Hybrid neural network for the prediction of damage patterns in open-hole composites

Karthik Venkatesan[a], Boyang Chen[a,*]

[a]*Department of Aerospace Structures and Materials, Faculty of Aerospace Engineering, Delft University of Technology, Kluyverweg 1, 2629 HS Delft, The Netherlands*

Abstract

Damage pattern predictions of open-hole laminates under different loading conditions are ubiquitous in the finite element modelling of composite structures. This work investigates the applicability of artificial neural networks for the fast and accurate generation of damage patterns for a composite plate with a cut-out under a variety of loading conditions. Data for training and evaluating these neural networks were generated through nonlinear finite element models. Different neural networks, such as a standard Feedforward Neural Network and a Hybrid Neural Network that combines a Feedforward Neural Network with a convolutional decoder, have been tested for this task. To quantify the resemblance between the predicted and actual outputs in terms of colours and contours, different performance metrics have been explored. The use of the Structural Similarity Index, in addition to the standard Mean Square Error, was explored to improve the visual quality of outputs from the neural network. The Hybrid Neural Network has been shown to accurately and efficiently predict the damage patterns of the open-hole laminate with small scatter in testing error, thereby constituting a promising candidate for a surrogate model of open-hole composite panels.

Keywords: damage patterns, machine learning, surrogate, neural network, composites, open-hole

1. Introduction

In the design of aerospace composite structures, the reliable prediction of damage and failure of open-hole composite coupons is a prerequisite for the subsequent design of larger and more complex components. Therefore, many works have been dedicated to the high-fidelity finite element modelling of open-hole composite coupons [1, 2, 3, 4, 5, 6]. With fine meshes and advanced modelling technologies, these models can reproduce accurately the experimental data on failure strengths and damage patterns of open-hole composite coupons. One could then employ these models in a larger scale model in regions where open-hole occurs under a suitable multi-scale framework [7, 8, 9]. If the larger structure contains many open-hole regions, the

*Corresponding author
Email addresses: karthikvenkatesan.kv@gmail.com (Karthik Venkatesan), B.Chen-2@tudelft.nl (Boyang Chen)

overall computational cost may quickly become prohibitive, as the high-fidelity coupon-scale models typically require fine meshes to resolve accurately the local stress fields in damage hot-spots (e.g., the cohesive zone). A popular approach to reduce the computational cost is to develop efficient surrogate models of the otherwise costly high-fidelity models for the open-hole coupons [10, 11, 12, 13, 14], such that the surrogate models would directly feed useful information to the larger scale model for multi-scale modelling.

Many work have been done in the past on the application of Machine Learning (ML) in composites research [15, 16, 17, 18, 19, 20, 21, 22]. However, there is still very limited work on the prediction of damage patterns using ML. Sepasdar et al. [23] trained a ML approach for damage pattern prediction at the microscale, with the microstructural geometry image as input to the ML model. To the best of the authors' knowledge, there is still no ML surrogate model which takes the loadings as inputs and outputs directly the damage patterns of the plies in a composite laminate. Such a surrogate would allow engineers to quickly visualize the damage patterns with respect to the applied loadings and identify critical regions without running through the time-consuming high-fidelity nonlinear finite element models. In this study, we establish a ML surrogate model which generates accurate predictions of the damage patterns for different plies, with the applied loading as input. An open-hole laminate at the coupon-scale is selected for demonstration, as open-hole coupons are frequently tested to generate design parameters for upper-scale structures in aircraft design.

The rest of the paper will start with the description of the finite element model which serves as the data generator in Section 2. Next, an initial small dataset is generated and different neural network models are tested in Section 3. Then, a special neural network model is chosen to be further trained and improved on an expanded dataset in Section 4, eventually arriving at a novel hybrid neural network. Finally, the conclusions are drawn in Section 5.

2. Finite Element Model - the data generator

A Finite Element (FE) model for the open-hole composite laminate is created to generate data for the subsequent training of surrogate models. The data of this study would focus on images of damage patterns of the different plies in the laminate under different biaxial loadings. In this section, details of this model are described, specifically the geometrical and mechanical properties, the elements and meshing strategy, the loads applied, and the setup of the numerical analysis procedure.

2.1. Geometry and Material

The geometry of the structure is a square plate with an elliptical hole at its centre, as shown in Figure 2.1. As described in the previous section, this is decided on the basis of being a simple geometry that exhibits a complex nonlinear phenomenon, and is a modular structure in mechanical 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. The layup of the laminate is chosen as $[45/90/\text{-}45/0]_s$.

