure 4.5(a) depicts the true orientation map, where the 4 seed model, 2 seed model and 5 seed model are shown going from left to right. Figure 4.5(b) illustrates the prediction outputted by the trained CNN model outputs; as the single pixel prediction outputs a grainy image output, the images have been processed with a Guassian kernel of size 3x3 to obtain a smoothed out image. This is typically a technique used to smooth out camera sensor outputs to reduce noise, achieved by averaging and removing outlier

pixels[14]; thus, the same principle will be applied on the CNN outputs using *GaussianBlur* from the OpenCV package [14]. Upon processing the images with the Gaussian smoothing, the results showed a visual improvement of the similarity between the true orientation maps against the smoothed predicted maps; examples of this is depicted in Figure 4.5(c). Further examples of predictions of the material maps are shown in Appendix A.3.3 for the 2 seed and 5 seed models, where examples of the 4 seed are illustrated in Appendix A.3.1.

As the ROC curve approach to evaluate the performance isn't applicable to this problem due to the zero accuracy issue, different metrics will be necessary. Thus, the two metrics introduced to evaluate the models between the true orientation map against the predicted orientation map are MAE and structural similarity index measure (SSIM), where these values are shown above the predictions within Figure 4.5. MAE will be used as it demonstrates the discrepancies between the true and predicted values; the output values has a range of MAE ≥ 0, where a value of 0 depicts a prediction with no errors [16, 19]. Whereas, SSIM will be used as a metric to evaluate the perceived similarity, where this measure considers luminescence, contrast and structure, where an SSIM value of 1 depicts a perfect prediction [58, 59]. Although the MAE values from Figure 4.5 suggests that the overall performance of the predictions are more accurate if the material map is more complex, such that there are more seeds in a random pattern, this may not be representative of the entire dataset. Thus, a histogram of MAE and SSIM values from optimised kernel predictions are generated to summarise the performance of the predictions, depicted in Figure 4.6(a) and Figure 4.6(b) respectively. See Appendix A.3.1 for the 4 seed version.

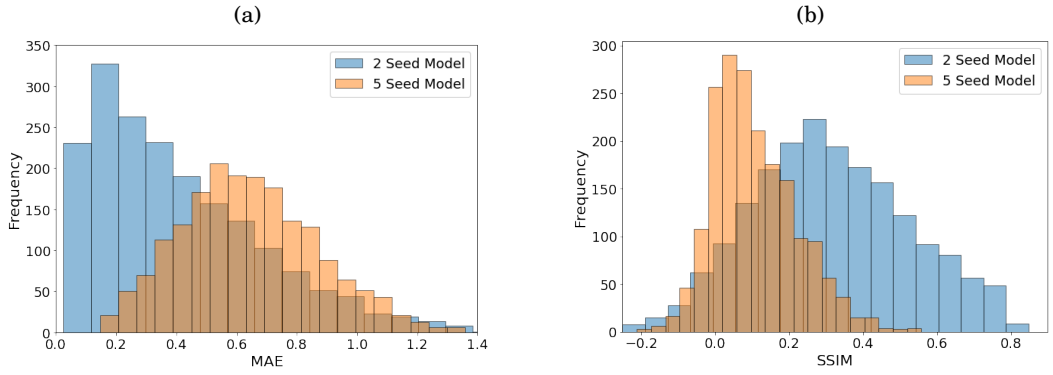

FIGURE 4.6. Histogram of MAE and SSIM shown in (a) and (b) respectively, generated using the 2 seed and 5 seed cell centre model predictions from CNN (see relevant schematic in Figure 3.5 and hyperparameters used to train the model in Table 3.3). Lower values MAE depicts a better performing prediction, whereas a higher value of SSIM suggests a better performing prediction.

From the distributions shown in Figure 4.6(a) and Figure 4.6(b), it can be deduced that generally the fewer seeds the model expects, the greater the performance of the predictions are. This is demonstrated by the skew towards 0, with an average of around 0.2 MAE with the 2 seed model,

where the MAE from the 5 seed model is clearly skewed towards larger values of MAE, averaged around 0.6. Furthermore, the SSIM scores from the 2 seed model is also evenly distributed around 0.3, ranging to 0.8; whereas the 5 seed model is averaged around an SSIM score of only 0.1, with the majority of the data distributed around the lower regions. The SSIM performance on the 5 seed model is particularly poor likely due to the increased complexity in the structure, resulting in more changes to the map as there are more boundaries within the area; thus, likely inducing a lower SSIM despite visually producing reasonable results. Nevertheless, the summarised results are as predicted, where more seeds would induce a lower accuracy; however, the results are capable of demonstrating the significance of the change in difficulty in the models.

To further assess the performance of the models, the trained CNN models will predict the material maps with two different types of noise within the input ToF data. The two noise types added will be multiplicative and additive, where the process of adding the noise to the input data is described within Section 3.2.3. Examples of the predicted orientation maps with multiplicative noise is illustrated in Figure 4.7, where examples of additive noise predictions are shown in Figure 4.8. The results from the figures suggest that predictions perform significantly worse under additive noise inputs, where the predictions become similar regardless of the true map. This is likely due to the difference between multiplicative and additive noise, where additive noise fills a specified amount of values with the minimum and maximum range values (-π to π in this case), typically generating black and white pixels [56]. Thus, for the current problem, as the majority of ToF input data is filled with minimum and maximum range values, the predicted output is likely constant as the noise scale increases, regardless of the true map. Furthermore, the results influenced under multiplicative noise indicates the method with CNN predictions is robust under the influence of multiplicative noise, whereas the effectiveness dropped significantly during the flaw classification problem (see Section 4.1). Although the accuracy of the predictions may be considered low under 75% noise (shown in Figure 4.7(e)), the rough outline of the structure is still visible. Furthermore, it should be noted that 75% noise is a high level of noise and is not likely an accurate representation of a real-life scenario. In addition, by introducing industrial computational resources, the accuracy is likely to further improve with additional training; thus, indicating the potential of using ML for NDT. However, although the Figure 4.7 suggest the overall performance of the predictions under multiplicative noise is satisfactory, where the method is less robust under additive noise shown in Figure 4.8; this may not be representative of the entire dataset. Thus, following the previous approach, a further evaluation will be undertaken with a histogram of MAE and SSIM values to summarise the performance of the entire dataset under a specified noise level. The generated histograms of MAE and SSIM for 50% noise levels are shown in Figure 4.9(a) and Figure 4.9(b) for the 2 seed model respectively, where Figure 4.9(c) and Figure 4.9(d) depicts the results for the 5 seed model respectively. However, due to the immense amount of data for both types of noise and different noise levels, multiple histograms have been generated for clarity; see Appendix A.3.6 for more generated data.

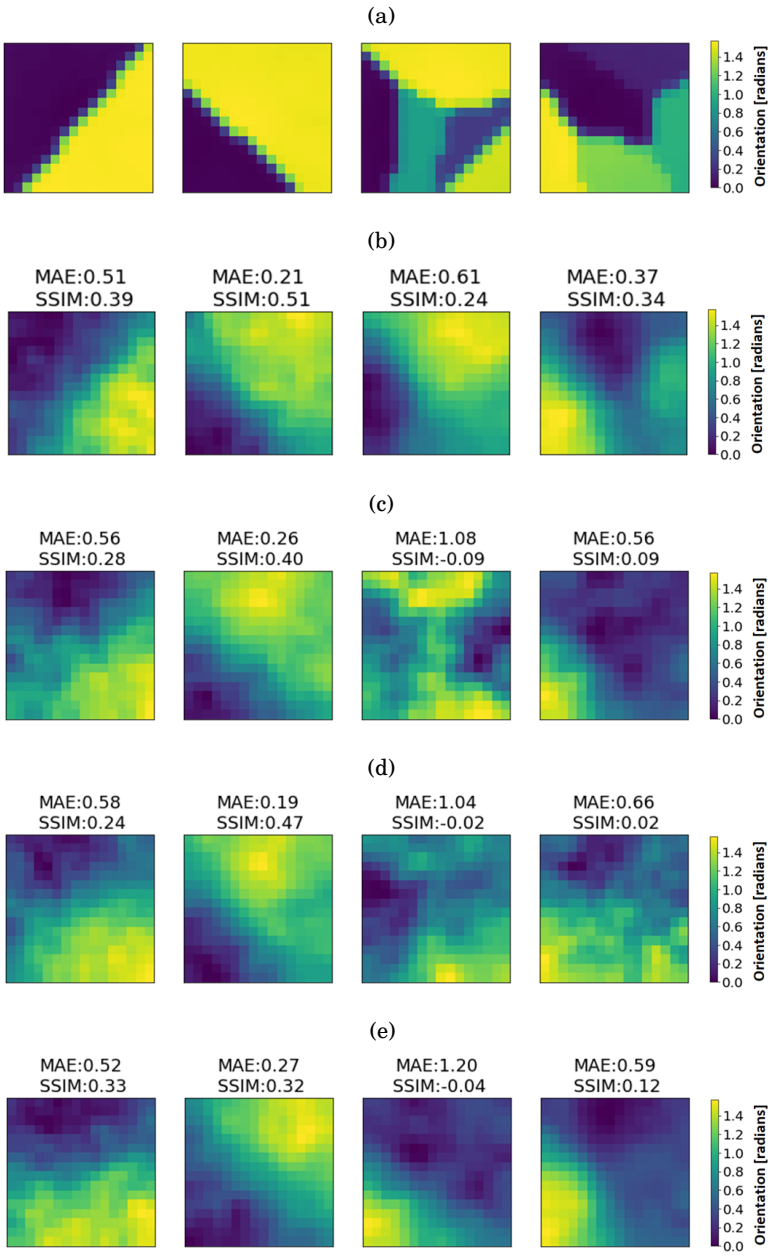

FIGURE 4.7. Illustration of the effect on CNN reconstruction predictions when multiplicative noise is introduced within the input data. (a) Denotes the true orientation maps, where the examples on the left are the 2 seed models and the examples on the right illustrates the 5 seed examples. (b) Shows the corresponding prediction of the orientations maps generated using the tuned CNN model (network architecture shown in Figure 3.5, where the corresponding hyperparameters used are depicted in Table 3.3). (c)-(e) Illustrates the effect of multiplicative noise on the corresponding predictions of the orientation maps, where the noise scale used were 25%, 50% and 75% respectively.

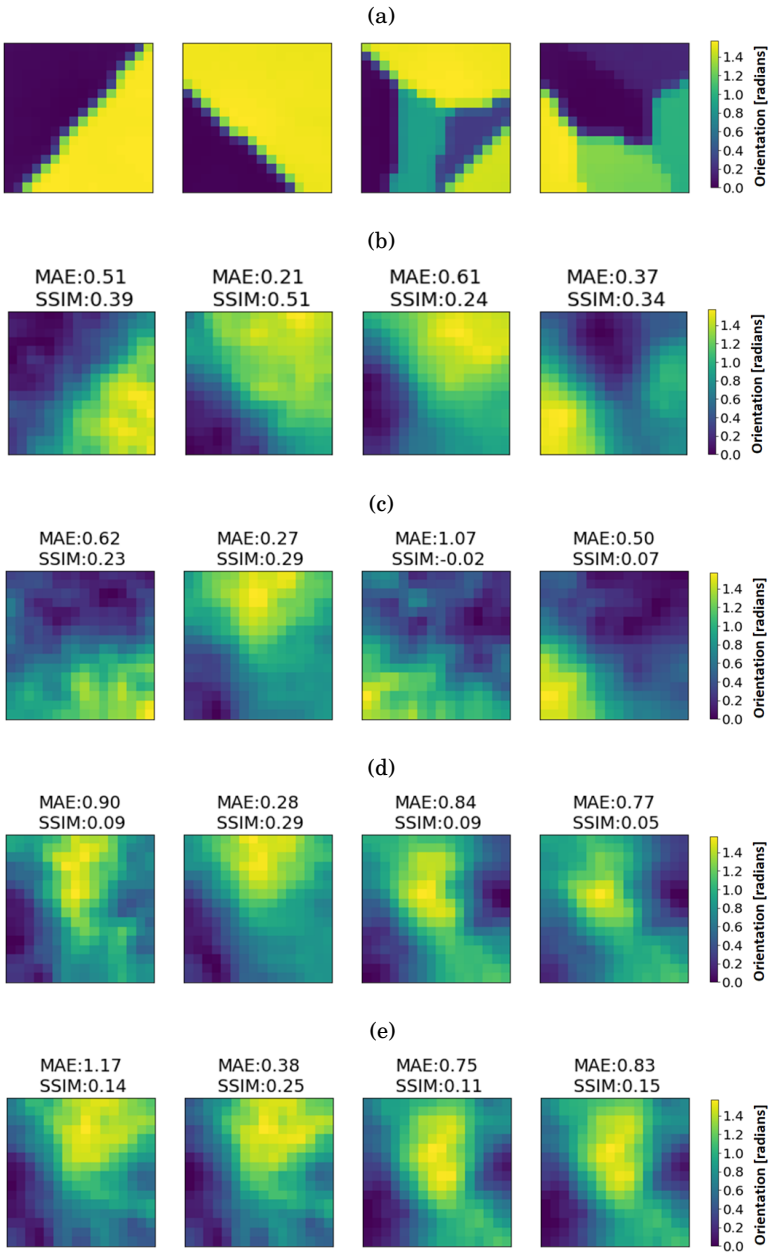

FIGURE 4.8. Illustration of the effect on CNN reconstruction predictions when additive noise is introduced within the input data. (a) Denotes the true orientation maps, where the examples on the left are the 2 seed models and the examples on the right illustrates the 5 seed examples. (b) Shows the corresponding prediction of the orientations maps generated using the tuned CNN model (network architecture shown in Figure 3.5, where the corresponding hyperparameters used are depicted in Table 3.3). (c)-(e) Illustrates the effect of additive noise on the corresponding predictions of the orientation maps, where the noise scale used were 25%, 50% and 75% respectively.

