MSc Thesis

Neural Network Modelling for Composite Damage Pattern Generation

by

Karthik Venkatesan

in partial fulfilment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology
to be defended publicly on Friday August 30, 2019 at 13:00 hrs.

Supervisor: Dr. Boyang Chen

TU Delft Delft University of Technology

Foreword

The process of carry out this thesis work has been the most exhilarating yet challenging experience of my academic life so far. Formulating the research objectives based on a literature study was one that needed to balance ambition with pragmatism, and was a process that saw me step into the unfamiliar territory of machine learning. This was a big decision that I'm very glad to have taken. The development work itself was one that had its share of both euphoria and uncertainty, which made it one of the most rewarding aspects of the thesis. The experience of managing the project from start to finish has certainly contributed to my growth as an individual, and will no doubt stay with me for the years to come.

At any stage in this process, my supervisor, Dr. Boyang Chen, was someone I could turn to for guidance and support. I was offered not only scientific assistance from him, but also positivity and reassurance during rough patches. I am extremely grateful for the influence he has had, and consider myself fortunate to have worked under his supervision.

Through the course of my entire master's study, but even more so during the thesis phase, a major source of support and motivation has been my parents - their love and encouragement have played a significant role in getting me across the finish line.

Finally, I would like to thank my friends from both within and beyond the faculty, who I've greatly enjoyed spending time with, and have been responsible for keeping my morale high. I hope to never lose touch with these wonderful people.

Karthik Venkatesan
Delft, August 2019

Contents

List of Tables

List of Figures

Abstract

This research project was initiated as a result of a curiosity and desire to investigate the applicability of surrogate modelling to analyse complex non-linear behaviour in aircraft structures. The study chose to focus on modelling damage in composite plates, and through a literature review, deemed that the generation of graphical outputs was a domain worthy of attention. Thus, the research questions subsequently formulated were centred around the modelling of artificial neural networks for the generation of damage patterns on composite plates.

Data for training and evaluating these neural networks was first generated through 20 finite element models solved using Abaqus 2017. Standard neural networks trained to directly reproduce these damage patterns, as well as reduced-image neural networks trained to reproduce a reduced form of these patterns (obtained using convolutional neural networks) were analysed, and both were found in want of more training data.

The generation of a further 421 finite element models resulted in a striking improvement in the performance of both networks, but with the standard neural network outperforming the reduce-image neural network. Thereafter, it was discovered that a hybrid network that combined facets of the standard and convolutional neural networks performed superior to both.

In the process of training these networks, it was recognised that while the performance metrics served as an indicator of the resemblance between the predicted and actual outputs in terms of colours and contours, the same trends did not apply to the image quality. In order to improve the visual quality of outputs from the hybrid network, the use of the Structural Similarity Index (SSIM) was explored. It was eventually determined that pre-training the network using the Mean Square Error (MSE) as its optimising metric before then doing the final training using SSIM resulted in a model with impressive results. This fine-tuned model carried out predictions with a mean error of 0.0014 on the MSE metric and 0.9804 on the SSIM metric when evaluated on an independent dataset.

Finally, the reliability and computational efficiency of the hybrid model was measured. It was found that approximately 95% of the MSE values on the independent dataset were within a value of 0.0040, while the same percentage of SSIM values were over 0.9100. The computation speed, meanwhile, improved by a factor of roughly 34 times on average, with the figure rising to over 443 on specific models.

1

Introduction

The motivation for instigating this research project stems from a curiosity to explore the prospects of surrogate modelling in structural design. For the uninitiated, surrogate models are approximation models used in engineering to imitate the results of experimental or simulation setups, but less expensively. These are most valuable when design variables are determined through repeated trials. For instance, when designing a composite plate on the basis of a complex phenomenon such as damage, the optimal value of plate thickness cannot be directly determined easily. Instead, simulations are run for different thickness values until the most suitable value is identified. Considering that each iteration can take several hours or even days, this process can easily run up computation costs. Research in surrogate modelling investigates practices that maximise computational efficiency and minimise approximation errors, using which a substantial number of dependable simulations may be carried out more cost-effectively.

This study carries out its research on surrogate models in the context of damage initiation in composite laminates. The choice is made on the basis of damage analysis being a complex non-linear problem, the solving of which is an exercise performed abundantly in composite design, particularly in the field of aerospace engineering. The current chapter serves to introduce the concepts of damage, and closes with a description of how the remainder of this report is structured.

1.1. Damage in Composite Materials

Damage is a phenomenon whereby the mechanical properties of a material deteriorates, and it experiences a decline in load-carrying capacity. This is formally defined in terms of a stiffness degradation, with a damage variable d ranging from 0 to 1 representing the fraction of stiffness lost. In other terms, d is defined as:

$$d = 1 - \frac{E}{E_0} \tag{1.1}$$

where E_o represents the original stiffness of the material, and E represents the stiffness after the onset of damage. This stiffness behaviour is illustrated in Figure 1.1, which shows the typical response of a material on either side of damage initiation.

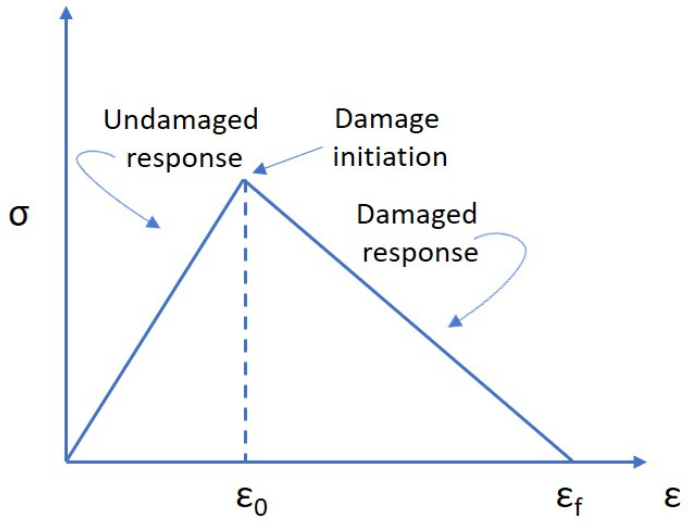


Figure 1.1: Typical response of material before and after damage

While such a trend applies to different degrees of linearity/non-linearity in most materials, this is often of little research interest in homogenous materials such as metals. The design constraints in those cases are straightforward - stay clear of the point of damage initiation, and limit their use to the region to the left of it. With composite materials, however, the considerations are more complex. Take the case of a laminate, for instance - these comprise of a number of plies stacked one on top of the other, each of which is made up of fibre and matrix materials at different orientations. The material is therefore extremely non-homogenous, and every region of it experiences a different state of damage (or proximity to it), when load is applied. The most structurally efficient approach in such a scenario is to analyse each region and assess the load-carrying capacity of the structure as a whole based on it. It is therefore, entirely possible that one component of the material is well past its point of damage initiation, but the structural integrity of the material as a whole has still not been lost. In the following subsections, the initiation and evolution of damage are discussed.

1.1.1. Damage Initiation

The response seen in Figure 1.1 indicates that there is a clear point at which damage initiates, before it goes on to evolve. Of the various theories that exist to explain the manner in which loading in tension, compression and shear combine to initiate damage, Hashin's theory [1, 2] is one that provides a set of criteria that is known to be reliable and is considered here. Since the material response varies for the fibre and matrix, as well as between tension and compression, these criteria provide four different cases to look at.

The four equations shown below give the Hashin initiation criteria for damage through fibre tension, fibre compression, matrix tension and matrix compression respectively. X and Y refer to the longitudinal and transverse strengths of the material, with the superscripts T and C referring to them under tension and compression. S^L, on the other hand refers to the longitudinal shear stress of the material, and S^T the transverse shear stress. When any of these variables given in Eq. (1.2) to (1.5) attain a value of 1, damage is initiated in the corresponding mode.

In fibre tension ($\hat{\sigma}_{11} \geq 0$):

$$F_{ft} = \left(\frac{\hat{\sigma}_{11}}{X^T}\right)^2 + \alpha\left(\frac{\hat{\sigma}_{12}}{S^L}\right)^2 \tag{1.2}$$

where α is a coefficient lying between 0 and 1 that determines the contribution of shear stress to the fibre tension mode of damage initiation.

In fibre compression ($\hat{\sigma}_{11} < 0$):

$$F_{fc} = \left(\frac{\hat{\sigma}_{11}}{X^C}\right)^2 \tag{1.3}$$

In matrix tension ($\hat{\sigma}_{22} \geq 0$):

$$F_{mt} = \left(\frac{\hat{\sigma}_{22}}{Y^T}\right)^2 + \left(\frac{\hat{\sigma}_{12}}{S^L}\right)^2 \tag{1.4}$$

In matrix compression ($\hat{\sigma}_{22} < 0$):

$$F_{mc} = \left(\frac{\hat{\sigma}_{22}}{2S^T}\right)^2 + \left[\left(\frac{Y^C}{2S^T}\right)^2 - 1\right]\frac{\hat{\sigma}_{22}}{Y^C} + \left(\frac{\hat{\sigma}_{12}}{S^L}\right)^2 \tag{1.5}$$

The above equations use the effective stress $\hat{\sigma}$, which is related to the applied stress σ through the damage variables, as shown in Equation (1.6).

$$\hat{\sigma} = \begin{bmatrix} \frac{1}{1-d_f} & 0 & 0 \\ 0 & \frac{1}{1-d_m} & 0 \\ 0 & 0 & \frac{1}{1-d_s} \end{bmatrix} \sigma \tag{1.6}$$

where d_f and d_m are the damage variables concerning the fibre and matrix respectively (corresponding to tension or compression with σ), while d_s is given by:

$$d_s = 1 - (1-d_{ft})(1-d_{fc})(1-d_{mt})(1-d_{mc}) \tag{1.7}$$

At first glance, it might appear that including the damage variables in these criteria is unnecessary because damage has not been initiated yet. However, damage is not initiated simultaneously in all modes - its initiation

and subsequent evolution in one mode will influence its initiation in others. The calculation of these d values as damage evolves is described in the next subsection.

1.1.2. Damage Evolution

Any load that is applied after the initiation of damage contributes to the evolution of damage. While Eq. (1.1) gives us the definition of damage, it is not immediately useful as there is no direct way of obtaining the post-damage stiffness. In fact, these new stiffness values are generally determined the other way around, using the damage variables.

To compute them, the energy dissipation and linear softening of the material are considered [2]. A result of these are the equivalent displacements at a point being different from that computed using the strain value δ from the stress-strain curve - their expressions are given by Eq. (1.8) to (1.11).

$$\left(\delta_{ft}\right)_{eq} = \sqrt{\langle\delta_{11}\rangle^2 + \alpha\delta_{12}^2} \tag{1.8}$$

where $\langle\rangle$ is the Macauley bracket operator, defined for every $p \in R$ as $\langle p\rangle = \dfrac{p+|p|}{2}$.

$$\left(\delta_{fc}\right)_{eq} = \langle -\delta_{11}\rangle \tag{1.9}$$

$$(\delta_{mt})_{eq} = \sqrt{\langle\delta_{22}\rangle^2 + \alpha\delta_{12}^2} \tag{1.10}$$

$$(\delta_{mc})_{eq} = \sqrt{\langle -\delta_{22}\rangle^2 + \alpha\delta_{12}^2} \tag{1.11}$$

These equivalent displacements have stresses corresponding to them. The equivalent stresses may be computed using the equivalent strains as per the following expressions.

$$\left(\sigma_{ft}\right)_{eq} = \frac{\langle\sigma_{11}\rangle\langle\delta_{11}\rangle + \alpha\sigma_{12}\delta_{12}}{\left(\delta_{ft}\right)_{eq}} \tag{1.12}$$

$$\left(\sigma_{fc}\right)_{eq} = \frac{\langle -\sigma_{11}\rangle\langle -\delta_{11}\rangle}{\left(\delta_{fc}\right)_{eq}} \tag{1.13}$$

$$(\sigma_{mt})_{eq} = \frac{\langle\sigma_{22}\rangle\langle\delta_{22}\rangle + \sigma_{12}\delta_{12}}{(\delta_{mt})_{eq}} \tag{1.14}$$

$$(\sigma_{mc})_{eq} = \frac{\langle -\sigma_{22}\rangle\langle -\delta_{22}\rangle + \sigma_{12}\delta_{12}}{(\delta_{mc})_{eq}} \tag{1.15}$$

The equivalent displacement at final failure is given by:

$$\delta_{eq}^{f} = \frac{2G}{\sigma_{eq}^{o}} \tag{1.16}$$

where σ_{eq}^{o} refers to the equivalent stress at initiation. Using the equivalent displacements at initiation and final failure, the general damage variable at a particular equivalent displacement is given by:

$$d = \frac{\delta_{eq}^{f}\left(\delta_{eq} - \delta_{eq}^{o}\right)}{\delta_{eq}\left(\delta_{eq}^{f} - \delta_{eq}^{o}\right)} \tag{1.17}$$

With the damage variable for the fibre and matrix calculated in this manner, the stiffness of the material as a whole is modified as:

$$C = \frac{1}{D}\begin{bmatrix} (1-d_f)E_1 & (1-d_f)(1-d_m)\nu_{21}E_1 & 0 \\ (1-d_f)(1-d_m)\nu_{12}E_2 & (1-d_m)E_2 & 0 \\ 0 & 0 & D(1-d_s)G \end{bmatrix} \tag{1.18}$$

where,

$$D = 1 - (1-d_f)(1-d_m)\nu_{12}\nu_{21} \tag{1.19}$$

The initiation and evolution criteria discussed in this section are the basis of modelling damage. In the following section, their solution is discussed.

1.2. Solution of Damage Model

Damage models are solved using the equations outlined in the previous section. In order to obtain accurate results, load is applied in small time steps, with the initiation and evolution criteria variables updated after each time step. In addition, by using the method of finite elements, the distribution of damage outputs over the entire structure is computed more precisely.

To demonstrate what a converged output conveys, the Hashin initiation criteria are computed for a plate with an elliptical hole in the centre. Shown in Figure 1.2, this choice is made in order to use a simple geometry to exhibit a complex non-linear phenomenon. Moreover, such a structure is also 'modular' and may be used repetitively in aircraft construction, thus making it immediately relatable. The outputs being computed here are the damage initiation criteria variables given in Eq. (1.2) to (1.5). Note that these are not the damage variables d themselves, but variables that indicate how close an element of the structure is to the initiation of damage in a particular mode.

The laminated plate described is sized 100 mm×100 mm×10 mm, with the axes of the elliptical hole being 20 mm horizontally and 40 mm vertically. It is constructed by stacking 8 layers of Carbon Fibre Reinforced Plastic (CFRP) in the sequence $[45/90/\text{-}45/0]_s$. The legend shown to its right side corresponds to the damage initiation criteria, which is 0 before loading (although the variable for damage initiation lies between 0 and 1, there often exists numerical noise that results in values that are fractionally over 1.0, and is accounted for through the range between 1 and 2 in the legend).

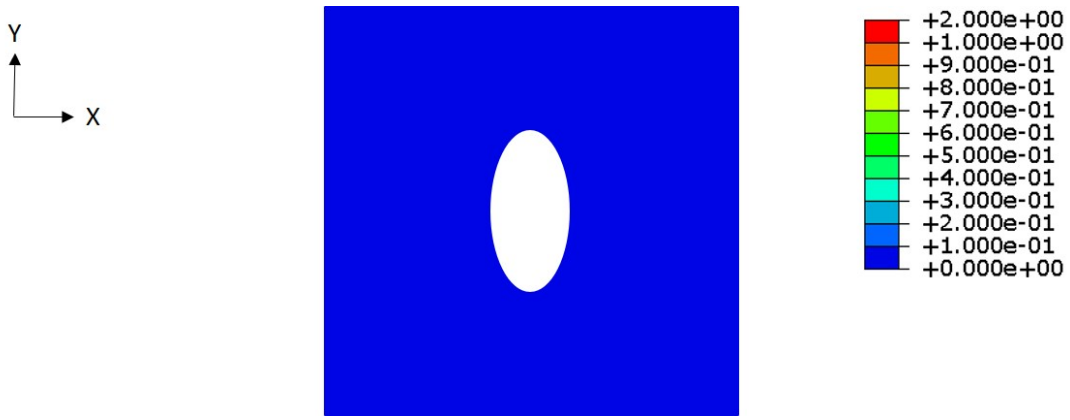


Figure 1.2: Laminated plate subject to biaxial loading

The plate is now subject to biaxial displacement loading - the left and right edges are displaced horizontally in opposite directions (in tension) by 0.1 mm. In similar fashion, the top and bottom edges are displaced vertically by 1 mm. These values are selected as they lie between the range of no failure and complete failure, and yield results that give a good indication of how these damage variable plots appear in different modes. The damage initiation variable is computed for each element over the entire surface of the plate, and is displayed as a contour plot according to the legend indicated in Figure 1.2. These are therefore referred to as *damage initiation patterns*, or simply *damage patterns* within the context of this analysis. The four modes that these patterns are applicable to are output for each ply. Figure 1.3 to Figure 1.6 show the results of the fibre tension mode for each of the four plies (ply 1 being the outermost one). All 16 damage patterns are shown together in Appendix-A.

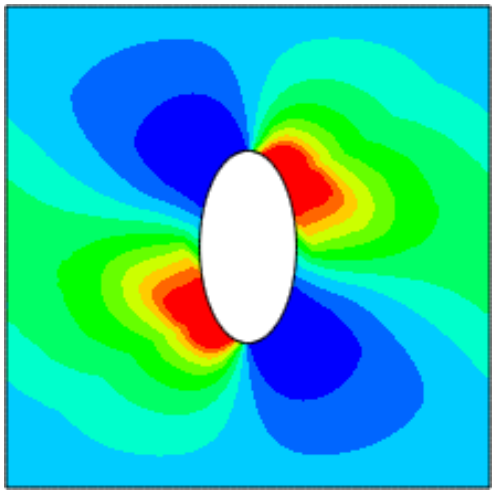


Figure 1.5: Fibre tension damage pattern for ply 3

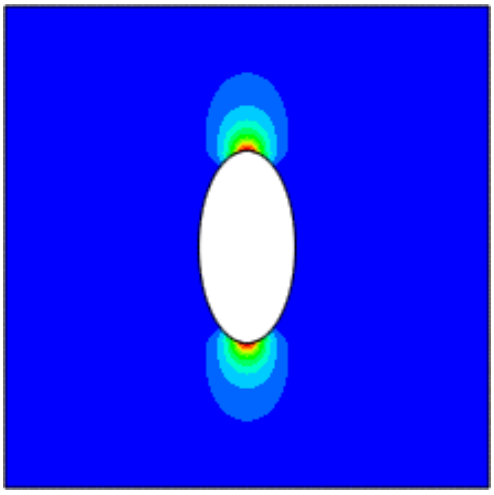


Figure 1.6: Fibre tension damage pattern for ply 4

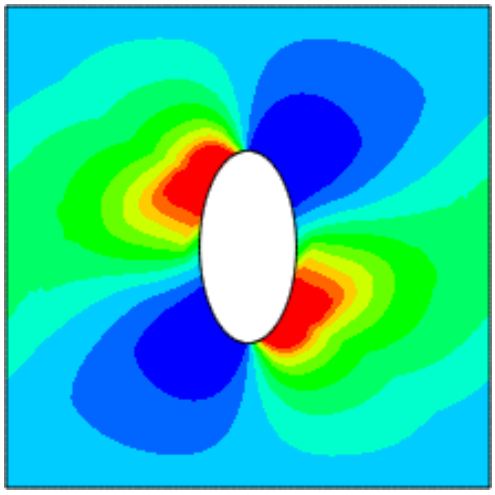


Figure 1.3: Fibre tension damage pattern for ply 1

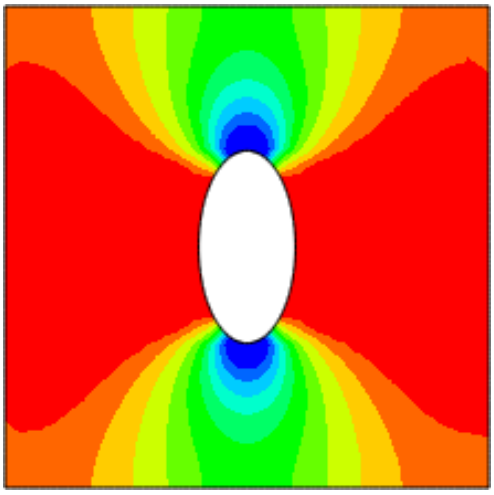


Figure 1.4: Fibre tension damage pattern for ply 2

The specific results of damage initiation aside, one observation that is of importance for us is the computation time. Using the Abaqus 2017 solver, this model is found to take 2927 CPU seconds (about 49 CPU minutes) to solve. This might be deemed acceptable while analysing a one-off model, but is less than ideal when the design is being optimised through repeated trials, or is a modular structure than is present in multiple locations on a higher-level component. The research being carried out in this study is therefore driven by a desire to emulate these results at lower computation costs through surrogate modelling.

1.3. Layout of Thesis

The investigation of surrogate models begins with a literature review in the next chapter. After introducing the concept of empirical relationships, previous research in their application to structural engineering is discussed, which serve as motivation for framing research questions. With the objective of developing neural network models to help answer them, the third chapter is devoted to the generation of training and testing data. The data is then used to train networks in Chapter 4, which are subsequently improved through modifications described in Chapter 5. The thesis concludes with a discussion of the results observed, and an attempt at answering the formulated research questions.

2

Literature Review

The illustrative model solved in the previous chapter served as an indicator of the limitations of a traditional finite element approach to structural design. Surrogate modelling offers an alternative that relies on approximations but is computationally less expensive - ensuring that these consequences are within acceptable ranges is the main challenge of such an approach. This literature review investigates empirical techniques that could potentially play a role in surrogate modelling, starting with simple techniques before exploring the use of machine learning approaches. Following this, an examination of previous work in the application of these methods in structural mechanics is carried out - the gaps that exist in this respect subsequently influence the research questions that are then laid out.

2.1. Empirical Science

Modelling and analysis in engineering is often approached in one of three ways - analytically, numerically, or experimentally. Purely analytical approaches are developed from first principles, and provide mathematically sound results with a high degree of reliability. However, these are often useful only in developing simplistic models, and is either impossible or impractical to use in more complex problems. Numerical techniques serve a useful purpose to overcome this. Instead of attempting to describe the entire system by a continuous function, a numerical method such as FEM discretises the system into miniscule elements which are each treated as continuous systems linked to each other. As one might expect, a greater number of elements or 'mini-systems' used to discretise the system leads to a greater likelihood of an accurate representation of information across it. Doing so, however, increases the computational burden on the numerical solver - the model solved in Section 1.2 is a case in point.

The third approach, which is based on experimental analysis, is one that is adopted either when there is no reliable manner of solving the problem computationally, or for verifying results obtained through that means. The costs involved in executing experiments with precision, however, generally makes it prohibitively expensive to adopt for performing iterative analysis. More often than not, data from experiments are recorded meticulously for future reference to avoid repeating them for identical models. In fact, the data is also used to identify trends from, so that a new model may be solved merely by interpolating/extrapolating from previous models. This is the basis of empirical science - the philosophy of solving problems and making decisions based on previous observations and experiences, which simplify the solution process greatly. Some well-known relationships in physics and engineering have been established empirically through interpolation of data, such as the ones listed below.

- Ideal gas law [3]: This is a relationship that relates the pressure (P), volume (V) and temperature (T) of an ideal gas as

$$PV = nRT \tag{2.1}$$

  where n is the number of moles of the gas, and R the universal gas constant.

- Surface water evaporation rate [4]: The rate of evaporation of water per hour (g_h) from an open surface has been empirically determined to be follow the line

$$g_h = \Theta A(x_s - x) \tag{2.2}$$

  where Θ is the evaporation coefficient, A the surface area of exposed water, x_s the maximum humidity of saturated air at the surface temperature, and x the humidity ratio of air in the its current state.

- Rydberg formula [5]: The expression used to calculate the wavelengths of a spectral line in a chemical element (λ_{vac}) is an interesting one that was first derived empirically, and then came to be proved theoretically once its existence was known. For principal quantum numbers n_1 and n_2, the equation is given by

$$\frac{1}{\lambda_{vac}} = RZ^2\left(\frac{1}{n_1^2} - \frac{1}{n_2^2}\right) \tag{2.3}$$

where R is the Rydberg constant for the element, and Z its atomic number.

- Air pressure loss [6]: The pressure drop (Δp) in air-carrying pipelines of length L and diameter d has been determined to vary in accordance with the equation

$$\Delta p = 7.57 \cdot 10^4 \cdot q^{1.85} \cdot \frac{L}{d^5 p} \tag{2.4}$$

where q is the volume flow rate of air, and p the initial pressure.

The data on the basis of which these relationships have been identified come from tests carried out in a laboratory, using measuring instruments to study physical parameters. However, the notion of an *experiment* need not be limited such a setup - it is entirely possible to generate data using a computational process such as, say, finite element analysis.

An observation that can be made from the empirical relationships (such as those given in Eq. (2.1) to (2.4)) is that while some may result from simple trends that can be spotted easily by the eye and fine-tuned through trial and error, a majority require a more methodical analysis procedure. Consider some illustrative data points given in Figure 2.1, where the each axis represents a parameter, and the relationship between them is to be determined in each of the three cases - it is easy to suggest the nature of a curve that fits them well, but the exact parameters of the curve that does this best is less trivial. This is carried out through a process known as *regression analysis*, which is described in the following section.

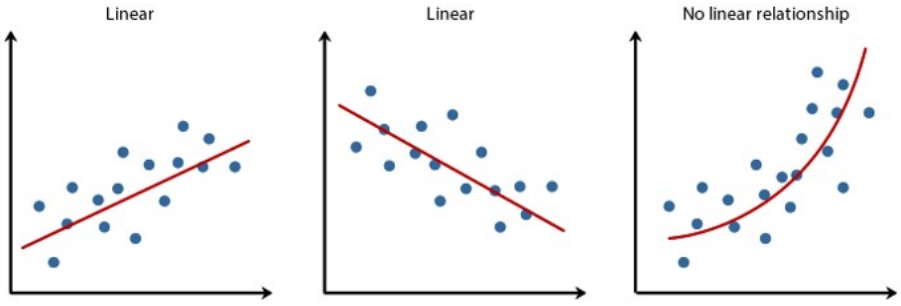


Figure 2.1: Fitting data to a curve [7]

2.2. Regression Analysis

Regression analysis is used to determine the dependence of output variables on input variables, a dependency determined from observations recorded on the behaviour of these variables. Consider the linear relationships shown in Figure 2.1 - these represent equations of the form $y = mx + c$. When just two data points are available, ie, two sets of (x, y), it is trivial to determine the values of m and c that fit the two points. However, when there is a larger set of data points available, as shown in the figure, an 'average' line of sorts is to be determined. This is done most commonly using the method of least squares.

2.2.1. Method of Least Squares

For a line chosen to fit the scatter of points, the square error for each x value is computed using the *true* y value corresponding to it (from the recorded observations), and the *predicted* value that is outputted by the equation for the line. The performance of the system as a whole is quantified by summing the square errors of each pair of true and predicted y values. To avoid penalising models that use a larger number of data points, the performance is often quantified using the mean of the square errors rather than the sum.

The method of least squares aims to determine the line that yields the smallest square error [8]. This line, as we have already seen, is a function of the form $y = f(x, m, c)$, and outputs the predicted y values. For a generalised data point (x_i, y_i), the error (or residual) is calculated as

$$r_i = y_i - f(x_i, m, c) \tag{2.5}$$

The optimal value of m and c are determined by reducing the sum of the squares of this residual, given by S.

$$S = \sum_i r_i^2 \tag{2.6}$$

This is minimised by setting the gradients with respect to the variables to be determined (m and c), to zero.

$$\frac{\partial S}{\partial m} = \sum_i 2 r_i \frac{\partial r_i}{\partial m} = 0 \tag{2.7}$$

$$\frac{\partial S}{\partial c} = \sum_i 2 r_i \frac{\partial r_i}{\partial c} = 0 \tag{2.8}$$

which reduces to the expressions given by Eq. (2.9) and (2.10). Solving them for m and c yields the best fit line for available the data points.

$$\sum_i 2\left(y_i - m x_i - c\right)(-x_i) = 0 \tag{2.9}$$

$$\sum_i 2\left(y_i - m x_i - c\right)(-1) = 0 \tag{2.10}$$

2.2.2. Higher Order Curve-Fitting

Determining a best-fit linear relationship between observed parameters can often by limiting - more complex relationships suffer from large square errors and are less reflective of the actual relationship between them. Consider for instance, the third chart shown in Figure 2.1 - a second degree polynomial appears to better explain the relationship between the variables than a straight line, which is of degree 1.

To obtain a best-fit quadratic polynomial, the method of least squares is used again, only that the process is carried out using an appropriately modified line (also called the *fitting curve*) - $f(x) = mx + c$ is replaced by $f(x) = ax^2 + bx + c$. Obviously, because of the additional independent variable in the polynomial, an extra step is involved in solving for a, b and c. This logic may be extended to polynomials of any degree, and even relationships in higher dimensions [9], examples of which are shown in Figure 2.2.

The curves we have seen until now, including the first one in Figure 2.2, have been models with one input and one output. This does not always have to be the case though - the second plot in the same figure is obtained for two inputs and one output, and applies the method of least squares with a fitting-curve of the form $z = f(x, y)$. Likewise, a similar model may have one input and two outputs of the form $f_1(x)$ and $f_2(x)$, with the overall error function to be minimised being the sum (or a weighted sum) of the errors of each.

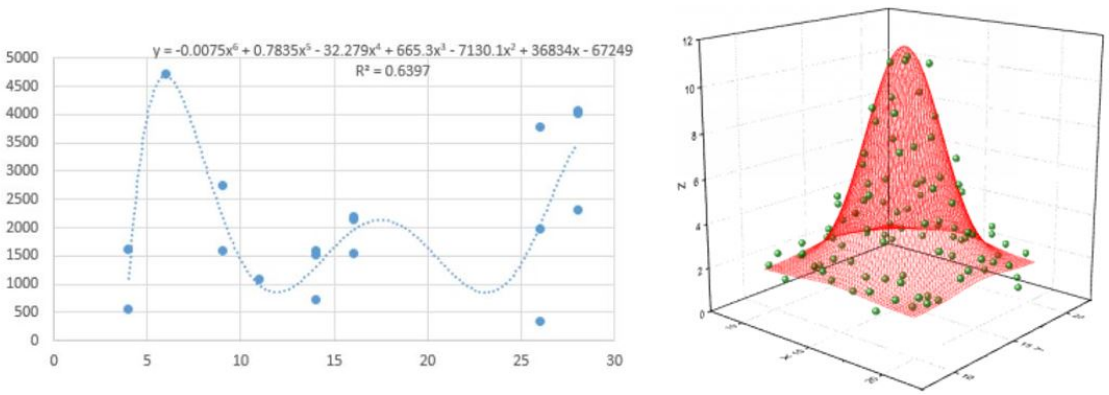


Figure 2.2: Fitting with higher degree polynomials [10] (left) and data points in three dimensions [11] (right)

A natural question that arises from this discussion is regarding the choice of fitting function - how is an appropriate selection of this done? An understanding of bias and variance helps answer this question.

2.2.3. Bias, Variance, Underfitting and Overfitting

Bias and variance are two key parameters that specify the performance of a regression model. In fact, the Mean Square Error (MSE), which is essentially the error computed in the method of least squares, is a performance indicator that may be decomposed into bias and variance terms [12] as

$$\text{MSE} = E\left[\left(y - f(x)\right)^2\right] = \left(\text{Bias}\left[f(x)\right]\right)^2 + \text{Var}\left[f(x)\right] + \sigma^2 \tag{2.11}$$

where $E(X)$ is the expected value of X. The σ^2 term represents the irreducible error that results from noise in the true curve. The bias and variance terms can be computed separately as

$$\text{Bias}\left[f(x)\right] = E\left[f(x)\right] - y \tag{2.12}$$

$$\text{Var}\left[f(x)\right] = E\left[f(x)^2\right] - E\left[f(x)\right]^2 \tag{2.13}$$

The first of these two expressions, the bias, is representative of the difference between the predicted output and the true output. This difference may be expected to be large when a simple function is used to fit a more complex relationship, such as the model shown in the first plot of Figure 2.3. Here, the fitting curve is said to be highly biased in its predictions, and performs better with data points that lie close to the line. In contrast, the second plot shows a model having practically no error between the predicted and true outputs, or in other words, very low bias - the fitting curve is 'unbiased' and does not favour one point or the other, irrespective of where they lie. Such a curve is obtained by applying the method of least squares with a much higher degree polynomial as the fitting-curve.