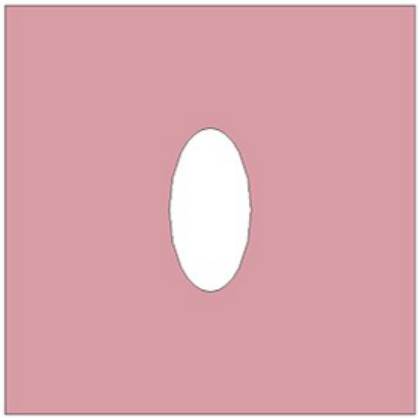


Figure 2.1: Geometry of composite plate with hole

2.2. Loading

The damage patterns of this structure are generated by applying biaxial tensile displacement loading on its boundaries. The focus of this study is to establish a ML surrogate model that can predict the damage patterns with respect to given boundary conditions. For simplicity and without loss of generality, compressive loading, which may lead to buckling, is not considered in this study.

2.3. Mesh Convergence

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 is on the prediction of damage patterns, the convergence criterion is based on the 'convergence' of the patterns. The Mean Square Error (MSE), which is computed between two images by applying it pixel-by-pixel, is used to evaluate the difference between the damage patterns of two successive models.

To determine what MSE value can be considered negligible, we refine the mesh until we observe that the damage patterns do not vary significantly, then record 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. Through this process, the cut-off value for the MSE to be considered negligible is determined to be 7×10^{-4}.

The model that gives this cut-off value of MSE, however, may not be the final, converged mesh because: 1) more damage modes may appear in more plies during the later stage of the loading; and 2) more severe damage patterns at the later stage may be more sensitive to mesh refinement. To mitigate these risks, this cut-off MSE value 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 2.2 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. We can see that the third model does not improve the results notably, therefore the proposed cut-off MSE value is reasonable and the second mesh is considered to be the final, converged mesh.

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 2.3.

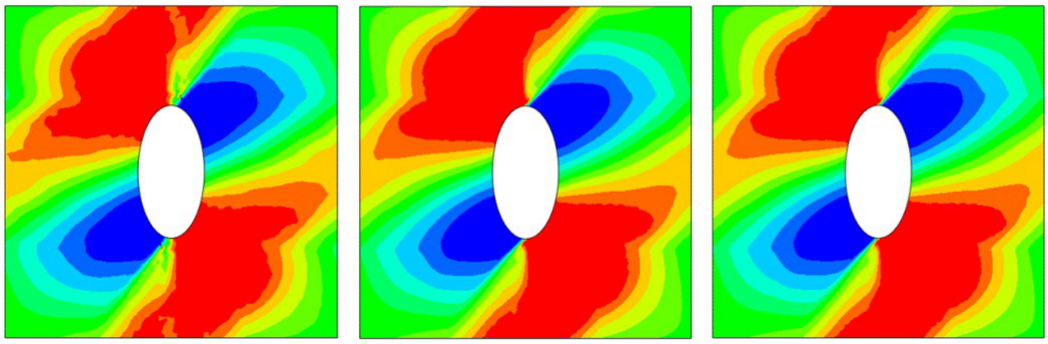


Figure 2.2: Ply 1 fibre damage in tension, computed with different levels of mesh refinement - MSE between the first two is 7.39×10^{-3}, and between the next two is 3.27×10^{-4}

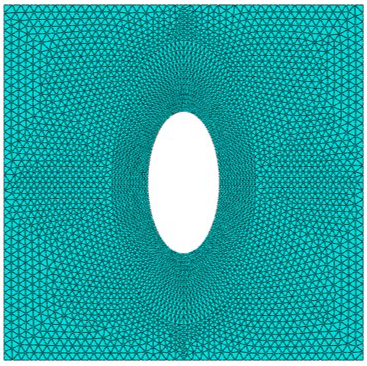


Figure 2.3: Plate after meshing

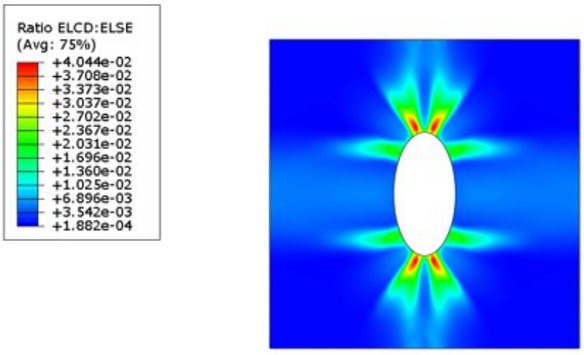


Figure 2.4: Ratio between total energy dissipated due to viscoelastic deformation (ELCD) and total elastic strain energy (ELSE), plotted over the plate

2.4. Solution Convergence

Setting up the configuration of the numerical model is important for the convergence of a nonlinear problem such as this. Note that this is done in parallel to mesh convergence. 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 increment 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, we check the ratio between the viscous energy dissipated and the total elastic strain energy. The plot in Figure 2.4 shows that in the area away from the hole, the viscoelastic dissipation is less than 0.7%. Closer to the hole, it is generally below 3%. Near the sharper bends, it goes up to just over 4%.