The distributions shown in Figure 4.9(a) and Figure 4.9(b) further support the previous evaluation of the CNN performance of how the method is robust under multiplicative noise; whereas, the method is more susceptible under additive noise, affecting the accuracy of the results as shown by the higher MAE and lower SSIM. Interestingly, the robustness of the CNN predictions under multiplicative noise appears to be more noticeable only in the 2 seed model as opposed to the 5 seed model, shown in Figure 4.9(c) and Figure 4.9(d). Nevertheless, the same scale on the distributions illustrates the difference in performance clearly, where the 2 seed model performs significantly greater than the 5 seed model again. Regardless, the reduced performance (lower MAE and higher SSIM) of both models with the addition of noise suggests that the CNN had greater predictions beforehand; thus, indicating that the predictions from the CNN are not random and relate to the true orientation map.

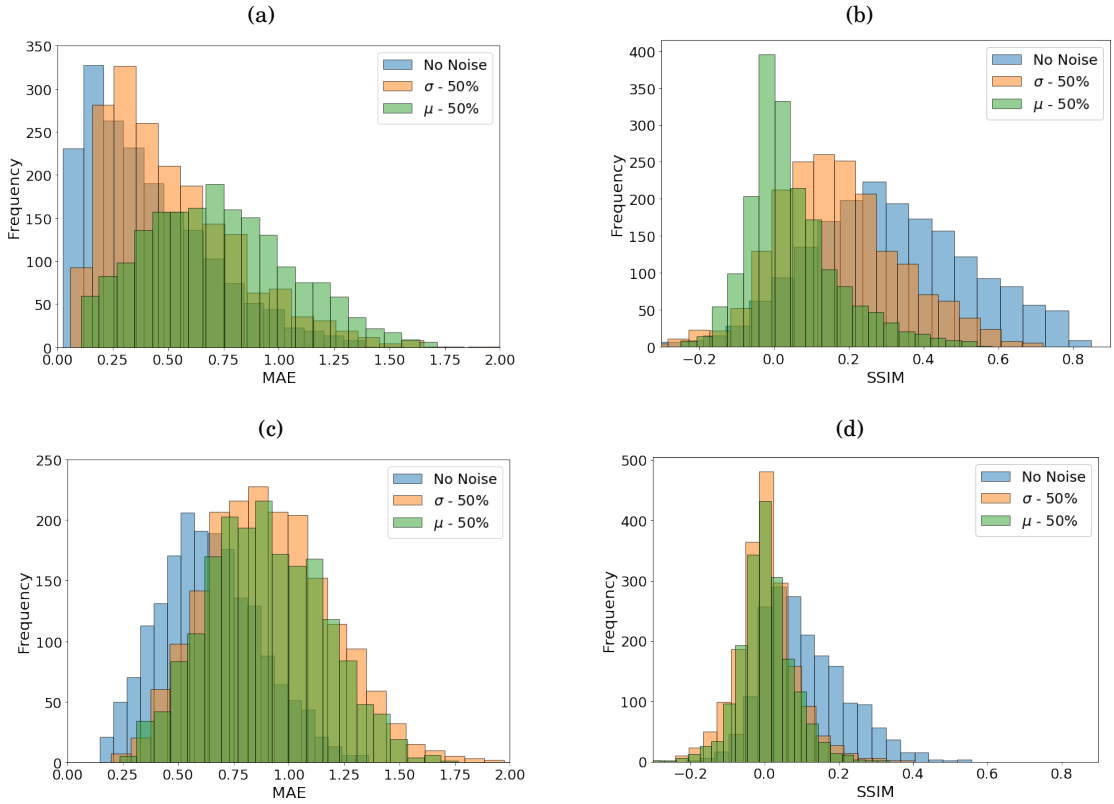

FIGURE 4.9. Histograms generated of MAE and SSIM values of the predicted orientation maps for the 2 seed model shown in (a) and (b) respectively; whereas (c) and (d) depicts the 5 seed model of MAE and SSIM respectively. The histogram shows the data of the predictions under no noise, with multiplicative noise (σ) and with additive noise (μ) at a level of 50% (see Section 3.2.3 for the method used to introduce noise levels).

As the histograms only represent performance at one noise level, an evaluation of the predicted orientation maps for the test dataset has been undertaken across 5% noise level increase increments to represent the entire test dataset. The MAE and SSIM data will be averaged at each increment to obtain an accurate representation of the performance, illustrated in Figure 4.10(a) and Figure 4.10(b) for the multiplicative noise; whereas the results from additive noise are depicted in Figure 4.10(c) and Figure 4.10(d).

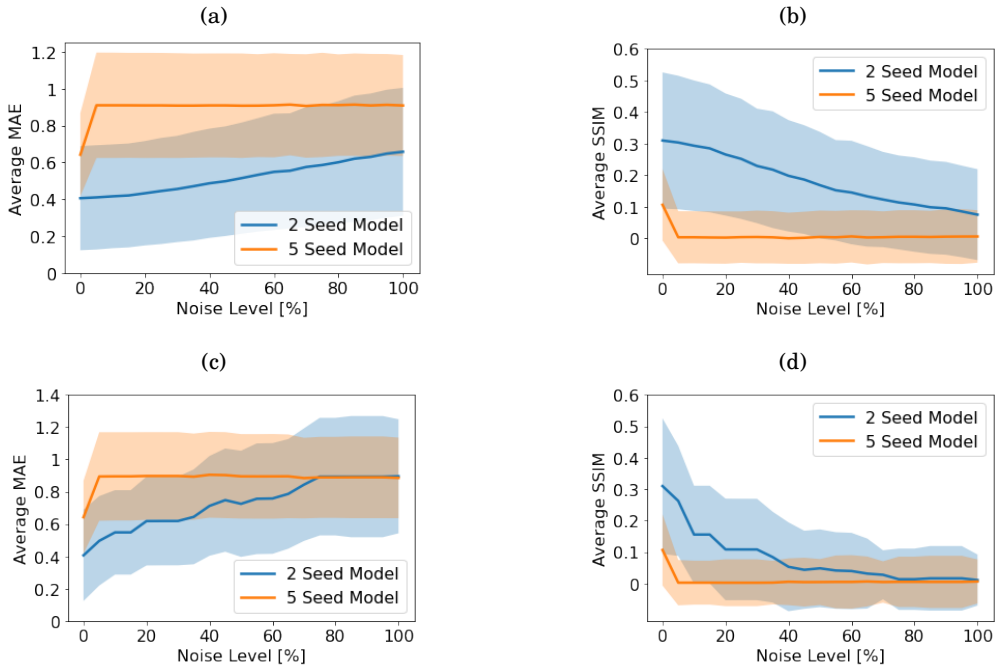

FIGURE 4.10. Evaluation of the effect of multiplicative noise on MAE and SSIM, shown in (a) and (b) respectively, where the values are averaged across the evaluated test dataset. The same evaluation is illustrated within (c) and (d) but with the effect of additive noise. The lighter bands on the graph depict the standard deviation, calculated across the test dataset at increasing 5% increments of noise.

The standard deviation bands from the results shown in Figure 4.10 illustrates the spread of the data at each noise level increment (such as the spread shown in the histogram distribution in Figure 4.10), where the spread suggests the distributions are relatively consistent at all levels of noise. Thus, the performance change is shown to be consistent throughout each increase of noise level increments, indicating the consistence of the trained model's performance. The results from Figure 4.10 suggests that the 5 seed model is highly susceptible to either type of noise; this is shown from the sudden peak in lower performance (higher MAE and lower SSIM), where the performance then stays consistent. Conversely, the 2 seed model shows a gradual decrease in performance with multiplicative noise and more sudden changes to performance under the influence of additive noise. Furthermore, the MAE of the 2 seed model is capable of sustaining a

low MAE over high multiplicative noise levels; in addition, the gradual decrease in SSIM suggests the structure is still pertained visually to the user which is ideal for practical use. Thus, it can be deduced that the shown method with CNN predictions is more susceptible to complex orientation maps. However, the method proves to be robust to multiplicative noise and less susceptible to additive noise compared to more complex orientation maps.

As the complexity of orientation maps within real materials is bound to be significantly higher, the practicality of CNN for accurate real-time predictions of material maps within NDT is questionable, even though high prediction speeds are present (requiring less than a minute to predict entire test dataset). Nevertheless, the visible change in accuracy of the performance between simpler and more complex models, highly suggests accurate predictions can be made with CNN. Furthermore, the low accuracy may be related to the lack of training data, where this is restricted within the presented implementation due to runtime disconnections (see Section 3.2.2); due to the more complex nature of the problem, it is highly likely that this model requires more data for accuracy improvements. Thus, with further improvements to the current method for higher accuracy, fast real-time inspection predictions may be viable using ML for NDT. However, further changes and implementations need to be made for the applicability of this method to more complex orientation maps, where the accuracy is likely capable of increasing upon improvements to the presented method; potential directions to tackle the complexity and extensions to this work are discussed within Section 5.

4.3 Stakeholder Relevance

The evaluated results for the flaw classification problem with ML (discussed in Section 4.1) suggests the current method is not suitable as a replacement for flaw inspections. Although fast predictions with high accuracy can be produced, the underlying issue of poor performance under the influence of little noise is too significant for practical use within NDT. The results obtained for the reconstruction problem (discussed in Section 4.2) illustrates a potentially robust method under the influence of multiplicative noise to predict orientation maps inside a material. However, as high performance is only present with simple material maps, the current method is not suitable for inspections within NDT as of yet. However, for both investigated problems, the excellent performance shown without noise highly suggests with further improvements to the presented methods, practical use for inspections within NDT in the future is highly possible.

4.4 Code Availability

To ensure the results discussed within this chapter can be reproduced, the code used to obtain the results are available at:

`https://github.com/RaymondWKWong/Seeing-the-Unseen`

CHAPTER 5

CONCLUSION AND FURTHER WORK

Current inspection methods within non-destructive testing (NDT) for flaw detection is sufficient; however, the process can be strenuous for the inspector when the volume of data is vast [3, 16, 48]. Such repetitive pattern recognition tasks are an ideal task for machine learning. Introducing this technology can potentially assist by inspecting thousands of data within seconds or allow secondary checks for the user [48]. However, flaw detection is not the only interest within the NDT industry, there is also a huge desire to look inside objects that are optically opaque [3, 16, 48]. This can be achieved by tomography using ultrasound waves by reconstructing the material map. Although this can be achieved with ultrasound tomography, current methods have many disadvantages such as long computation times and the incapability to provide real-time imaging, a feature highly desired in the NDT industry [12]. Thus, the report presents an in-depth evaluation of the viability of introducing machine learning to NDT.

The tasks undertaken focuses on specifically using convolutional neural networks (CNN) due to their notorious feature of performing exceptionally well with images and signal data [17, 19, 40]. The first task executed involves two classification problems, the first is a flaw detection problem, where the model intakes an ultrasonic inspection image, taking advantage of the online database USimgAIST to train and test the model [36]. The second task involves a further step of the model predicting the flaw size. The results outline exceptional accuracy and fast predictions using CNN; however, the method is susceptible to small amounts of noise. This suggests further improvements to the current method is necessary before it is suitable for inspections in NDT.

The second task involves using CNN for the reconstruction of orientation maps within a material using synthetic data. The results suggests the method is capable of producing rapid reconstructions with some accuracy. Furthermore, the results shows the method is robust to some added noise within the input data, ideal for practical use. However, the presented method is susceptible to added noise when the true orientation map is more complex. Thus, the current method may not be suitable for NDT inspections as real material maps are significantly more complex. However,

the results illustrates that CNN is capable of reconstructions. Thus, with further implementations and greater computational resources, higher accuracy is likely achieved, such that machine learning may potentially be suitable for real-time practical inspections in the future.