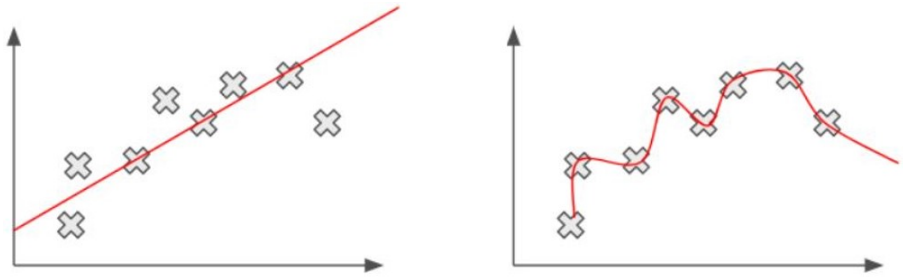


Figure 2.3: Fitting data points with high bias and low variance (left), and with low bias and high variance (right) [13]

Variance, on the other hand, represents the width of the spectrum within which the predicted outputs lie. The high-bias example, for instance, is a linear model whose outputs do not move significantly around its mean, thereby having a low variance (which can also be understood as a lower 'variation' in values). The low-bias example, on the other hand, has a larger variation around its mean, and therefore has a large variance.

These two scenarios are also referred to as *underfitting* and *overfitting* respectively. In the centre of Figure 2.4 lies an optimal fit with respect to which the other two are considered to be fit 'under' or 'over'. As the names suggest, the underfit model oversimplifies the relationship between the parameters and does not conform to the data points sufficiently, while the overfit model goes overboard with trying to fit the data points, and ends up fitting the noise as well. The optimal model, meanwhile, fits the data with a smooth curve and has a good balance between bias and variance.

The question raised in the previous subsection can now be better addressed - on what grounds is the choice of this optimal function made? Unfortunately, there is no hard-and-fast rule for doing so. A general rule of thumb is to use as low a degree as possible for the fitting-polynomial - lower degree polynomials tend to be 'smoother' curves, while higher degree polynomials generally comprise of several inflection points. In engineering problems, it is quite common to observe a 'smooth' relationship between parameters, which makes it seem more natural to select such a curve to fit observations. Often, the data analyst also knows either from experience with similar data, or through theories that contribute to the understanding of observations, the kind of curve that would fit the data best.

One of the reasons why overfitting is avoided, even though it appears to fit the data points 'perfectly', is that the resulting relationship is overly dependent on the data set used. It is therefore quite likely that new observations

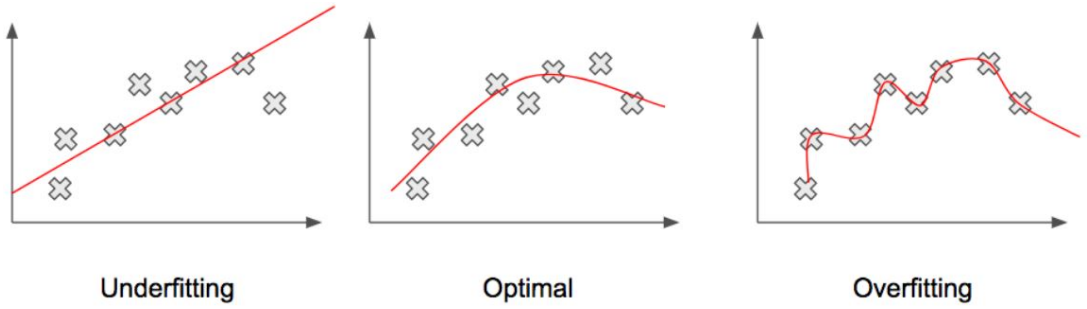


Figure 2.4: Data points being underfit (left), optimally fit (centre) and overfit (right) [13]

made on these parameters would deviate from the established relationship. When a large enough dataset is available, it is recommended to set aside a portion of it so that the model can be later evaluated on an independent dataset that was not involved in the curve-fitting process. In fact, when large datasets, higher order functions and multi-dimensional relationships such as those observed in structural engineering are involved, it is these very concepts that are expanded to form the domain of machine learning, which is the subject of discussion in the next section.

2.3. Machine Learning

Machine learning is the process of an 'intelligent' system inferring relationships between data through patterns, as opposed to explicit instructions [14]. The principles on which it recognises these patterns are very similar to those pertaining to regression analysis, and are even more useful when dealing with complex datasets - a large number of inputs and outputs can be handled. For ease of visualisation through plots, the techniques used are explained using 2D or 3D relationships.

A short description of the nature of problems that machine learning is commonly applied to is given below.

- **Regression:** Computing numerical outputs for a given set of numerical inputs (for example - determining housing price in a neighbourhood given the floor size and number of rooms)
- **Classification:** Assigning a label to a particular input (for example - classifying an image as 'contains a cat' or 'does not contain a cat')
- **Clustering:** Analysing similarities and differences within a dataset and segregating them into groups that are not immediately obvious (for example - determining categories of website users and tailoring content/ads for each)
- **Dimensionality Reduction:** Compressing data possessing a large number of features to fewer features without losing vital information (for example - reducing a large image to a smaller image without losing information on whether it contains a cat or not)

In each case, the model *learns* to observe patterns through examples that are fed to it.

2.3.1. Learning Process

The learning process is characterised by the feeding in of datasets comprising of inputs and outputs - the model is *trained* to develop a relationship between them, with the objective being to then predict an output reliably given an input that it had not seen before in the training process. As described in Section 2.2.3, this performance is usually evaluated by splitting the entire dataset into two parts - one called the *training set*, which is used to train the model in developing a relationship, and the other the *test set*, which serves as an independent dataset for performance evaluation. During the training process, the following situations are normally encountered:

- **Large training set error:** A large error on the training set indicates that the model is underfitting the data, the relationship between which is more complex than initially assumed while configuring the model. A suitable way of dealing with this is to repeat the training process using a more complex model (analogous to using a higher degree polynomial in basic regression). Until the training set error is satisfactory, the test set is not given any consideration.

- **Small training set error but large test set error:** A small training set error indicates that the model is fitting the training data well. However, the error may still be large on the test set - when this is indeed the case, it is an indication that the model is overfitting the training data and is therefore not performing well on an independent dataset. An appropriate way of dealing with this is to reduce the complexity of the model (analogous to using a lower degree polynomial in basic regression).
- **Small training set error and small test set error:** A small error on both the training set and the test set indicates an optimal configuration for the model. While the training set error in this case may be higher than in the overfit model, this model possess more reliability with its predictions due to its performance on an independent dataset.

The ideas of 'large' and 'small' errors are relative and are dependent on the preference of the user. Usually, the process of determining the optimal configuration is carried out until the test set error cannot be improved discernibly. If the user is still unsatisfied with the performance of the system, using a different model type, or obtaining a larger dataset for training and testing is recommended.

The complexity that was being referred to analogously with the polynomial degree in basic regression, is defined differently in each form of machine learning model. It is no surprise then, that bias and variance considerations are to be made in all cases to obtain the optimal model configuration. A description of these different forms of machine learning models is provided in the following section.

2.3.2. Model Types

While all forms of machine learning models operate with the principle of assuming a function and modifying it to reduce the error with respect to the data being fed into it, these models differ structurally, and the manner in which computations are made. Given below is a summary of the fundamental structural elements of five commonly used model types - decision tress, Bayesian networks, genetic algorithms, support vector machines, and artificial neural networks.

Decision Trees

Decision Trees are models that work very similarly to the way flowcharts do - decisions are made, or conclusions are arrived at, depending on the path taken. The choices available are represented as branches of the tree, and the decisions/conclusions as its leaves. Figure 2.5 shows an example of such a model [15] - the model concludes that the chances of a survivor aboard the Titanic was greater if they were either female, or male and younger than 9.5 years old and having fewer than 2.5 siblings. The values of the probabilities are arrived at by feeding passenger data to the model.

While a model of this form appears simple and intuitive, it is often very dependent on training data and can be difficult to get to work well on an independent dataset while still maintaining performance [16].

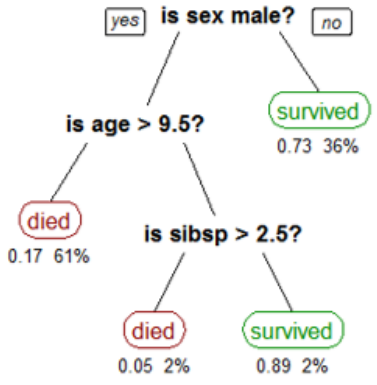


Figure 2.5: A decision tree that predicts the survival chances of passengers aboard the Titanic [15]

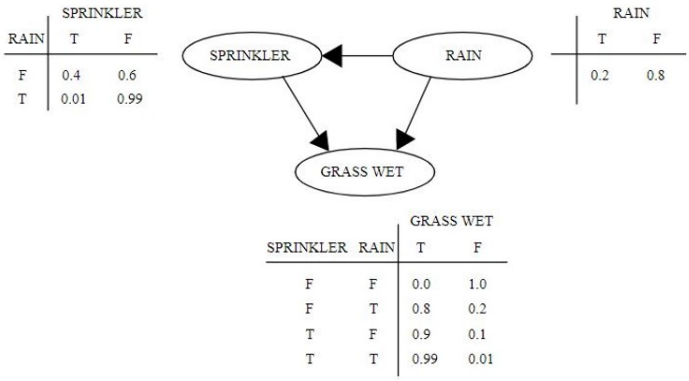


Figure 2.6: Bayesian network that estimates the probability of grass being wet given the possibilities of rain and sprinklers

Bayesian Networks

A Bayesian network is a graphical model that represents the probabilistic relationships between variables. Consider the model shown in Figure 2.6 - given the probabilities of rain, the grass getting wet by rain and/or sprinklers, and the relationship between rain and the use of sprinklers, the probability of the occurrence of

events surrounding them can be computed. As is the case with decision trees, these probabilities are learnt through the dataset fed into it.

Such networks are useful for understanding casual relationships between parameters, and is less of a 'black box'. However, with more complex models, these networks are computationally very intense, which is something to be avoided for surrogate modelling [17].

Genetic Algorithms

Genetic algorithms are optimisation algorithms that are inspired by the evolutionary process of natural selection, and involve the use of operators such as mutation, crossover and selection. They work by initiating several potential solutions, and let them evolve through mutations and alterations - in the end, better solutions survive, resulting in well-optimised models.

The wide range of solutions it offers is useful, particularly when we wish to compare their uses. However, finding a well-fitting model for a complex problem often requires prohibitively expensive computational resources. Another drawback is that the solutions obtained do not necessarily include the most optimal solution, and we are therefore limited to only being able to pick the 'best' from those available [18].

Support Vector Machines

Support Vector Machines (SVMs) work by constructing hyperplanes that separate data, and are commonly used for classification and object detection, but can also be extended to regression [19]. Figure 2.7 shows an example of this on a 2D plane - however, these hyperplanes may be constructed in higher dimensional spaces as well [20].

When configured properly, SVMs are capable of solving complex problems, even though the resulting models may be somewhat of a black box.

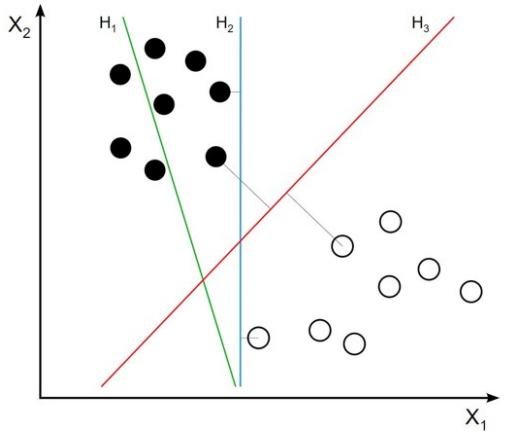


Figure 2.7: Data points being sorted using hyperplanes H_1, H_2 and H_3 [21]

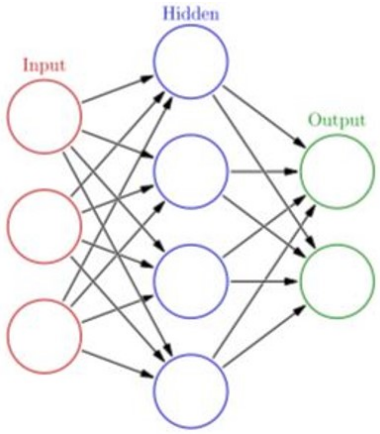


Figure 2.8: A neural network with 3 input units, 4 hidden units, and 2 output units

Artificial Neural Networks

Artificial Neural Networks (ANNs) are systems that have over the years become extremely robust ways of carrying out classification and regression. A standard neural network consists of an input layer, an output layer, and in between them, one or more hidden layers. Figure 2.8 shows a model that computes an output set comprising of two parameters, given an input comprising of three. Within the network, these are also called *units* or *nodes*, with the hidden layer shown comprising of four. Information is passed from the input units, through the hidden units, until they reach the output units.

Like SVMs, these are also often black boxes, with the hidden units not explicitly processing separate, perceivable subdivisions of the data [22, 23]. However, through years of research and development, these networks have impressive capabilities when it comes handling data efficiently and learning trends quickly [24]. ANNs are therefore an attractive choice of machine learning model to explore an empirically-driven structural analysis scheme, and is the focus of the remainder of this literature review.

2.3.3. Network Modelling

The modelling process of a machine learning model is one that requires a great deal of decision-making from the user, and is not one that is based on adherence to hard-and-fast rules. Several tunable variables influence the speed of training, and the performance of the network. For reasons outlined in the previous subsection, these aspects of modelling and training are described with regard to artificial neural networks (commonly referred to as simply *neural networks).*

Standard Neural Network

In a typical neural network, each layer takes the previous layer as the input and performs operations on it. The resultant outputs are then treated as inputs for the layer that follows it. To be more precise - each unit in a hidden layer receives inputs from each unit in the preceding layer, and upon combining them in a suitable manner, they pass on the output to each unit of the next layer. Figure 2.8 and Figure 2.9 show examples of such an interconnected network. Obviously, the input layer of the network is directly fed with inputs and do not possess a 'previous layer' to process information from. Likewise, the output layer gives the final outputs of the network with no further processing required, and therefore have no further layers succeeding it.

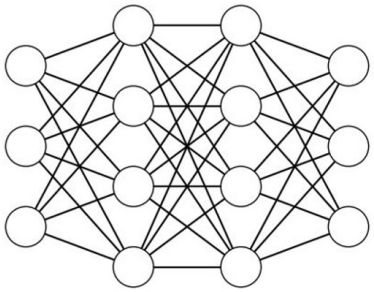


Figure 2.9: A standard neural network, where every unit is connected to every unit of both the preceding and succeeding layers

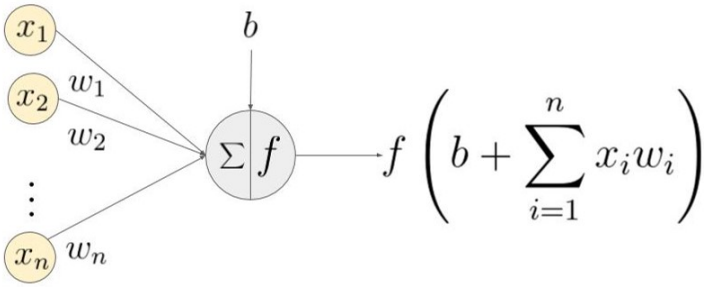


Figure 2.10: A single unit in a layer first linearly combines all its inputs using individual weights, and then applies a function f to it to obtain its output [25]

Consider the schematic in Figure 2.10 to understand what a single unit (also known as a neuron) in a neural network is doing. It receives n inputs from the previous layer, which it linear combines to obtain a single quantity. It applies a weight $w_1, w_2, ...w_n$ to each of these inputs before summing them together and then adding a *bias* term b to it [26]. This is analogous to multiplying the input x with the slope m and then adding the constant term c to it in 2D linear regression. Now, in order to truly utilise the power of neural networks in determining complex relationships, a non-linear function f, called the *activation function* is applied to it. Without this activation function, all the hidden layers of the network can be mathematically reduced to a single hidden layer, which underutilises its potential [27]. With its application, the complexity of problems it can solve increases with an increasing number of neurons, both in the form of additional units in a layer, as well as additional layers itself.

Looking back at Figure 2.9 now, we can see that given a dataset of input values and their corresponding output values, it is reasonable to expect to be able to relate the two approximately through appropriate choices of weights and biases, as long as a sufficiently large dataset is available. These weights and biases are termed the *parameters* of the network. In the example shown, the first hidden layer has 4 units, each having parameters of the form w_1, w_2, w_3, b - a total of 16 independent parameters. Likewise, the units of the second hidden layer comprises of parameters of the form w_1, w_2, w_3, w_4, b (20 independent parameters for the entire layer). The units of the output layer have the same number of parameters as the units of the second hidden layer, but for 3 units instead of 4, resulting in 15 more independent parameters. Thus, in total, the model contains 51 independent parameters used to generate a set of 3 output values given a set of 3 input values.

Unlike the method of least squares used in simple regression, however, it is in general not possible to directly obtain the parameters w and b that minimise the error from a system of equations using a computationally efficient operation [28]. The higher dimension and non-linearity of the problem means that the iterative process of training is necessary to converge towards the minimum loss, for which an algorithm called gradient descent is used.

Training using Gradient Descent

Gradient descent is an iterative optimisation algorithm that minimises a given cost function. When used to train neural network models, the cost function is set as the error between the true output and the predicted output. Often, as seen in the case of simple regression, the Mean Square Error (MSE) is used. For ease of understanding, the working of this algorithm is explained using a single dimensional model.

Consider a model with a single parameter w, and a cost function $J(w)$. The value of w that minimises J is to be determined using gradient descent. To start with this, a random initial value for w is assumed, and J is computed using it. This is depicted in Figure 2.11, where the objective is to go from the initial weight, to the weight that gives $J_{min}(w)$. Performing this 'descent' requires shifting w to the left, an operation that is carried out using the gradient at the initial point [29], as show in Eq. (2.14).

$$w := w - \alpha \frac{dJ}{dw} \tag{2.14}$$

The ':=' operation indicates an updating of the value of w, which makes it take a step towards the desired value. When the operation is repeated in subsequent steps, it moves closer and closer towards the global cost minimum.

α in the equation is termed the *learning rate*, and controls the magnitude of the shift. It is one of the several variables that influence the training of the model - together, they are termed as *hyperparameters* (note that these are different from what the term *parameters* refers to). A small learning rate means that a small step is taken from the current point towards the minimum. The consequence of this is that the algorithm takes longer to determine the optimal value of w. However, if α is increased drastically in order to speed up convergence, there is a greater likelihood of the minimum being overshot, and possibly even diverge from there.

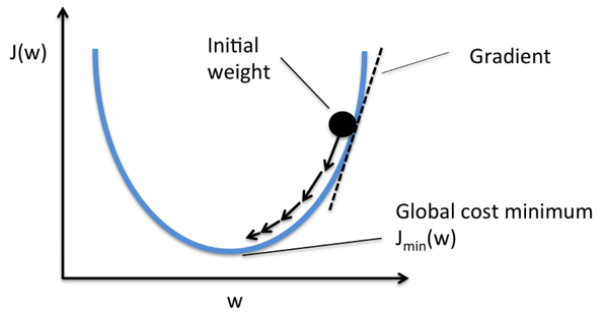


Figure 2.11: Gradient descent with a single parameter w [30]

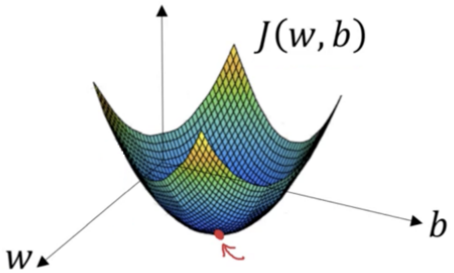


Figure 2.12: Gradient descent with two parameters w and b

Figure 2.12 shows another such function, but with 2 parameters w and b. Starting from a random initial point (w, b), the update rule is performed on both parameters to approach the global minimum.

$$w := w - \alpha \frac{\partial J}{\partial w} \tag{2.15}$$

$$b := b - \alpha \frac{\partial J}{\partial b} \tag{2.16}$$

The same logic may be extended to any number of parameters, as is the case in larger networks. It is worth noting, however, that some models will tend to converge to a local optimum as opposed to a global optimum, depending on what initial value is taken. For this reason, models are often evaluated with multiple random initialisations before accepting the results [31]. Additionally, convergence often slows down significantly as you approach the minimum - it is therefore not practical to carry out iterations endlessly to reach the exact minimum. Instead, once the cost is deemed to have stopped improving to a sufficient extent, the process is terminated, and the parameter values in that present state are accepted as optimal.

Modified Gradient Descent Algorithms

The standard gradient descent algorithm, especially with a large number of parameters, can often take a long time to converge [32]. This is largely because every update can take the parameter values in a direction that

causes it oscillate several times before it reaches the minimum. Figure 2.13 shows this occurring in two dimensions. In order to reduce the oscillations and speed up the process of convergence, several modified algorithms have been developed - three such popular ones are discussed below.

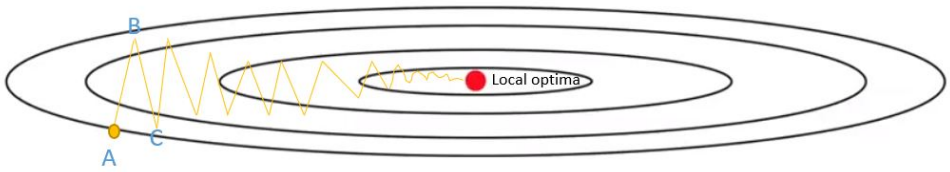


Figure 2.13: Standard gradient descent oscillating towards the local optima [33]

1. **Gradient descent with momentum:** This algorithm aims to dampen oscillations and drive the parameters towards the direction of the minima. Instead of directly using the computed gradients in the update rule, weighted averages between the computed gradients and those from the previous iteration are used instead. This serves as way of dampening the oscillations in the derivatives. Eq. (2.17) to (2.20) show this mathematically, with the *momentum* term β indicating the weight. The value of β ranges from 0 to 1, and is recommended to be set at approximately 0.9 [34].

$$v_{dw} = \beta v_{dw} + (1-\beta)\frac{\partial J}{\partial w} \tag{2.17}$$

$$w := w - \alpha v_{dw} \tag{2.18}$$

$$v_{db} = \beta v_{db} + (1-\beta)\frac{\partial J}{\partial b} \tag{2.19}$$

$$b := b - \alpha v_{db} \tag{2.20}$$

The values of v_{dw} and v_{db} are initialised as 0 for the first iteration. Fig. 2.14 shows the effect of using momentum - the iterative process quickly approaches the local optima as opposed to oscillating around it.

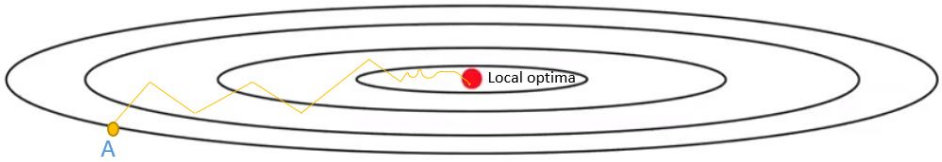


Figure 2.14: Gradient descent with momentum showing dampened oscillations and heading faster towards the local optima [33]

2. **RMSprop:** The Root Mean Square Propagation (RMSprop) algorithm also aims to provide more directionality of the gradient towards the optima, but differs from the previous algorithm in terms of how the gradient is calculated. When performed in the manner given by Eq. (2.21) to (2.24), the gradient puts more emphasis in the direction towards the local optima, and less emphasis in the direction perpendicular to it, thereby allowing the use of higher learning rates than possible by simply using momentum [35].

$$v_{dw} = \beta v_{dw} + (1-\beta)\left(\frac{\partial J}{\partial w}\right)^2 \tag{2.21}$$

$$w := w - \alpha \cdot \frac{1}{\sqrt{v_{dw}} + \epsilon} \cdot \frac{\partial J}{\partial w} \tag{2.22}$$

$$v_{db} = \beta v_{db} + (1-\beta)\left(\frac{\partial J}{\partial b}\right)^2 \tag{2.23}$$

$$b := b - \alpha \cdot \frac{1}{\sqrt{v_{db}} + \epsilon} \cdot \frac{\partial J}{\partial b} \tag{2.24}$$

ϵ is a small factor (usually taken as 10^{-7}) used to ensure that Eq. (2.22) and (2.24) do not encounter divide-by-zero errors.

3. **Adam:** The Adaptive Moment Estimation (Adam) algorithm combines the concepts of the previous two algorithms. The gradients and their squares are exponentially weighted using separate momentum terms β_1 and β_2, and are calculated as shown in the equations below [36].

$$v_{dw} = \beta_1 v_{dw} + (1 - \beta_1) \frac{\partial J}{\partial w} \tag{2.25}$$

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) \left(\frac{\partial J}{\partial w} \right)^2 \tag{2.26}$$

$$v_{db} = \beta_1 v_{db} + (1 - \beta_1) \frac{\partial J}{\partial b} \tag{2.27}$$

$$S_{db} = \beta_2 S_{db} + (1 - \beta_2) \left(\frac{\partial J}{\partial b} \right)^2 \tag{2.28}$$

Since exponentially weighted averages are inherently inaccurate during early phases of estimation, bias correction is applied to them. The superscript c in Eq. (2.29) to (2.32) indicate the corrected gradients, with the t denotes the iteration number.

$$v_{dw}^c = \frac{v_{dw}}{1 - \beta_1^t} \tag{2.29}$$

$$S_{dw}^c = \frac{S_{dw}}{1 - \beta_2^t} \tag{2.30}$$

$$v_{db}^c = \frac{v_{db}}{1 - \beta_1^t} \tag{2.31}$$

$$S_{db}^c = \frac{S_{db}}{1 - \beta_2^t} \tag{2.32}$$

The update rules may now be applied.

$$w := w - \alpha \frac{v_{dw}^c}{\sqrt{S_{dw}} + \epsilon} \tag{2.33}$$

$$b := b - \alpha \frac{v_{db}^c}{\sqrt{S_{db}} + \epsilon} \tag{2.34}$$

Activation Functions

We now know how to determine the optimal parameters using different gradient descent-based algorithms. The basis of these is the cost function, which is in most cases the MSE between the true outputs and the predicted ones. Computing the latter makes use of an activation function f, as we had observed earlier in Figure 2.10. In the early days of neural networks, the sigmoid function, given by Eq. (2.35) was a popular choice which gave output values between 0 and 1. Then there came a time when having the output centred at zero was deemed better for optimisation, an option that was provided by the hyperbolic tangent (tanh) function (Eq. (2.36)), whose outputs lie between -1 and 1 [37]. Both these functions, however, suffer from the issue of vanishing gradients - beyond a small region on either side of $x = 0$, their gradients begin to diminish quickly, which leads to slower learning. An activation function that overcomes this and is popular today is the Rectified Linear Unit (ReLU), given in Eq.(2.37) [38].

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.35}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.36}$$

$$\text{ReLU}(x) = \max(0, x) \tag{2.37}$$

Figure 2.15 shows these equations in graphical form, which makes clearer the issue of vanishing gradients in the sigmoid and tanh functions.

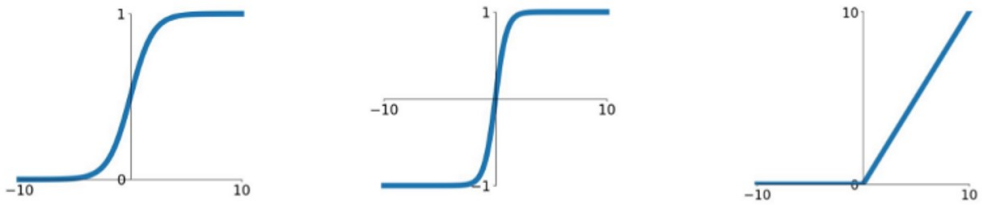


Figure 2.15: Sigmoid function (left), tanh function (centre), and ReLU function (right)

Convolution Neural Network

Convolutional Neural Networks (CNNs) are networks that make use of the convolution operation in the computation of its output, and a modified architecture style to suit it. The principles of training and learning, however, are largely the same. Such a network is most commonly used to extract features from an image [39], which often helps in reducing patterns to a single block of information. This offers potential in processing FEM outputs, and is studied further.

The convolution operation, denoted by '$*$', is applied between a matrix and a *filter*. The filter is generally smaller than the matrix, and is applied to it region-by-region. It extracts features from the regions by multiplying the corresponding values and then summing them. Figure 2.16 shows an example of this. The filter size f refers to the dimension of the filter, the stride s to magnitude of the movement of the filter (before repeating the convolution operation on another region of the matrix), and the padding p to the thickness of a pad of 0s or 1s applied to the matrix before convolving (this is sometimes done to control the dimensions of the output matrix).

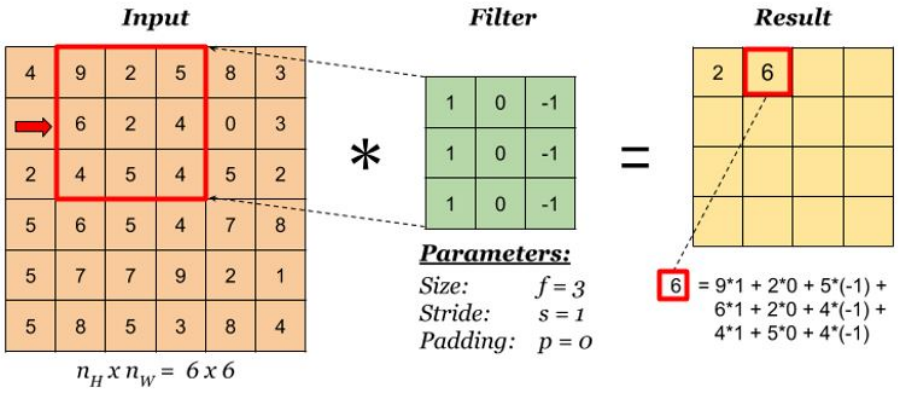


Figure 2.16: Example of an input being convolved with a filter [40]

An image is essentially a matrix, with each pixel representing an intensity. Based on the choice of filter, therefore, different features can be extracted from an image. Figure 2.17 shows a picture of a house on the left, and the result of two separate convolutions on the right - the first convolution (on top) is done using a horizontal line filter, while the second one uses an edge filter. As their names suggest, the two filters extract the horizontal

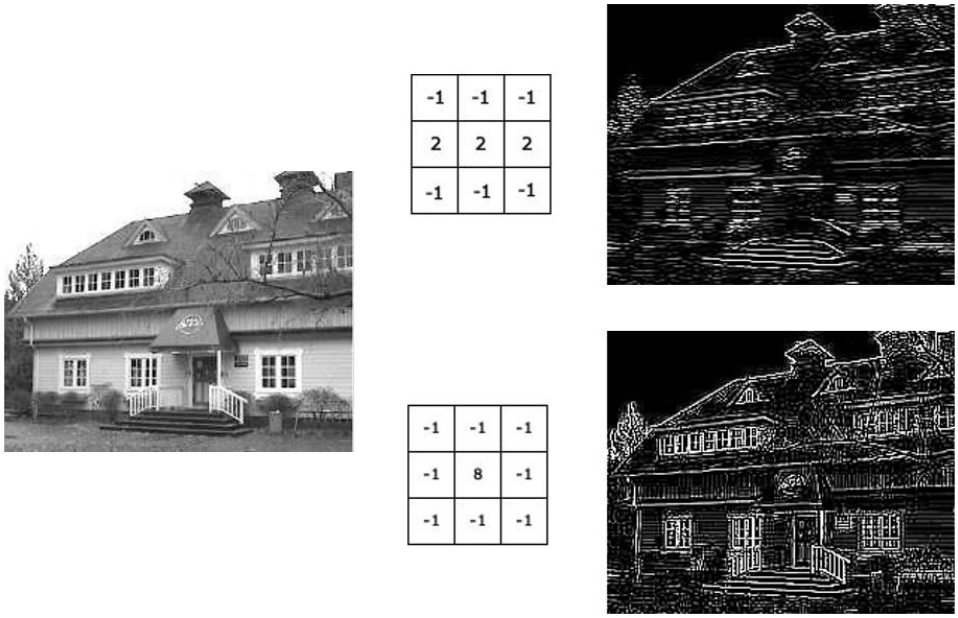


Figure 2.17: Image of a house (left) being subject to a horizontal line filter (top), and an edge filter (bottom) [41]

lines, and all edges respectively from the image.