3. Preliminary Neural Network Models

3.1. Initial 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 models. 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, 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 3.1.

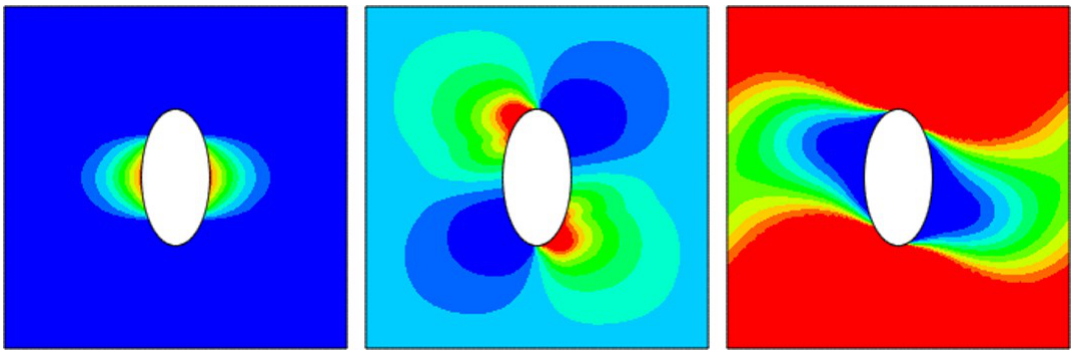


Figure 3.1: Outputs corresponding to the inputs (0.75, 1.00, 4, 1), (0.25, 0.75, 1, 2) and (0.50, 0.00, 1, 4) respectively

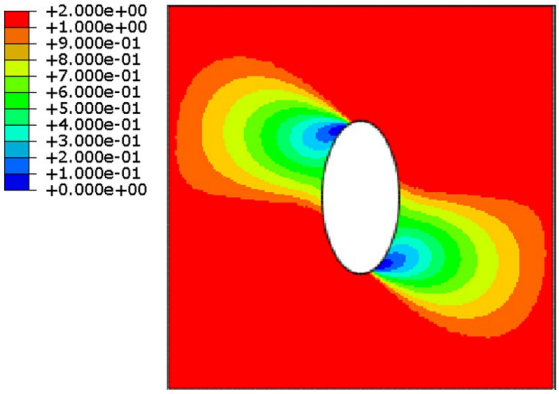


Figure 3.2: Damage pattern set aside as test image

Another damage pattern from the dataset (shown in Figure 3.2), is set aside as a test image - it is planned to be used as the reference for visualising results from the trained models. This particular one is selected because it is deemed 'detailed' and 'complex' enough to study variations in results (as opposed to say, the first pattern in Figure 3.1, which can be expected to be easier to reproduce). Of the other 399 images, 300 are randomly selected 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. The performance of the model on the validation set can however guide the selection of hyperparameters.

The initial dataset of 400 images can now be used for training and testing of different candidate neural networks. Here, we test two candidates, namely the standard Feedforward Neural Network (FFNN) and a hybrid network based on the Convolutional Neural Network (CNN).

3.2. Feedforward Neural Network

To train a FFNN, the output images in the dataset need to be transformed as a suitable final layer of the network. Figure 3.3 shows how an image with three colour channels is vectorized into a one-dimensional array.

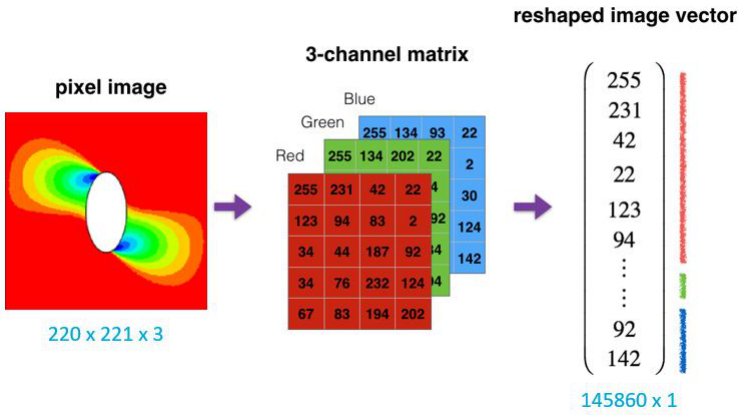


Figure 3.3: Transformation of a 3-index image into a vector

Now, the architecture comprises an output layer with 145860 nodes, an undetermined number of hidden nodes (which may be in one or more layers), and 4 input nodes, as shown in Figure 3.4. 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.