5.0.1 Novel Project Contributions

The novelty presented of the flaw detection research undertaken contributes as an extension towards the existing literature by Ye et al. [36], where the reconstruction work contributes as an extension towards the work demonstrated by Bourne et al. [12] in the following ways:

- Constructing a distinct CNN to solve flaw classification problems as opposed to utilising pre-designed networks such as AlexNet and VGGNet as demonstrated by Ye et al. [36]
- Extended the work shown by Ye et al. [36] by further evaluating the performance of CNN under the influence of noise to simulate more realistic scenarios, as opposed to evaluating data obtained from rigorously controlled environments alone
- Implementing novel CNN to predict material maps to attempt at resolving current underlying issues with reconstruction methods presented by Bourne et al. [12]
- Adapting and deploying hyperparameter tuning and training model techniques (see Section 4.3 for Python scripts) for the flaw classification networks (see Figure 3.2 and Figure 3.3 for schematic) and the reconstruction network (see Figure 3.5 for schematic)
- Constructing working code for processing USimgAIST dataset [36] to simulate multiplicative and additive noise data; furthermore, constructed code to simulate the same types of noise within the synthetic ToF data generated for the reconstruction problem (see Section 4.3 for the full details of the 3 constructed Python scripts)
- Implemented Python scripts to process obtained results for performance evaluation using ROC curves, MAE and SSIM histograms and visual images of reconstructions and noisy data, refer to Section 4.3 for the working code

5.0.2 Further Work

As the report focuses mainly on CNN for flaw classifications and reconstructions, future research could explore the potential with different approaches. This could include dense neural nets and generative adversarial networks; the potential of these networks are discussed thoroughly within Section 2.4.1. Due to computational resource limitations experienced with timeouts (see Section 3.1.2 and Section 3.2.2), this may have resulted in lower accuracies; thus, further work may be explored by expanded computational resources with longer training times, allowing training with larger datasets. Furthermore, the synthetic data generated takes advantage of a 16x16 grid with 2 transducers on each side to hasten processing times. An expansion on this work to attempt at increasing accuracy may be implemented by allowing a larger grid and higher definition by increasing the number of transducers.

APPENDIX

A

FURTHER EXPERIMENTAL DETAILS

A.1 Flaw Classification

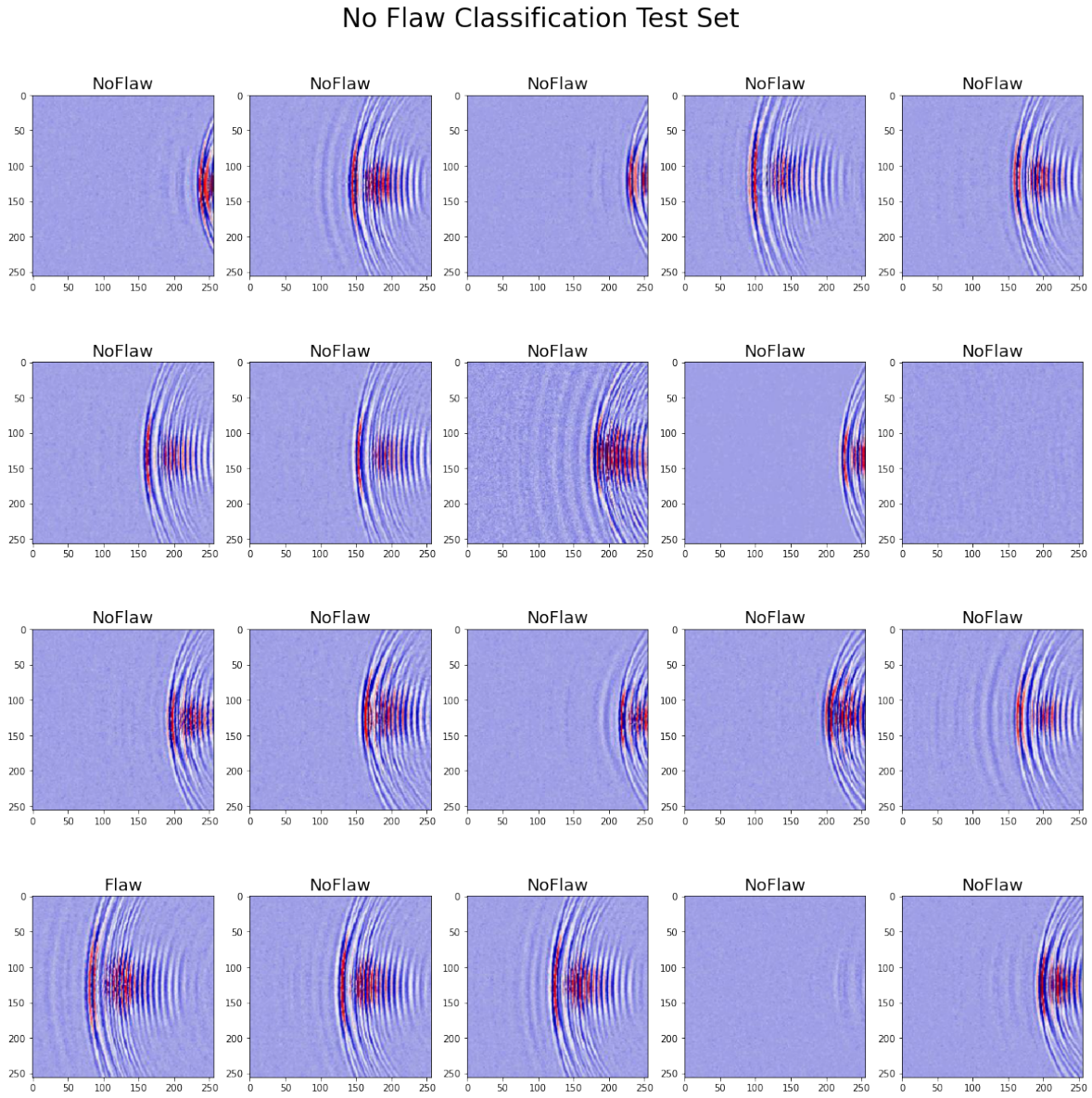

FIGURE A.1. Results from the binary classification model (see schematic in Figure 3.2), testing with 20 example classifications on the "no flaw" test set, scoring 95% validation accuracy.

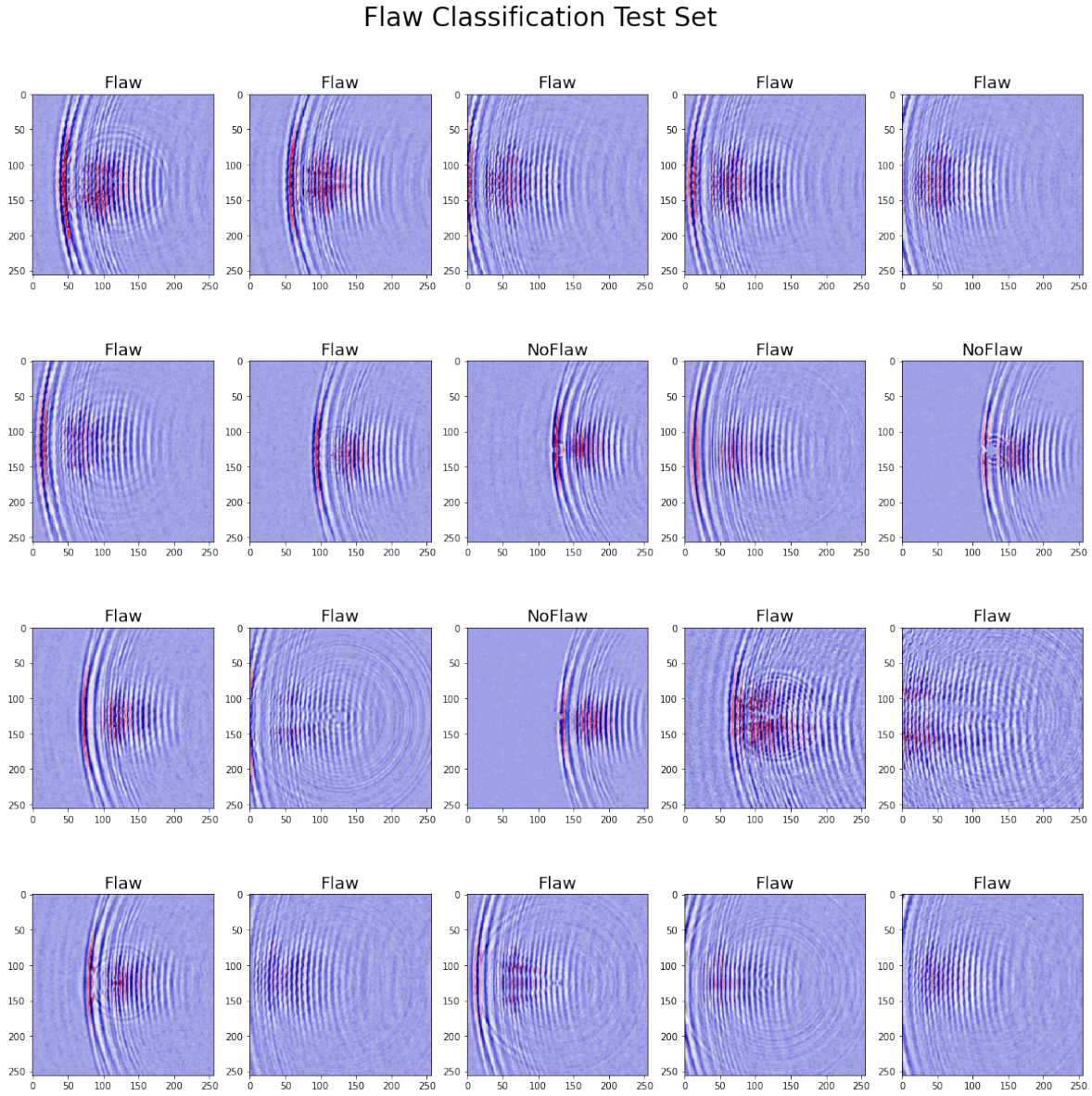

FIGURE A.2. Results from the binary classification model (see schematic in Figure 3.2), testing with 20 example classifications on the "flaw" test set, scoring 85% validation accuracy.

A.2 Flaw Size Classification

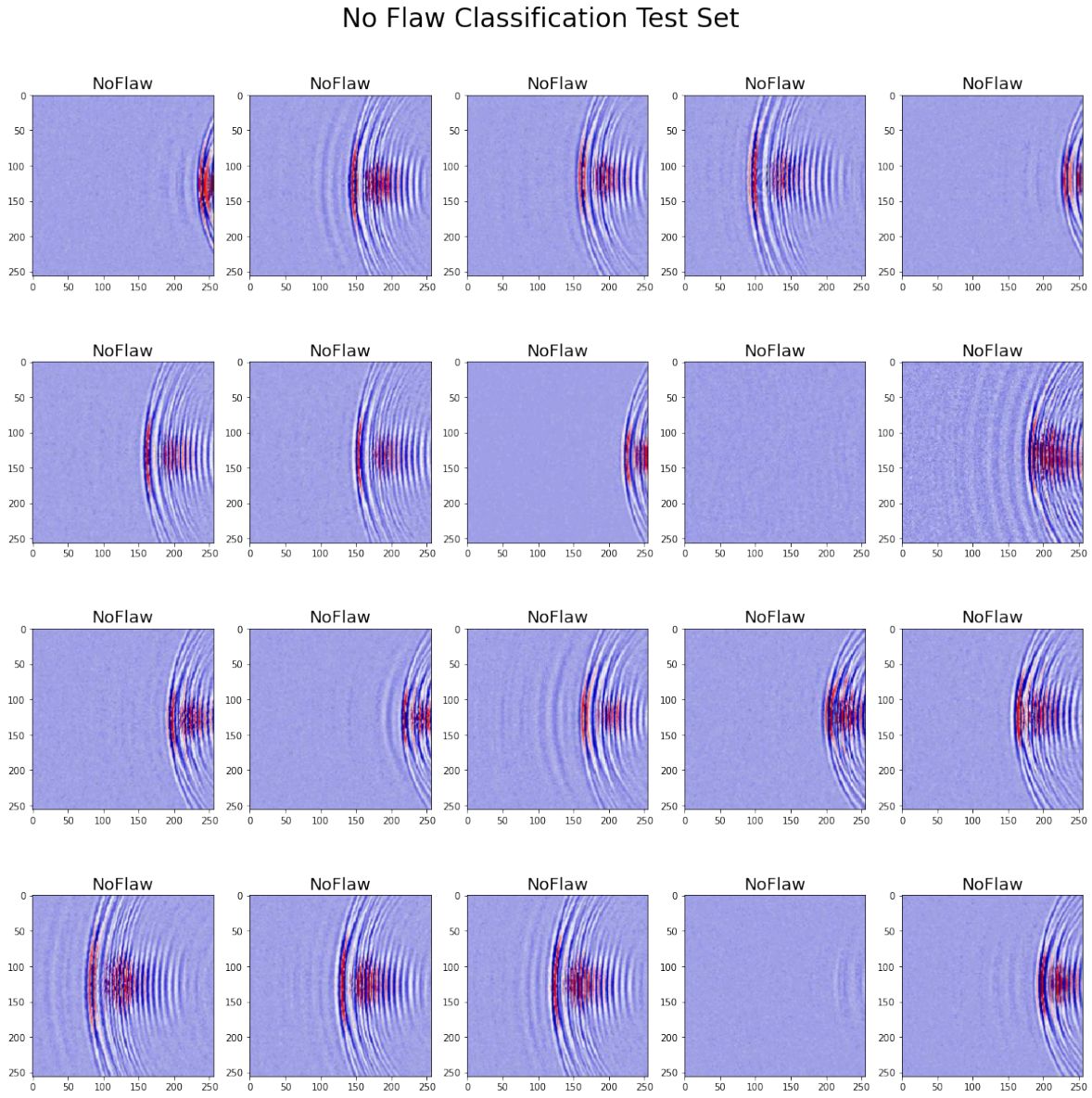

FIGURE A.3. Results from the flaw size classification model (see schematic in Figure 3.3), testing with 20 example classifications on the "no flaw" test set, scoring 100% validation accuracy.