The two filters shown in the image are two among several such that have been developed for extracting common features. The idea with a CNN is to make use of a 'trainable' filter that performs the same task - instead of using one with predetermined values as is the case with known filters, those in a CNN are comprised of variables that are randomly initiated and the then trained to find the optimal values. This helps extract features without explicitly knowing what kind of features are repeating in the images. Often, certain features are not obvious and structured enough to be identified by a human user, but exist nonetheless, making training of this nature extremely useful.

The potential that both the standard and the convolutional neural networks possess have been tapped to different degrees in various applications. In the past few decades, the domain of structural analysis has also seen attempts to develop predictive models using data from both computer simulations and laboratory experiments. An outline of some of this research is provided in the following section.

2.4. Neural Networks in Structural Analysis

The use of neural networks in structural engineering research began to popularise in the early 90s. Problems analysed in these ranged from simple static structures, to more complex dynamic responses. In this section, a few of these models are described, and the results obtained from them are discussed.

2.4.1. Truss Analysis

A simple 10-bar truss sought to be weight-optimised is shown in the left of Figure 2.18. Based on results obtained through analytical methods, a neural network was trained by Berke and Hajela [42] to predict displacements d_6 and d_8, given the cross-sectional areas of each of the 10 members. The architecture of the network, comprising of two hidden layers with 6 units each, is shown in the right of the figure. With the assumption that the area of each member is proportional to its weight, the truss can be optimised through quick computations of displacements for different area combinations. Table 2.1 shows an optimal truss configuration in terms of cross-sectional area, obtained using the neural network (NN), and compared to the exact analytical solution. For a set of 100 training examples, the cost function shows a high degree of similarity (a difference of barely 1%), while the individual areas computed for each member are identical in certain cases (X_2 for instance), but differ by up to 40% in some others (such as X_7). This early model appears to give acceptable ballpark estimates, but does not seem to have a high degree of reliability.

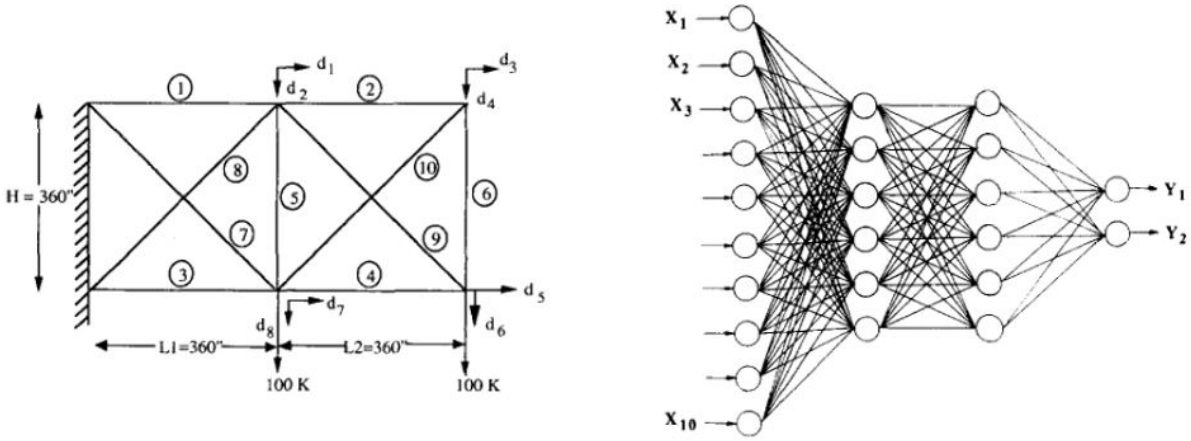


Figure 2.18: 10-bar truss displacing under load (left), and the neural network architecture used to predict displacements d_6 and d_8 (right) [43]

Cross-Section	NN Result (in^2)	Analytical Result (in^2)
X_1	30.508	30.688
X_2	0.100	0.100
X_3	26.277	23.952
X_4	11.415	15.461
X_5	0.100	0.100
X_6	0.413	0.552
X_7	5.593	8.421
X_8	21.434	20.605
X_9	22.623	20.554
X_{10}	0.100	0.100
Cost Function	5010.22 lb	5063.81 lb

Table 2.1: Comparison between NN and analytical results for an optimised truss [43]

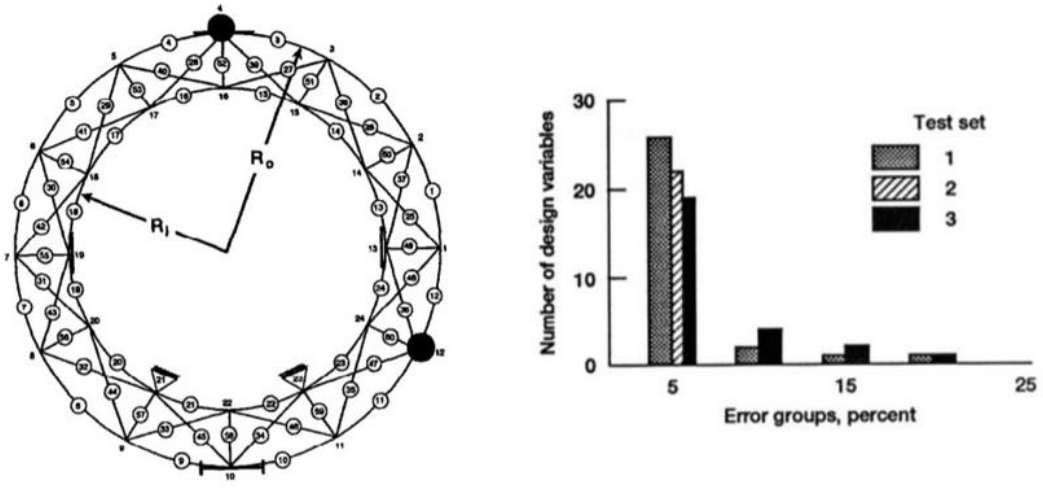


Figure 2.19: Trussed ring with variable number of elements (left) and observed errors between neural network and FEM results (right) [44]

In similar vein, a 60-bar trussed ring was analysed by Berke *et al.* [44] to optimise the radii (R_i and R_o) and link combinations between nodes shown in the left of Figure 2.19. The weight and stress were considerations with respect to which the optimisation was carried out. The neural network was trained for 20,000 iterations using 120 data points each for 3 different loading conditions, with another 5 data sets in each case for testing. The right of Figure 2.19 shows error groups for each loading condition, with the y-axis indicating the number

of variable node-link combinations. A majority of the design combinations showed an error in the 5% range, while a few went up to the 20% mark. The dataset used for training/testing took over 35 CPU hours - generating a larger dataset would have taken even more, but may have resulted in the error values shown shifting more to the left. The balance between the two is a cost-benefit situation that is dependent on the application of the network.

2.4.2. Beam and Plate Analysis

Problems of higher dimensional structures were also analysed. For instance, Kapania and Liu [45] estimated stiffness quantities for three dimensional beams using neural networks comprising of 10 hidden units and a single output unit. The number of input units were varied as 2, 3 and 4, with one of the inputs reserved for the length of the beam, and the other(s) for one, two or three surface areas. These simple models showed very close correspondence to results obtained from FEM. The shearing rigidity values predicted with 2 inputs, for instance, threw a maximum error value of less than 0.1%, which is not a surprise given the size of the network.

In another research by Waszczyszyn and Ziemianski [46], elastoplastic plane stress conditions were simulated on a notched plate. Figure 2.20(a) shows the loading conditions of the plate, with Figure 2.20(b) showing the FE mesh solved by ANKA code (a finite element solver). The resulting displacements at different load and strain hardening (χ) values are shown in Figure 2.20(c) - the dotted line indicates the results from the ANKA-H code, which is a modified ANKA code that incorporates the NN model. The results show strong correspondence with those computed by the unaltered ANKA code - the deviations at higher χ values indicate the heightened non-linear effects that the NN model has not learned precisely. This network was trained using two hidden layers of 40 and 20 units respectively, and 3349 training examples - the large number of training examples appears to have had a positive effect on its performance. Nevertheless, it is possible that a greater number of hidden units (at the risk of overfitting) or an even greater number of training examples (at the risk of exceeding allocated computational resources) could have resulted in the model learning the non-linear relationship more precisely.

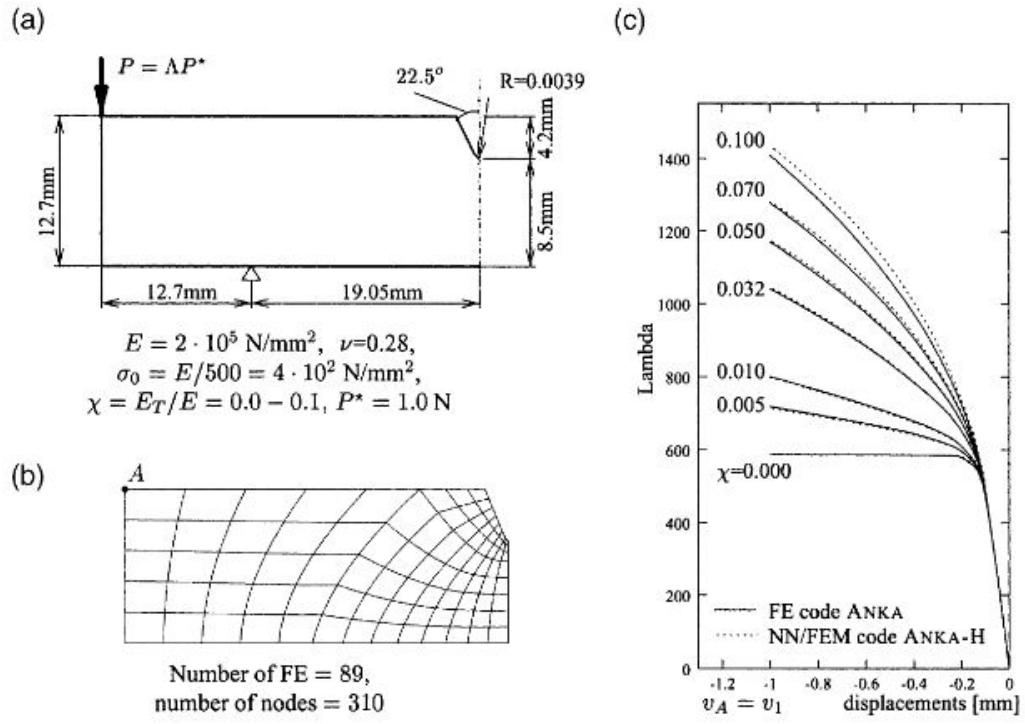


Figure 2.20: Incorporation of NN in ANKA FE code: (a) Geometry and mechanical characteristics of notched plate under load; (b) FE mesh of notched plate; (c) Output comparison between standard FE code and NN-based FE code [46]

2.4.3. Stiffened Structures

The design of stiffened structures are popular optimisation problems due to the delicate balance between the number of stiffening elements and the weight of these structures. Their solution, therefore, has been ap-

proached using neural networks, given the potential that simple structures have shown. Consider, for instance, the intermediate complexity wing shown in Figure 2.21 [44] - the unstiffened wing is show on the left, and a stiffened design on the right.

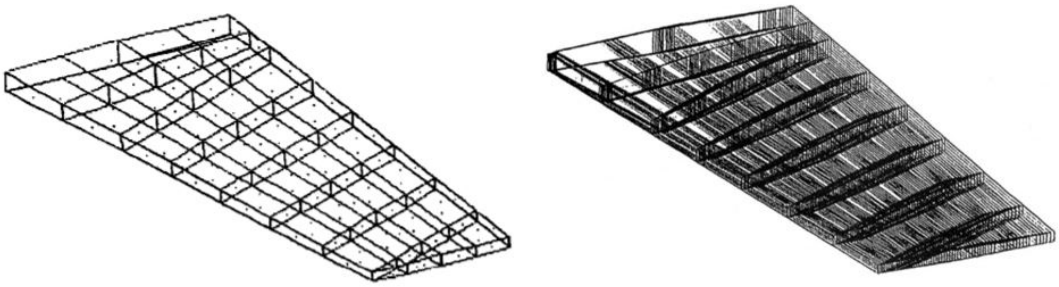


Figure 2.21: Base model of intermediate complexity wing structure (left), and stiffened model (right) [44]

The methodology adopted in training this network is similar to that described for the trussed ring, and is done using 13 datasets alongside another 2 for testing. Once again, the majority of design variables have errors in the lower range, but there are some that are as high as 90%. This may possibly be attributed to the size of the dataset. The same paper also attempted to optimise a forward-swept wing using 45 training sets and 5 test sets, and showed a similar trend in results - Figure 2.22 shows these.

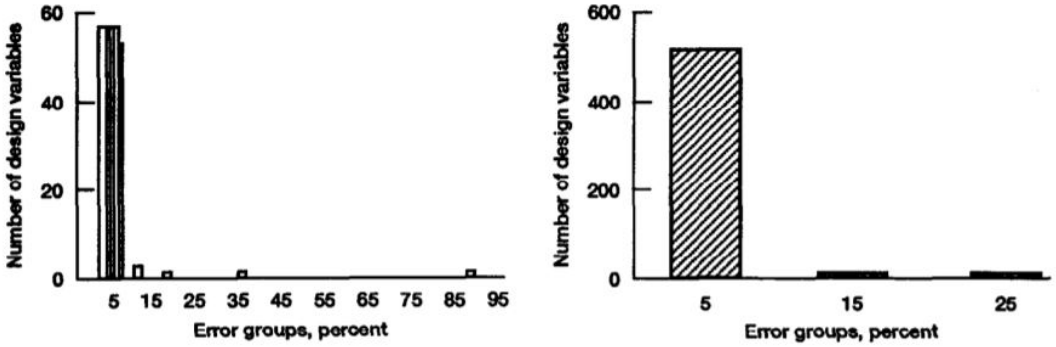


Figure 2.22: Errors between FE model and NN model for varying stiffener numbers in intermediate complexity wing design (left), and the same for forward-swept wing design (right) [44]

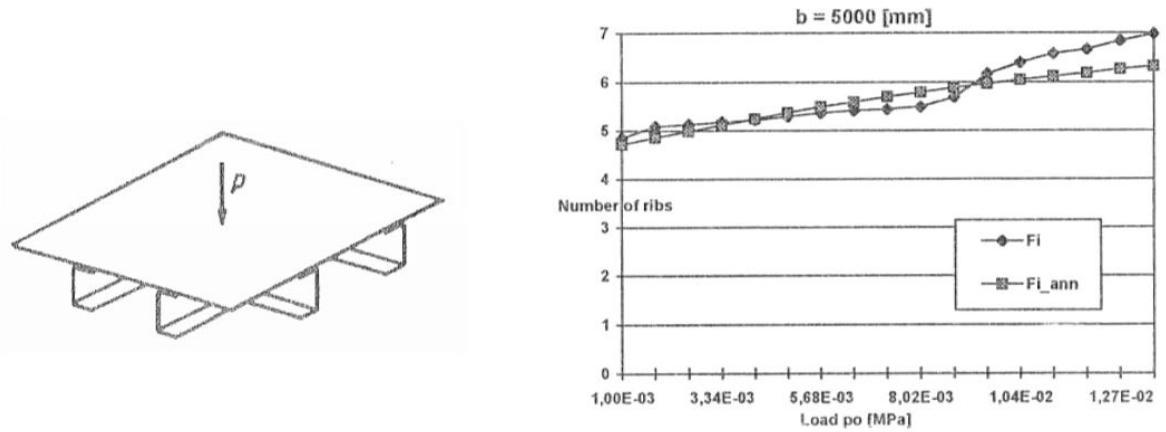


Figure 2.23: Ribbed plate subject to load (left), and the comparison between failure stresses computed using FE models and NN models (right) [47]

Likewise, stiffened plates have also been optimised using neural networks [47]. Figure 2.23 shows such a simply supported plate of width $b = 5000$ mm. The optimal number of ribs for a range of loading values is plotted to

the right, computed both numerically and by using the neural network. The author of this work does not reveal the exact details of the network or the training process, but the two curves conform to one another at lower values of loading, and show signs of diverging from each other at higher loads. In the particular range shown here, this is not a significant issue as fractional values of 'number of ribs' are always rounded up in design.

2.4.4. Dynamic Response

Amidst the exploration of static problems, Lavaei and Lohrasbi [48] investigated the dynamic response of a steel frame structure using regression approaches. The frame is disturbed using a load of 500 kg/m^2, and the displacement is recorded. Using a dataset of 30 training examples and 9 test examples, a Generalised Regression (GR) approach is first used, which is essentially a simple regression analysis adapted to suit non-linear problems. Following this, a Backward Propagation Wavenet (BPW), which is nothing but a standard neural network employing backward propagation, is trained. The results of both are presented in Figure 2.24 and compared to the exact solution. It is very evident from the two responses that the neural network model is superior in performance. In the initial phase of the BPW response, the displacements are practically identical to the exact response, while in the latter phase, the error between the two are less than 5%.

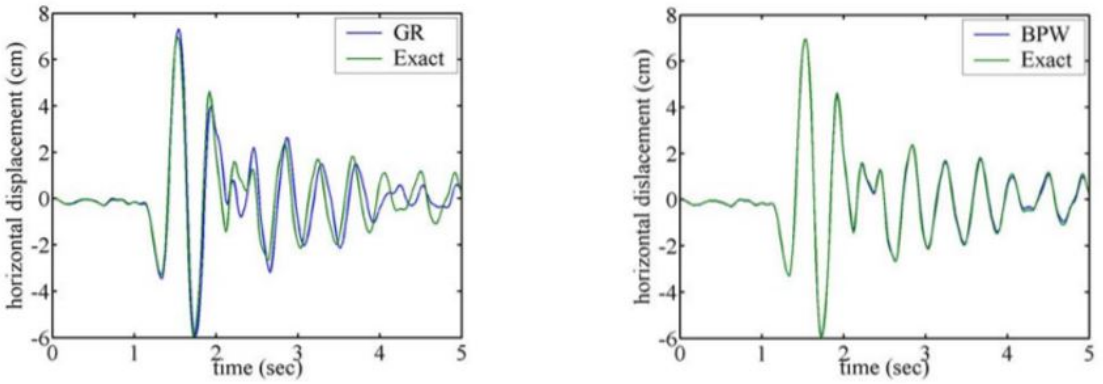


Figure 2.24: Comparison between FE-based and regression-based dynamic responses of steel frame structure using generalised regression (left), and backward propagation wavenets (right) [48]

2.4.5. Damage Analysis

The analysis of dynamic responses using neural networks has not been limited to displacement-time plots - they have also been used to detect damage. In one such research [46], data was generated through laboratory experiments where cracks in steel beams were studied using the setup and schematic shown in Figure 2.25(a) and Figure 2.25(b). The crack is introduced as a notch of width 1 mm, and is present in varying locations and depths to study their effect. The frequency response functions are shown in Figure 2.25(c). With 8 different crack locations and and 5 different crack depths, a total of 40 data points are generated for training and validating a neural network. In the network architecture, these locations and depths are output quantities to be predicted, with four excitation frequencies serving as the input. A single hidden layer is used, and the performance is evaluated for varying numbers of hidden units ranging from 2-16 - the peak performance (in terms of validation error) is obtained with 8 hidden units, as shown in Figure 2.25(d). The increase in validation error while the training error continues to decrease beyond 8 hidden units suggests that the network is overfitting. The results obtained from this optimal configuration are within an MSE of 0.002 with respect to those obtained through laboratory testing.

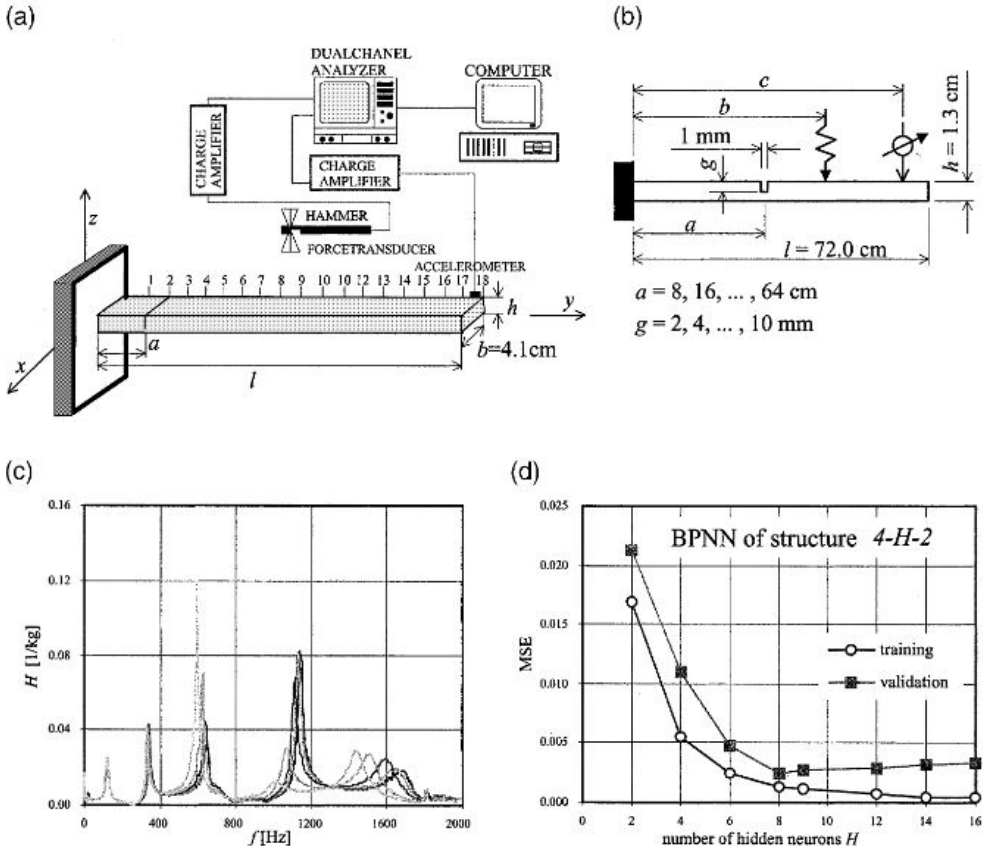


Figure 2.25: Detection of crack in steel beam through excitation: (a) Experimental setup; (b) Schematic showing beam dimensions and crack locations; (c) Frequency response functions; (d) Dependency of NN model performance on network architecture [46]

2.4.6. Remarks on Previous Research

The studies outlined in this section serve as encouragement for the use of neural networks in solving structural engineering problems, and seem to perform better than basic regression does. Models with a small number of inputs and outputs appear to perform particularly well. On the other hand, problems that are dependent on a larger number of variables and/or predict a greater number of quantities require to be trained on a large dataset for good performance. Unsurprisingly, structural behaviours that exhibit a higher degree of non-linearity are harder to model. The use of composites instead of metals [49], for instance, is one such non-linearity that has been explored.

The power of neural networks, however, is not fully extracted by developing models that are dependent on only a small number of variables, or emulate easily predictable behaviour. The prediction of more complex phenomena is a prospect worth exploring, and such networks can potentially serve as powerful surrogate models. Notably, graphical outputs from non-linear analyses are a commonly utilised source of information in design that has not received enough attention from empirical scientists. It will be curious to observe how accurately a trained network can recreate these outputs, how demanding such models will be, and what shortcomings they possess. With these objectives, the research questions are formulated in the next section.

2.5. Research Questions

This literature review explored the options of carrying out structural analysis through empirical means. Taking encouragement from previous research in this area while at the same time recognising the gaps that exist in them, the problem that was introduced in the first chapter is sought to be addressed by answer the below question.

What is the level of accuracy with which neural networks can predict the damage patterns on a composite plate?

While attempting to break down this question more specifically, the sub-questions that arise are listed below.

- What is the most suitable network architecture style to generate these patterns?
  - How do different architecture styles compare in terms of performance?
  - What is the optimal configuration of this model?
- How do the performance metrics observed reflect on the damage patterns from a visual perspective?
- What is the gain in efficiency in carrying out the computation using neural networks instead of traditional finite element analyses?

The remainder of this study aims to answer the above questions, and draw conclusions from the observations made in the process. The methodology adopted in doing so starts with the generation of training/testing data using a finite element solver such as Abaqus, and then developing a neural network model by training, tuning, and comparing different architectures. The optimal configuration is refined further to arrive at the most suitable model.

2.6. Summary

Surrogate models offer a means of carrying out approximate analyses of engineering problems with the benefit of being computationally less intensive. Empirical techniques allow for this type of modelling by using a set of recorded data points and interpolating/extrapolating using them to predict outputs for given inputs. Regression analysis, a process of determining a suitable function to relate the data points, helps formulate a mathematical expression that relates the variables of the problem together. When higher dimensional and generally more complex relationships are involved, machine learning is a powerful technique that may be adopted. Using the principles of regression, detailed models can be trained to predict outputs accurately. Of the different types of machine learning models available, neural networks are one that have evolved to be robust, efficient, and capable of handling complex datasets. They are, therefore, deemed to be a suitable choice for this study.

Neural networks have been made use of previously in structural engineering research. Studies have ranged from simple truss analysis problems to non-linear plate deflection studies. An aspect of structural design that has not yet received a significant amount of attention is the generation of graphical outputs that are so often desired while carrying out finite element analysis. This gap in the research, in combination with a desire to address the concerns introduced in the previous chapter with regard to damage modelling, has resulted in the formulation of research questions centred around the generation of damage patterns in composite materials using neural networks. The first step in modelling such a network is the gathering of training data. In the following chapter, the details of how this data is generated and organised for training purposes are elaborated.

3

Finite Element Model

Through a literature review of empirical techniques used in structural modelling, the objective of developing a neural network that can predict damage patterns on a composite plate such as the one described in Chapter 1 has been established. For training and optimising this network, a dataset of a reasonable size is necessary, and is generated out by modelling the problem on a finite element solver, and then solving the model under different conditions to obtain a set of results. The first part of this chapter focuses on the details of the FEM model. Before the results obtained can be used for training, they are to be organised in an appropriate manner and stored in a useable dataset corresponding to the inputs. The details of this process are explained in the second part of the chapter.

3.1. Model Parameters

The Finite Element (FE) model is created using Abaqus 2017, a popular platform used for such problems. This section describes the details of the model, specifically the geometrical and mechanical properties, the elements and meshing strategy, loads applied, and the setup of the numerical analysis procedure.

Geometry and Material

The geometry of the structure is a square plate with an elliptical hole at its centre, as shown in Figure 3.1. As described in Chapter 1, this is decided on the basis of being a simple geometry that exhibits a complex non-linear phenomenon, and is a modular structure in aircraft design. The edge of the plate is 100 mm in the length, and has a uniform thickness of 10 mm. The hole is 20 mm wide along the horizontal axis, and 40 mm wide along the vertical axis. It is constructed using 8 CFRP layers stacked in the sequence $[45/90/-45/0]_s$. The material properties of the CFRP lamina is given in Table 3.1.

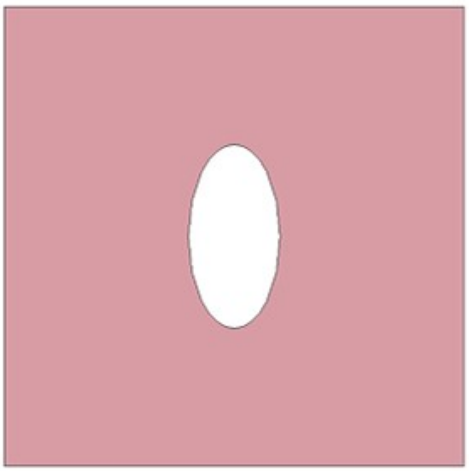


Figure 3.1: Geometry of composite plate with hole

Modulus Properties (GPa)					Strength Properties (MPa)					Fracture Energies (N/mm)			
E_1	E_2	$\nu_{12}(-)$	G_{12}	G_{13}	σ_1^T	σ_1^C	σ_2^T	σ_1^C	σ_{12}	δ_1^T	δ_1^C	δ_2^T	δ_2^C
161	11	0.32	5.17	5.17	2800	1700	60	125	90	100	100	0.22	0.72

Table 3.1: Mechanical properties of CFRP lamina used

Loading

The damage initiation patterns are sought to be generated by applying biaxial displacement loading. Buckling is a phenomenon that is not considered for this study - since the focus is to replicate these patterns using neural networks, a model that is simple enough to keep the variables low, while complex enough to analyse performance on non-trivial cases is considered. This has two outcomes - firstly, the biaxial loading being limited to tensile displacements, and secondly, an upper limit on the tensile displacements, as the Poisson effect can result in even tensile loads causing buckling. On this basis, the loading limits are identified by analysing different combinations of biaxial loads, and are finalised as 1 mm of edge displacement for each of the four edges, in the direction normal to them.

Mesh Convergence

Keeping in mind the non-linearity of the model, the part is meshed using quadratic shell elements whose mesh density increases as it approaches the stress concentration present (ie, the hole). As the emphasis in this study are damage patterns as a whole, the convergence criterion is based on the 'convergence' of the patterns themselves - the outputs from successive models are observed to evaluate whether the difference between the two are negligible or not. If they are, then the mesh is considered to be converged.

Making this decision purely subjectively is error prone and leads to inconsistency, for which reason objective metrics are considered. The most obvious metric that comes to mind is the Mean Square Error (MSE), which is computed between two images by applying it pixel-by-pixel. For the comparison of images, two other metrics - the Peak Signal-to-Noise Ratio (PSNR), and the Structural Similarity Index (SSIM) [50], are not uncommon. However, those are more useful to assess the quality of an image, and when two FE-generated images having several common image properties (such as brightness and contrast) are compared, they do not offer any more than MSE does (we will see later in Section 5.3, the situations in which they might be useful).

There are no hard-and-fast rules to determine what MSE value can be considered to be 'negligible', and there will, therefore, be a certain degree of subjectivity and judgement involved in arriving a number for it. This is done by refining the mesh until we observe plies/modes where the damage patterns do not vary appreciably, and recording the MSE values between them. For the sake of performing this process efficiently, this is done using a load value in the lower end of the range (0.25 mm in this case), which takes less time to compute. The model that gives this cut-off value of MSE, however, is not the final, converged mesh for two reasons:

1. The model is deemed to have converged only when all 16 damage patterns converge, as opposed to only the first few ply/modes that do so. The cut-off MSE value is obtained only from one pattern, and must be applied to the others before making a judgement on their convergence status. This, however, is trivial in comparison to making a subjective decision for all of them.

2. The final convergence study cannot be carried out with confidence when using a loading value at the lower end of the range - it is at the higher end that the damage patterns exhibit maximum sensitivity to the mesh.

Through this process, the cut-off value for the MSE to be considered negligible is determined to be 7×10^{-4}. This is then used as the criterion at maximum loading to check for convergence of all damage patterns. To gain an idea of the sort of visual distinction obtained by values higher and lower than this cut-off value, consider the tensile fibre damage in the first ply (ie, outer ply). Figure 3.2 shows three different stages of mesh refinement - the MSE between the first two patterns is 7.39×10^{-3}, which does not meet our convergence criterion, while that between the second and the third is 3.27×10^{-4}, which does. Since (by the established criterion) the third model does not improve the results notably, the second is considered to be the final, converged mesh.

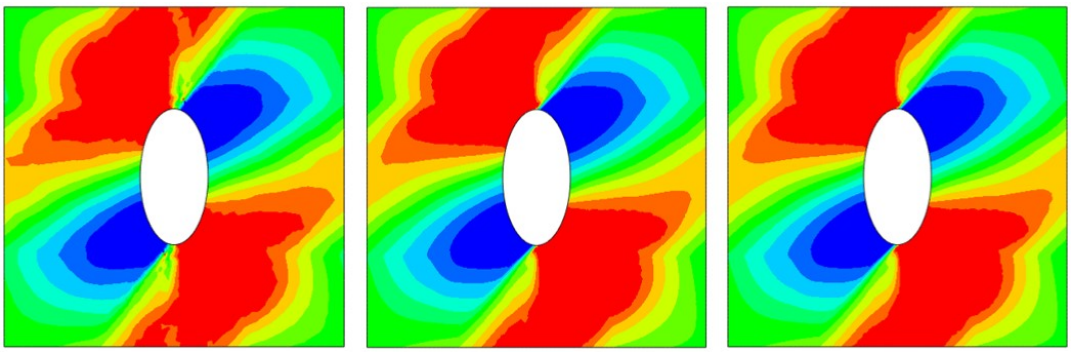


Figure 3.2: Ply 1 fibre damage in tension, computed with different levels of mesh refinement - MSE between the first two is 7.39×10^{-3}, and between the next two is 3.27×10^{-4}

For the sake of visual clarity, the mesh is hidden from the damage patterns. The mesh on its own, for reference, is shown in Figure 3.3.