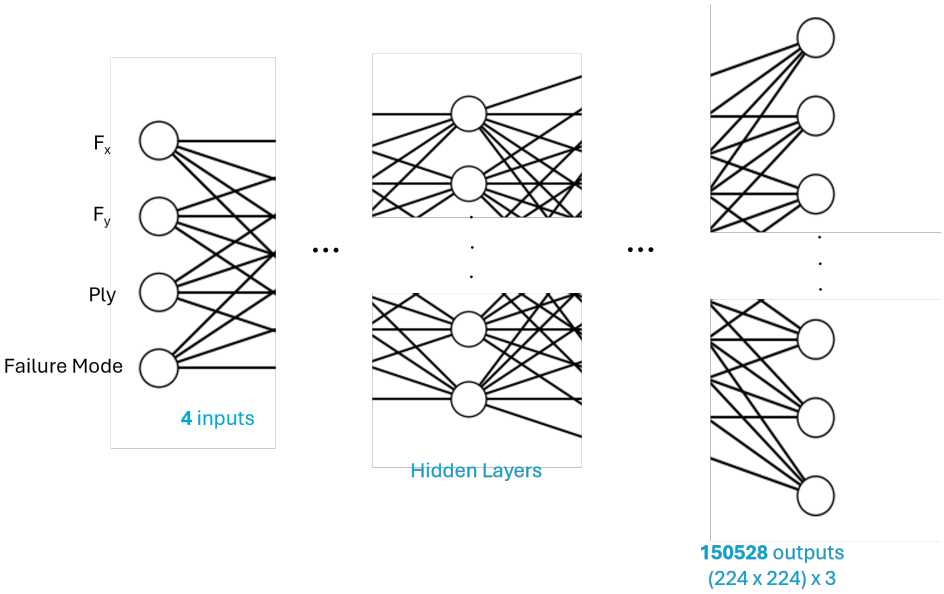


Figure 3.4: Architecture of FFNN for generating image outputs

3.2.1. 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 MSE 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.

Optimization algorithms. Gradient descent is a basic optimization algorithm that is robust and easy to visualize [24], but there exist modified versions of this algorithm that help speed up convergence. Figure 3.5 compares them on that front, and points favourably to the use of the Adam algorithm. As outlined while splitting the generated data in Section 3.1, 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). The Adaptive gradient (Adagrad) algorithm [25], the ADADELTA algorithm [26] and the Nesterov Accelerated Gradient (NAG) algorithm [27] 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 3.5 for the sake of visual clarity.

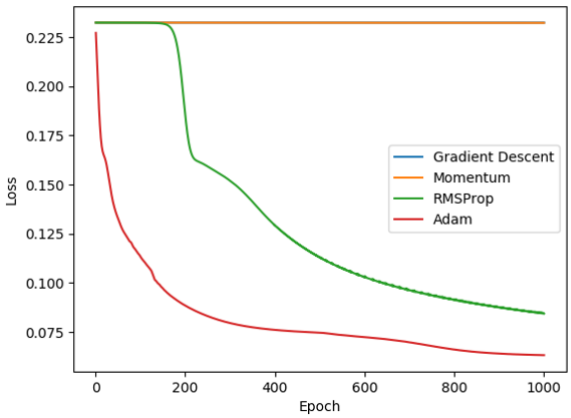


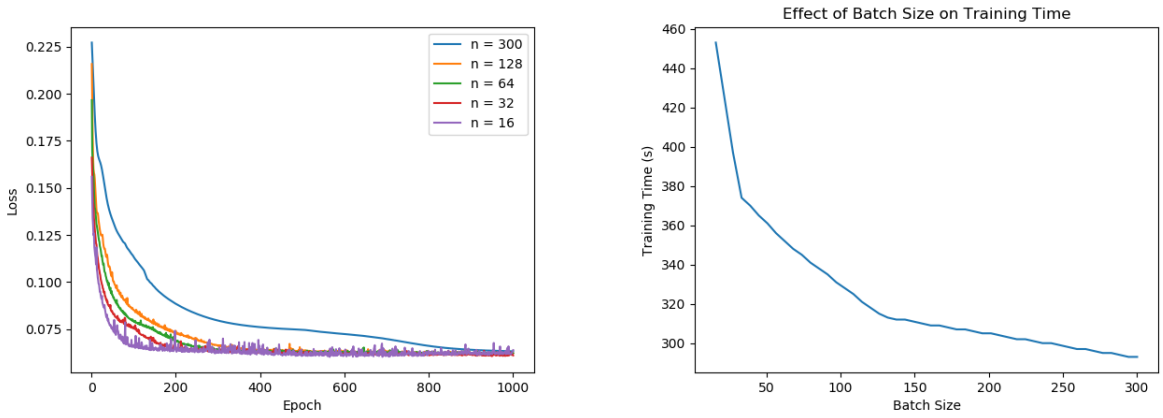
Figure 3.5: Rate of convergence of different optimization algorithms

This comparison is made using a model configured according to Table 3.1 - 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 [28].