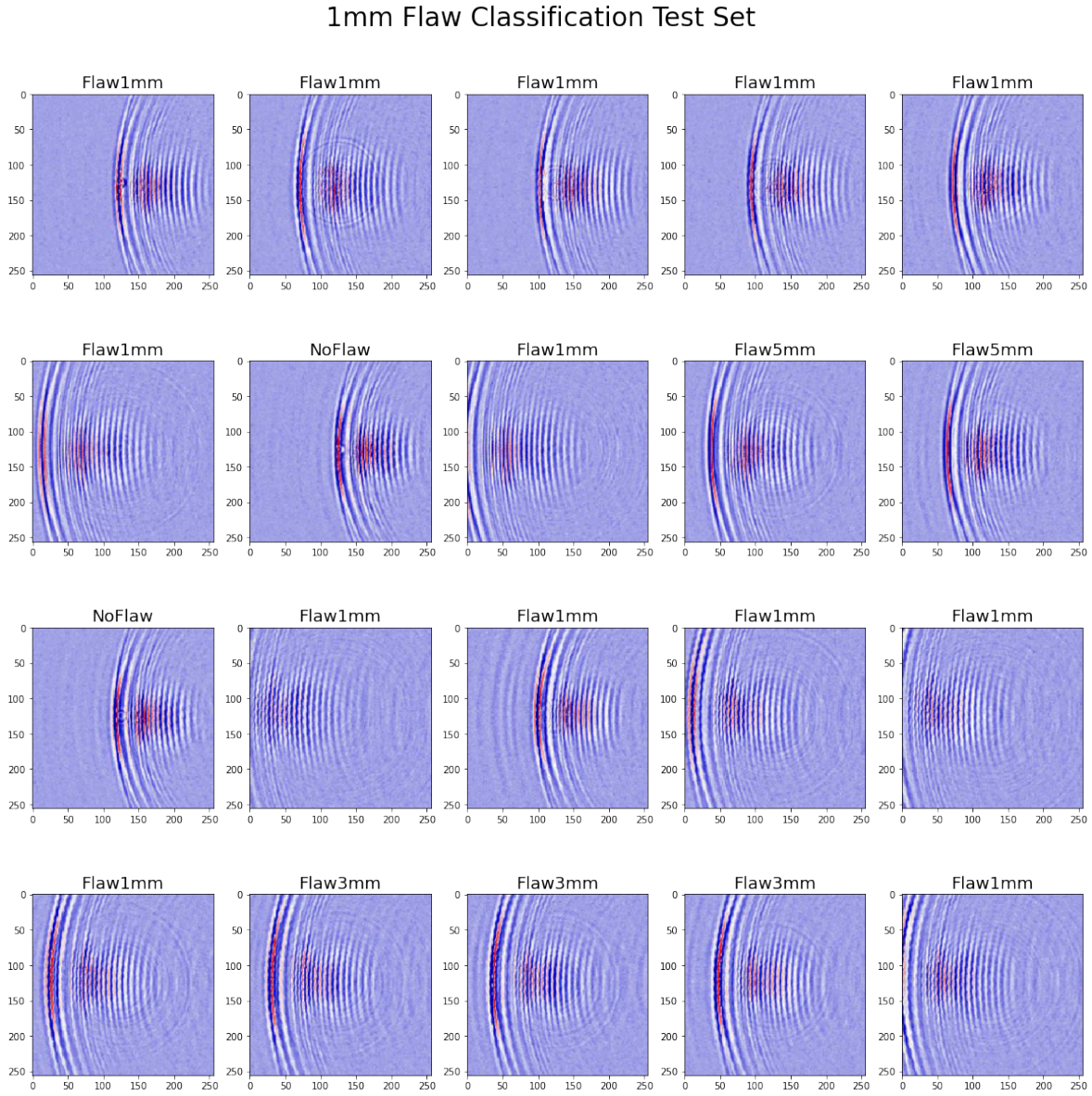

FIGURE A.4. Results from the flaw size classification model (see schematic in Figure 3.3), testing with 20 example classifications on the "flaw 1mm" test set, scoring 65% validation accuracy.

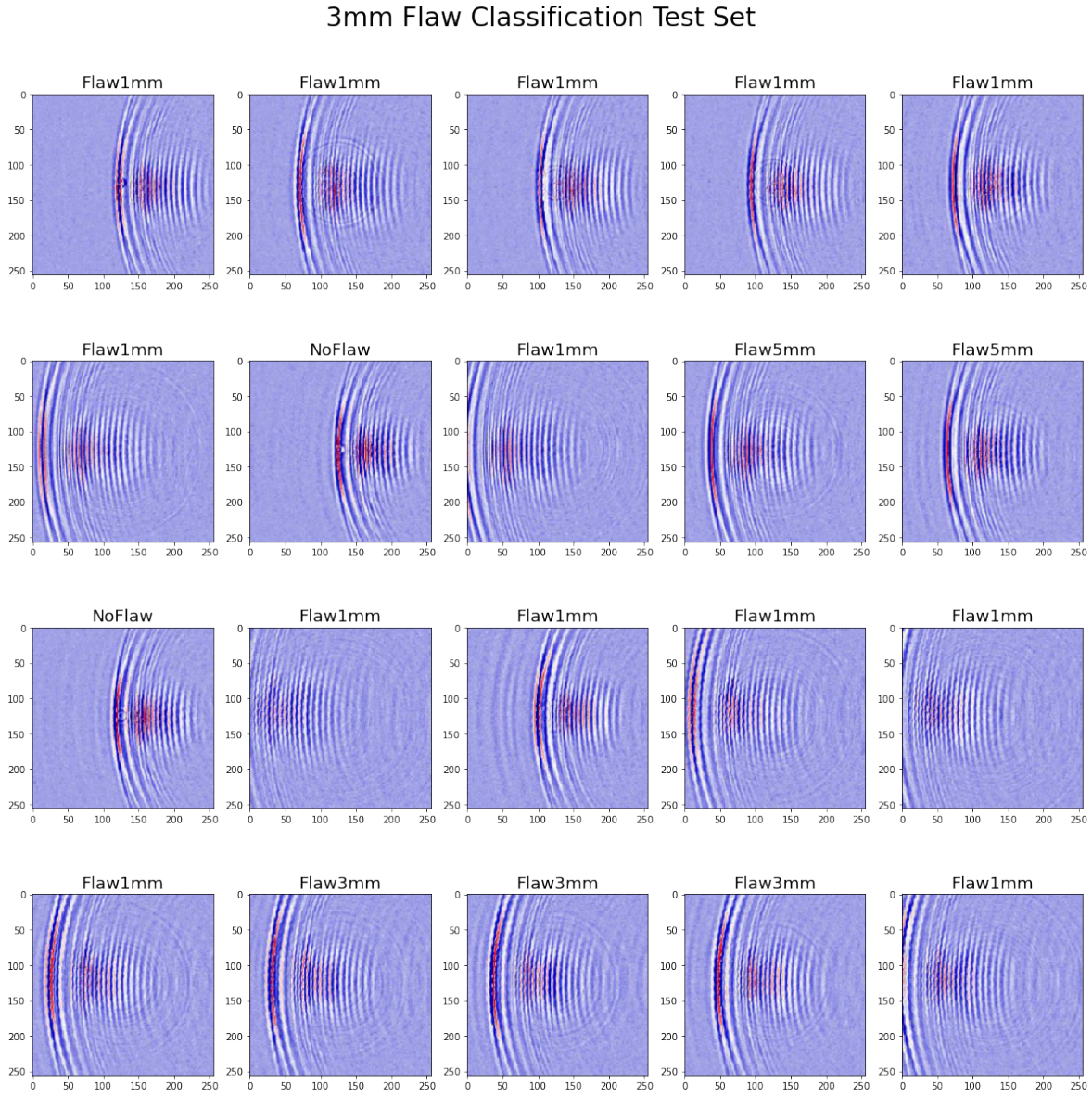

FIGURE A.5. Results from the flaw size classification model (see schematic in Figure 3.3), testing with 20 example classifications on the "3mm flaw" test set, scoring 15% validation accuracy.

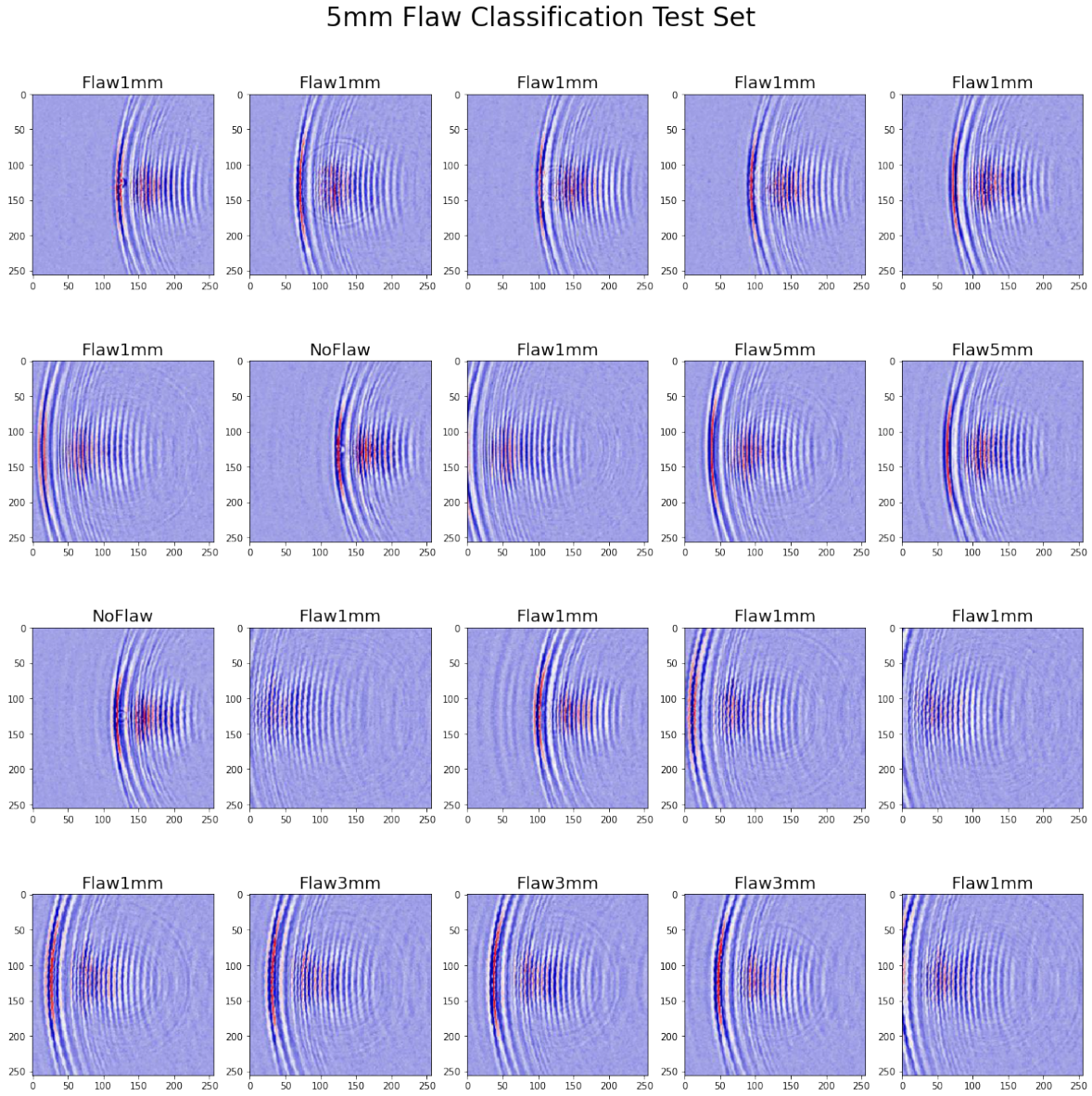

FIGURE A.6. Results from the flaw size classification model (see schematic in Figure 3.3), testing with 20 example classifications on the "5mm flaw" test set, scoring 10% validation accuracy.