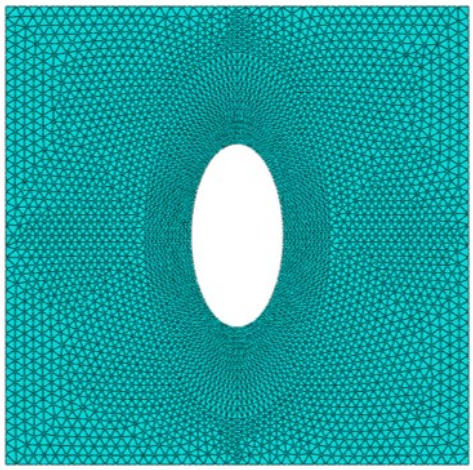


Figure 3.3: Plate after meshing

Solution Convergence

Setting up the configuration of the numerical model is important for the convergence of a non-linear problem such as this. Note that this is not done in sequence after the completion of mesh convergence, but alongside it - the solution must converge for the effect of meshing to be studied, while the mesh refinement in turn influences the configuration required for convergence. For the convergence of this model, the maximum number of increments is set to 1000, and the maximum number of iterations per step to 30.

The viscous regularisation parameter, which is desired to be as small as possible while still resulting in convergence, is set to 5×10^{-3}. To ensure that the artificial viscoelastic deformation associated with the regularisation parameter does not pollute the results, consider the ratio between the energy dissipated due to it, and the total elastic strain energy. The plot in Figure 3.4 shows this over the area of the plate - away from the hole, the viscoelastic dissipation is less than 0.7%, while closer to the hole, it is generally below 3%, and near the sharper bends, goes up to just over 4%. These values are considered to be an acceptable degree of 'noise', giving us the final configuration for obtaining convergence.

This model may now be solved at different loading values to generate results, a process that is detailed in the following section.

3.2. Data Generation

With the model parameters established, results are generated and processed to create the dataset. Using Abaqus scripting, the models are evaluated in a loop, with the load going from 0 to 1 mm in steps of 0.25 mm. This is done in all combinations of loading in the horizontal and vertical directions, resulting in 25 mod-

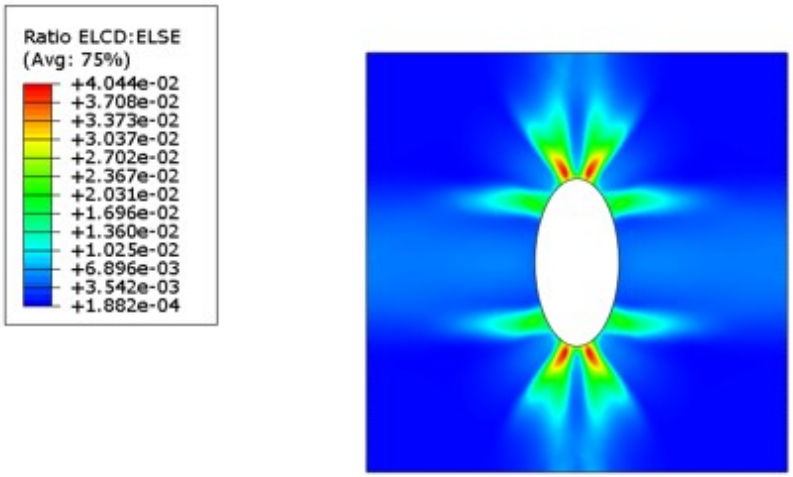


Figure 3.4: Ratio between total energy dissipated due to viscoelastic deformation (ELCD) and total elastic strain energy (ELSE), plotted over the plate

els. With 16 damage patterns in each of them, a total of 400 images are available, which is a good starting point.

The dataset is compiled by first trimming the border from the images, and then storing them in an output vector Y. The corresponding input parameters are stored in an input vector X, and have the following features:

1. Loading in X-direction (value in centimetres of displacement)

2. Loading in Y-direction (value in centimetres of displacement)

3. Ply that is of interest (represented by a number from 1-4, increasing from the outer-most ply to the central ply)

4. Failure mode that is of interest (represented by a number from 1-4, where 1 corresponds to Fibre-Compression, 2 to Fibre-Tension, 3 to Matrix-Compression, and 4 to Matrix-Tension)

We now have a dataset with inputs X and outputs Y, both with 400 entries corresponding to each other - each entry in X possesses 4 *features*, while those in Y posses $220 \times 221 \times 3$ of them. For instance, the inputs $(0.75, 1.00, 4, 1)$, $(0.25, 0.75, 1, 2)$ and $(0.50, 0.00, 1, 4)$ correspond to the outputs shown in Figure 3.5.

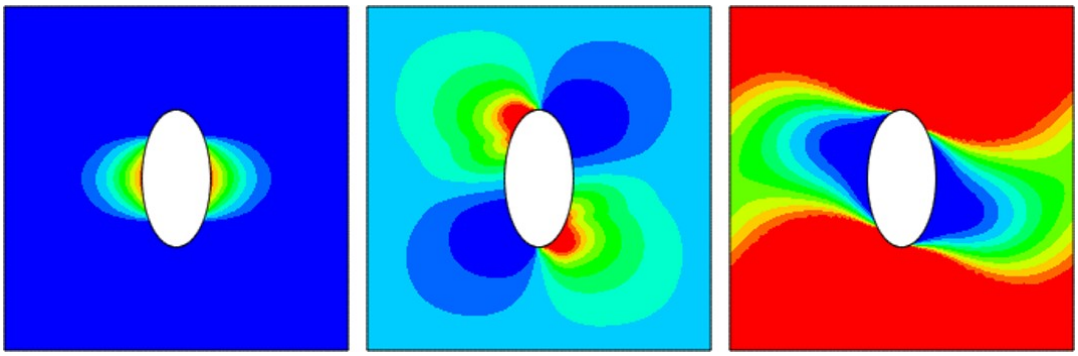


Figure 3.5: Outputs corresponding to the inputs $(0.75, 1.00, 4, 1)$, $(0.25, 0.75, 1, 2)$ and $(0.50, 0.00, 1, 4)$ respectively

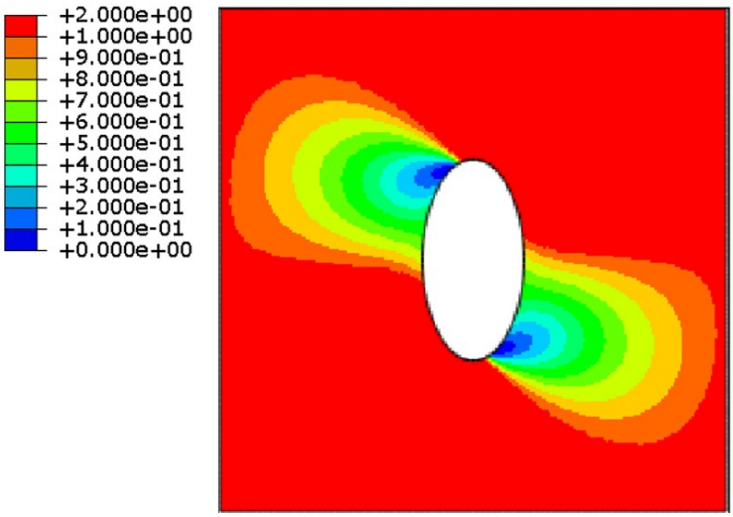


Figure 3.6: Damage pattern set aside as test image

One other damage pattern from the dataset (shown in Figure 3.6), is set aside as a test image - it is planned to be used as the reference for visualising results from the trained models. This one in particular is selected because it is deemed 'detailed' and 'complex' enough to study variations in results (as opposed to say, the first pattern in Figure 3.5, which one can expect to be easier to reproduce). Out of the other 399 images, 300 are selected randomly to form the training set, and 99 to form the validation set. The validation set is an independent dataset that is not used in the training process, and is used to evaluate the performance of the model. The training set is not used for performance evaluation so as to ensure that the models are judged on their ability to form relationships between the inputs and outputs, and not their ability to memorise pairs of inputs and outputs.

3.3. Summary

The training data necessary to train the neural network is generated using an FE-solver. By modelling the composite plate on Abaqus 2017 and carrying out convergence studies, damage patterns are generated for different biaxial loading conditions. For developing an initial network, 400 such patterns are generated, cropped, and stored in a file with each image labelled appropriately with information regarding the loading, ply, and failure mode. The dataset is then split into training, validation and test sets, and ready for neural network modelling. This is the focus of the following chapter.

4

Neural Network Training

Having established the motivation for developing a neural network model for damage analysis, the previous chapter described the process of generating data for it. In this chapter, the data is used for training - first, a standard neural is trained, with various options for improving its performance explored during the hyperparameter tuning process. Following this, the use of convolutional neural networks for reducing the number of features in the images is investigated, and the performance of the networks are compared.

4.1. Standard Neural Network

To train a standard neural network, the output images in the dataset need to be transformed as a suitable final layer of the network. Figure 4.1 shows how an image with three colour channels is vectorized into a one-dimensional array.

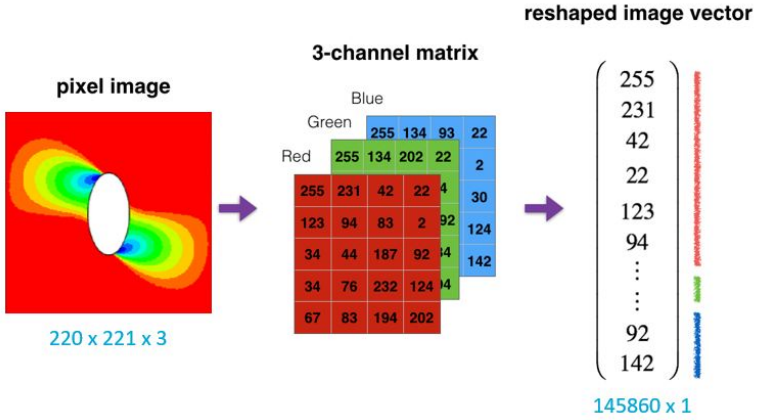


Figure 4.1: Transformation of a 3-index image into a vector

Now, the architecture comprises of an output layer with 145860 nodes, an undetermined amount of hidden nodes (which may be in one or more layers), and 4 input nodes. With this established, the training process is initiated, and the hyperparameters of the network (which include the architecture of the hidden layer(s)) are tuned for optimal performance.

4.1.1. Hyperparameter Tuning

The objective of hyperparameter tuning is to determine the optimal configuration of hyperparameters such that the network model performs at its peak. The performance here is quantified using the average Mean Square Error (MSE) (a proven *distance metric* [51] that can be used for iterative training) between the *true* images (generated using Abaqus), and the *predicted* images (generated using the neural network). At the end of the tuning process, a configuration is obtained that generates damage patterns as close to those of the FE solver as possible.

It is generally recommended to tune hyperparameters by analysing random configurations as opposed to systematically changing one at a time [52] - this is in order to avoid putting too much focus on one hyperparameter before identifying which ones the model is most sensitive to. However, for the sake of clarifying their influence on the model's behaviour to the reader, the results of tuning are presented in a systematic manner in the following subsections.

Optimisation Algorithms

Gradient descent is a basic optimisation algorithm that is robust and easy to visualise [53]. For these reasons, it is generally a common choice for first-implementation. However, as discussed in Section 2.3.3, it has been

found that modified versions of this algorithm help speed up convergence. Figure 4.2 compares them on that front, and points favourably to the use of the Adam algorithm. As outlined while splitting the generated data in Section 3.2, the loss indicated in the plot is the validation loss (note that the curve for standard gradient descent is 'hidden' behind the one for momentum as they are practically coincident).

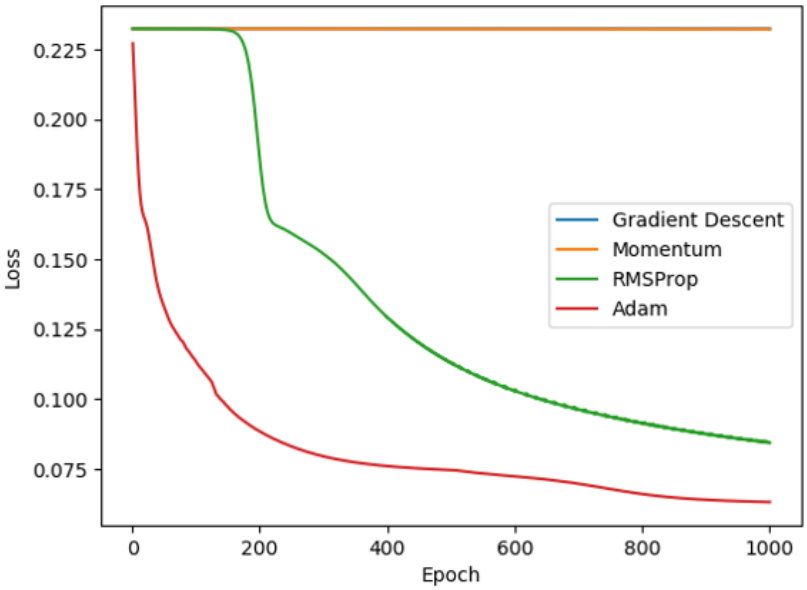


Figure 4.2: Rate of convergence of different optimisation algorithms

This comparison is made using a model configured according to Table 4.1 - the format for architecture is shown as the number of nodes in the input layer, hidden layer(s) and output layer respectively, with the row below it showing the activation functions for each of them ('n/a' refers to the fact that no activation is applied to the input). Hyperparameters that not explicitly specified take on the default values in TensorFlow 1.13 [54].

Architecture	4, 10, 145860
Activation Functions	n/a, ReLU, Sigmoid
Learning Rate	0.01
Batch Size	300
Number of Epochs	1000

Table 4.1: Model configuration for comparison of optimisation algorithms

The Adaptive gradient (Adagrad) algorithm [55], the ADADELTA algorithm [56] and the Nesterov Accelerated Gradient (NAG) algorithm [57] are some other modifications to the standard gradient descent algorithm that were applied to this network. The results obtained are practically coincident with that of momentum, and are therefore ignored from Figure 4.2 for the sake of visual clarity.

Batch Size

The previous pointed favourable towards the use of the Adam optimisation algorithm to carry out further analysis of the network. A batch size of 300 (ie, the entire training set) was used for the training of that model, which itself had an influence over the results - Figure 4.3 shows this through the training curves of models trained with different batch sizes (n in the legend refers to the batch size).

The curves indicates that while the loss in each case eventually converges to the same point, the rate at which it does so differs. The reason for this is that with larger batches, there is less learning happening within a single epoch - for instance, when the batch size is equal to the size of the training set, the parameters of the model are updated only at the end of each epoch. With smaller batches, on the other hand, parameters are updated

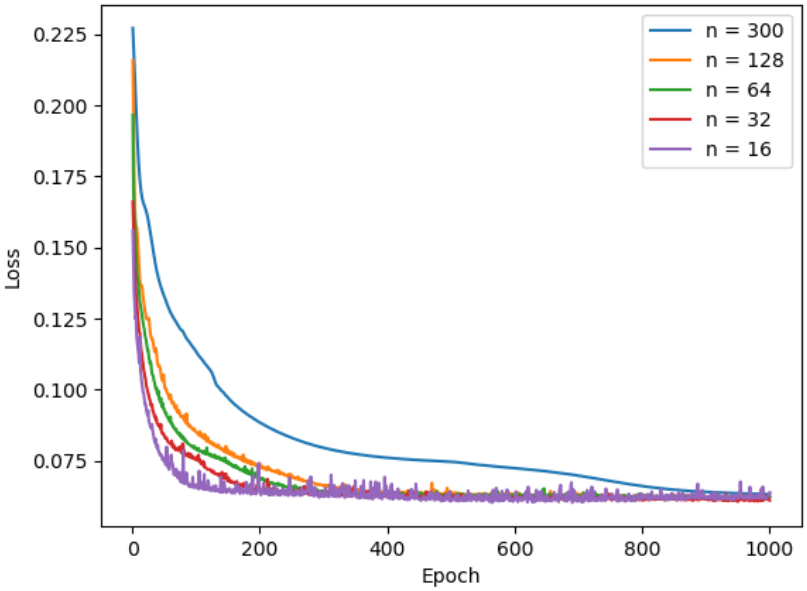


Figure 4.3: Training curves for different batch sizes

after each batch is processed, and therefore multiple times in each epoch. The multiple updates per epoch also explains why there is a degree of oscillation in the training curves of smaller batches, as opposed to the cases of larger batches where the loss curves are continuously decreasing.

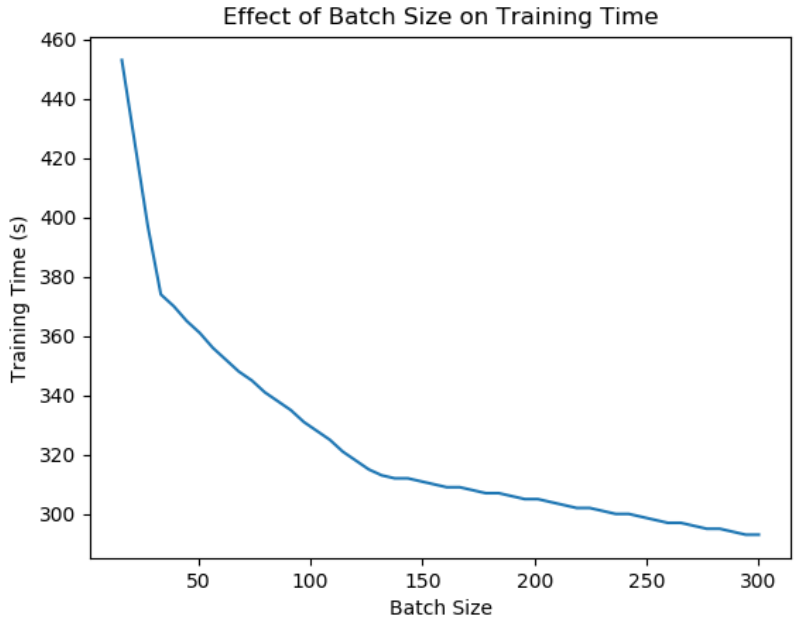


Figure 4.4: Effect of batch size on training time

It might seem logical therefore, to reduce the batch size to 1 in order to obtain the fastest possible convergence, and subsequently the fastest possible hyperparameter tuning process. However, Figure 4.4 suggests that this might not be the best idea - in addition to more frequent parameter updating, the training time for smaller batch sizes increases steeply due to under-utilisation of the power of vectorisation (to comprehend the magnitude of that, consider the difference between a vectorized operation between two arrays, and an element-by-element operation run in a loop - the latter is computationally a significantly slower process [58]).

A compromise, therefore, is to be made between the batch size and the number of epochs that are desirable for each run during the tuning process. With this consideration, a batch size of 32 is selected for further tuning.

Hidden Layer Architecture

The number of hidden layers and hidden units in the neural network affect the complexity of the model, and therefore its ability to learn complex relationships between the input and the output. In theory, the size of the hidden layers of the network is limited only by the availability of computational resources. However, it becomes clear by the end of tuning this hyperparameter that an increasingly complex model has its limitations. The model used to perform this tuning is influenced by the previous two subsections, and is configured as shown in Table 4.2 - the reason for the number of epochs remaining at 1000 despite reducing the batch size to 32 is that more complex architectures required more epochs for convergence (defined by a performance gain of less than 10% during the tuning process, and 1% for the training of the final model).

Architecture	4, <Hidden Layer(s)>, 145860
Activation Functions	n/a, <...ReLU...>, Sigmoid
Learning Rate	0.005
Optimisation Algorithm	Adam
Batch Size	32
Number of Epochs	1000

Table 4.2: Model configuration for analysing hidden layer architecture

For this model consisting of a single hidden layer, Figure 4.5 shows the effect of increasing the number of hidden units it consists of. In general, it follows the expected trend of better performance with an increased hidden layer size - the deviations observed can be explained by the small amount of randomness that is introduced during the initialisation of weights.

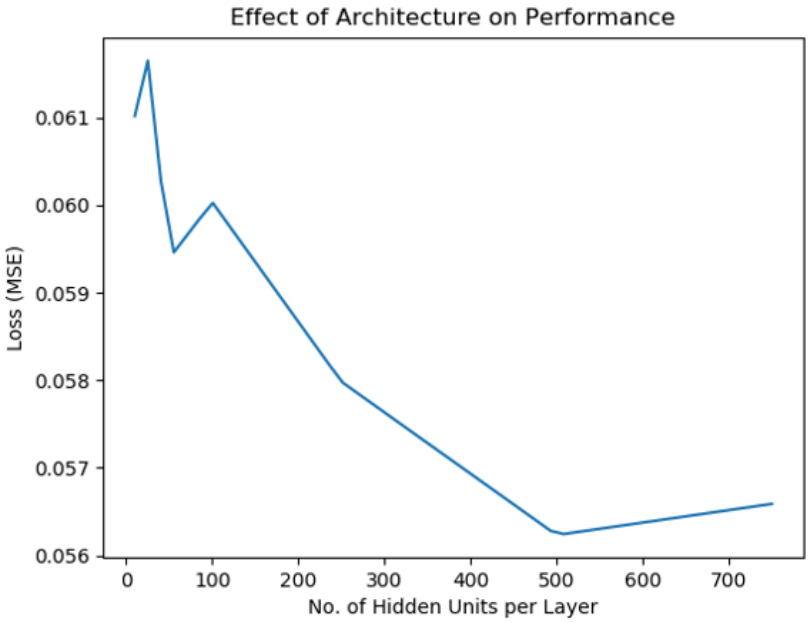


Figure 4.5: Effect of increasing size of hidden layer on performance

To understand what this improvement in performance means in terms of the output visually, consider the test example discussed in Section 3.2. This image serves as a reference to compare the output of the neural network with. The output generated by the model corresponding to the highest point of the curve in Figure 4.5, and that corresponding to lowest, are shown in Figure 4.6 and Figure 4.7 respectively. Comparing the two, it is clear that the increased complexity that comes with 500 hidden units has resulted in the development of a dominant colour closer to that of the reference, in addition to beginning to develop some of its contours. Note

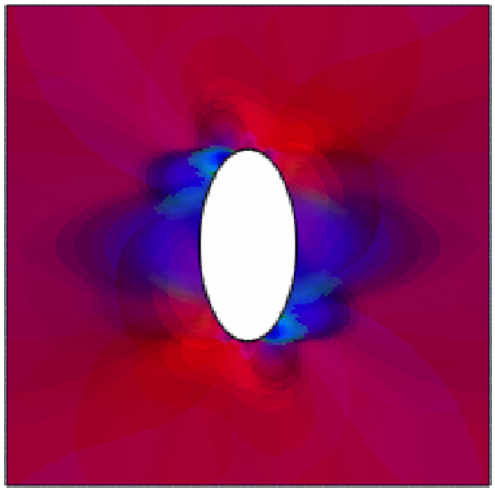


Figure 4.6: Predicted damage pattern with 1 hidden layer comprising of 20 units (MSE = 0.1294)

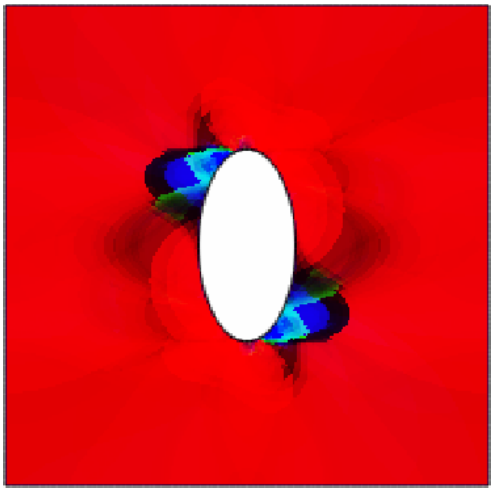


Figure 4.7: Predicted damage pattern with 1 hidden layer comprising of 500 units (MSE = 0.0642)

that the MSE values indicated in the captions of the figures are different from that shown for the corresponding models in Figure 4.5 - this is because the latter is the average error of the validation set, while the former corresponds to test example which has been deliberately chosen to be a complex pattern.

It is obvious, however, that the ability of the model to reproduce features from the reference is far from ideal. In an attempt to aid it in doing so, a second hidden layer is now added. Figure 4.8 shows a notable improvement in performance on doing so - in fact, there is a more discernable impact of adding hidden units to a layer when there are two layers. The likely reason for this is that because the second hidden layer is a function of the first hidden layer, each node of the second hidden layer is on its own a complex non-linear function that contributes to the overall complexity and therefore learning capacity of the network.

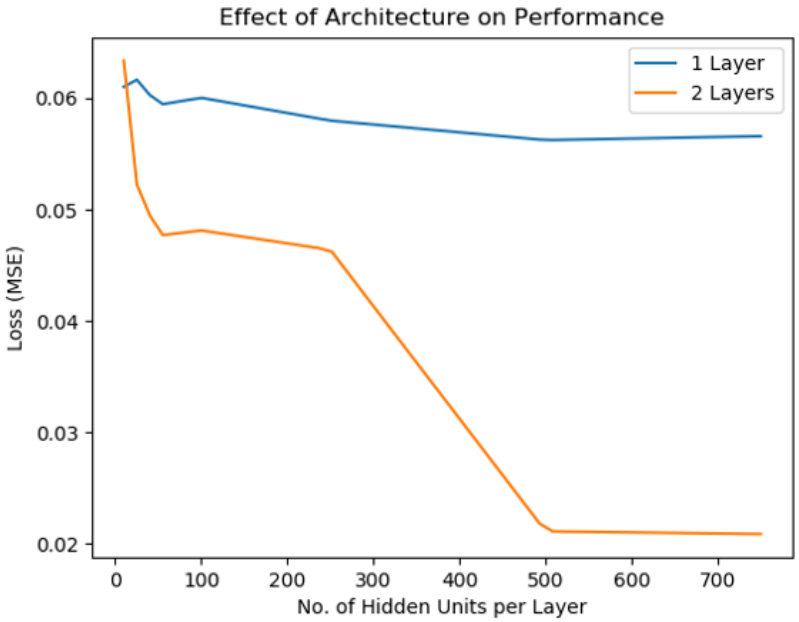


Figure 4.8: Effect of adding a second hidden layer on performance

The natural inclination, then, is to add a third hidden layer and a fourth hidden layer and expect drastic improvements in performance. However, a look at Figure 4.9 suggests that the returns are diminishing, and improvements soon cease. In fact, the curves suggest that too many hidden layers nodes may even hurt the performance of the network!

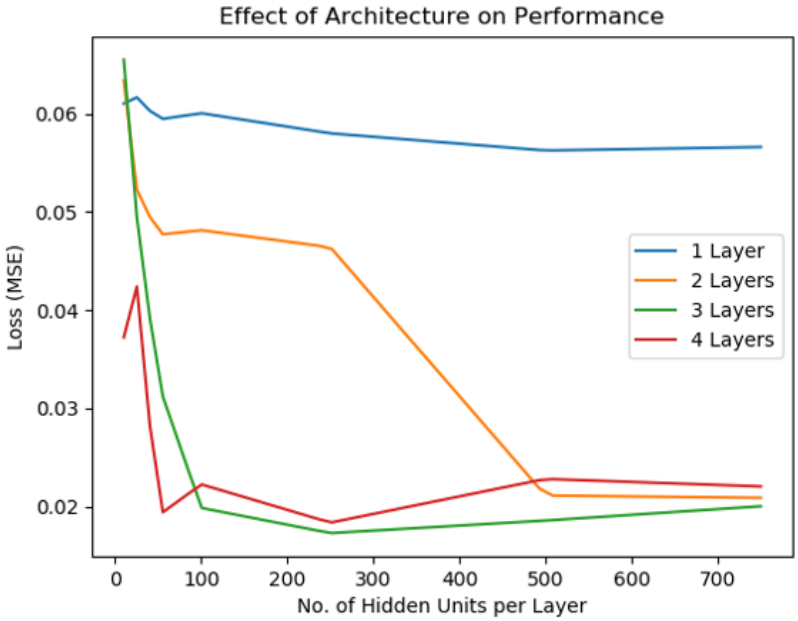


Figure 4.9: Effect of adding a third and a fourth layer on performance

This behaviour appears to be strange at first glance, but is logical when you consider that the loss being studied is the validation loss, and not the training loss - the former is computed using a data set that is not part of the training process at all. It is no surprise then, to see in Figure 4.10 that the training losses are only decreasing (even if by only an indistinguishable extent in some cases), while the validation losses increase after a point. This is due to overfitting (explained in Section 2.2.3), a situation where the model is fitting the training data so well that it is unable to generalise the results for an independent set of inputs from the same distribution.

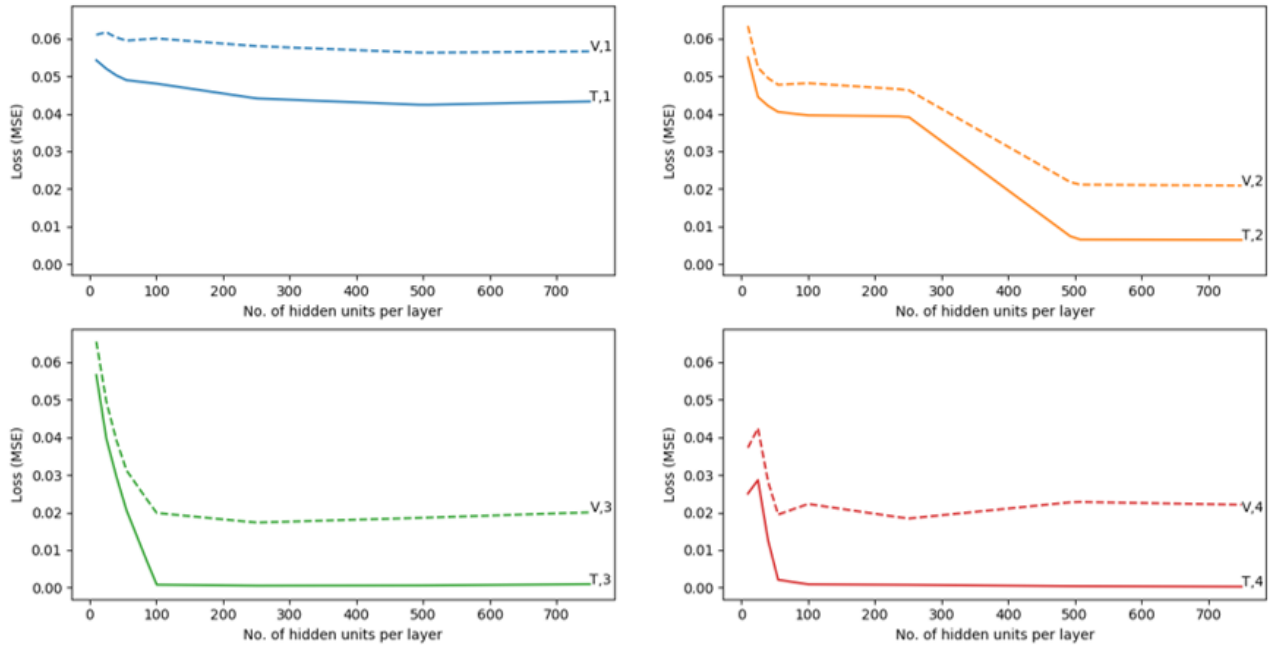


Figure 4.10: Training (T) and validation (V) errors plotted for different architectures - the numbers that are paired with the 'T' and 'V' markings indicate the number of hidden layers

There are three possible ways to deal with this situation:

1. Adopt a regularisation technique
2. Use a different kind of neural network
3. Generate a larger data set

Of these, adopting regularisation is an option that requires minimal tweaking, and is therefore tested first.

Regularisation

Dropout is a popular regularisation technique that is used to reduce overfitting [59]. This involves dropping out hidden nodes from the network, in accordance to a pre-determined fraction. A dropout fraction of 0.2, for instance, implies that one-fifth of the hidden nodes, selected randomly in each iteration, is ignored from the training process. The simplified idea is that this gets each node to actively learn a piece of information instead of being surplus to the process.

Figure 4.9 suggests that a model with 4 hidden layers comprising of 50 units each be used as a starting point for studying the effect of regularisation. In Figure 4.11, we see the effect that dropout has on the training and validation losses of this network. With more dropout, we see the gap between the training and validation errors close, which suggests less overfitting, but the value of the losses themselves increase, which is not desirable. In fact, we see in Figure 4.12 that even with a different number of hidden units (but still 4 hidden layers), dropout restricts the tendency for the validation error to rise after dropping, but the curves are entirely shifted upwards. This is not a desirable consequence, and dropout is therefore not applied to the model.