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

Table 3.1: Model configuration for comparison of optimization algorithms

Batch size. The batch size used can affect the rate of convergence even though the loss eventually converges to the same point - Figure 3.6 shows this through the training curves of models trained with different batch sizes (n in the legend refers to the batch size). 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 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.



(a) Training curves for different batch sizes

(b) Effect of batch size on training time

Figure 3.6: Effect of different batch sizes

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 3.6 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 [29]. 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.

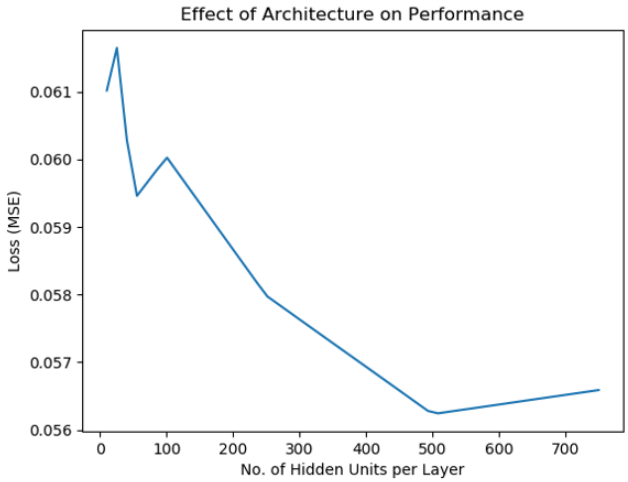


Figure 3.7: Effect of increasing size of hidden layer on performance

While attempting to increase the number of units with a single hidden layer, the expected trend of better performance with an increased hidden layer size was observed. An extension of this logic is to add hidden layers themselves, starting with a second hidden layer. Figure 3.8 shows a notable improvement in performance when doing so - in fact, there is a more discernible impact of adding hidden units to a layer when there are two layers. The likely reason for this is that the second hidden layer is a function of the first hidden layer, hence each node of the second hidden layer is itself a complex nonlinear function which contributes more significantly to the overall complexity and the learning capacity of the network than a node of the first layer.

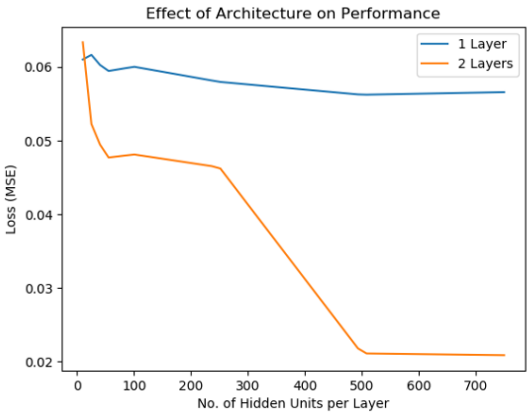


Figure 3.8: Effect of adding a second hidden layer on performance

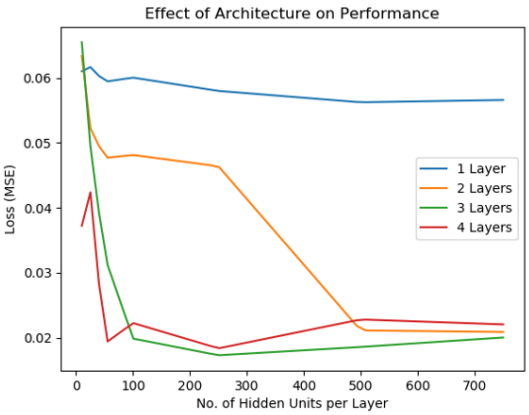


Figure 3.9: Effect of adding a third and a fourth hidden layer on performance

It is apparent from Figure 3.9 that the returns are diminishing if we keep increasing both the number of hidden layers and the number of units per layer. In fact, the curves suggest that the increasing network complexity will eventually hurt the performance of the network, i.e., overfitting would have occurred in the training of overly expressive models. For this reason, we chose the architecture of 4 hidden layers and 50 units per layer for performance evaluation.