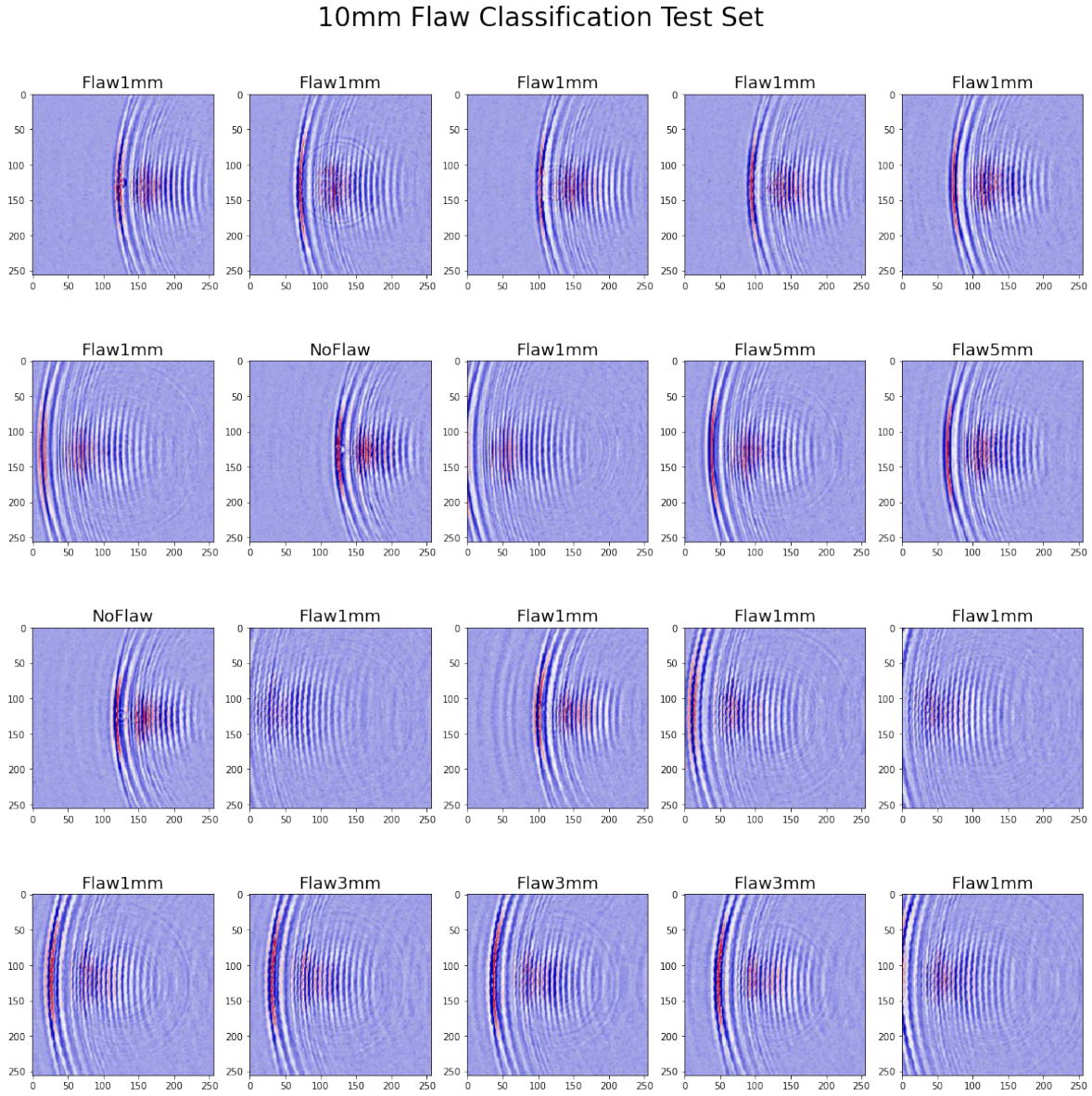

FIGURE A.7. Results from the flaw size classification model (see schematic in Figure 3.3), testing with 20 example classifications on the "10mm flaw" test set, scoring 0% validation accuracy.

A.3 Reconstruction Problem

A.3.1 4 Seed Model Evaluation Data

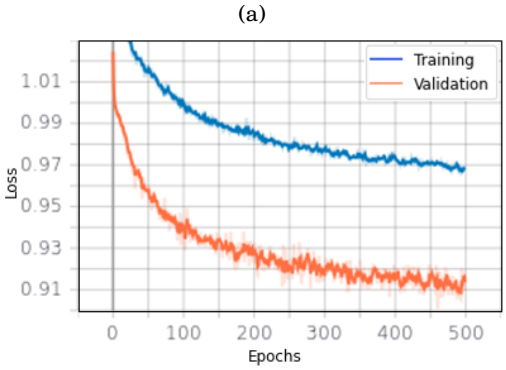

FIGURE A.8. (a) Depicts the epoch loss graph for the 2 seed cell centre model, where the graph was produced by *Tensorflow* during training. The graph shows convergence towards a lower loss, suggesting that the reconstruction models are learning when using a CNN architecture (for schematic see Figure 3.5).

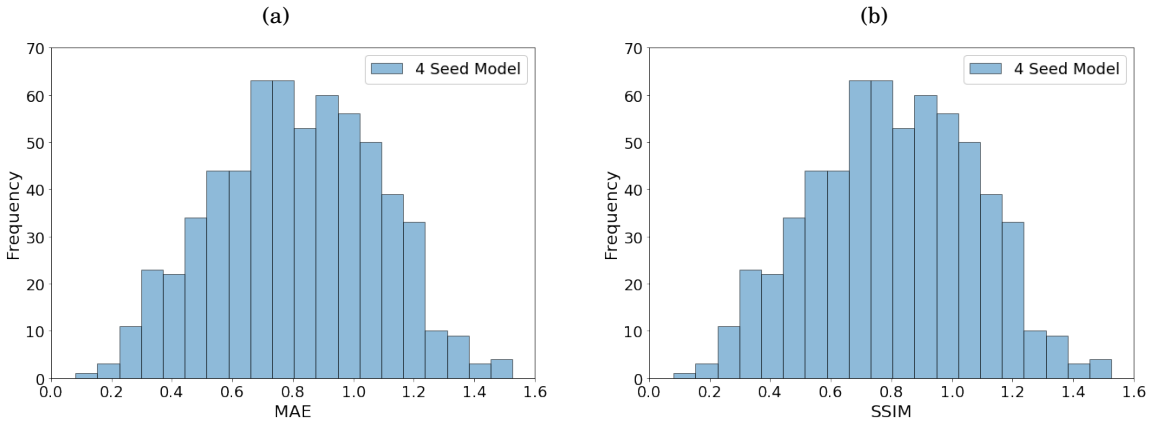

FIGURE A.9. Histogram of MAE and SSIM shown in (a) and (b) respectively, generated using the 4 seed cell centre model predictions from CNN (see relevant schematic in Figure 3.5 and hyperparameters used to train the model in Table 3.3). Lower values MAE depicts a better performing prediction, whereas a higher value of SSIM suggests a better performing prediction.

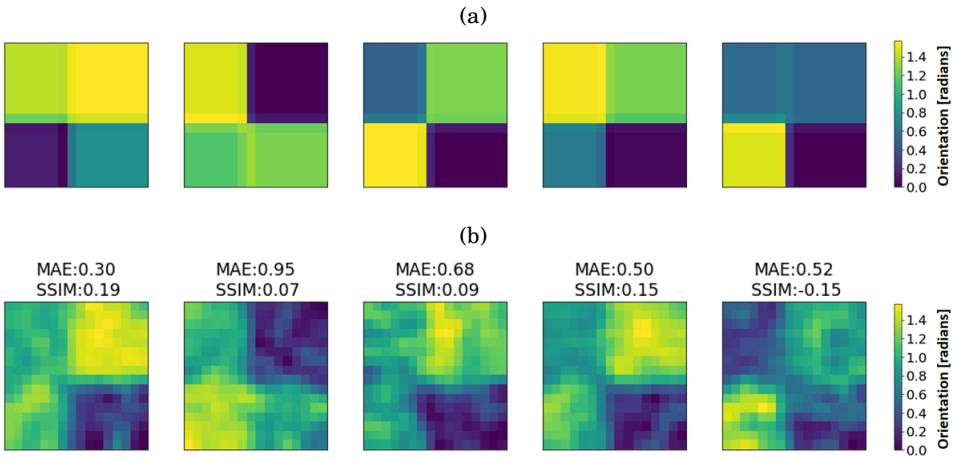

FIGURE A.10. (a) Illustrates examples of the true orientation maps for the 4 seed models (network architecture shown in Figure 3.5, the hyperparameters used to train the model are depicted in Table 3.3), where the ToF data will be generated from this map accordingly. (b) Depicts the corresponding predicted orientation maps using a CNN after Gaussian smoothing. As the results suggests the CNN model is capable of recognising the square features, progression towards the random 2 seed and 5 seed models will be undertaken as the code appears to work.

A.3.2 10 Seed Predicted Orientation Map Examples

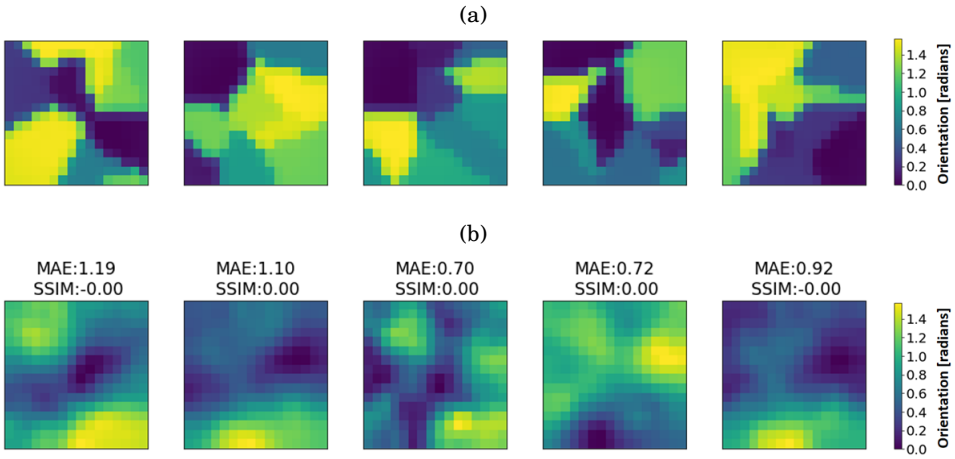

FIGURE A.11. (a) Illustrates examples of the true orientation maps for the 10 seed and model (network architecture shown in Figure 3.5), where the ToF data will be generated from this map accordingly. (b) Depicts the corresponding predicted orientation maps using a CNN after Gaussian smoothing. The results suggests poor performance with the 10 seed model; thus, less seeds will be used to evaluate the proposed method.

A.3.3 Further Predicted Orientation Map Examples

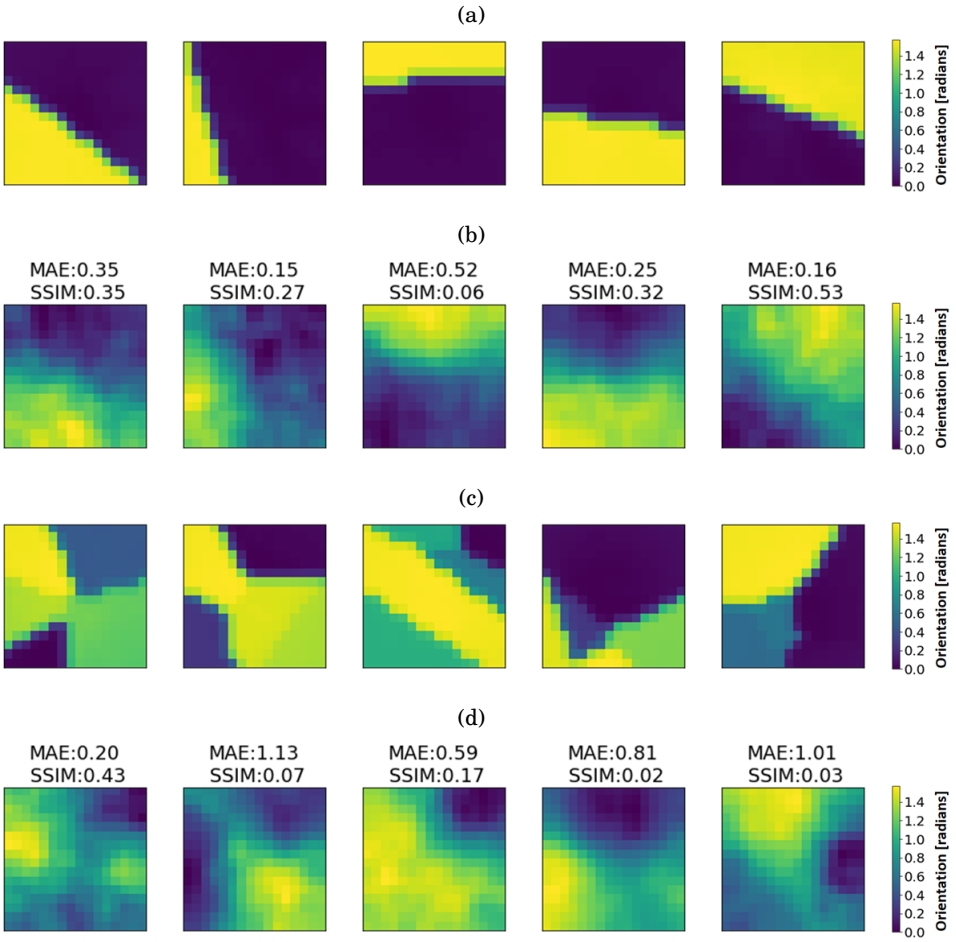

FIGURE A.12. (a) And (c) illustrates examples of the true orientation maps for the 2 seed and 5 seed models respectively (network architecture shown in Figure 3.5, the hyperparameters used to train the model are depicted in Table 3.3), where the ToF data will be generated from this map accordingly. (b) and (c) depicts the corresponding predicted orientation maps using a CNN after Gaussian smoothing.