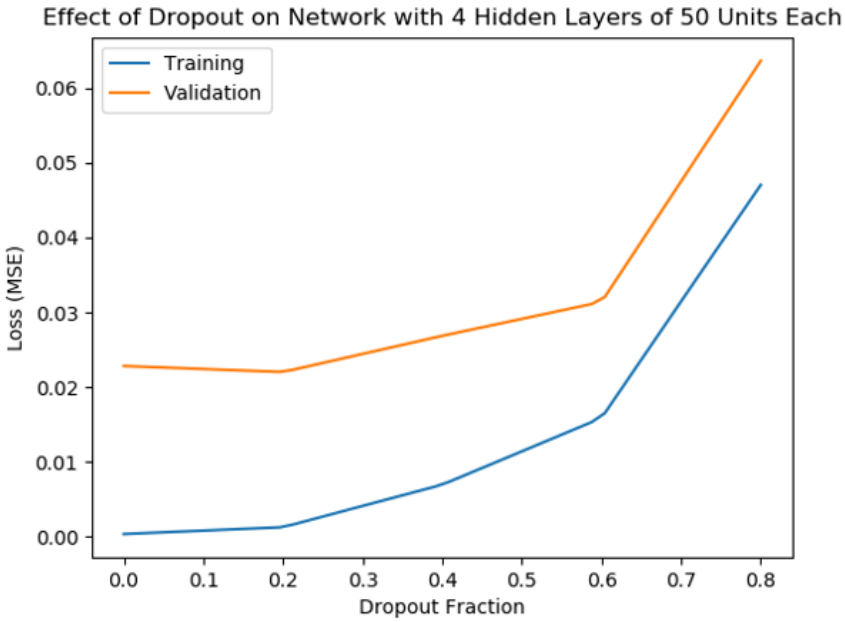


Figure 4.11: Effect of dropout on training and validation losses

Initialisers and Activations

The models trained so far have used weights that have been randomly initialised. Research has gone into alternate initialisation patterns such as picking from a normal distribution, or using a different random initialiser for each layer that is a function of its size [60, 61] These have been tried but with no perceivable variations or interesting trends, and are therefore not published here for brevity.

Likewise, different activation functions, such as those discussed in Section 2.3.3 have been compared, but with results conforming to what is already known from previous studies about the superior performance of the ReLU activation function. Note that, however, the sigmoid activation function is used for the output layer in order to restrict its values to the range 0-1.

This brings us to the end of the hyperparameter tuning process for the selected architecture style (standard neural network) - the results are presented in the following subsection.

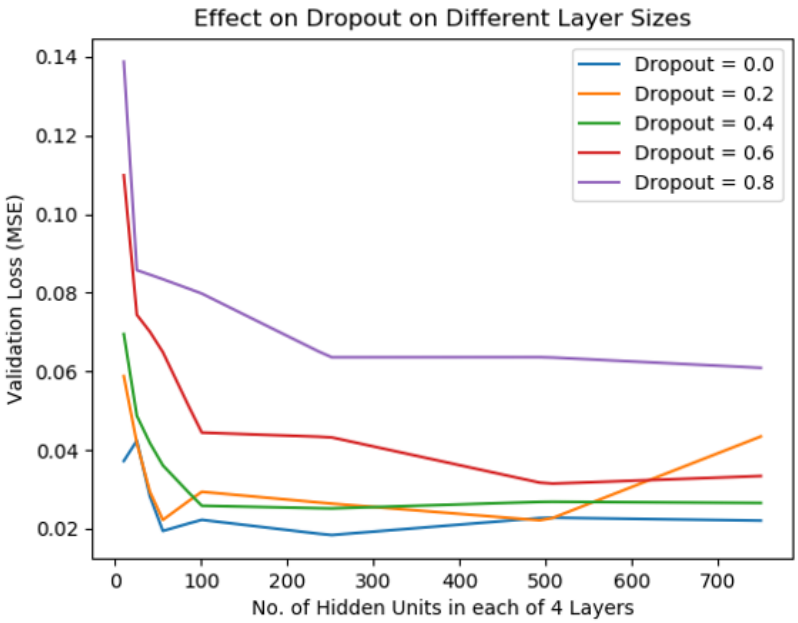


Figure 4.12: Effect of dropout on networks with 4 hidden layers losses

4.1.2. Results

The model optimised by the hyperparameter tuning process is described in Table 4.3, while the output for the test example is shown in Figure 4.13.

Architecture	4, 50, 50, 50, 50, 145860
Activation Functions	n/a, ReLU, ReLU, ReLU, ReLU, Sigmoid
Learning Rate	0.001
Optimisation Algorithm	Adam
Batch Size	32
Number of Epochs	5000
Training Error	0.0047
Validation Error	0.0213
Test Error	0.0621

Table 4.3: Configuration and performance metrics of tuned standard neural network model

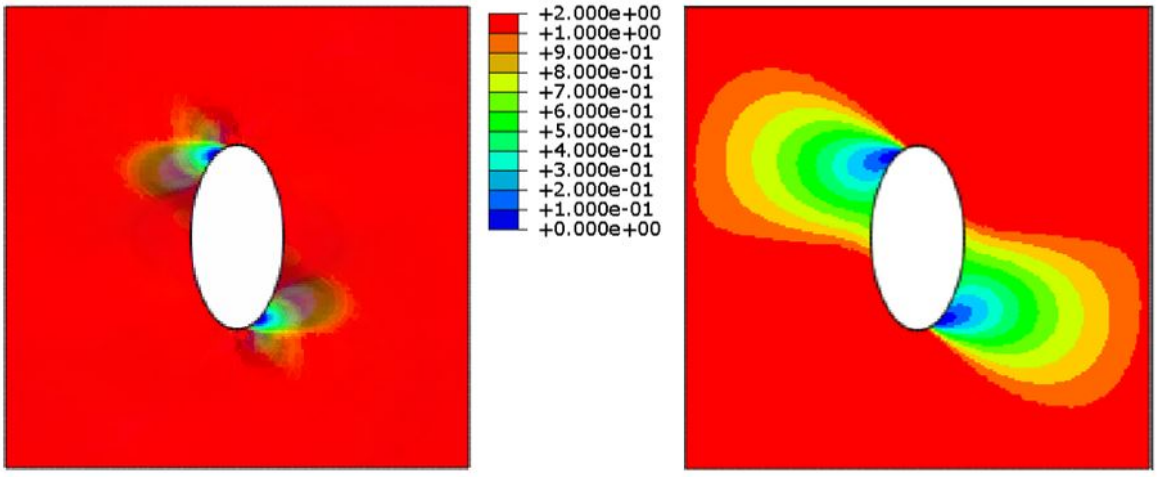


Figure 4.13: Output of tuned standard neural network model (left) compared to the reference (right)

The results of the tuned model show that it performs significantly better on the training set than it does on the validation set, or for that matter the test example. If the issue of overfitting is addressed, the performance of the model might in fact be acceptable.

Performance metrics aside, it can be said visually that the tuned model is able to reproduce the dominant colour of the pattern accurately. To a small extent, it is able create the pattern near the hole as well. However, it is clearly not anywhere close to being identical to the reference, and also exhibits some degree of noise.

As seen under the subsection 'Hidden Layer Architecture' in Section 4.1.1, regularisation does not provide any improvement in terms of the network's performance. The second possibility proposed in that subsection is an alternative worth pursuing - the use of a convolutional neural network, for instance, could potentially reduce the number of features in the image from 145860 to a significantly smaller amount, which in turn could make it easier to reproduce using the mere 4 inputs. The analysis of such a network is the focus of the next section.

4.2. Convolutional Neural Network

A Convolutional Neural Network (CNN) is a special class of neural networks that is most commonly used to extract features from images. We have seen in Section 2.3.3 how this is done through the use of filters. In addition to these filters, pooling operations are used to compress a localised set of pixels into a single one, with the aim of making the image invariant to rotations and translations, in addition to a reduction in size.

Figure 4.14 shows the sequence of convolutional filters and pooling operations used to extract features from the image. The final output layer here, as an example, has a size of $56 \times 56 \times 16$, which is then flattened into a feature index of size 50176. This is a reduction of features to about a third of its original value (note that the original 220x221x3 image is provided a thin black border to make its dimensions $224 \times 224 \times 3$, which is easier to divide several times by 2 in the network).

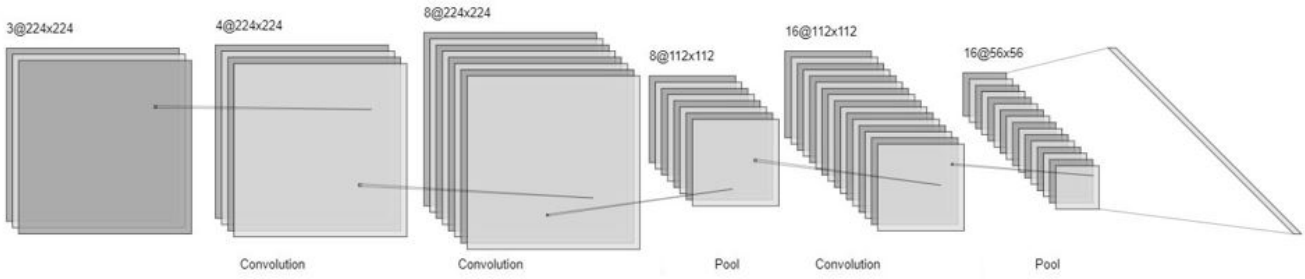


Figure 4.14: Architecture of a CNN reducing a $224 \times 224 \times 3$ image to an index of size 50176

The objectives of this process are to reduce the image to an index that is smaller, train a network to be able to generate this index given the inputs, and regenerate the image from the index. The last task can be performed using deconvolution operations that work similar to convolution operations, but for transforming smaller matrices to larger ones - this is not an exact inversion of the convolution process, and will therefore consist of its own weights that need to be learnt in the training process. Thus, to begin with, the goal will be determine the weights that reduce an image to a feature index, and those that regenerate the image from the feature index. The metric that will be used to train this network is the error between the true image fed into the convolution layers and the regenerated image outputted from the deconvolution layers, with MSE serving as the loss function (the inputs regarding loading, ply and damage mode do not play a role at this point).

4.2.1. Hyperparameter Tuning

As was the case with the standard neural network in Section 4.1.1, hyperparameter tuning is also carried out with the convolutional neural network to obtain the optimal configuration. The hyperparameters whose behaviours were identical to that of standard neural network do not throw light on new or interesting patterns and are therefore not included as part of the discussion in the following subsections.

Feature Index Size

The reduced output shown in the schematic Figure 4.14 is of size $56 \times 56 \times 16$, which has 50176 features. However, extending the sequence filtering and pooling operations can reduce this size further. The effect of reduc-

ing the image to a smaller feature index is shown in Figure 4.15.

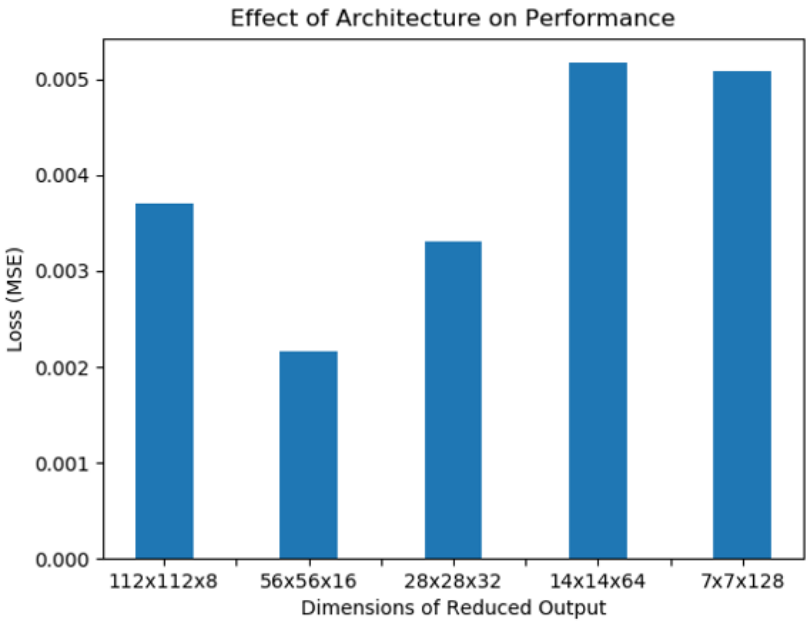


Figure 4.15: Convolution-Deconvolution performance for different feature index sizes

It can be observed that the loss drops to a minimum, before it continues to rise again. The losses first drop when convolutional layers are added, because they ensure that the network is actually learning patterns. However, continuing to add layers beyond a point increase the error between the true image and the regenerated image because of the loss of information that is occurring in the process of reducing the image to a small number of features. The optimal size is one that balances the two considerations.

Despite the optimal feature index size appearing to be 50176, it is deemed that the increase in error observed for a feature index size of $7 \times 7 \times 128 = 6272$ is small enough at this stage to be acceptable in lieu of the benefit of a notably reduced feature index - the reduction of the number of features of the image from 145860 to 6272 allows for further exploitation of the power of convolutional layers, which is of importance when the input parameters are introduced to the model. When these are combined together, the feature index size will be fine-tuned again, as seen in 5.2.3.

Kernel Size	(3, 3)
Pooling Technique	Average Pooling
Convolution/Deconvolution Activations	ReLU
Learning	0.001
Optimisation Algorithm	Adam
Batch Size	32
Number of Epochs	100

Table 4.4: Model configuration for analysing hidden layer architecture

The configuration of the model used to perform this analysis is given in Table 4.4, and the test image regenerated with this model is displayed in Figure 4.16. One observation that can be immediately made is that the image comprises of 'checkerboard artifacts' - a grid-like pattern on the image, seen more visibly at the interface between two colours. The following subsection on upsampling addresses this matter.

Deconvolution vs Upsampling

Previously, the image was regenerated by using deconvolution layers for both reducing the number of channels as well as expanding it in its first two dimensions. The latter uses a convolutional operation with a fractional

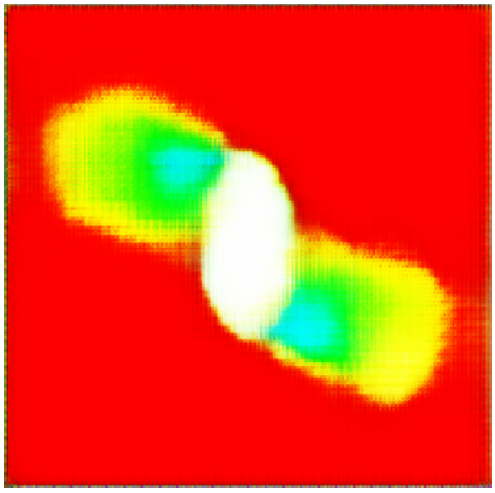


Figure 4.16: Test image regenerated after reducing to feature index of size 6272

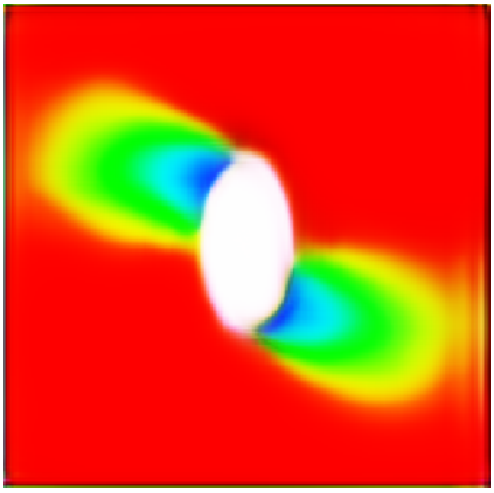


Figure 4.17: Test image regenerated using upsampling to extrapolate pixels instead of deconvolutional layers

stride, resulting in an uneven overlap between pixels. Upsampling, on the other hand, is a technique that extrapolates pixel values using neighbouring pixel values evenly. As this prevents uneven overlap, checkerboard artifacts [62] are avoided. Figure 4.17 shows this improvement in image quality, and its ability to regenerate some of the features observed in the true image. This justifies the use of upsampling, although deconvolution is still used to decrease the number of channels in the transformation - in the remainder of this work, the term 'deconvolution' is used to refer to both processes as a whole.

Pooling Technique

Max pooling and average pooling are two common techniques used to collect sets of features from a convolution output and reduce each set to a single feature either by selecting the highest value (max pooling) or the mean value (average pooling) of that set. Max pooling is often adopted while classifying photographs as it makes the results invariant to translations or rotations within the image, while average pooling is used almost exclusively to reduce size [63]. Figure 4.18 compares the performance of the two for this model, on the basis of which average pooling is selected as the choice for further tuning.

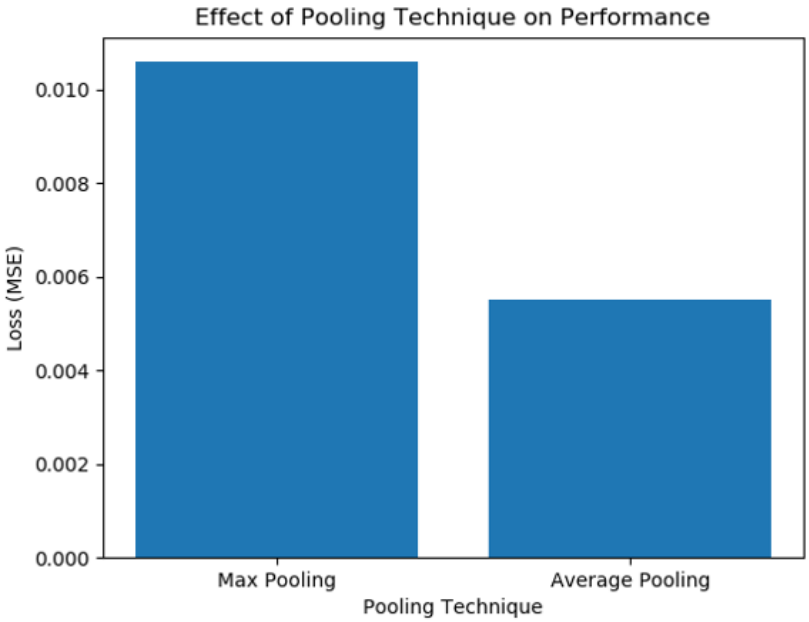


Figure 4.18: Comparison of performance between Max Pooling and Average Pooling

The hyperparameters of the CNN are now fully tuned, and are the results of the training process are given in

the following subsection.

4.2.2. Results

Table 4.5 shows the final configuration and performance metrics for reducing the image to a feature index of size 6272, whose performance was measured by its ability to regenerate the same image it was fed. The test image regenerated using this model is shown in Figure 4.19.

Kernel Size	(3, 3)
Pooling Technique	Average Pooling
Convolution/Deconvolution Activations	ReLU
Learning Rate	0.001 - 0.0001
Optimisation Algorithm	Adam
Batch Size	8
Number of Epochs	800
Training Error	0.0008
Validation Error	0.0015
Test Error	0.0027

Table 4.5: Configuration and performance metrics of tuned CNN

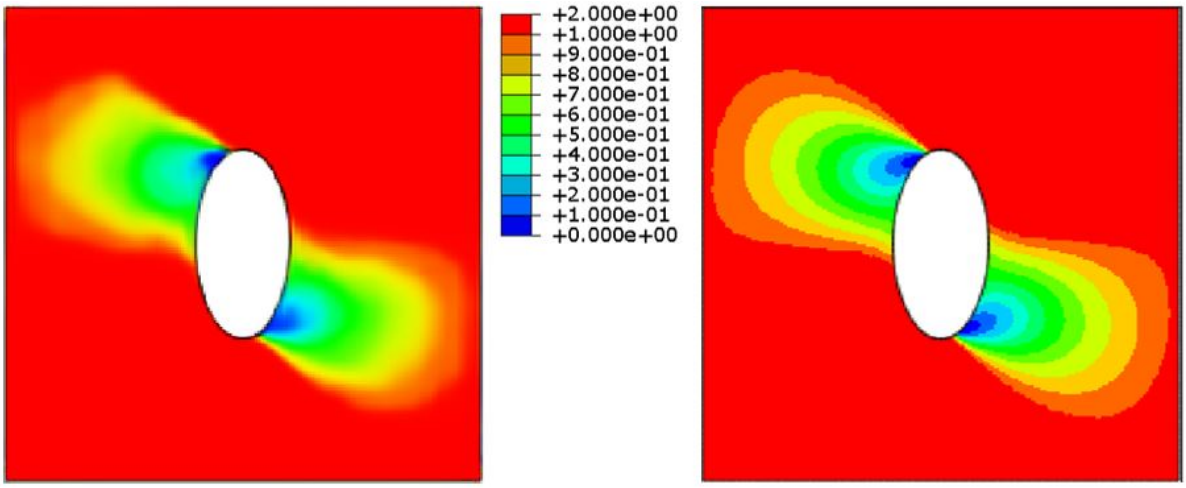


Figure 4.19: Output of tuned CNN model (left) compared to the reference (right)

Visually, it is apparent that this image is much closer to the true test image than the one generated by the standard neural network (seen in Fig.4.13). However, recall that this is merely the result of feeding in an image to the network, and attempting to obtain the same image as the output - the actual inputs of the mechanical model have not been introduced yet. This is done in the following section, where a standard neural network is used to train the inputs to the feature index.

4.3. Reduced-Image Neural Network

The previous section described how the images in the dataset of size $224 \times 224 \times 3$ were reduced to a feature index of size 6272. This section introduces the input parameters of the model to it, and follows a methodology similar to Section 4.1 to train what is termed the *Reduced-Image Neural Network (RINN)*.

The first step to doing so is creating the dataset of inputs and reduced images. To carry this out, each image in the original dataset is reduced to its feature index using the weights obtained from training the convolutional network. Subsequently, the input parameters are trained to reproduce the feature indices corresponding to the images.

The hyperparameter tuning process followed in Section 4.1.1 is followed once again - to evaluate the network, the loss function is applied between the true image, and the image generated by the predicted feature index

(the deconvolution weights obtained from the previous section are used to generate an image from a feature index). The results of this process are presented in the following subsection.

4.3.1. Results

The configuration and performance metrics of the optimised is model given in Table 4.6, with its performance on the test example shown in Figure 4.20.

Architecture	4, 2000, 2000, 2000, 6272
Activation Functions	n/a, ReLU, ReLU, ReLU, Sigmoid
Learning Rate	0.01 - 0.001
Optimisation Algorithm	Adam
Batch Size	128
Number of Epochs	20000
Training Error	0.0009
Validation Error	0.0251
Test Error	0.0716

Table 4.6: Configuration and performance metrics of tuned RINN model

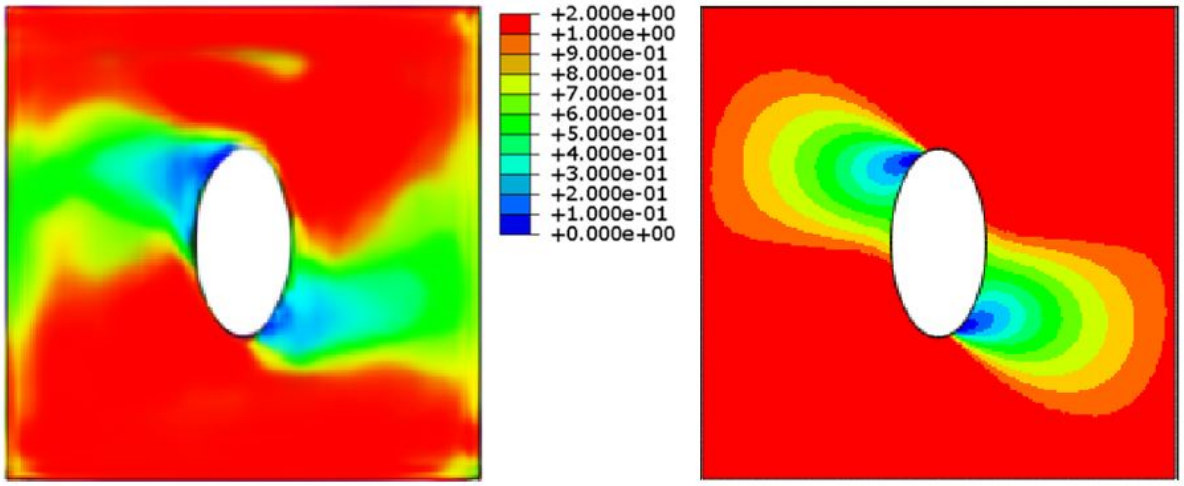


Figure 4.20: Output of tuned RINN model (left) compared to the reference (right)

Unsurprisingly, the errors associated with going from input parameters to feature index accumulate with those associated with going from feature index to image, and the performance is poorer than we observed in the CNN. On visual comparison with the test image from original standard neural network (shown in Figure 4.21 for ease of reference), we observe the patterns emerging from the hole roughly extend over the correct regions, while its colour and form are fairly inaccurate. This is likely because the reduction in the number of features enables the network to reproduce the first set of features well, but the that of the second set of features suffers due to the error in extrapolating to the complete image. Quantitatively, one observation stands out - the validation and test errors of the two networks are similar, but the training error of the RINN is superior. In fact, this network shows a notable difference between the training and validation errors, suggesting that if the third option listed towards the end of the subsection 'Hidden Layer Architecture' in Section 4.1.1 is exercised (increasing the size of the dataset), the performance of the network may be improved many-fold.

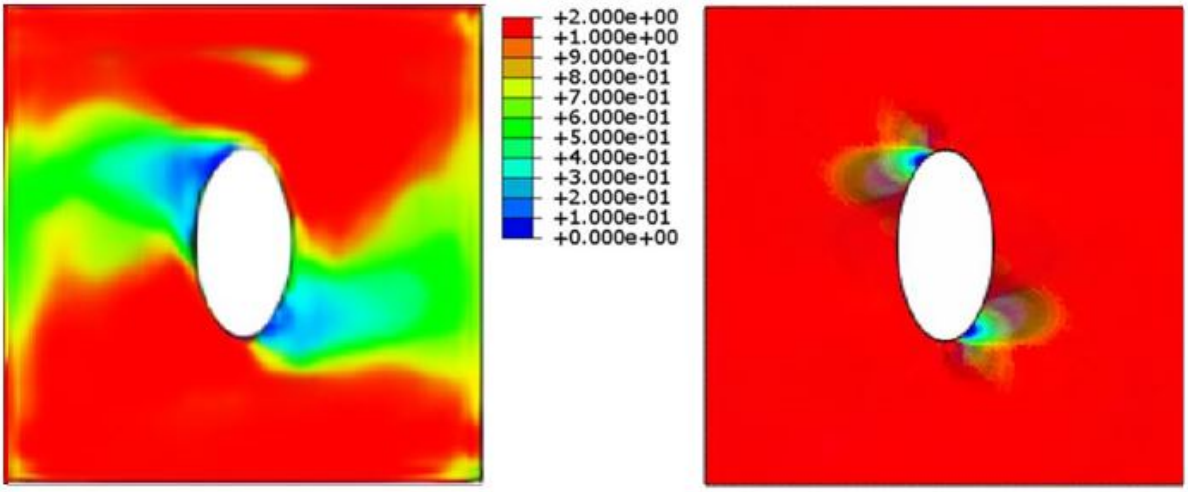


Figure 4.21: Output of tuned RINN model (left) compared to tuned standard network model (right)

4.4. Summary

Training a neural network to generate damage patterns given a set of input conditions requires extensive hyperparameter tuning to determine the optimal configuration. A standard neural network trained in this manner resulted in a model with an average MSE value of 0.0213 on the validation set. Visually, it was observed that this model was able to reproduce the dominant colour well, but not the contours of the plot. In order to attempt to improve this performance, a CNN was set up to reduce the number of features in the images from 145860 to 6272. Subsequently, an RINN model was trained to predict the feature index for a given set of inputs, to which deconvolution weights from the CNN were applied to generate the damage patterns.

On first glance, the validation set MSE value of 0.0251 suggests a slightly worse performance than the standard neural network. Visually, the RINN model seemed to do a better job of estimating the regions of contours of the plot. More than this, however, a look at the training set errors of both models gave good reason to stick with the RINN - an average MSE of 0.0009 in comparison to the standard network's 0.0047 suggests that if the overfitting is attended to, it could result in a high-performance model. This is attempted in the next chapter by generating a larger dataset for training. Further modifications are then made to the model to improve the quality of its outputs.

5

Improved Models for Damage Pattern Generation

With the previous chapter laying the groundwork for neural network modelling by identifying trends between the network configuration and performance with a small dataset, this chapter aims to improve upon the Reduced-Image Neural Network developed. The first step taken in doing so is generating a larger dataset for training and validation with the expectation of reducing overfitting. Subsequently, adjustments to the new model are carried out to improve its performance metrics further. Finally, in order to achieve greater visual quality to the outputs generated, an investigation of image quality assessment metrics is carried out for potential incorporation into the model.

5.1. Dataset Expansion and Training

The dataset used to train the neural networks discussed in the previous chapter was generated using FEM models loaded at different X- and Y-displacements, starting from 0, and going up to 1 mm displacement of a single edge. Done in steps of 0.25 mm, this resulted in 25 models, or 400 damage patterns. Now, this step size is decreased to 0.05 mm to generate a much larger volume of data - 441 models are produced in this process, or 7056 damage patterns. The expectation is that with a larger training set, the network is able to fine-tune its parameters to better fit a validation and test set.

About 80% of this dataset (5600 images) is used as the training set, just over 10% (800 images) as the validation set, and the same image as earlier as the test image. Generating all datasets took just over 100,000 seconds (close to 28 hours) of computation time (without counting the time spent on user-intervention), so in order to avoid spending time on regenerating data again (in the event that it is required), the remaining 655 images from the dataset are set aside as a reserve (note that other than the test example, all other sets are randomly selected and shuffled).

Before training the RINN model with this new dataset, a CNN is required once again to generate the feature indices, and learn the deconvolutional weights to convert them into damage patterns. The training of these two networks is carried out in follow subsections.

5.1.1. Convolutional Neural Network

For optimal performance, the CNN (and subsequently also the RINN) is trained from scratch following the same processes applied in the previous chapter, as opposed to merely using the new dataset on training the old model. This means that the hyperparameter processes are repeated in full again. Since the tuning process is well-established by now, it is not discussed further, and we skip straight to the results.

Results

The model configuration is not too different from the earlier case, and is shown in Table 5.1 along with its performance metrics. Figure 5.1 shows the regenerated test image.

Kernel Size	(3, 3)
Pooling Technique	Average Pooling
Convolution/Deconvolution Activations	ReLU
Learning Rate	0.001 - 0.0005
Optimisation Algorithm	Adam
Batch Size	8
Number of Epochs	875
Training Loss	0.0003
Validation Loss	0.0004
Test Loss	0.0012

Table 5.1: Configuration and performance metrics of CNN model tuned and trained with expanded dataset

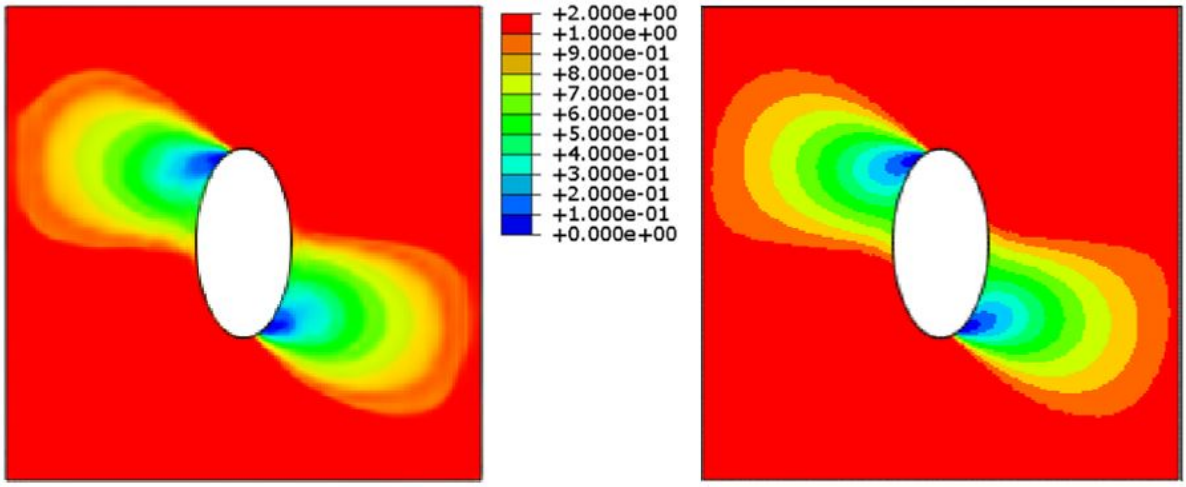


Figure 5.1: Output of CNN model tuned and trained with expanded dataset (left) compared to the reference (right)

Both qualitatively and quantitatively, a remarkable improvement in performance is observed. A low and nearly identical training and validation loss indicate that the model is guilty of neither underfitting nor overfitting. The test example shows that patterns are being replicated accurately all over the face of the plate. Again, it must be kept in mind that this is merely for the regeneration of the input image. Nonetheless, the expansion of the dataset does show promise, and feature indices are generated for training the Reduced-Image Neural Network.