3.2.2. Results of Feedforward Neural Network model

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

Table 3.2: Configuration and performance metrics of tuned FFNN model

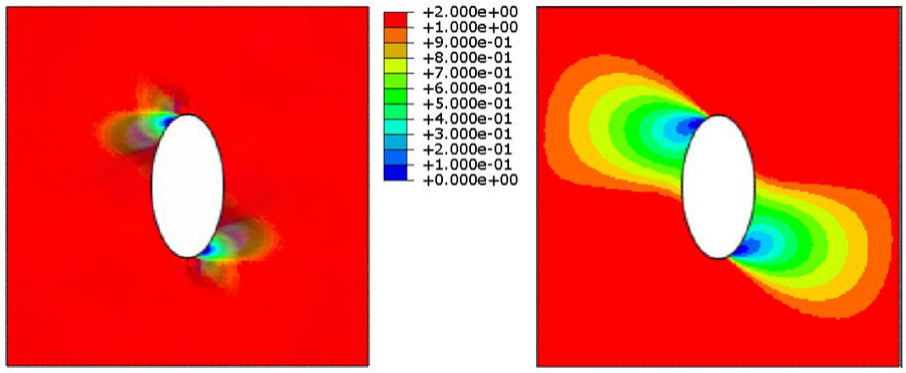


Figure 3.10: Output of tuned FFNN model (left) compared to the reference (right)

The results of the tuned FFNN model show clear signs of overfitting - it performs significantly better on the training set than it does on the validation set and the test example. Visually speaking, the tuned model is able to reproduce the dominant colour of the pattern accurately. To a small extent, it is able to predict the damage patterns near the hole as well. However, it is still clearly far from the reference.

3.3. Convolutional Neural Network - Autoencoder

A Convolutional Neural Network (CNN) is a special class of neural networks that is most commonly used to extract features from images using various 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. As an example, Figure 3.11 shows the sequence of convolutional filters and pooling operations used to extract features from the image, which reduces a $32 \times 32 \times 3$ image to a latent feature space of size 768, before passing it onto a fully connected neural network for binary classification. The size of the latent feature space here is a quarter of that of the original image.

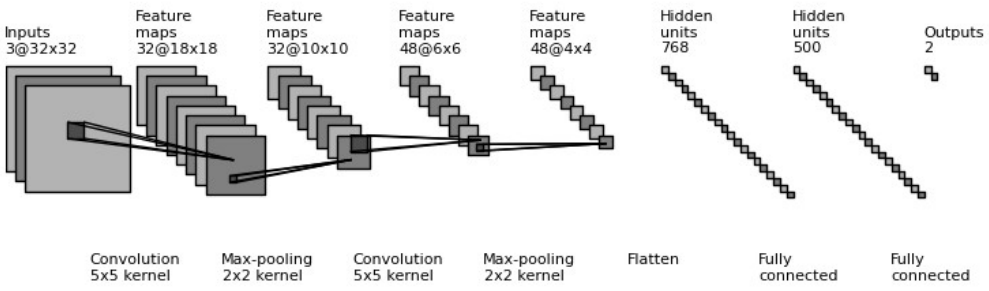


Figure 3.11: Architecture of an example CNN (figure credit: `github.com/gwding/draw_convnet`)

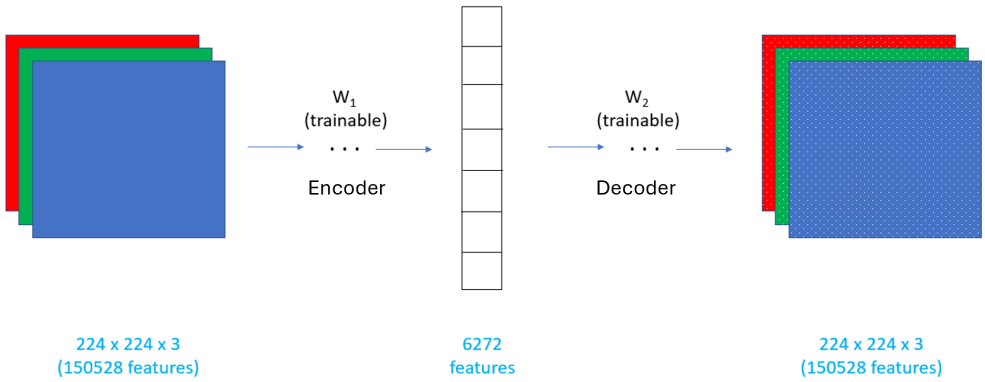


Figure 3.12: Architecture of a convolutional autoencoder

Once we reduce the image to a vector in the latent feature space, we can then train an autoencoder network to generate this vector given the inputs, and regenerate the image from the vector, as shown in Figure 3.12. 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 learned in the training process. Thus, both the weights that reduce an image to a feature vector, and those that regenerate the image from the feature vector, will need to be determined. The loss function is the MSE between the true image fed into the convolution layers and the regenerated image out from the deconvolution layers (note that the inputs regarding loading, ply and damage mode do not play a role in the training of this autoencoder).