A.3.4 Effect of Noise on 2 Seed Reconstruction Model

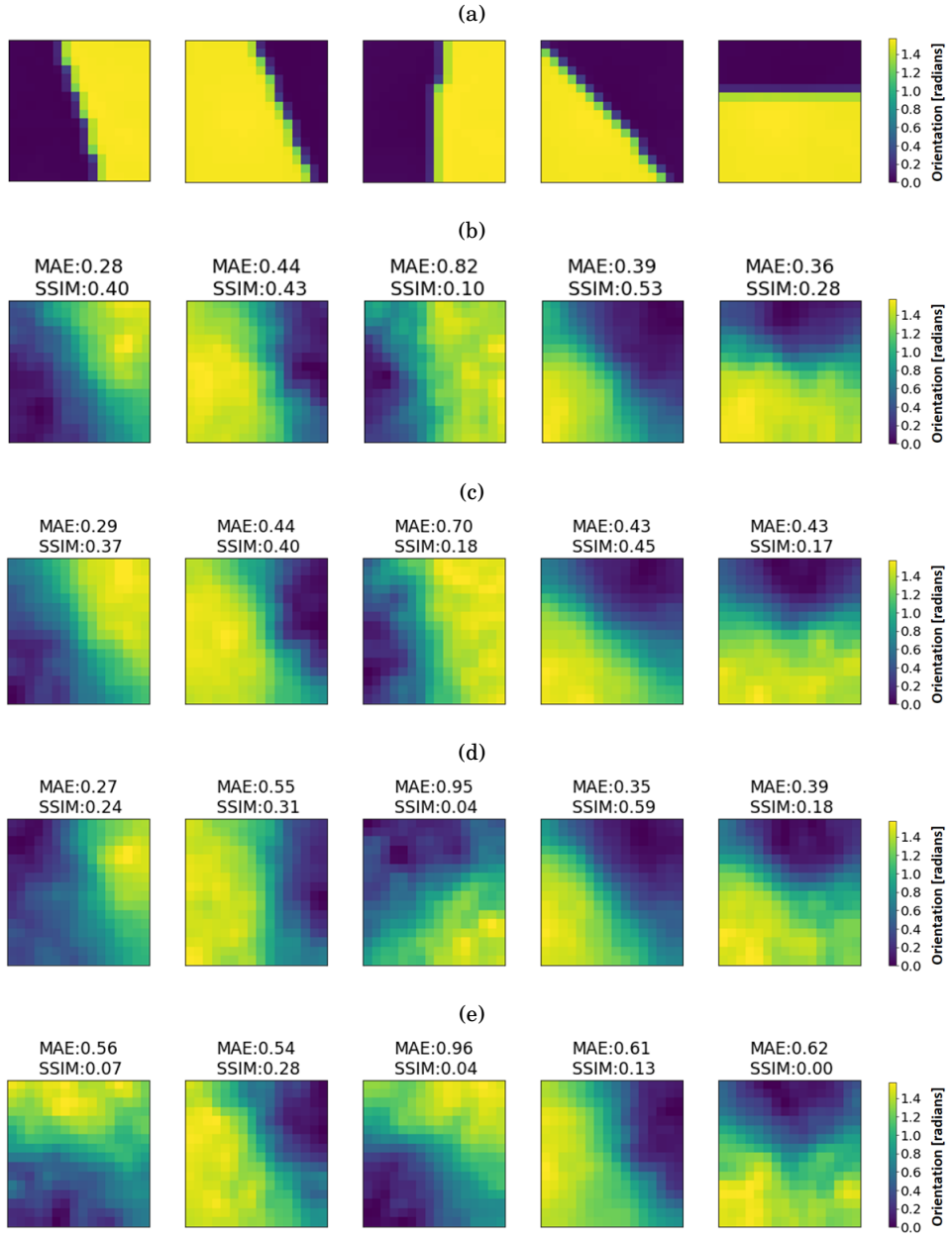

FIGURE A.13. Illustration of the effect on the 2 seed CNN model reconstruction predictions when multiplicative noise is introduced within the input data. (a) Denotes the true orientation maps, (b) shows the corresponding prediction of the orientations maps generated using the tuned CNN model (network architecture shown in Figure 3.5, where the corresponding hyperparameters used are depicted in Table 3.3). (c)-(e) Illustrates the effect of multiplicative noise on the corresponding predictions of the orientation maps, where the noise scale used were 25%, 50% and 75% respectively.

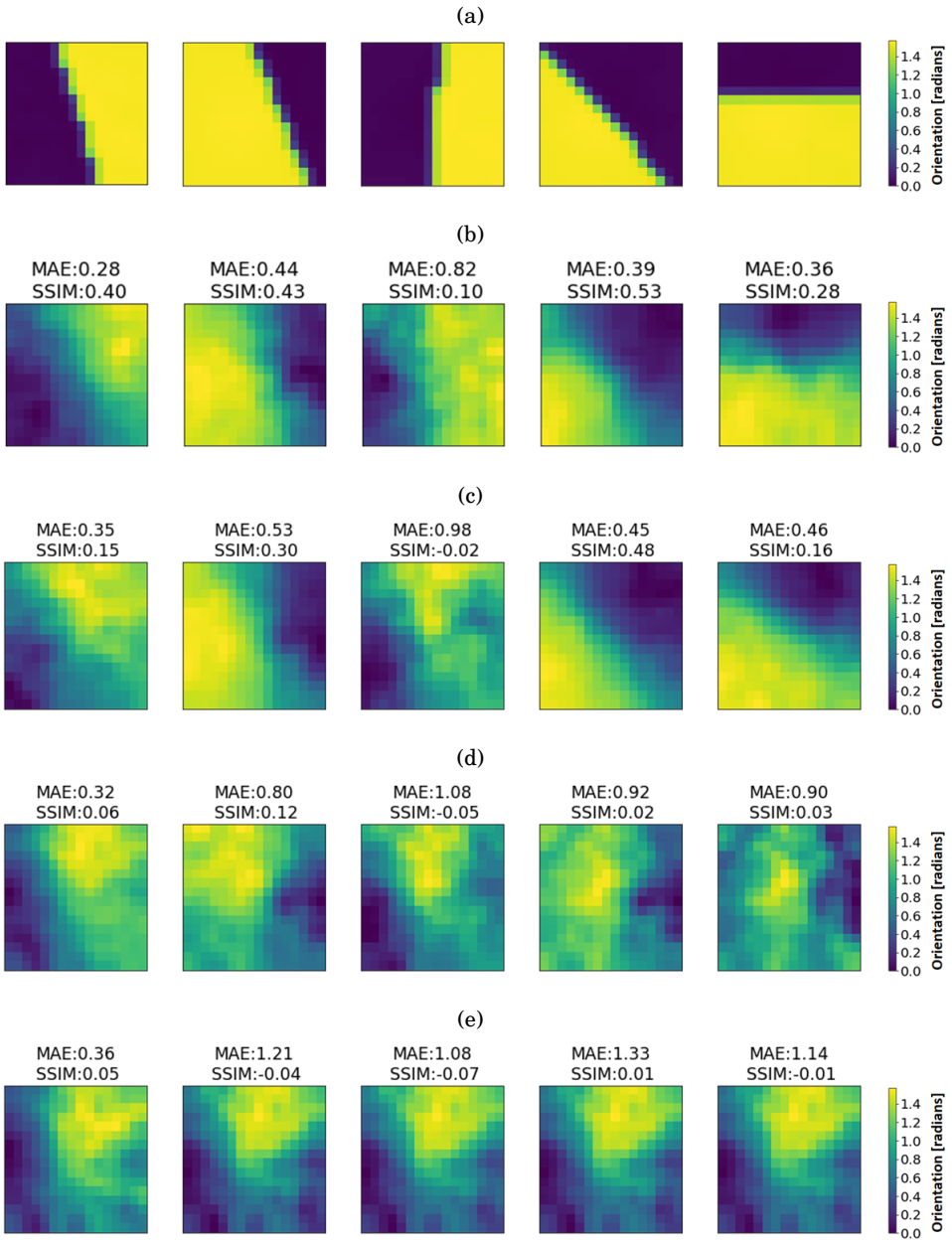

FIGURE A.14. Illustration of the effect on the 2 seed CNN model reconstruction predictions when additive noise is introduced within the input data. (a) Denotes the true orientation maps, (b) shows the corresponding prediction of the orientations maps generated using the tuned CNN model (network architecture shown in Figure 3.5, where the corresponding hyperparameters used are depicted in Table 3.3). (c)-(e) Illustrates the effect of additive noise on the corresponding predictions of the orientation maps, where the noise scale used were 25%, 50% and 75% respectively.

A.3.5 Effect of Noise on 5 Seed Reconstruction Model

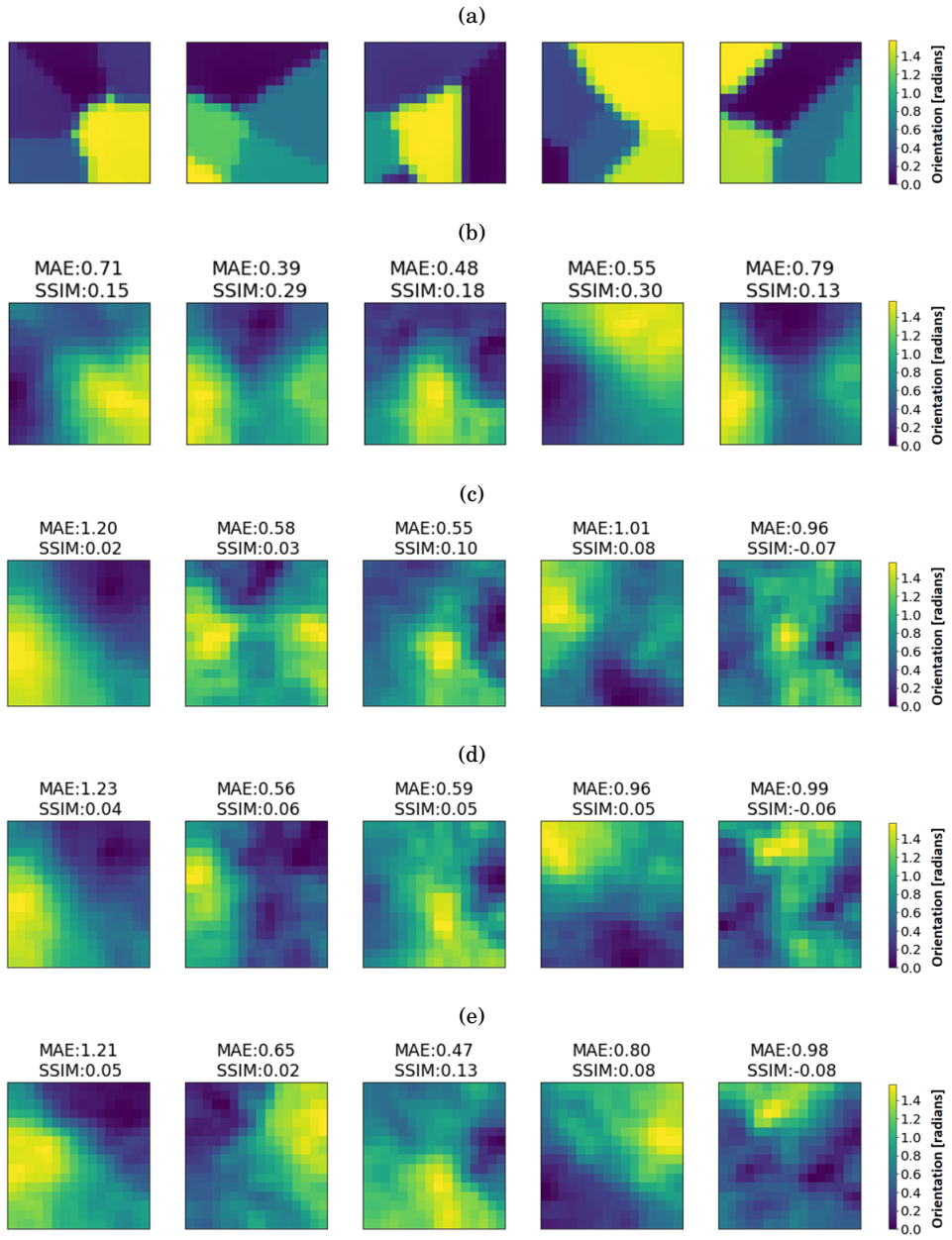

FIGURE A.15. Illustration of the effect on the 5 seed CNN model reconstruction predictions when multiplicative noise is introduced within the input data. (a) Denotes the true orientation maps, (b) shows the corresponding prediction of the orientations maps generated using the tuned CNN model (network architecture shown in Figure 3.5, where the corresponding hyperparameters used are depicted in Table 3.3). (c)-(e) Illustrates the effect of multiplicative noise on the corresponding predictions of the orientation maps, where the noise scale used were 25%, 50% and 75% respectively.