5.1.2. Reduced-Image Neural Network

With the expanded dataset showing notable improvements in its ability to regenerate images, the feature indices obtained from it are trained with the input parameters. The results obtained for this model are presented below.

Results

The model configuration obtained with hyperparameter tuning, as well as the performance metrics from the optimised network are shown in Table 5.2. The output using the test example is displayed in Figure 5.2.

As expected, the losses are greater than with the CNN because generating the image is now a two-step process. However, the improvement observed by expanding the dataset is notable. Observing visually, the model is able to reproduce the colours and patterns from the reference quite accurately. There is however, some noise associated with the output, as visible from the unexpected pattern towards the bottom-left of the plate. The reason for this is that we are carrying out a fixed combination of deconvolution and upsampling operations on a feature index - therefore, any slight deviation from the 'correct' feature index results in the amplification of the error when these operations to generate the image are carried out.

Now that we have a model that predicts a reasonably good damage pattern, it is worthwhile to attempt generating outputs other than just the test image, and observing their quality. Three patterns that were first illustrated

Architecture	4, 250, 250, 250, 6272
Activation Functions	n/a, ReLU, ReLU, ReLU, Sigmoid
Learning Rate	0.001
Optimisation Algorithm	Adam
Batch Size	16
Number of Epochs	1300
Training Error	0.0023
Validation Error	0.0025
Test Error	0.0077

Table 5.2: Configuration and performance metrics of RINN model tuned and trained with expanded dataset

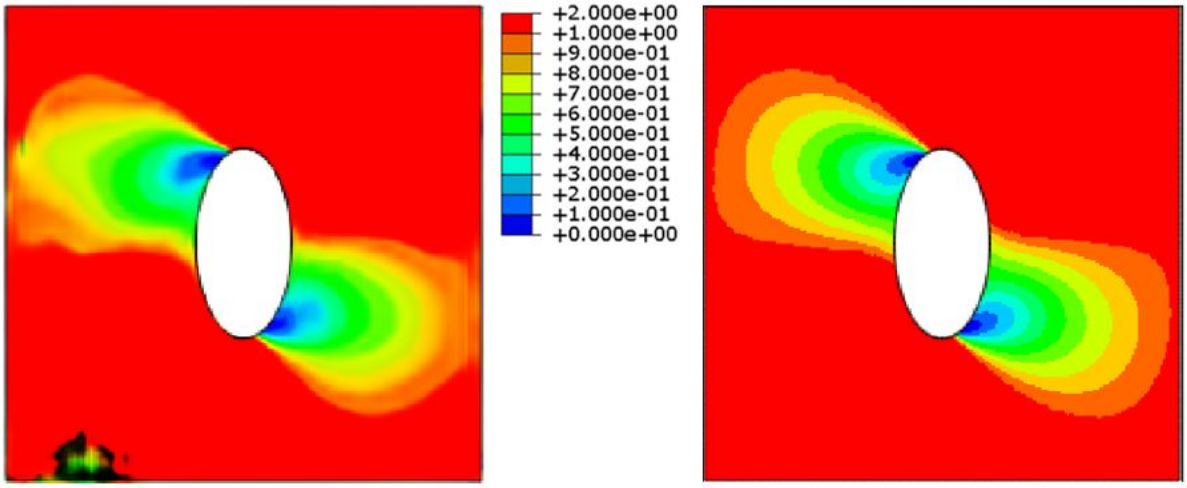


Figure 5.2: Output of RINN model tuned and trained with expanded dataset (left) compared to the reference (right)

in Figure 3.5 are regenerated using this model and shown in Appendix-B. While the first of those images is a relatively less complex pattern that is very accurately reproduced, the other two do indicate the presence of noise.

In an attempt to reduce the sensitivity of the network to deviations in the feature index, it is remodelled as a network that combines features of a standard network with a CNN. How this works is explained in the following section.

5.2. Hybrid Network

In order to desensitise the network to deviations in the predicted feature index, weights used in the deconvolution process are changed from fixed values to trainable ones. This makes the first half of the network, going from the inputs to 'feature index', a standard neural network, and the second part, going from the feature index to the image, a convolutional neural network, and is therefore termed a *hybrid network* (the phrase *feature index* is within inverted commas because the network now learns the parameters afresh, giving them values that reduce the final error without the intermediate layer necessarily being a reduced feature index).

5.2.1. Training Hybrid Network

Upon creating the architecture which combines these elements, training is carried out in a similar manner to previous networks. The architecture-related hyperparameters that are tuned in the process are the standard layers before the feature index - the deconvolution layers keep their architecture for comparison purposes. The results of training are provided below.

Results

The results of this modified architecture is provided in Table 5.3, and the output of the test example is shown in Figure 5.3.

The performance metrics suggest that using deconvolution layers that are trainable allows for a model that is more than twice as good as using fixed-parameter layers. Qualitatively, we observe that the noise signal present in the generated output of the RINN is absent, just as we had hypothesised. This also applies to the additional

Architecture	4, 200, 6272, Deconvolution
Hidden Layer Activation Functions	<...ReLU...>
Output Layer Activation Function	Sigmoid
Kernel Size	(3, 3)
Pooling Technique	Average Pooling
Learning Rate	0.005 - 0.0001
Optimisation Algorithm	Adam
Batch Size	64
Number of Epochs	1700
Training Error	0.0011
Validation Error	0.0011
Test Error	0.0026

Table 5.3: Configuration and performance metrics of hybrid network

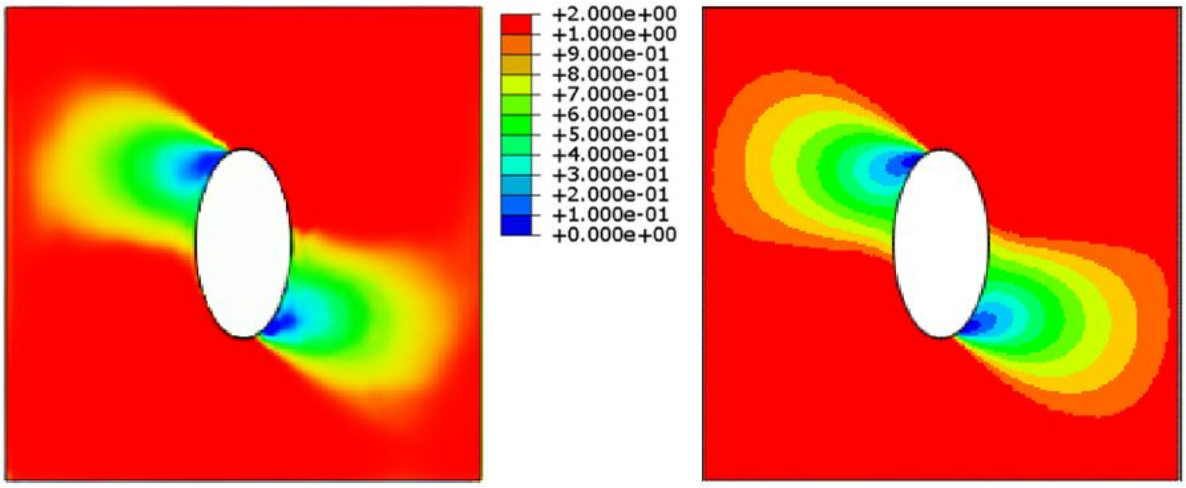


Figure 5.3: Output of hybrid network (left) compared to the reference (right)

examples shown in Appendix-B. Another qualitative observation is that the output image lacks 'sharpness' in comparison to the reference, which is an unfortunate by-product of the deconvolution process [64]. This may be observed even in the comparison in Figure 4.21, where the output from the deconvolution process shows better overall accuracy, but possesses poorer image quality with respect to the output from the standard network operation. In order to effect such a comparison again, this time with the expanded dataset, a standard neural network is trained, and is described in the next subsection.

5.2.2. Performance Comparison against Standard Neural Network

Having taken the path of incorporating deconvolution layers and developing a hybrid network, we have not taken a look back at the standard neural network again. With the expanded dataset now available, creating a standard network might be worthwhile, and comparing its performance to the hybrid network will be interesting. This is carried out, and results of training are discussed below.

Results

The hyperparameter tuning process yields the configuration shown in Table 5.4, where the performance metrics from training are also provided. The predicted test image is shown in Fig.5.4.

Quantitatively, we are most interested in the validation loss, by which metric this network does not perform as well as the hybrid network. The training loss, on the other hand, is slightly more impressive for the standard network. This difference between the training and validation losses is a reflection of the 'noisy' features observed in the output image. These features can be observed even in the additional damage patterns given in Appendix-B. It also suggests that with an even larger dataset, the standard neural network could potentially perform better than the hybrid network. Considering the sharpness of the image obtained here (as expected), this might in fact be a better option when a high volume of data is available. For the remainder of this study, however, we would like to focus on a network which is not excessively demanding in terms of data require-

Architecture	4, 250, 250, 250, 145860
Activation Functions	n/a, ReLU, ReLU, ReLU, Sigmoid
Learning Rate	0.01 - 0.0005
Optimisation Algorithm	Adam
Batch Size	32
Number of Epochs	2500
Training Loss	0.0009
Validation Loss	0.0014
Test Loss	0.0025

Table 5.4: Configuration and performance metrics of standard neural network

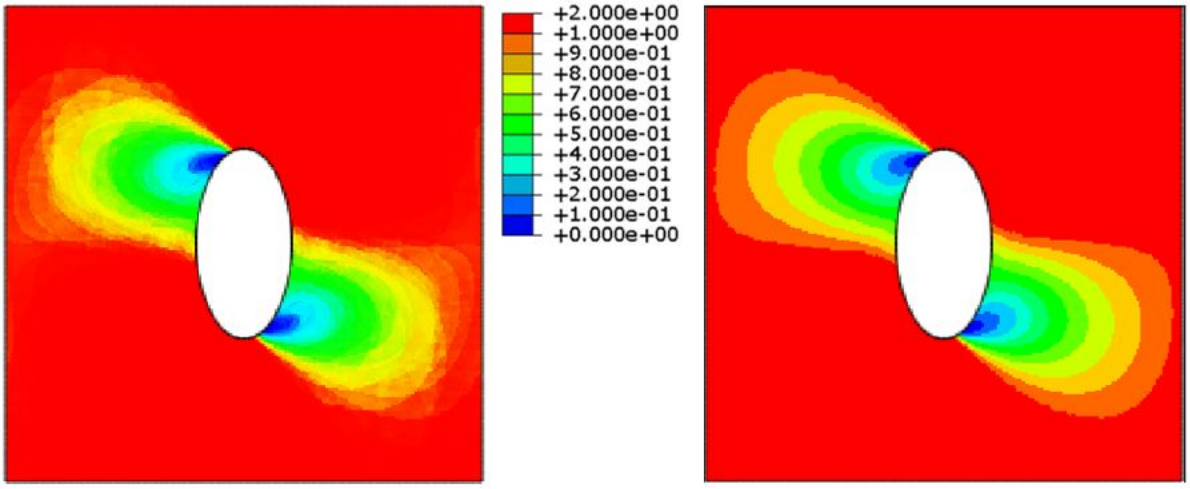


Figure 5.4: Output of standard neural network tuned and trained with expanded dataset (left) compared to the reference (right))

ment, which is more likely to be useful in practice.

While training the reduced-image network, we used a feature index size of 6272 as that was the size to which we reduced our image to using the CNN. We then carried over the same idea to the hybrid network. However, we also noted that that vector is no longer a feature index as such, but merely a layer that is similarly sized - hence, using a different size is a possibility. In an attempt to fine-tune the hybrid network, we investigate the performance of the network with different feature index sizes.

5.2.3. Fine-Tuning Hybrid Network

Up until this point, we have explored the effect of expanding the dataset on various neural network models, and constructed a hybrid network that performed better than other networks that were analysed. With this model as the basis point, we will now focus on making improvements in order to develop it further.

While tuning the hyperparameters of this network earlier in the section, our focus on the architecture was limited to up until the point of generating a feature index of size 6272. This latter value was not adjusted, much like the deconvolution architecture. Here however, these two are also tuned. The process of doing so is nothing new, and is therefore not elaborated upon. The optimised parameters obtained from the tuning process, apart from those pertaining to network architecture, are given in Table 5.5. The optimised architecture is tuned for different feature index sizes, and subsequently evaluated.

Results

The performance metrics for different feature index sizes, along with the optimised architecture in each case, is shown in Table 5.6. The outputs generated in each of the 4 cases are shown in Figure 5.5 to Figure 5.8. These sizes specifically are selected because of their relative ease in conversion to the original image through deconvolution. The deconvolution process follows the same pattern in all cases, only that the starting point and number of operations reduce as the index size increases.

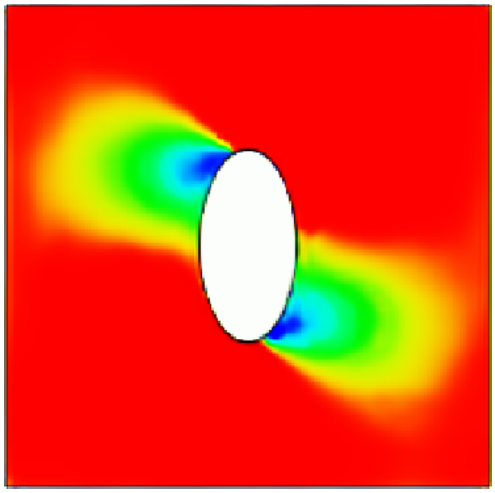


Figure 5.5: Hybrid network output for feature index size 6272

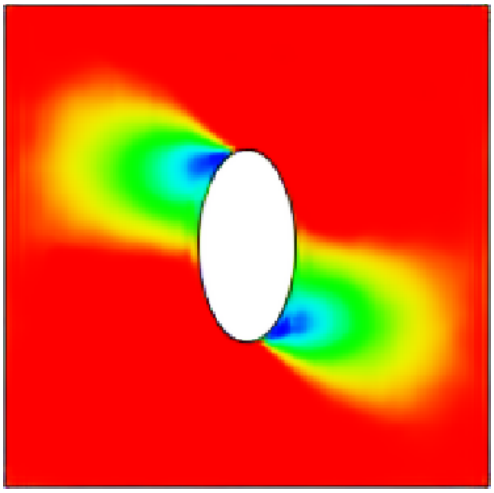


Figure 5.6: Hybrid network output for feature index size 12544

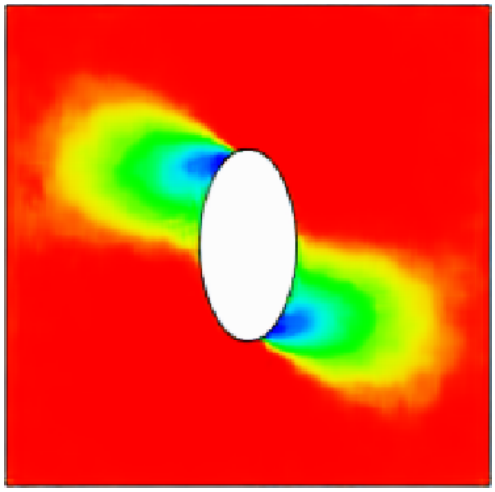


Figure 5.7: Hybrid network output for feature index size 25088

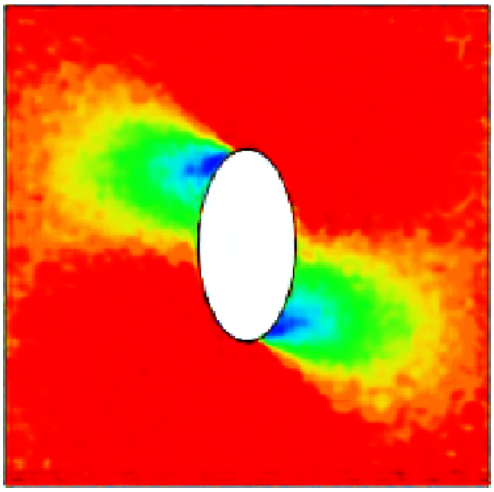


Figure 5.8: Hybrid network output for feature index size 50176

Hidden Layer Activation Functions	ReLU
Output Layer Activation Function	Sigmoid
Kernel Size	(3, 3)
Pooling Technique	Average Pooling
Learning Rate	0.005 - 0.0001
Optimisation Algorithm	Adam
Batch Size	64

Table 5.5: Model configuration (excluding architecture) for fine-tuning hybrid network

Architecture	Training Loss	Validation Loss	Test Loss
4, 200, 6272, Deconvolution	0.0011	0.0011	0.0026
4, 200, 200, 12544, Deconvolution	0.0006	0.0007	0.0018
4, 1000, 1000, 25088, Deconvolution	0.0007	0.0008	0.0024
4, 500, 50176, Deconvolution	0.0017	0.0017	0.0044

Table 5.6: Comparison of performance metrics for hybrid networks comprising of different feature index sizes

Just as we had observed while analysing the effect of network architecture on performance in Section 4.2, we see again see that the performance reaches a peak and subsequently drops again while tuning the index size. The reason for this an index that is too small does not comprise of sufficient features to extrapolate accurately from, while an index too large is lacking in deconvolution weights to learn complex relationships well enough. A qualitative inspection also reveals that an index size of 12544 yields the best results. Further damage patterns from this model are provided in Appendix-B.

The analyses carried out in this study so far have helped determined the most suitable network architecture, and fine-tune it obtain to a refined model that is able to predict damage patterns with a high degree of accuracy. The final stage of this model used a deconvolution process to generate the output image, which has been observed to be responsible for a drop in image sharpness. During this search for an optimal model, we discovered some networks which gave sharp outputs, even if they did so with lower accuracies. Such an observation kindles hope that there exists be a possibility to tweak a model to obtain sharpness even while receiving the low-noise benefits of a deconvolved output.

In the following section, some ways of assessing the image quality are studied with the intention of improving the sharpness of the results.

5.3. Enhancement of Image Quality

An optimal network configuration has been determined for the generation of damage patterns, and now the objective is to explore the possibility of visually improving the outputs even further. To upgrade the image quality, a discussion on the ways of assessing it is important, and is the topic of discussion in the next subsection. In the subsections that follow, a suitable assessment technique is applied to the model.

5.3.1. Assessment Metrics

The Mean Square Error (MSE) between two images has been used as the basis for training and evaluating models so far. This is by far the most popular loss function used for regression analysis - intuitively, these loss values give a good indication of the extent to which the values deviate from the ideal as well as being reasonably scalable while comparing models. In addition to this, the size of the gradient varies in accordance with the magnitude of the loss, thereby enabling fast learning when the loss is large, and more precise learning as the loss approaches convergence.

Despite these qualities, it is worth investigating if alternate metrics could result in the improvement of the model. Three potential options are discussed below.

Mean Absolute Error (MAE)

As the name suggests, the Mean Absolute Error (MAE) is computed by taking the mean of the absolute difference between the true output and the predicted output. This metric is useful when multiple outliers are involved - because the MSE maximises large errors and minimises fractional losses due to the squaring oper-

ation, the model is encouraged to adjust its weights in order to satisfy the outliers. The MAE, however, does not amplify or diminish any errors involved, making it suitable when outliers are a problem. With the model at hand, however, outliers are not a concern as the dataset used is generated by an FE algorithm as opposed to experimental testing. Therefore, using MAE at the cost of slower learning (due to a constant gradient) is deemed to be detrimental.

Peak Signal-to-Noise Ratio (PSNR)

When describing the quality of a set of several images, the Peak Signal-to-Noise Ratio (PSNR) is often preferred to MSE. Consider how the PSNR is calculated:

$$\text{PSNR} = -10 \log \frac{\text{MSE}}{S^2} \tag{5.1}$$

This is useful because when comparing images with different scales of pixel intensities (such as 0-1, 0-255 and 0-1023), it normalises the losses to a common scale, unlike MSE where losses vary widely with scale (S in Eq. (5.1) is the maximum pixel value of that scale). In the dataset used for this study, however, all pixel intensities lie between 0 and 1, thereby rendering the use of PSNR superfluous.

Structural Similarity Index (SSIM)

The Structural Similarity Index (SSIM) is another metric that is in many instances preferred over MSE for analysing the quality of an image. While MSE compares two images pixel-by-pixel and channel-by-channel, the human visual system tends to spot dependencies between pixels in a structured image. SSIM aims to replicate this tendency but comparing features of the image locally [50] - this is carried out by incorporating the luminance l, contrast c and structure comparison s to give the SSIM value S as:

$$S(x, y) = f(l(x, y), c(x, y), s(x, y)) \tag{5.2}$$

x and y here refer to the two images that are to be compared. The resultant is a number between 0 and 1, where a higher value indicates closer correspondence between the two images. For the four predicted damage patterns shown in Figure 5.5 to Figure 5.8, for instance, the SSIM values with respect to the reference are 0.8891, 0.9039, 0.8787 and 0.8145 respectively. These numbers suggest that ranking the outputs on either metric would result in the same order, but are clearly not proportional - it is therefore possible that training a network to maximise the SSIM value could potentially result in outputs that are considered 'high quality' by the human visual system, and therefore more sharp. This proposal is implemented in the next subsection.

5.3.2. Training Hybrid Network using SSIM

A hybrid network is trained much like they were earlier in the chapter, except that the optimisation metric being used is SSIM instead of MSE. Since a higher SSIM value is desirable, $(1-\text{SSIM})$ is used as the cost function to be minimised.

Results

The training curve and the test output are shown in Figure 5.9. Clearly, these suggest that the model is not learning anything. There is an initial drop in the first epoch due to the gradient association with random initialisation, but soon the algorithm is in a situation where with such a large structural dissimilarity to the reference, no gains are there to be made irrespective of the direction in which the gradient descends. It eventually settles at a local minima given by a uniform colour as it achieves uniform luminance and contrast throughout. This is observed even while using hyperparameters with a wide range of values. But before jumping to the conclusion to discard SSIM as a potential training metric, the possibility of making it work through a better choice of weight initialisation is discussed in the following section.

5.3.3. Refining MSE-trained Hybrid Model using SSIM

In order to give a chance for gradient descent to focus sufficiently on all three components of SSIM, a weight initialisation that allows the process to start with a relatively low loss could prove useful. In order to do this, the hybrid model is initialised with the weights obtained from training the model using MSE. In a sense then, we are attempting to refine an MSE-based pre-trained model using SSIM.

The training process comprises of two parts now - first, training until convergence using MSE, and subsequently, doing the same using SSIM. Each part may have its own optimal configuration, barring of course the architecture, which needs to remain the same for both processes (the second part cannot be initialised with the weights from the first part if the architecture is modified).

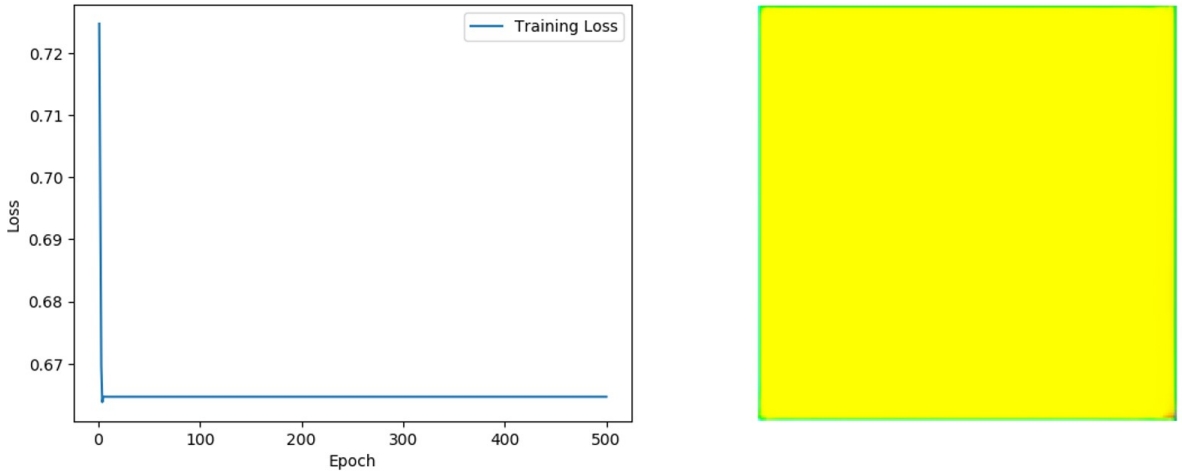


Figure 5.9: Learning curve (left), and output image (right) upon training with SSIM

Results

The model configuration used to obtain optimal results for both parts together, along with the final SSIM values are shown in Table 5.7. The output generated with this model is shown in Figure 5.10.

Architecture	4, 1000, 1000, 12544, Deconvolution
Hidden Layer Activation Functions	<...ReLU...>
Output Layer Activation Function	Sigmoid
Kernel Size	(3, 3)
Pooling Technique	Average Pooling
Learning Rate	0.0001 - 0.00005
Optimisation Algorithm	Adam
Batch Size	32
Number of Epochs with MSE	1150
Number of Epochs with SSIM	900
Training Set SSIM	0.9822
Validation Set SSIM	0.9816
Test SSIM	0.9418

Table 5.7: Configuration and performance metrics of SSIM-based tuned hybrid network

Quite evidently, initialising the SSIM-based models with pre-trained weights leads the image to converge towards a much-improved result in comparison to that obtained through random initialisation. As a function that is used to grade the quality of images, using SSIM as a training metric results in an output with improved sharpness - notice how the test image has its SSIM value improve from 0.9039 to 0.9418. Unsurprisingly, with the objective being to maximise SSIM, the focus has been taken off of MSE, leading to the increase in MSE loss values, as shown in Table 5.8. This is acceptable increase, however, given the visual quality of images obtained from the test example and additional ones shown in Appendix-B.

Training Set MSE	0.0013
Validation Set MSE	0.0014
Test Example MSE	0.0034

Table 5.8: Mean Square Errors on model trained using SSIM

5.4. Final Remarks

Chapters 4 and 5 have methodically enabled the evolution of a simple neural network to a hybrid network that is partly trained using MSE, and partly trained using SSIM. This final model is seen to perform impressively both visually, and on the validation set. However, it must be kept in mind that while the validation set is not

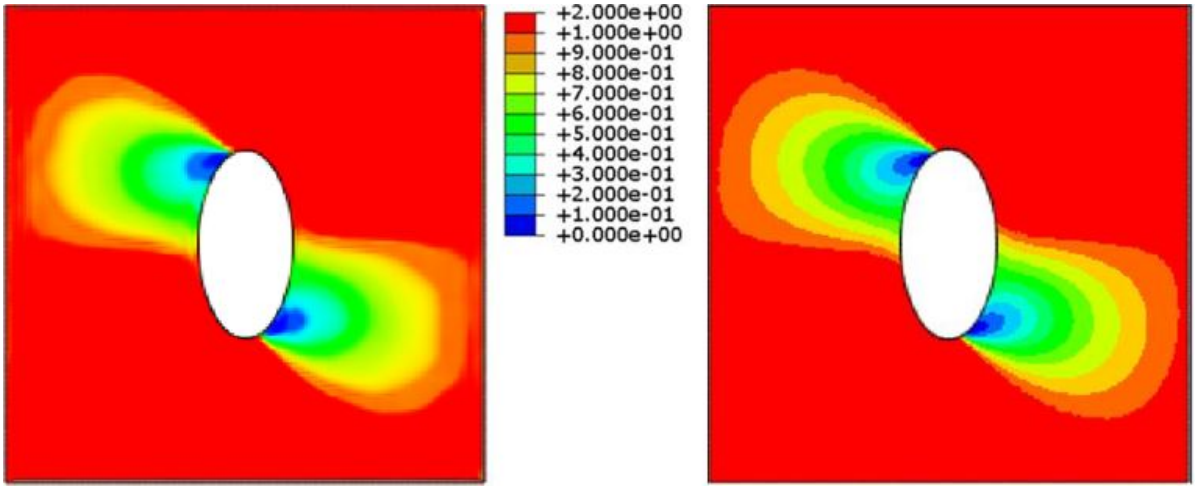


Figure 5.10: Output of SSIM-tuned tuned hybrid network (left) compared to the reference (right)

directly involved in training the network parameters, the hyperparameter tuning process is carried out with the objective of minimising the validation error. Therefore, in some respects, the network is inherently biased towards both the validation data as well, albeit to a much smaller extent than it is towards the training data. The subsection below addresses this concern, while the one that follows looks at the computation time of the network.

5.4.1. Reliability of Trained Model

A true measure of the performance of the model is obtained using a completely independent dataset that played no role in the training process, which discounts even the validation set. The test image used throughout the study is one such 'set' that fits this criterion. To evaluate the model on a larger independent dataset, the 655 reserve examples mentioned at the start of this chapter are used. The MSE and SSIM values on this set are measured, and given in Table 5.9 below.

Mean Square Error (MSE)	Structural Similarity Index (SSIM)
0.0014	0.9804

Table 5.9: MSE and SSIM values of final model evaluated on an independent dataset

These results confirm the performance in terms of the mean error on a set of unseen examples, which adds to its reliability. To analyse its degree of reliability even further, however, the distribution of errors are as important as the mean. Consider the histogram of MSE values shown in the left of Figure 5.11 - nearly 300 of the 655 examples yield an error of less than 0.0002, while the maximum error is 0.0112. This is also seen in cumulative density plot in the right of the figure - 100% of the values lie below 0.0112, while approximately 95% of the errors are within the 0.0040 mark.

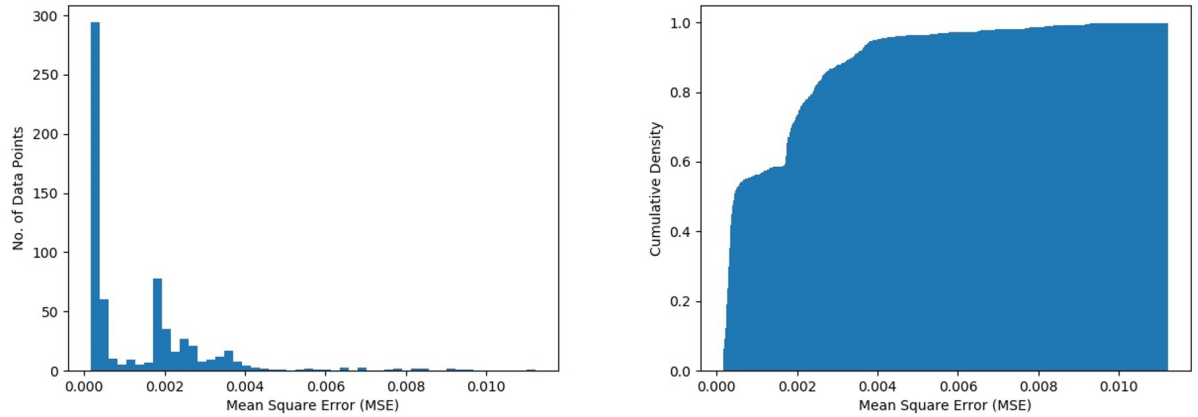


Figure 5.11: Histogram of MSE values (left), and their cumulative densities (right)

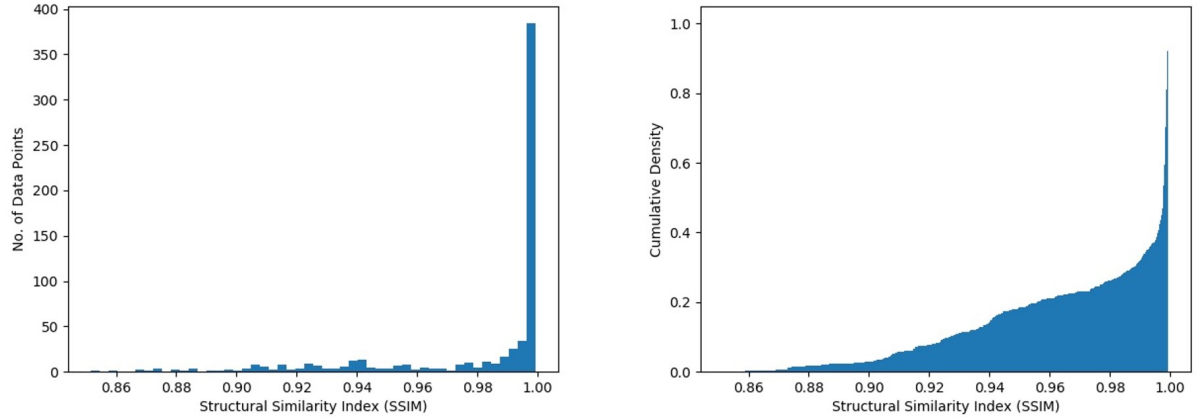


Figure 5.12: Histogram of SSIM values (left), and their cumulative densities (right)

The plots are made for SSIM as well, and are shown in Figure 5.12. Over 375 data points in the reserve set have an SSIM value of greater than 0.9950. All images have an SSIM value greater than 0.8500, and approximately 95% of them are over 0.9100.