3.3.1. Hyperparameter Tuning

As was the case with the FFNN in Section 3.2.1, hyperparameter tuning is also carried out with the CNN to obtain the optimal configuration.

Feature Vector Size. The size of the latent feature space depends on the sequence of filtering and pooling operations. The influence of latent space size on the training loss is shown in Figure 3.13. It can be observed that the loss drops to a minimum, before it starts to rise again. The loss initially drops when convolutional layers are added, because they ensure that the network is actually learning patterns. However, continuing to add layers beyond a certain point increases the error between the true image and the regenerated image due to the loss of information in the process of reducing the image to a small number of features. The optimal size is the one that balances the two considerations.

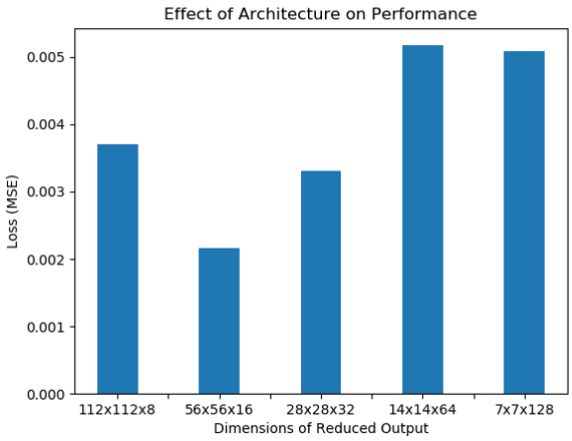


Figure 3.13: Convolution-Deconvolution performance for different feature vector sizes

Despite the optimal feature size appearing to be $56 \times 56 \times 16 = 50176$, it is deemed that the increase in error observed for a feature space size of $7 \times 7 \times 128 = 6272$ is small enough at this stage to be acceptable in lieu of the benefit of a much reduced feature space.

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

Table 3.3: Model configuration for analysing hidden layer architecture

Upsampling & Deconvolution. The configuration of the model used to perform this analysis is given in Table 3.3, and the test image regenerated with this model is displayed in Figure 3.14. 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. This is a typical drawback of the standard deconvolutional operation which employs a fractional stride, resulting in an uneven overlap between pixels [30]. This can be improved by using upsampling before deconvolution (of stride 1). Upsampling is a technique that scales an image through nearest-neighbour extrapolation. Figure 3.15 shows this improvement in image quality, and its ability to regenerate some of the features observed in the true image. In the remainder of this work, the term 'deconvolution' is used to refer to both upsampling and deconvolution as a whole.

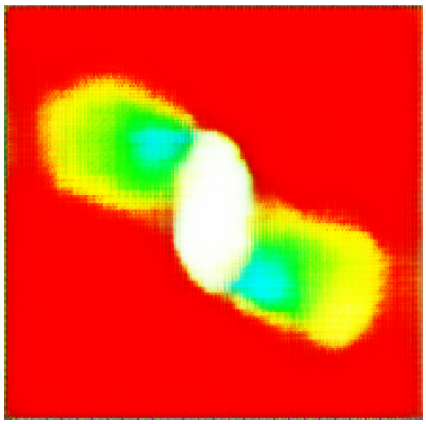


Figure 3.14: Test image regenerated with standard deconvolution

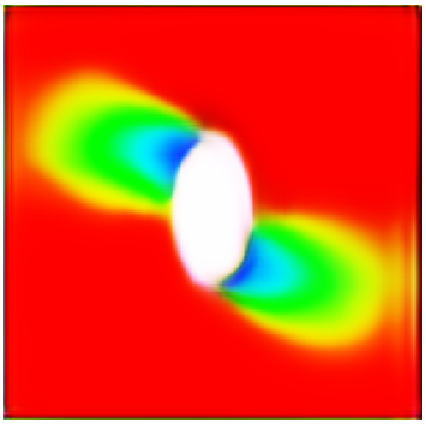


Figure 3.15: Test image regenerated with upsampling improvement

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 helps make the results invariant to translations or rotations within the image, while average pooling is used almost exclusively to reduce size [31]. Based on an MSE of less than 0.006 with average pooling and more than 0.010 with max pooling, the former is selected as the pooling technique for subsequent study.

3.3.2. Results of the convolutional autoencoder

Table 3.4 shows the final configuration and performance metrics for reducing the image to a feature vector of size 6272 through the convolutional autoencoder, 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 3.16.