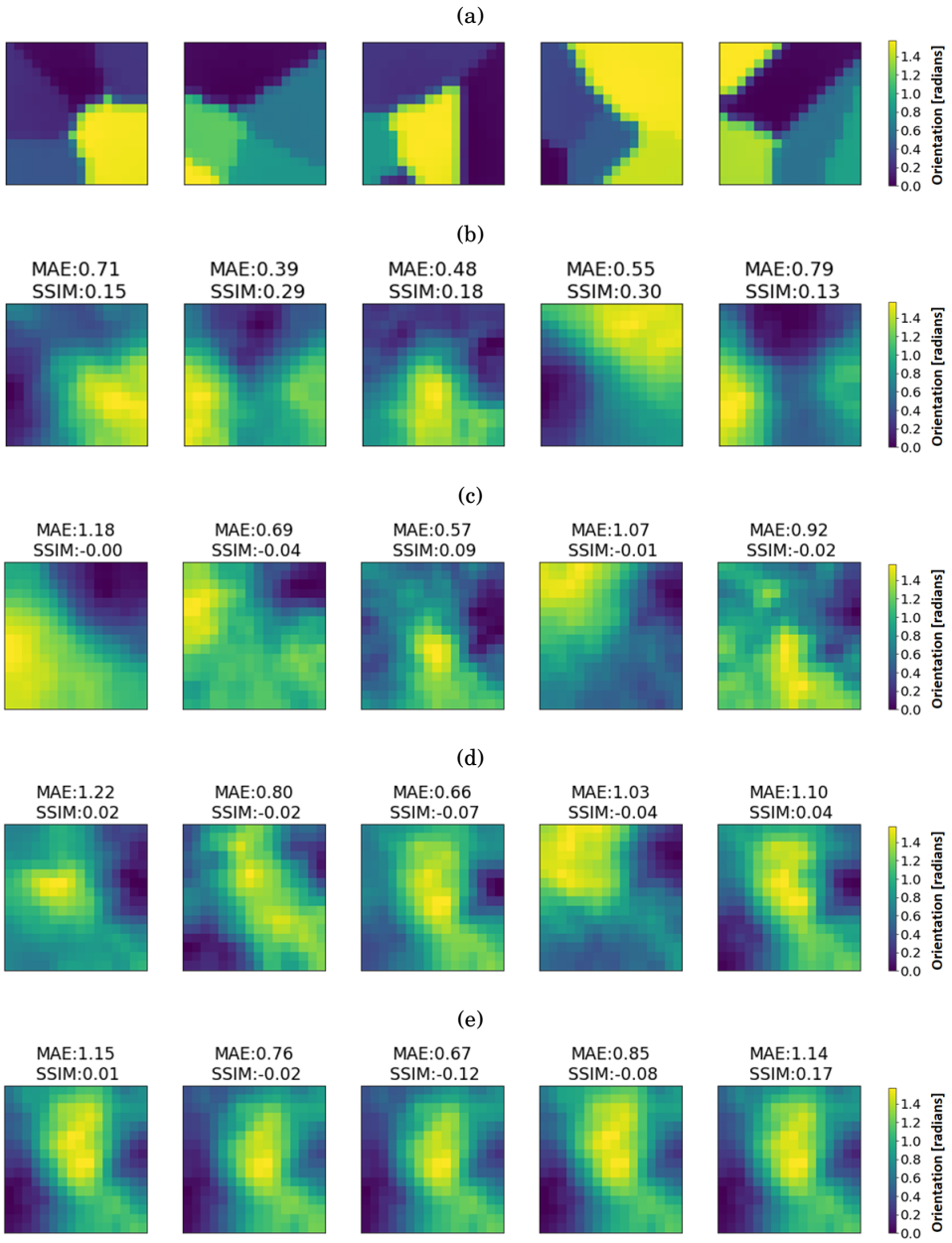

FIGURE A.16. Illustration of the effect on the 5 seed CNN model reconstruction predictions when additive noise is introduced within the input data. (a) Denotes the true orientation maps, (b) shows the corresponding prediction of the orientations maps generated using the tuned CNN model (network architecture shown in Figure 3.5, where the corresponding hyperparameters used are depicted in Table 3.3). (c)-(e) Illustrates the effect of additive noise on the corresponding predictions of the orientation maps, where the noise scale used were 25%, 50% and 75% respectively.

A.3.6 MAE and SSIM Histogram at Different Noise Levels

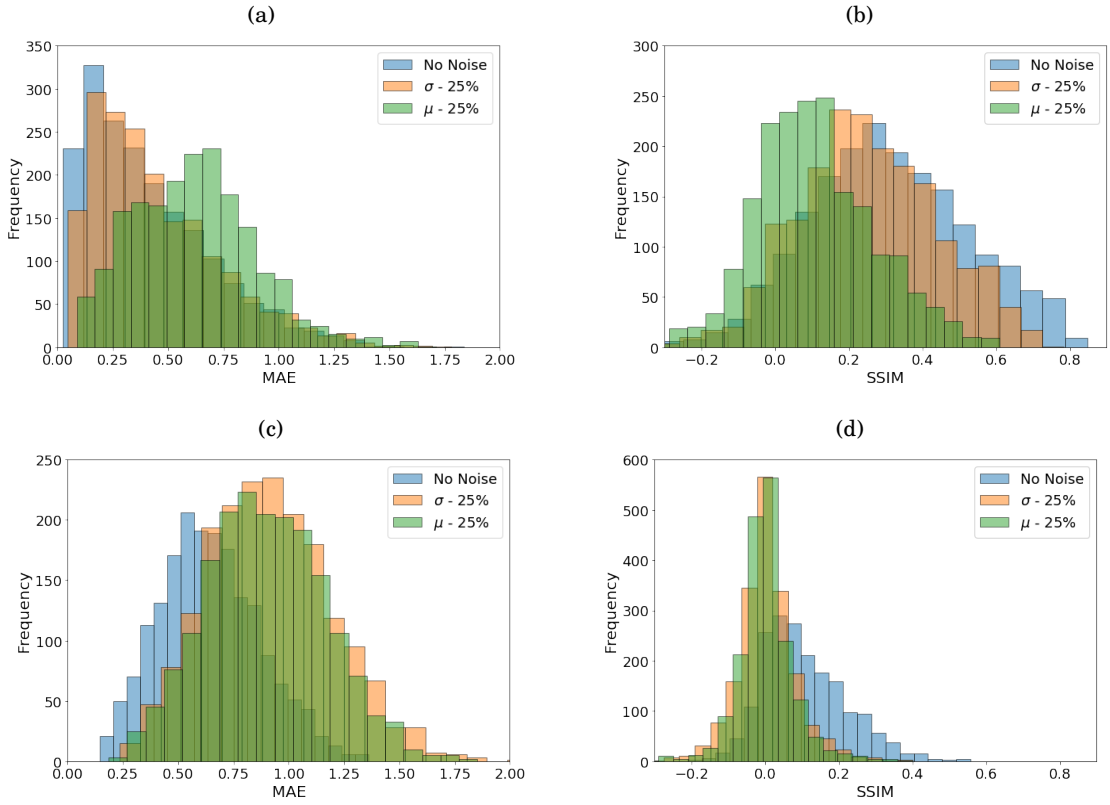

FIGURE A.17. Histograms generated of MAE and SSIM values of the predicted orientation maps for the 2 seed model shown in (a) and (b) respectively; whereas (c) and (d) depicts the 5 seed model of MAE and SSIM respectively. The histogram shows the data of the predictions under no noise, with multiplicative noise (σ) and with additive noise (μ) at a level of 25% (see Section 3.2.3 for the method used to introduce noise levels).

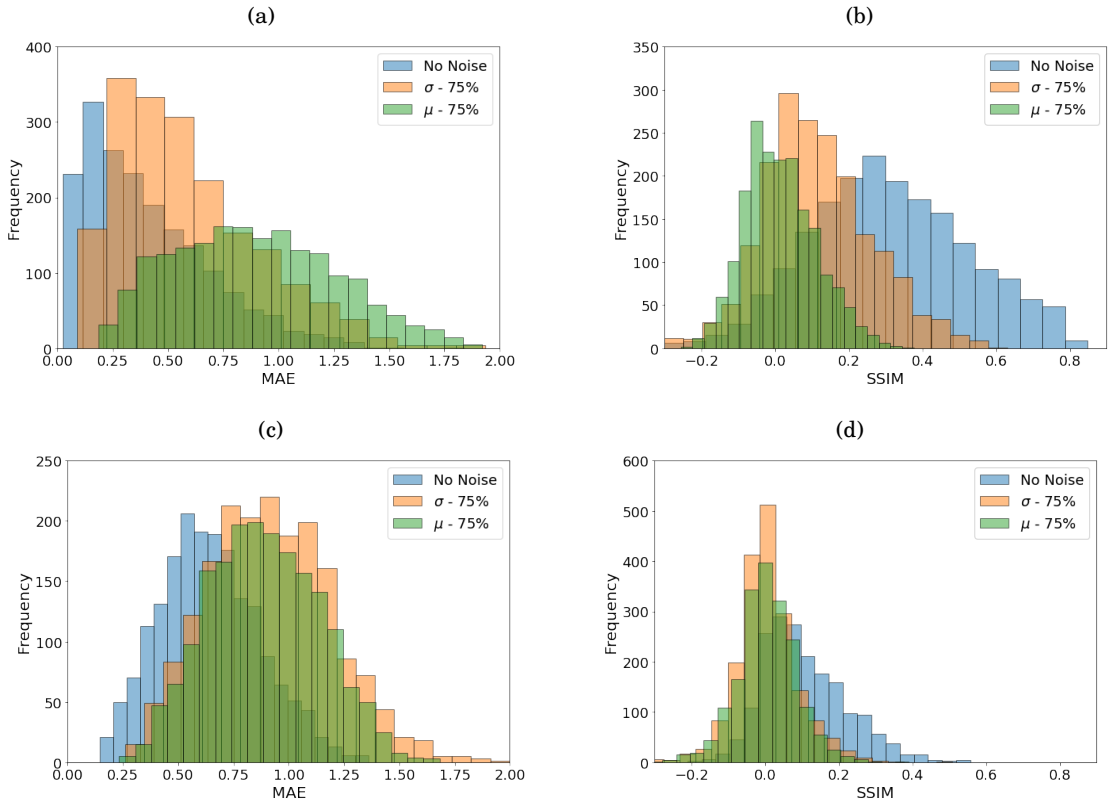

FIGURE A.18. Histograms generated of MAE and SSIM values of the predicted orientation maps for the 2 seed model shown in (a) and (b) respectively; whereas (c) and (d) depicts the 5 seed model of MAE and SSIM respectively. The histogram shows the data of the predictions under no noise, with multiplicative noise (σ) and with additive noise (μ) at a level of 75% (see Section 3.2.3 for the method used to introduce noise levels).

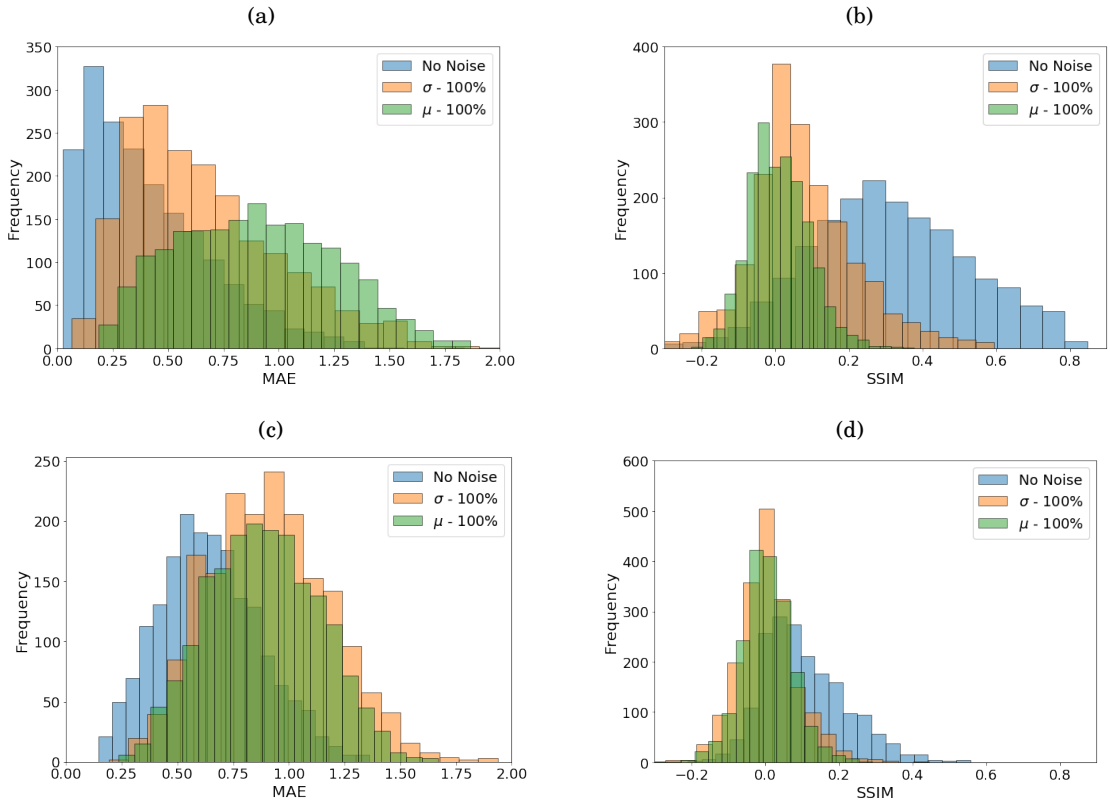

FIGURE A.19. Histograms generated of MAE and SSIM values of the predicted orientation maps for the 2 seed model shown in (a) and (b) respectively; whereas (c) and (d) depicts the 5 seed model of MAE and SSIM respectively. The histogram shows the data of the predictions under no noise, with multiplicative noise (σ) and with additive noise (μ) at a level of 100% (see Section 3.2.3 for the method used to introduce noise levels).