5.4.2. Computation Costs

The neural network models developed in this work were optimised purely on the basis of performance metric values. Prediction times for an architecture style were found to be practically identical irrespective of hyper-parameter values, and were therefore not a serious consideration. However, bearing in mind that one of the major motivating factors for undertaking this study was to explore the gains obtained through surrogate modelling, an examination of computation times was carried out.

The prediction time on a complete analysis using the hybrid network, ie, for the generation of 16 damage patterns, was found to 6.6 seconds on average, with negligible spread. This last detail suggests that the complexity of the physical model is not a factor that influences the computation, which is unsurprising considering that every input is processed in exactly the same way, unlike finite element models which require more iterations to converge on more complex problems. The finite element models solved in Chapters 3 and 5 to generate data, meanwhile, took an average of 227 seconds to complete, making it 34.4 times more expensive than the neural network model. In fact, more complex cases took even longer, such as the problem analysed in Chapter 1, where the computation took 2927 seconds, or 443.5 times longer than the hybrid network.

With all the gains observed in terms of prediction times, however, it must be recognised that the network required to be trained for 76225 seconds, or just over 21 hours. Not to mention the generation of data, which required another 28 hours. This is a worthwhile investment only when the model is called upon a substantial

number of times, or represents a modular structure that is used extensively in high-level design. In any case, the time spent on generating data and training a network to a desired level of accuracy is something that must be weighed against the extent of usage in order to ascertain the feasibility of a surrogate model.

5.5. Summary

The chapter began with the objective to improve upon the RINN model developed in the previous chapter. An expanded dataset was generated for this purpose, which significantly improved its performance - the MSE on the validation set went from 0.0251 to 0.0025. The outputs, however, were not free of noise, which were found to be a result of keeping the deconvolution weights fixed. A hybrid network, ie, one that comprised of both standard and deconvolutional trainable layers was developed, helping remove the noise from the RINN, and improving the validation set error further to 0.0011.

This network was also found to perform better than the standard neural network, on the basis of which the decision to fine-tune the hybrid network was made, resulting in another rise in performance, taking the validation error to 0.0007. However, the standard neural network appeared to output images of higher visual quality, which served as an inspiration to explore achieving the same with the hybrid network.

The Structural Similarity Index (SSIM) was identified as a potential training metric to improve image quality, but failed while trying to train the network from scratch. However, when an MSE-based pre-trained network was trained, the model performed impressively. The validation set was found to have an average SSIM value of 0.9816, which suggests a high similarity with its corresponding reference patterns.

This model was accepted as the final, tuned model, and was assessed on a new, independent dataset, on which the average MSE and SSIM values were 0.0014 and 0.9804 respectively. An assessment of the reliability of the model was also carried out, and it was found that 95% of the MSE values in the new dataset were below 0.0040, while the same fraction had an SSIM value greater than 0.9850.

The study concluded with a few remarks on computational efficiency. The hybrid network was found to be around 34 times more efficient that the average finite element model, and up to 443 times so in certain cases. The investment required to achieve this, however, was approximately 21 hours of training, which itself was carried out on data generated over a period 28 hours. Therefore, the feasibility of training such a network is dependent largely on the extent to which the developed model is made use of.

6
Conclusions

From the outset of this research, the objective was to develop a surrogate model that could predict damage patterns on a composite plate. This was achieved in the form of a hybrid neural network model trained using both the Mean Square Error (MSE) and the Structural Similarity Index (SSIM). Several results were observed through the course of this work, the main ones of which are summarised in the first section. An attempt is then made to formally answer the research questions laid down in Section 2.5, during the process of which additional remarks are made. Finally, suggestions for future work in this domain are discussed.

6.1. Summary of Results

The development of neural network models in this study began with the generation of 20 finite element based damage models whose results were used to train both a standard and a reduced-image neural network. The results of these were less than satisfactory, and it was found that generating another 421 such damage models for training resulted in a striking improvement in their performances. A comparison between the validation losses in the two cases is provided in Table 6.1.

Architecture Style	20 models	441 models
Standard Neural Network	0.0213	0.0014
Reduced-Image Neural Network	0.0251	0.0025

Table 6.1: Validation loss reduction with dataset expansion for two architecture styles

The network modelling process eventually led to the evolution of the hybrid network, a network that combined facets of both the standard and reduced-image network, and displayed performance superior to each of them. When optimised for minimal MSE values, the results shown in Table 6.2 are obtained.

Architecture Style	Training Loss	Validation Loss	Test Loss
Standard Neural Network	0.0009	0.0014	0.0025
Reduced-Image Neural Network	0.0023	0.0025	0.0027
Hybrid Network	0.0006	0.0007	0.0018

Table 6.2: Performance comparison between network architecture styles

Subsequently, it was identified that optimising the hybrid network for maximum SSIM value after training with MSE as its performance metric produced visually higher quality patterns. To a certain degree, this tended to increase the MSE values again, but the overall performance of the network was deemed to be an improvement. As a final measure of its capability, the network was tested on an independent dataset, the results of which are provided in Table 6.3.

Mean Square Error (MSE)	Structural Similarity Index (SSIM)
0.0014	0.9804

Table 6.3: Performance of hybrid network trained with MSE and SSIM metrics, and evaluated on an independent dataset

The spread of loss values for this refined model showed that roughly 95% of the MSE values were under 0.0040, and the same percentage of SSIM values were over 0.9100, indicating the reliability of the hybrid network as a surrogate model. These levels of accuracy were obtained by subjecting the network to approximately 21 hours of training on a dataset generated over 28 hours, and resulted in a surrogate model that had on average 34.4

times the computationally efficiency as a finite element model, a number which went up to 443.5 in certain cases.

6.2. Addressing of Research Questions

In addressing the research questions and drawing conclusions, let us begin with the sub-questions first, as they are more specific and therefore easier to answer.

- **What is the most suitable network architecture style to generate these patterns?**

  A hybrid neural network, where the inputs are first processed to a larger standard hidden layer, and then deconvolved to produce a full-sized image, were found to be leading in performance. However, the damage patterns from standard neural networks were visually most impressive, and with a much larger dataset, they could potentially be a more suitable option.

  - **How do different architecture styles compare in terms of performance?**

    Hybrid networks performed better than standard neural networks, both of which performed appreciably better than reduced-image networks. For the specific optimised models trained in this work, their MSE measures are 0.0011, 0.0014 and 0.0025 respectively.
  - **What is the optimal configuration of this model?**

    The model is found to generate outputs that boast a good balance between visual quality and objective accuracy when trained with the SSIM metric. However, to avoid getting stuck in a local minima during this process, the model weights need to be initiated closer to the global minima, which is taken of by training first with MSE as its performance metric. The optimal configuration and SSIM performance indicators of such a model are given below.

Architecture	4, 1000, 1000, 12544, Deconvolution
Hidden Layer Activation Functions	<...ReLU...>
Output Layer Activation Function	Sigmoid
Kernel Size	(3, 3)
Pooling Technique	Average Pooling
Learning Rate	0.0001 - 0.00005
Optimisation Algorithm	Adam
Batch Size	32
Number of Epochs with MSE	1150
Number of Epochs with SSIM	900
Training Set SSIM	0.9822
Validation Set SSIM	0.9816
Test SSIM	0.9418

Table 6.4: Optimal configuration of hybrid network

- **How do the performance metrics observed reflect on the damage patterns from a visual perspective?**

  When outputs resulting from a single model are compared, the qualitative comparison using objective metrics is in tune with those made visually. However, when different networks architecture styles are compared, there is usually a tendency to favour the visual quality of standard neural networks over the others. This is attributed to the nature of standard and deconvolution outputs.

- **What is the gain in efficiency in carrying out the computation using neural networks instead of traditional finite element analyses?**

  The hybrid model has very little variance in its output computation times, as opposed to a finite element model for which the range is wide. This is along expected lines as neural network models carry out exactly the same set of operations on all inputs fed to it, while the convergence of finite element computations depends on the complexity of the problem. Taking the average duration of the latter, it has been found that the hybrid network is 34.4 times more efficient. Alternatively, with a more complex problem, the hybrid network becomes 443.5 times more efficient. These gains, however, come with a caveat - a data generation process of 28 hours, along with a network training time of 21 hours were instrumental in achieving these figures. In view of this, it is recommended that a cost-benefit analysis be carried out to determine whether a similar investment of resources is beneficial or not. It is likely to be so when

the trained model is called into use with high frequency, or when they represent modular units that are repeated extensively in a larger system. In many cases, it is also preferred from an ease-of-use point of view to carry out data generation and network training in one or two sittings with minimal human intervention, so that the human user can later on analyse models quickly - this might be the case from a convenience point of view even if the total time the user otherwise spends on finite element analysis is shorter than the time spent on data generation and training.

The main research question can now be addressed.

What is the level of accuracy with which neural networks can predict the damage patterns on a composite plate?
The final, tuned hybrid network predicts damage patterns with an accuracy of 0.9804 on the SSIM scale and an error of 0.0014 on the MSE scale, with both values measured on an independent dataset. The spread of these errors within the dataset is given in Figure 6.1.

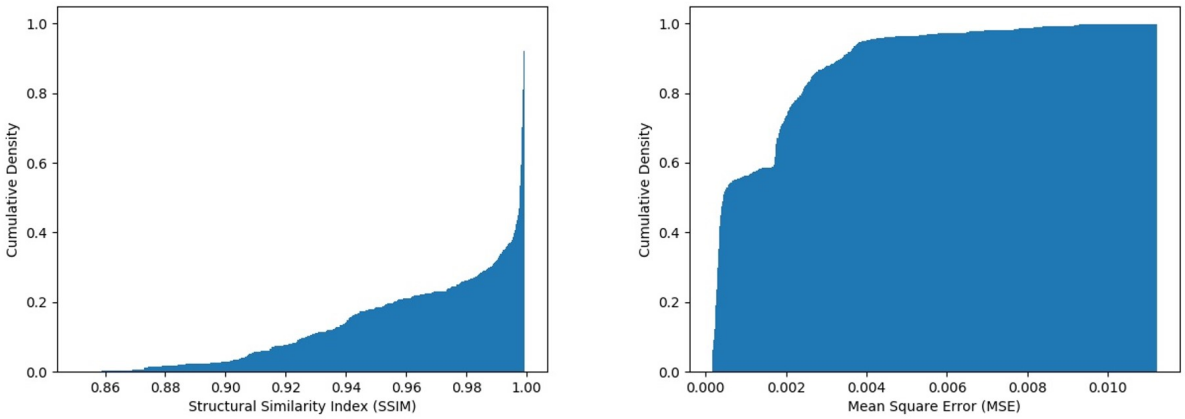


Figure 6.1: Cumulative density histograms of SSIM (left), and MSE (right) measures

6.3. Suggestions for Future Work

We had discussed at the beginning of this study, the utility of surrogate models in engineering - the results of this research appear to strengthen those claims. However, as an approximate model, there is always scope for taking it one step closer towards 'exact' models - some potential ways in which this can be done are:

- The final hybrid network in this study was developed using a two-step training process. When stricter quality requirements (and therefore convergence criteria) arise, the exact point of stopping each step will be difficult to ascertain, especially while considering the fact that improving one metric could act against the other. It will thus be of use to develop a single metric that combines both SSIM and MSE using a weighted average, to reduce the process to a single-step that can be iterated as many times are necessary.
- The training of Generative Adversarial Networks (GANs) also help produce high quality images [65], potentially more than we have seen generated in this study. These are based on the existence of two competing networks, one that is trained to generate images, and the other than classifies them as "high quality" or "low quality". The first network constantly improves itself in order to be accepted as "high quality", while the second has standards that keep increasing as it classifies them in that manner. Therefore, the two networks end up being responsible for each others' performance, which results in the generation of high quality images.
- We observed in carrying out this work that the availability of data could influence the choice of network architecture style. A more precise study on this influence could help aid decision-making when it comes to selecting a style to train for a given amount of data.
- Throughout this research, the only variables present in the FE model were the biaxial load values. Training the model with a greater number of variable parameters could increase its flexibility, and therefore potential for applied use.

Bibliography

[1] Z. Hashin and A. Rotem, "A Fatigue Criterion for Fiber- Reinforced Materials", Journal of Composite Materials, vol. 7, no. 4, pp. 448–464, 1973, ISSN: 1530793x. DOI: 10.1177/002199837300700404.

[2] I. Lapczyk and J. A. Hurtado, "Progressive damage modeling in fiber-reinforced materials", Composites Part A: Applied Science and Manufacturing, vol. 38, no. 11, pp. 2333–2341, 2007, ISSN: 1359835X. DOI: 10.1016/j.compositesa.2007.01.017.

[3] B. P. É. Clapeyron, "Mémoire sur la puissance motrice de la chaleur", Journal. de. l'Ecole. Polytechnique, vol. 14, no. 23, pp. 153–190, 1834.

[4] Engineering Toolbox, Evaporation from Water Surface, 2004. [Online]. Available: https://www.engineeringtoolbox.com/evaporation-water-surface-d_690.html (visited on 08/16/2019).

[5] N. Bohr, "LXXIII. On the constitution of atoms and molecules", The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, vol. 26, no. 155, pp. 857–875, 1913, ISSN: 1941-5982. DOI: 10.1080/14786441308635031.

[6] Engineering Toolbox, Compressed Air - Pressure Loss in Pipe Lines, 2005. [Online]. Available: https://www.engineeringtoolbox.com/pressure-drop-compressed-air-pipes-d_852.html (visited on 08/16/2019).

[7] Laerd Statistics, Linear Regression Analysis using SPSS Statistics, 2014. [Online]. Available: https://statistics.laerd.com/spss-tutorials/linear-regression-using-spss-statistics.php (visited on 08/16/2019).

[8] M. Merriman, "Note on the History of the Method of Least Squares", The Analyst, vol. 4, no. 5, p. 140, 1877, ISSN: 07417918. DOI: 10.2307/2635662.

[9] C. F. Gauss, "Theoria motus corporum coelestium sectionibus conicis solem ambientium", Werke, vol. 7, pp. 1–280, 1809. DOI: 10.1016/0377-0427(91)90094-Z.

[10] S. Jain, A comprehensive beginners guide for Linear, Ridge and Lasso Regression in Python and R, 2017. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/ (visited on 08/19/2019).

[11] OriginLab Corporation, Fitting 3D Surfaces. [Online]. Available: https://www.originlab.com/doc/Origin-Help/Fitting-2DSurf (visited on 08/16/2019).

[12] T. Hastie, R. Tibshirani, and J. Friedman, "Springer Series in Statistics", The Elements of Statistical Learning, vol. 27, no. 2, 2009, ISSN: 03436993. DOI: 10.1007/b94608. arXiv: arXiv:1011.1669v3.

[13] M. Deshpande, A Guide to Improving Deep Learning's Performance, 2017. [Online]. Available: https://pythonmachinelearning.pro/a-guide-to-improving-deep-learnings-performance/ (visited on 08/16/2019).

[14] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane, Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming. 1996. DOI: 10.1007/978-94-009-0279-4_9.

[15] S. Milborrow, An example of a CART classification tree, 2011.

[16] G. James, D. Witten, T. Hastie, and R. Tibshirani, An Introduction to Statiscal Learning with Applications in R. 2013, ISBN: 9780387781884. DOI: 10.1016/j.peva.2007.06.006.

[17] D. Niedermayer, "An introduction to Bayesian networks and their contemporary applications", Studies in Computational Intelligence, vol. 156, pp. 117–130, 2008, ISSN: 1860949X. DOI: 10.1007/978-3-540-85066-3_5.

[18] M. Melanie, "An introduction to genetic algorithms", Computers & Mathematics with Applications, vol. 32, no. 6, 1996, ISSN: 08981221. DOI: 10.1016/S0898-1221(96)90227-8.

[19] H. Drucker, C. J. Surges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines", in Advances in Neural Information Processing Systems, 1997, ISBN: 0262100657.

[20] Sergios Theodoridis and K. Koutroumbas, Pattern Recognition (Fourth Edition). 2009, ISBN: 9781597492720.

[21] Z. Weinberg, SVM separating hyperplanes, 2012.

[22] B. V. Lewenstein, "Yes, We Have No Neutrons: An Eye-Opening Tour through the Twists and Turns of Bad Science . A. K. Dewdney", Isis, vol. 89, no. 3, pp. 566–567, 1998, ISSN: 0021-1753. DOI: 10.1086/384132.

[23] R. S. Govindaraju, "Artificial neural networks in hydrology. I: Preliminary concepts", Journal of Hydrologic Engineering, vol. 5, no. 2, pp. 115–123, 2000, ISSN: 10840699. DOI: 10.1061/(ASCE)1084-0699(2000)5:2(115).

[24] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification", in IJCAI International Joint Conference on Artificial Intelligence, 2011, ISBN: 9781577355120. DOI: 10.5591/978-1-57735-516-8/IJCAI11-210.

[25] A. Sharma, Understanding Activation Functions in Deep Learning, 2017.

[26] C. W. Dawson and R. Wilby, "An artificial neural network approach to rainfall-runoff modelling", Hydrological Sciences Journal, vol. 43, no. 1, pp. 47–66, 1998, ISSN: 0262-6667. DOI: 10.1080/02626669809492102.

[27] L. K. Hansen and P. Salamon, "Neural Network Ensembles", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, no. 10, pp. 993–1001, 1990, ISSN: 01628828. DOI: 10.1109/34.58871.

[28] J. Opfermann, "Kinetic analysis using multivariate non-linear regression. I. Basic concepts", Journal of Thermal Analysis and Calorimetry, vol. 60, no. 2, pp. 641–658, 2000, ISSN: 13886150. DOI: 10.1023/A:1010167626551.

[29] J. Barzilai and J. M. Borwein, "Two-point step size gradient methods", IMA Journal of Numerical Analysis, vol. 8, no. 1, pp. 141–148, 1988, ISSN: 02724979. DOI: 10.1093/imanum/8.1.141.

[30] Suryansh.S, Gradient Descent: All You Need to Know, 2018. [Online]. Available: https://hackernoon.com/gradient-descent-aynk-7cbe95a778da (visited on 08/17/2019).

[31] L. F. Wessels and E. Barnard, "Avoiding False Local Minima by Proper Initialization of Connections", IEEE Transactions on Neural Networks, vol. 3, no. 6, pp. 899–905, 1992, ISSN: 19410093. DOI: 10.1109/72.165592.

[32] W. A. Gardner, "Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique", Signal Processing, vol. 6, no. 2, pp. 113–133, 1984, ISSN: 01651684. DOI: 10.1016/0165-1684(84)90013-6.

[33] M. Rizwan, Gradient Descent with Momentum, 2018. [Online]. Available: https://engmrk.com/gradient-descent-with-momentum/ (visited on 08/17/2019).

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", Nature, vol. 323, no. 6088, pp. 533–536, 1986, ISSN: 00280836. DOI: 10.1038/323533a0.

[35] T. Tieleman, G. E. Hinton, N. Srivastava, and K. Swersky, "Divide the gradient by a running average of its recent magnitude", COURSERA: Neural Networks for Machine Learning, 2012. DOI: https://www.coursera.org/learn/neural-networks/lecture/YQHki/rmsprop-divide-the-gradient-by-a-running-average-of-its-recent-magnitude.

[36] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic gradient descent", in ICLR: International Conference on Learning Representations, 2015. arXiv: arXiv:1412.6980v9.

[37] B. Kalman and S. Kwasny, "Why tanh: choosing a sigmoidal function", in IJCNN International Joint Conference on Neural Networks, 1992. DOI: 10.1109/ijcnn.1992.227257.

[38] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning", Nature, vol. 521, no. 7553, pp. 436–444, 2015, ISSN: 14764687. DOI: 10.1038/nature14539.

[39] I. Goodfellow, Y. Bengio, and A. Courville, "Ch9 Convolutional Networks", in Deep Learning, 2016, pp. 330–372, ISBN: 0309-2402. arXiv: 1411.4555v1.

[40] B. Prijono, Student Notes: Convolutional Neural Networks (CNN) Introduction, 2018. [Online]. Available: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/ (visited on 08/17/2019).

[41] U. Sinha, Convolutions: Image convolution examples, 2017. [Online]. Available: http://aishack.in/tutorials/image-convolution-examples/ (visited on 08/16/2019).

[42] L. Berke and P. Hajela, "Applications of artificial neural nets in structural mechanics", Structural Optimization, vol. 4, no. 2, pp. 90–98, 1992, ISSN: 09344373. DOI: 10.1007/BF01759922.

[43] P. Hajela and L. Berke, "Neural networks in structural analysis and design: An overview", in 4th Symposium on Multidisciplinary Analysis and Optimization, 1992, 1992, pp. 902–914.

[44] L. Berke, S. N. Patnaik, and P. L. Murthy, "Optimum design of aerospace structural components using neural networks", Computers and Structures, vol. 48, no. 6, pp. 1001–1010, 1993, ISSN: 00457949. DOI: 10.1016/0045-7949(93)90435-G.

[45] R. K. Kapania and Y. Liu, “Applications of Artificial Neural Networks in Structural Engineering with Emphasis on Continuum Models”, Virginia Polytechnic Institute and State University, Tech. Rep., 1998. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20000101656.pdf.

[46] Z. Waszczyszyn and L. Ziemiański, “Neural networks in mechanics of structures and materials - New results and prospects of applications”, Computers and Structures, vol. 79, no. 22-25, pp. 2261–2276, 2001, ISSN: 00457949. DOI: 10.1016/S0045-7949(01)00083-9.

[47] K. Jármai, “Expert Systems and Artificial Neural Networks in Structural Optimization”, Series C. Mechanical Engineering, vol. 47, pp. 111–122, 1997.

[48] A. Lavaei and A. Lohrasbi, “Dynamic Analysis of Structures Using Neural Networks”, in 15th World Conference on Earthquake Engineering, Lisbon, 2012.

[49] L. Roseiro, U. Ramos, and R. Leal, “Neural networks in damage detection of composite laminated plates”, WSEAS Transactions on Systems, vol. 4, no. 4, pp. 430–434, 2005, ISSN: 11092777.

[50] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: From error measurement to structural similarity”, IEEE transactions on image processing, vol. 13, no. 4, pp. 600–612, 2004.

[51] P. M. B. Vitányi, “Information Distance in Multiples”, IEEE Transactions on Information Theory, vol. 57, no. 4, pp. 2451–2456, 2011.

[52] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization”, Journal of Machine Learning Research, vol. 13, pp. 281–305, 2012, ISSN: 15324435.

[53] D. P. Mandic, “A generalized normalized gradient descent algorithm”, IEEE Signal Processing Letters, vol. 11, no. 2, pp. 115–118, 2004, ISSN: 10709908. DOI: 10.1109/LSP.2003.821649.

[54] TensorFlow, “TensorFlow Guide”, Tech. Rep. [Online]. Available: https://www.tensorflow.org/guide/.

[55] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization”, in COLT 2010 - The 23rd Conference on Learning Theory, 2010, pp. 257–269, ISBN: 9780982252925.

[56] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method”, Tech. Rep., 2012. [Online]. Available: https://arxiv.org/abs/1212.5701.

[57] Y. Nesterov, “A method for unconstrained convex minimization problem with the rate of convergence o(1/k2)”, Doklady ANSSR, vol. 27, no. 2, pp. 543–547, 1983.

[58] S. Maleki, Y. Gao, M. J. Garzarán, T. Wong, and D. A. Padua, “An evaluation of vectorizing compilers”, in Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT, 2011, pp. 372–382, ISBN: 9780769545660. DOI: 10.1109/PACT.2011.68.

[59] G. E. Hinton, A. Krizhevsky, I. Sutskever, and N. Srivastva, System and method for addressing overfitting in a neural network, 2016.

[60] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks”, Journal of Machine Learning Research, vol. 9, pp. 249–256, 2010, ISSN: 15324435.

[61] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”, in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034, ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.123.

[62] J. Gauthier, “Conditional generative adversarial nets for convolutional face generation”, Tech. Rep., 2014. [Online]. Available: http://cs231n.stanford.edu/reports/2015/pdfs/jgauthie_final_report.pdf.

[63] D. Ciresan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification”, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2012, ISBN: 9781467312264. DOI: 10.1109/CVPR.2012.6248110.

[64] A. Dosovitskiy and T. Brox, “Generating images with perceptual similarity metrics based on deep networks”, in Advances in Neural Information Processing Systems, 2016.

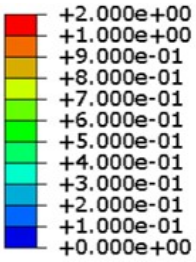
[65] M. S. Sajjadi, B. Scholkopf, and M. Hirsch, “EnhanceNet: Single Image Super-Resolution Through Automated Texture Synthesis”, in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 4501–4510, ISBN: 9781538610329. DOI: 10.1109/ICCV.2017.481.

Appendices

A

Damage Patterns from Introduction Model

The model solved in Section 1.2 outputs 16 damage patterns (4 for each of its 4 plies). These patterns are given in this appendix for reference.