BIBLIOGRAPHY

[1] N. A. ABD RAHMAN, L. E. HONG, R. H. ABDUL RAHIM, H. ABDUL RAHIM, N. AHMAD, S. BUNYAMIN, K. H. ABAS, N. M. NOR AYOB, F. R. MOHD YUNOS, AND M. S. B. MANSOR, *A review: Tomography systems in medical and industrial processes*, Jurnal Teknologi, 73 (2015), pp. 1–11.

[2] A. S. ALMRYAD AND H. KUTUCU, *Automatic identification for field butterflies by convolutional neural networks*, Engineering Science and Technology, an International Journal, 23 (2020), pp. 189–195.

[3] N. AMIRI, G. H. FARRAHI, K. R. KASHYZADEH, AND M. CHIZARI, *Applications of ultrasonic testing and machine learning methods to predict the static & fatigue behavior of spot-welded joints*, Journal of Manufacturing Processes, 52 (2020), pp. 26–34.

[4] ANON, *Maxima and minima*.
Available at `https://commons.wikimedia.org/wiki/File:Maxima_and_Minima.svg` (2021/05/01).

[5] M. ARAYA-POLO, J. JENNINGS, A. ADLER, AND T. DAHLKE, *Deep-learning tomography*, Leading Edge, 37 (2018), pp. 58–66.

[6] ASNT, *Introduction to non-destructive testing*.
Available at `https://asnt.org/MajorSiteSections/About` (2021/07/28).

[7] M. ASSENS, X. GIRO-I-NIETO, K. MCGUINNESS, AND N. E. OCONNOR, *SaltiNet: Scan-Path Prediction on 360 Degree Images Using Saliency Volumes*, Proceedings - 2017 IEEE International Conference on Computer Vision Workshops, ICCVW 2017, 2018-Janua (2017), pp. 2331–2338.

[8] S. BARRIOS, D. BULDAIN, M. P. COMECH, I. GILBERT, AND I. ORUE, *Partial discharge classification using deep learning methods - Survey of recent progress*, Energies, 12 (2019), pp. 1–16.

[9] I. BENEDETTI AND M. H. ALIABADI, *A three-dimensional grain boundary formulation for microstructural modeling of polycrystalline materials*, Computational Materials Science, 67 (2013), pp. 249–260.

[10] E. BISONG, *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, 01 2019.

[11] B. BOASHASH, *Time-Frequency Signal Analysis and Processing*, Academic Press, Oxford, second edition ed., 2016.

[12] D. P. BOURNE, A. J. MULHOLLAND, S. SAHU, AND K. M. TANT, *An inverse problem for Voronoi diagrams: A simplified model of non-destructive testing with ultrasonic arrays*, Mathematical Methods in the Applied Sciences, 44 (2021), pp. 3727–3745.

[13] S. BOYD AND L. VANDENBERGHE, *Convex Optimization*, Cambridge University Press, 3 2004.

[14] G. BRADSKI AND A. KAEHLER, *Learning OpenCV: Computer vision with the OpenCV library*, " O'Reilly Media, Inc.", 2008.

[15] L. BURTSEVA, F. WERNER, B. V. SALAS, A. PESTRYAKOV, R. ROMERO, AND V. PETRANOVSKII, *Modeling of the material structure using Voronoi diagrams and tessellation methods*, (2015), pp. 1–11.

[16] Y. J. CHA, W. CHOI, AND O. BÜYÜKÖZTÜRK, *Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks*, Computer-Aided Civil and Infrastructure Engineering, 32 (2017), pp. 361–378.

[17] F. CHOLLET, *Deep Learning with Python*, Manning Publications Co., USA, 1st ed., 2017.

[18] A. DALMIA, *Optimization for training deep models part i*.
Available at `https://medium.com/inveterate-learner/deep-learning-book-chapter-8-optimization-for-training-deep-models-part-i-20ae75984cb2` (2021/08/17).

[19] B. DERVIŞ, *Hands-on Machine Learning with Sklearn, keras & tensorflow*, vol. 53, 2013.

[20] T. DETTMERS, *Deep learning in a nutshell*.
Available at `https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/` (2021/08/02).

[21] J. DLUHOŠ, L. SEDLÁČEK, AND J. MAN, *Application of electron channeling contrast imaging in study of polycrystalline materials and visualization of crystal lattice defects*, METAL 2012 - Conference Proceedings, 21st International Conference on Metallurgy and Materials, (2012), pp. 216–220.

[22] B. W. DRINKWATER AND P. D. WILCOX, *Ultrasonic arrays for non-destructive evaluation: A review*, NDT and E International, 39 (2006), pp. 525–541.

[23] S. EARP AND A. CURTIS, *Probabilistic neural network-based 2D travel-time tomography*, Neural Computing and Applications, 32 (2020), pp. 17077–17095.

[24] T. FAWCETT, *An introduction to ROC analysis*, Pattern Recognition Letters, 27 (2006), pp. 861–874.

[25] A. F. FERCHER, *Optical Coherence Tomography*, Journal of Biomedical Optics, 1 (1996), pp. 157–173.

[26] E. GALETTI, A. CURTIS, G. A. MELES, AND B. BAPTIE, *Uncertainty Loops in Travel-Time Tomography from Nonlinear Wave Physics*, Physical Review Letters, 114 (2015), pp. 1–5.

[27] H. GEMMEKE AND N. V. RUITER, *3D ultrasound computer tomography for medical imaging*, Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 580 (2007), pp. 1057–1065.

[28] S. GHOLIZADEH, *A review of non-destructive testing methods of composite materials*, Procedia Structural Integrity, 1 (2016), pp. 50–57.

[29] I. GOODFELLOW, J. POUGET-ABADIE, M. MIRZA, B. XU, D. WARDE-FARLEY, S. OZAIR, A. COURVILLE, AND Y. BENGIO, *Generative Adversarial Nets*, (2014).

[30] C. R. HARRIS, K. J. MILLMAN, S. J. VAN DER WALT, R. GOMMERS, P. VIRTANEN, D. COURNAPEAU, E. WIESER, J. TAYLOR, S. BERG, N. J. SMITH, R. KERN, M. PICUS, S. HOYER, M. H. VAN KERKWIJK, M. BRETT, A. HALDANE, J. FERNÁNDEZ DEL RÍO, M. WIEBE, P. PETERSON, P. GÉRARD-MARCHANT, K. SHEPPARD, T. REDDY, AND W. WECKESSER, *Array programming with NumPy*, Nature, 585 (2020), p. 357–362.

[31] S. G. HASLINGER, M. J. LOWE, R. V. CRASTER, P. HUTHWAITE, AND F. SHI, *Prediction of reflection amplitudes for ultrasonic inspection of rough planar defects*, Insight: Non-Destructive Testing and Condition Monitoring, 63 (2021), pp. 28–36.

[32] A. HIDAKA AND T. KURITA, *Consecutive Dimensionality Reduction by Canonical Correlation Analysis for Visualization of Convolutional Neural Networks*, Proceedings of the ISCIE International Symposium on Stochastic Systems Theory and its Applications, 2017 (2017), pp. 160–167.

[33] T. HOPP, N. RUITER, J. C. BAMBER, N. DURIC, AND K. W. A. V. DONGEN, *International Workshop on Medical Ultrasound Tomography*, (2017).

[34] Y. JAVADI, C. N. MACLEOD, S. G. PIERCE, A. GACHAGAN, D. LINES, C. MINEO, J. DING, S. WILLIAMS, M. VASILEV, E. MOHSENI, AND R. SU, *Ultrasonic phased array inspection of a Wire + Arc Additive Manufactured (WAAM) sample with intentionally embedded defects*, Additive Manufacturing, 29 (2019), p. 100806.

[35] JAYAKUMAR, T, RAO, B.P.C., THIRUNAVUKKARASU, AND S, *Non-Destructive Testing Methods for Investigation of Surfaces of Materials*, Proc. International Conf on Surface Techniques, (2001).

[36] Y. JIAXING AND T. NOBUYUKI, *Benchmarking deep learning models for automatic ultrasonic imaging inspection*, IEEE ACCESS, (2021).

[37] J. KAJBERG AND G. LINDKVIST, *Characterisation of materials subjected to large strains by inverse modelling based on in-plane displacement fields*, International Journal of Solids and Structures, 41 (2004), pp. 3439–3459.

[38] D. P. KINGMA AND J. BA, *Adam: A Method for Stochastic Optimization*, (2014).

[39] B. A. KRIZHEVSKY, I. SUTSKEVER, AND G. E. HINTON, *ImageNet Classification with Deep Convolutional Neural Networks*, Communications of the ACM, 60 (2012), pp. 84–90.

[40] Y. LECUN, Y. BENGIO, AND G. HINTON, *Deep learning*, Nature, 521 (2015), pp. 436–444.

[41] C. LI, K. XU, J. ZHU, AND B. ZHANG, *Triple generative adversarial nets*, Advances in Neural Information Processing Systems, 2017-Decem (2017), pp. 4089–4099.

[42] M. MONGA, *Computed tomography - An increasing source of radiation exposure: Editorial comment*, International Braz J Urol, 33 (2007), p. 855.

[43] M. A. NIELSEN, *Neural networks and deep learning*, 2015.

[44] Z. NIU, W. WANG, X. HUANG, AND J. LAI, *Integrated assessment of concrete structure using Bayesian theory and ultrasound tomography*, Construction and Building Materials, 274 (2021), p. 122086.

[45] P. PEDAMKAR, *Difference between python vs matlab*.
Available at `https://www.educba.com/python-vs-matlab/` (2021/08/23).

[46] F. PEDREGOSA, G. VAROQUAUX, A. GRAMFORT, V. MICHEL, B. THIRION, O. GRISEL, M. BLONDEL, P. PRETTENHOFER, R. WEISS, V. DUBOURG, J. VANDERPLAS, A. PASSOS, D. COURNAPEAU, M. BRUCHER, M. PERROT, AND E. DUCHESNAY, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.

[47] R. PRATT, *Medical ultrasound tomography: lessons from exploration geophysics*, (2017).

[48] R. J. PYLE, R. L. BEVAN, R. R. HUGHES, R. K. RACHEV, A. A. S. ALI, AND P. D. WILCOX, *Deep Learning for Ultrasonic Crack Characterization in NDE*, IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, 3010 (2020).

[49] T. RAY, Y. CAO, Z. QIAO, AND G. CHEN, *2D and 3D ISAR image reconstruction through filtered back projection*, Radar Sensor Technology XVI, 8361 (2012), pp. 836107–836107.

[50] C. ROBERT, *Machine Learning, a Probabilistic Perspective*, CHANCE, 27 (2014), pp. 62–63.

[51] N. V. RUITER, M. ZAPF, T. HOPP, R. DAPP, E. KRETZEK, M. BIRK, B. KOHOUT, AND H. GEMMEKE, *3D ultrasound computer tomography of the breast: A new era?*, European Journal of Radiology, 81 (2012), pp. S133–S134.

[52] C. SIMON, *Roc curves to evaluate binary classification algorithms*.
Available at `http://corysimon.github.io/articles/what-is-an-roc-curve/` (2021/08/17).

[53] P. K. SINGH, A. K. KAR, Y. SINGH, M. KOLEKAR, AND S. TANWAR, *Proceedings of ICRIC 2019*, 2019.

[54] N. SMITH AND A. WEBB, *Introduction to Medical Imaging*, 2011.

[55] K. M. TANT, E. GALETTI, A. J. MULHOLLAND, A. CURTIS, AND A. GACHAGAN, *A transdimensional Bayesian approach to ultrasonic travel-time tomography for non-destructive testing*, Inverse Problems, 34 (2018).

[56] S. VAN DER WALT, J. L. SCHÖNBERGER, J. NUNEZ-IGLESIAS, F. BOULOGNE, J. D. WARNER, N. YAGER, E. GOUILLART, T. YU, AND THE SCIKIT-IMAGE CONTRIBUTORS, *scikit-image: image processing in python*, PeerJ, 2 (2014), p. e453.

[57] A. VAN PAMEL, C. R. BRETT, P. HUTHWAITE, AND M. J. S. LOWE, *Finite element modelling of elastic wave scattering within a polycrystalline material in two and three dimensions*, The Journal of the Acoustical Society of America, 138 (2015), pp. 2326–2336.

[58] Z. WANG, A. BOVIK, H. SHEIKH, AND E. SIMONCELLI, *Image quality assessment: from error visibility to structural similarity*, IEEE Transactions on Image Processing, 13 (2004), pp. 600–612.

[59] Z. WANG AND A. C. BOVIK, *Mean squared error: Love it or leave it? a new look at signal fidelity measures*, IEEE Signal Processing Magazine, 26 (2009), pp. 98–117.

[60] B. WEBER, *Matlab vs python: Why and how to make the switch*.
Available at `https://realpython.com/matlab-vs-python/` (2021/08/23).

[61] M. J. WILLEMINK AND P. B. NOËL, *The evolution of image reconstruction for CT—from filtered back projection to artificial intelligence*, European Radiology, 29 (2019), pp. 2185–2195.

[62] C. J. WILLMOTT AND K. MATSUURA, *Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance*, Climate Research, 30 (2005), pp. 79–82.