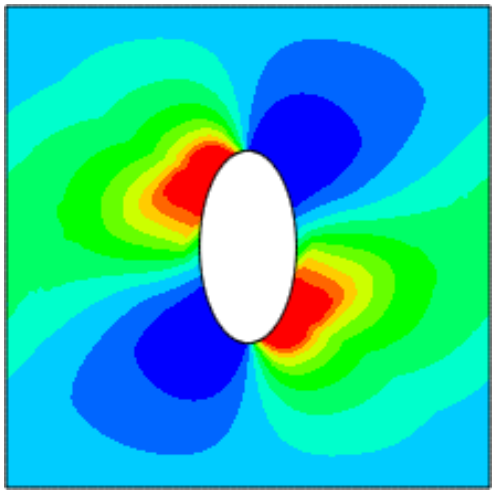


Figure A.1: Fibre tension damage pattern for ply 1

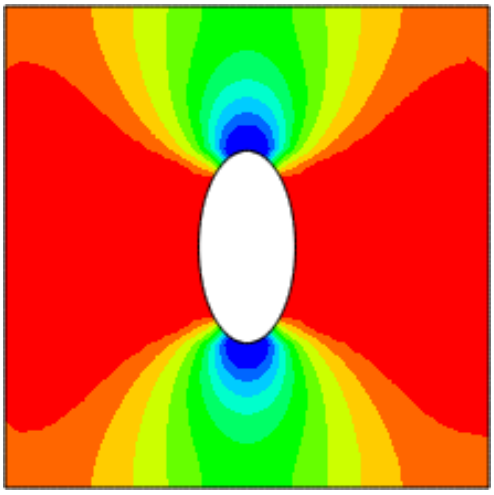


Figure A.2: Fibre tension damage pattern for ply 2

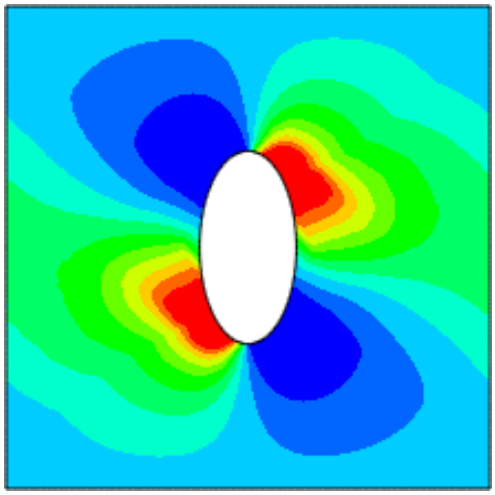


Figure A.3: Fibre tension damage pattern for ply 3

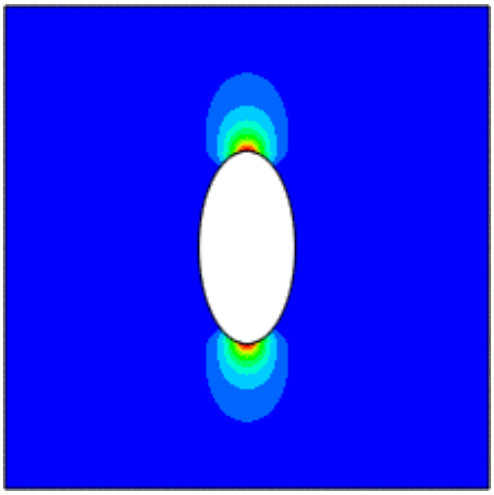


Figure A.4: Fibre tension damage pattern for ply 4

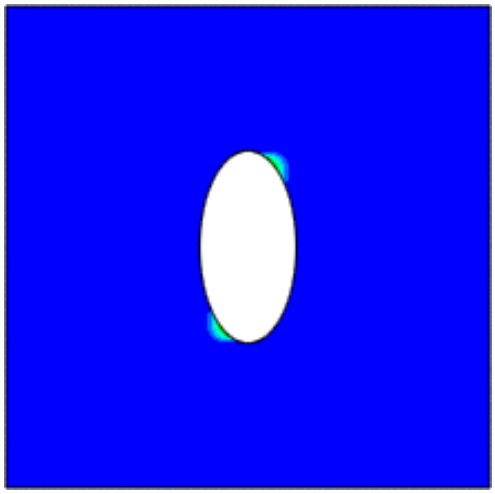


Figure A.5: Fibre compression damage pattern for ply 1

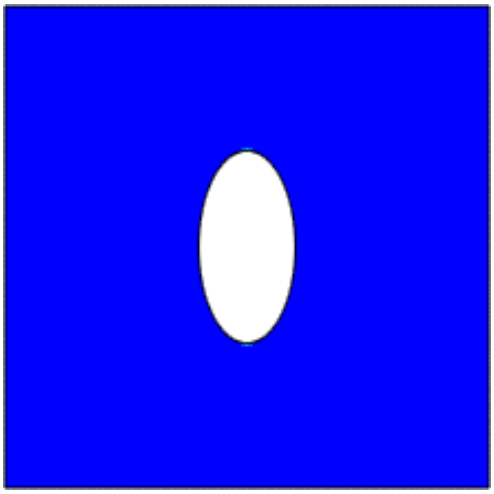


Figure A.6: Fibre compression damage pattern for ply 2

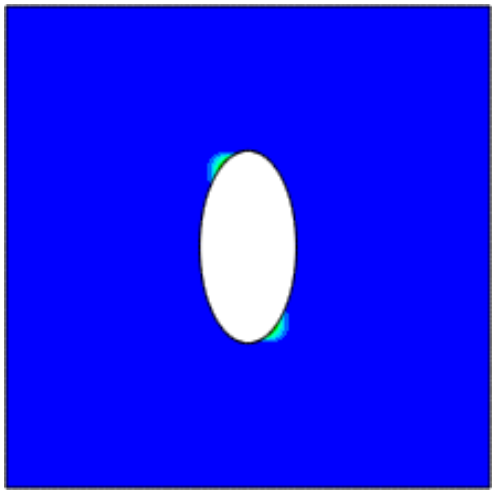


Figure A.7: Fibre compression damage pattern for ply 3

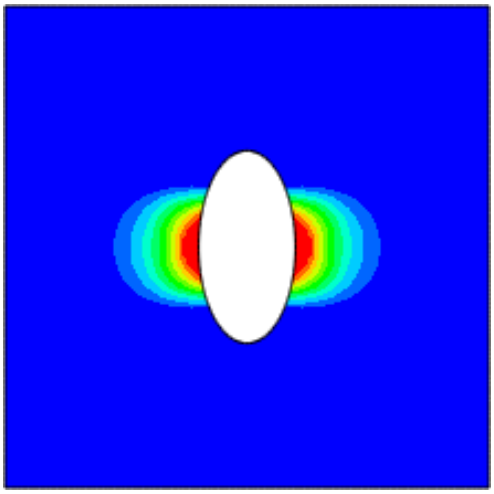


Figure A.8: Fibre compression damage pattern for ply 4

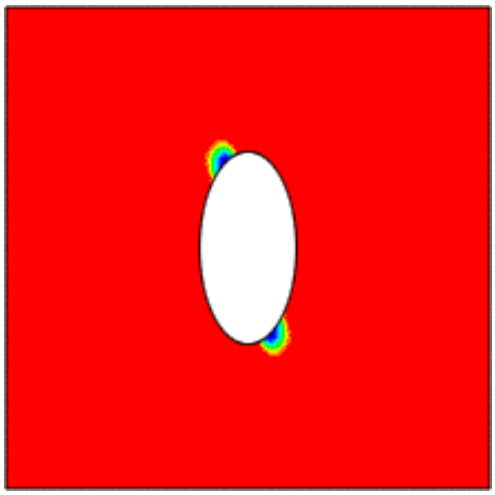


Figure A.9: Matrix tension damage pattern for ply 1

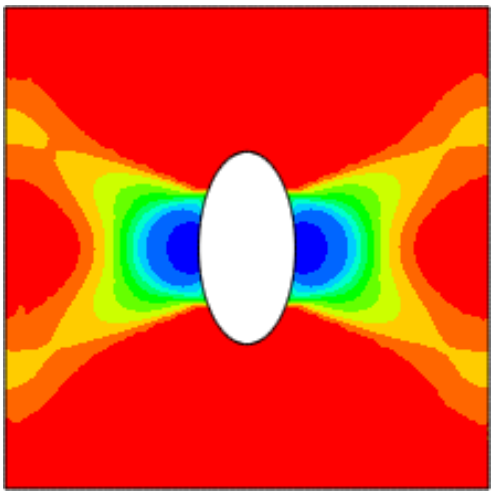


Figure A.10: Matrix tension damage pattern for ply 2

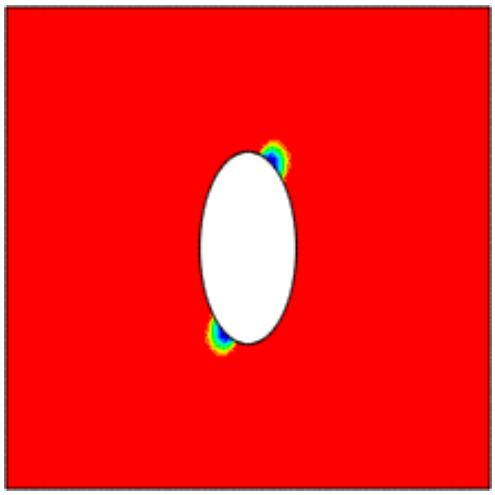


Figure A.11: Matrix tension damage pattern for ply 3

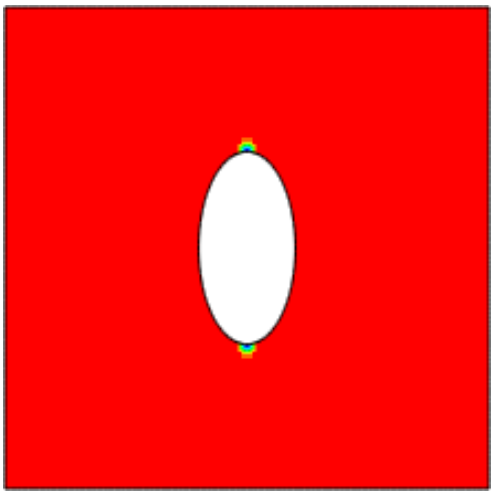


Figure A.12: Matrix tension damage pattern for ply 4

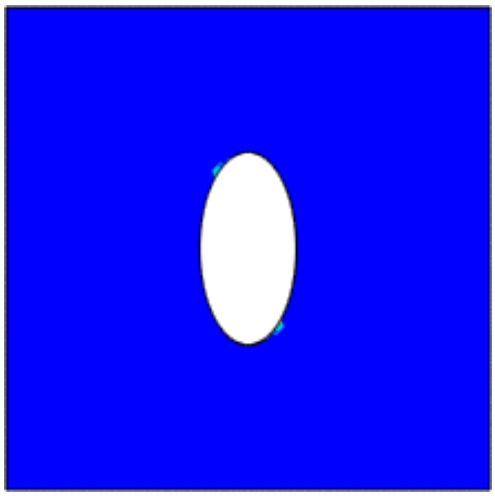


Figure A.13: Matrix compression damage pattern for ply 1

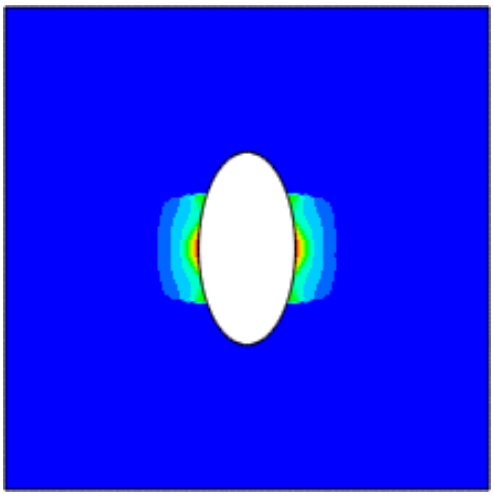


Figure A.14: Matrix compression damage pattern for ply 2

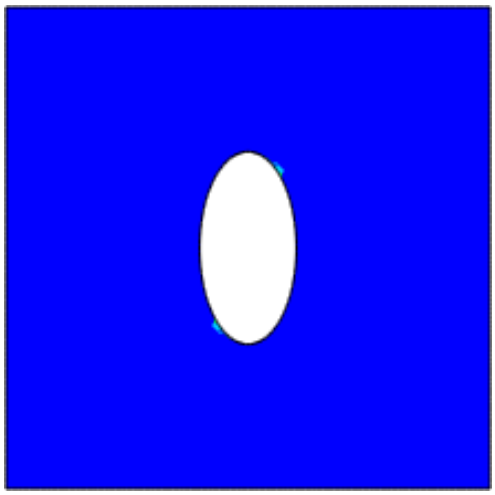


Figure A.15: Matrix compression damage pattern for ply 3

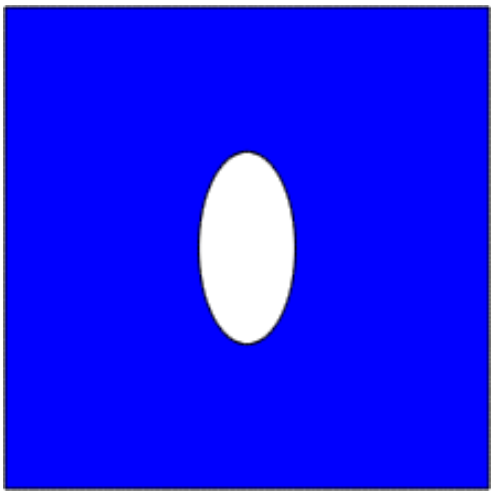


Figure A.16: Matrix compression damage pattern for ply 4

B

Improved Model Predictions

The improved models discussed in Chapter 5 are used to generate damage patterns besides just the test image. These are shown in this appendix for each of these networks.

Reduced Image Neural Network

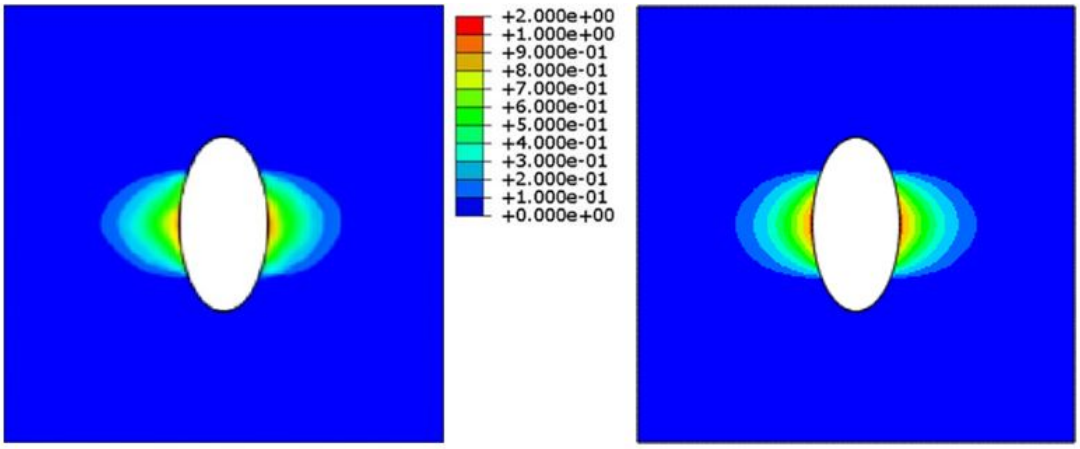


Figure B.1: Output of RINN model for an input $(0.75, 1.00, 4, 1)$ (left) compared to the true damage pattern (right)

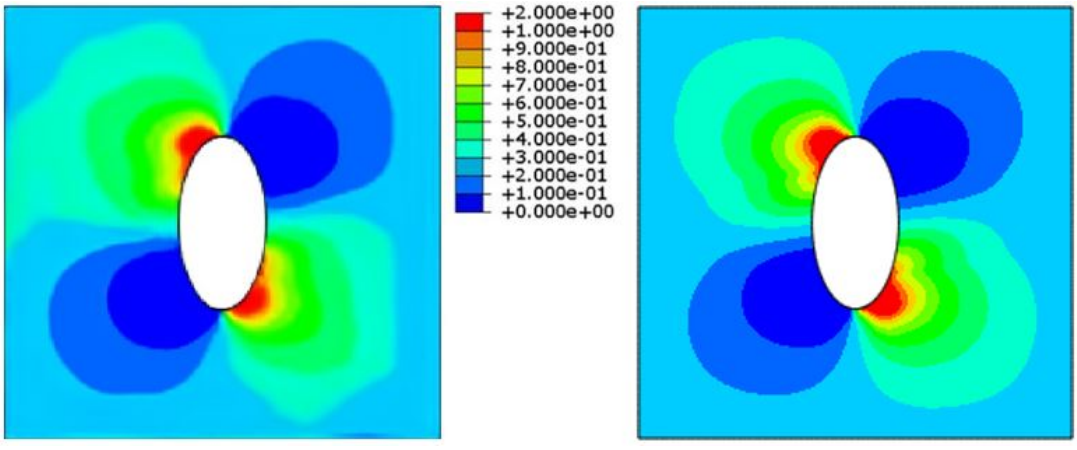


Figure B.2: Output of RINN model for an input $(0.25, 0.75, 1, 2)$ (left) compared to the true damage pattern (right)

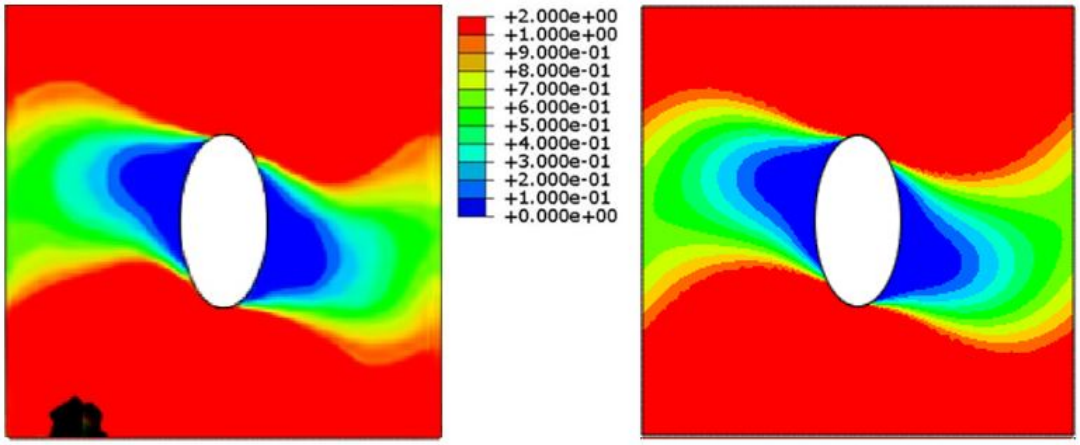


Figure B.3: Output of RINN model for an input $(0.50, 0.00, 1, 4)$ (left) compared to the true damage pattern (right)

Hybrid Network

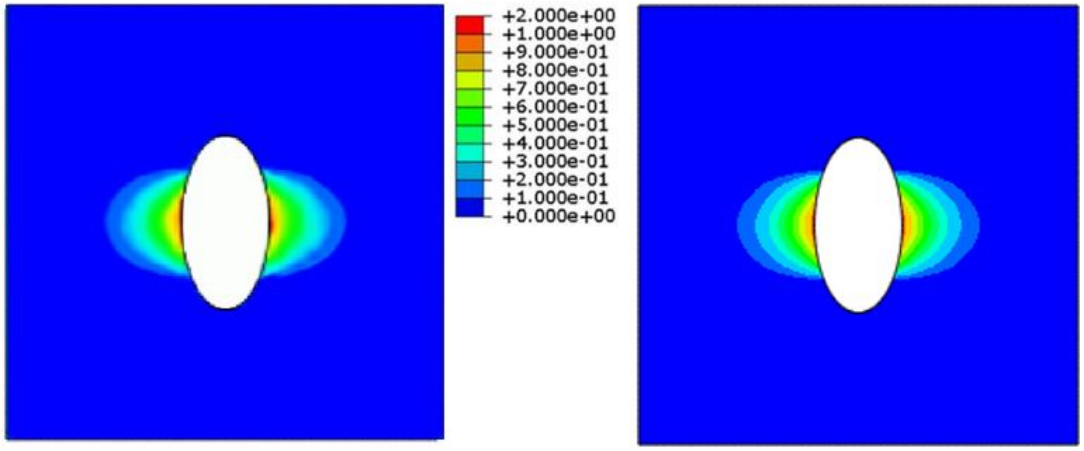


Figure B.4: Output of hybrid network for an input $(0.75, 1.00, 4, 1)$ (left) compared to the true damage pattern (right)

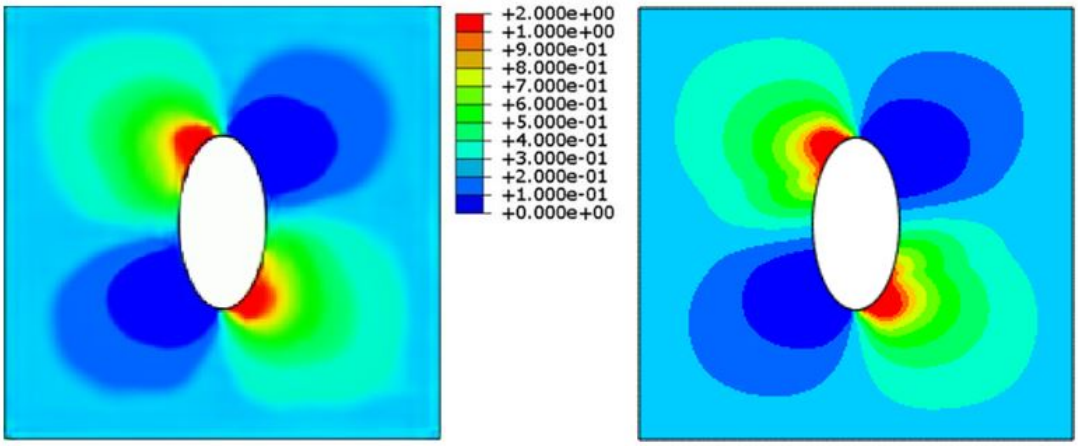


Figure B.5: Output of hybrid network for an input $(0.25, 0.75, 1, 2)$ (left) compared to the true damage pattern (right)

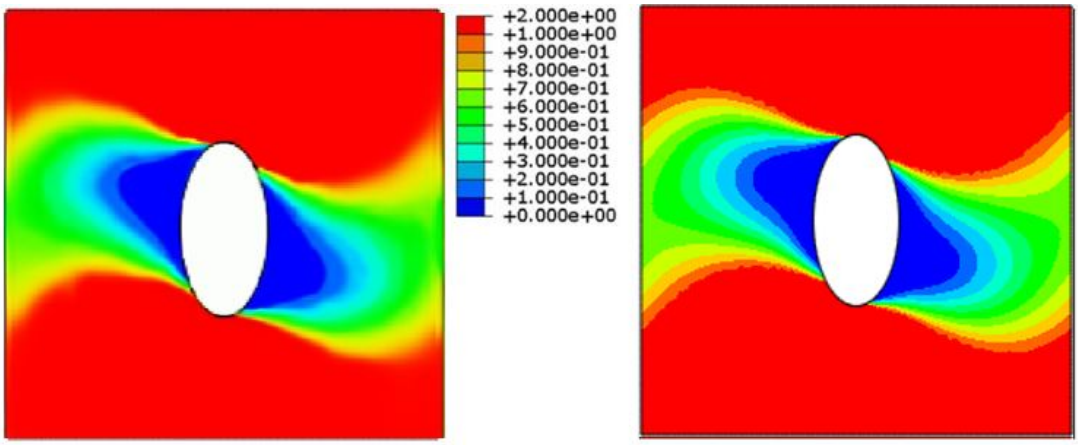


Figure B.6: Output of hybrid network for an input $(0.50, 0.00, 1, 4)$ (left) compared to the true damage pattern (right)

Standard Neural Network

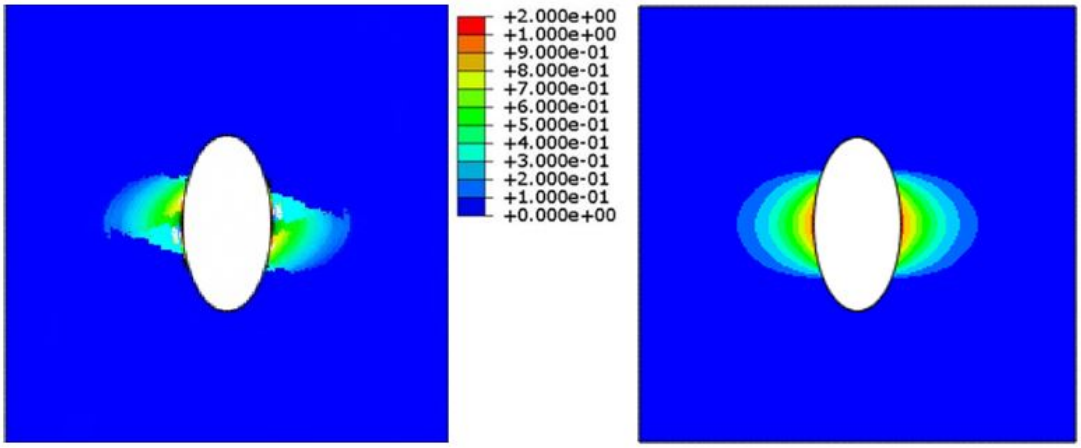


Figure B.7: Output of standard neural network for an input $(0.75, 1.00, 4, 1)$ (left) compared to the true damage pattern (right)

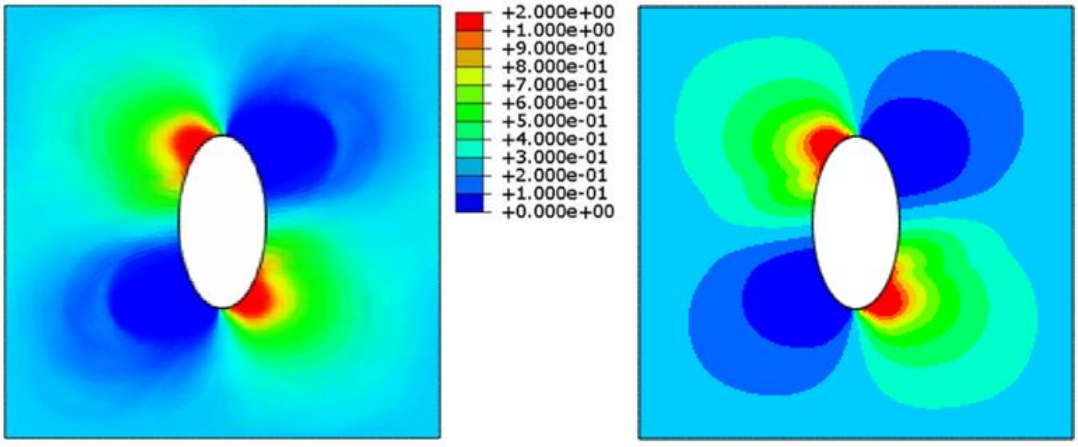


Figure B.8: Output of standard neural network for an input $(0.25, 0.75, 1, 2)$ (left) compared to the true damage pattern (right)

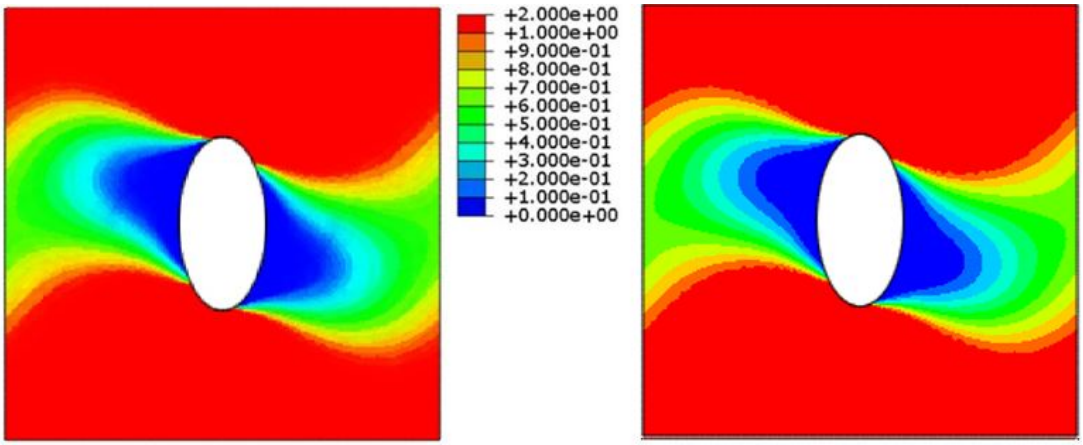


Figure B.9: Output of standard neural network for an input $(0.50, 0.00, 1, 4)$ (left) compared to the true damage pattern (right)

Fine-Tuned Hybrid Network

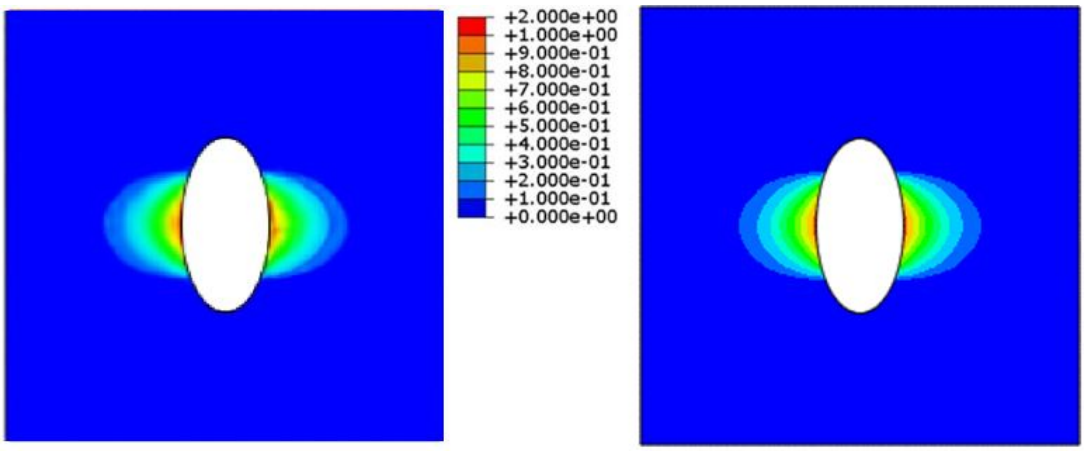


Figure B.10: Output of fine-tuned hybrid network for an input $(0.75, 1.00, 4, 1)$ (left) compared to the true damage pattern (right)

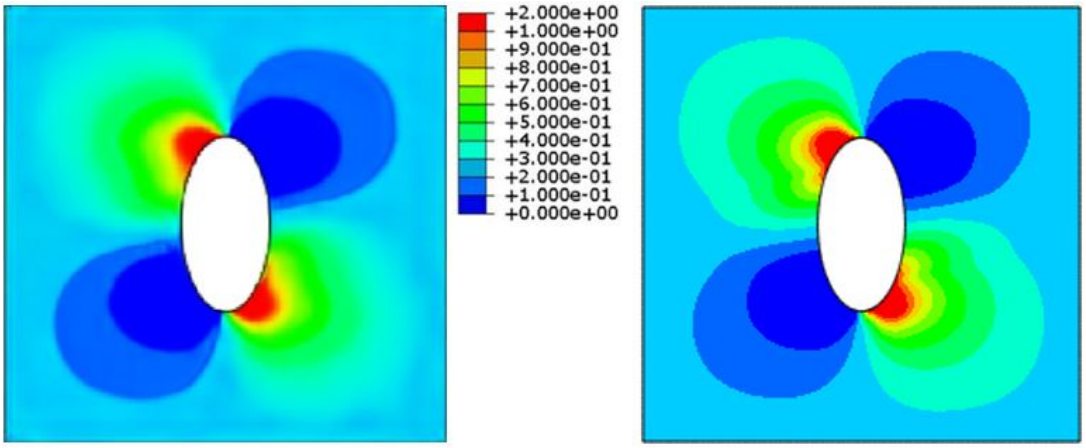


Figure B.11: Output of fine-tuned hybrid network for an input $(0.25, 0.75, 1, 2)$ (left) compared to the true damage pattern (right)

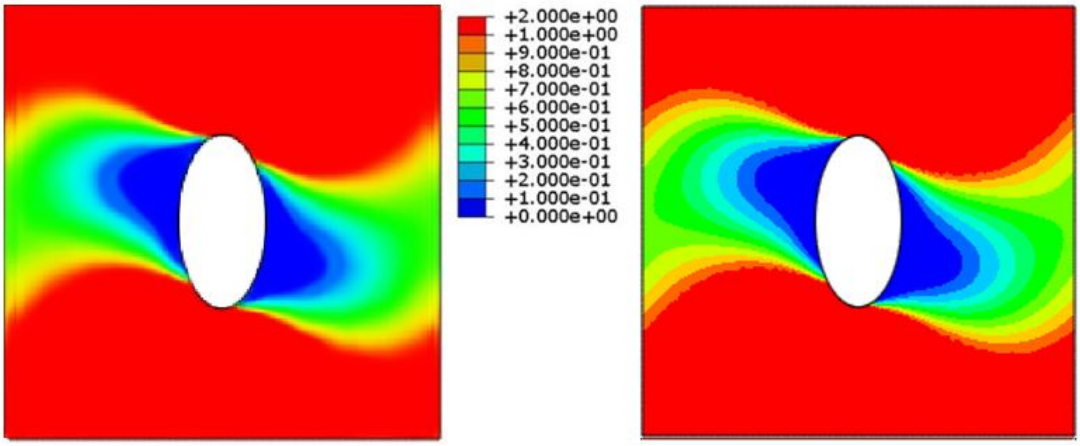


Figure B.12: Output of fine-tuned hybrid network for an input $(0.50, 0.00, 1, 4)$ (left) compared to the true damage pattern (right)

Hybrid Network Trained with SSIM

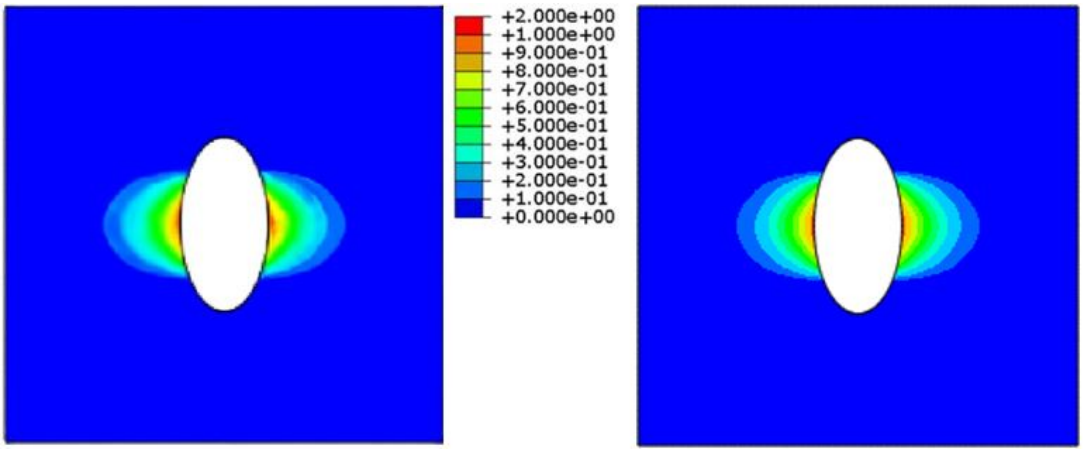


Figure B.13: Output of SSIM-tuned hybrid network for an input $(0.75, 1.00, 4, 1)$ (left) compared to the true damage pattern (right)

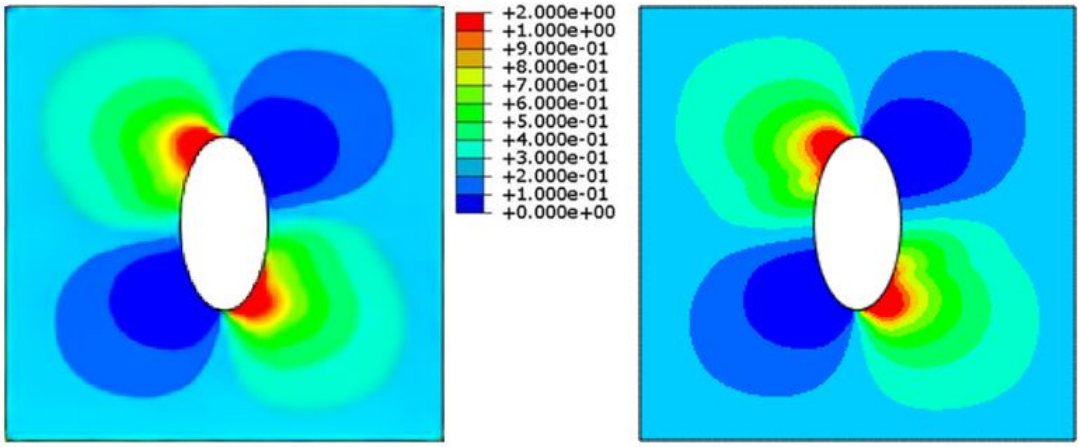


Figure B.14: Output of SSIM-tuned hybrid network for an input $(0.25, 0.75, 1, 2)$ (left) compared to the true damage pattern (right)

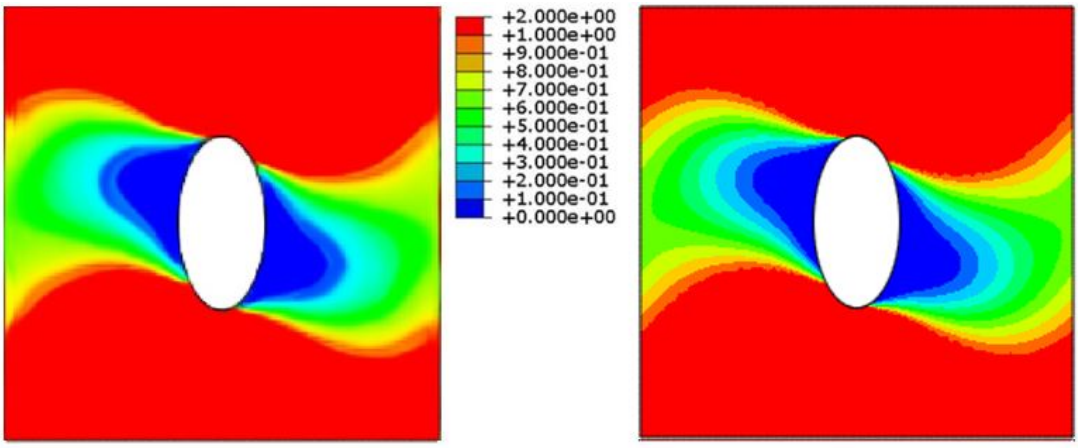


Figure B.15: Output of SSIM-tuned hybrid network for an input $(0.50, 0.00, 1, 4)$ (left) compared to the true damage pattern (right)