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

Table 3.4: Configuration and performance metrics of the trained CNN

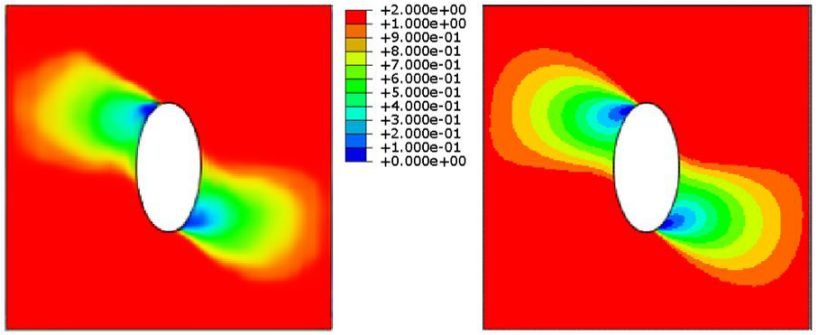


Figure 3.16: Output of the trained CNN model (left) compared to the reference (right)

The test error here is much smaller than that of the FFNN. Visually speaking, it is also apparent that this image is much closer to the true test image than the one generated by the Feedforward Neural Network (seen in Figure 3.10). 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.

3.4. Reduced-Image Neural Network

This section introduces the input parameters (i.e., loading, ply number and damage mode) of the model to it, and follows a methodology similar to Section 3.2 to train what is termed the Reduced-Image Neural Network (RINN), as illustrated in Figure 3.17.

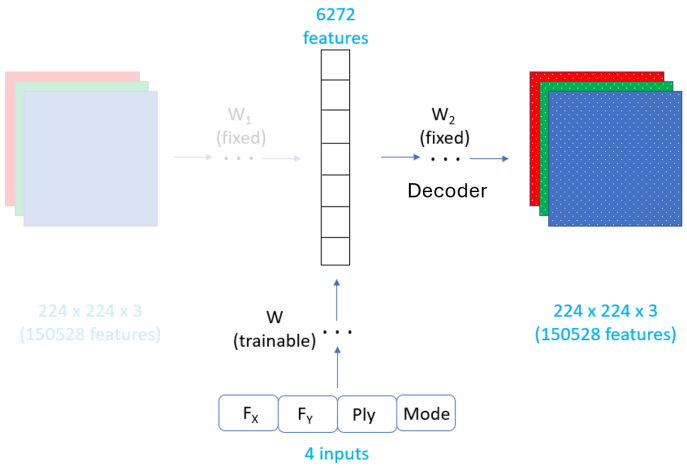


Figure 3.17: Architecture of the RINN. The encoding part of the CNN is grayed out as it is not used.

The first step is creating the suitable dataset, which is composed of the inputs and the feature vectors of the corresponding damage images. To carry this out, each image in the original dataset is reduced to its feature vector using the previously trained convolutional autoencoder.

Subsequently, a FFNN is trained to take the input parameters and output the corresponding feature vector. The hyperparameter tuning process presented in Section 3.2.1 is followed here. The loss function is applied between the true image, and the image generated by the predicted feature vector using the previously trained autodecoder. During training, the parameters of the FFNN are optimized to minimize the MSE loss. The configuration and performance metrics of the optimized model are given in Table 3.5.

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

Table 3.5: : Configuration and performance metrics of tuned RINN model

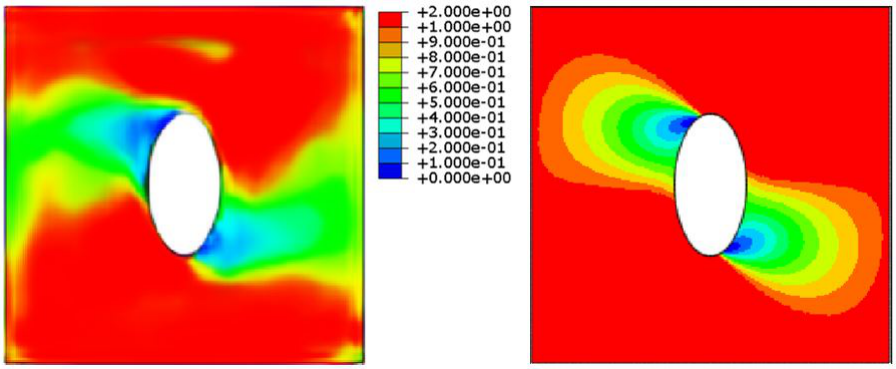


Figure 3.18: Output of tuned RINN model (left) compared to the reference (right)

The performance on the test example is shown in Figure 3.18. The prediction is clearly poorer than that by the CNN. The errors associated with going from input parameters to the feature vector seem to magnify when going further from the feature vector to the image output. The training error in Table 3.5 is consistent with that in the case of CNN in Table 3.4, but the validation and test errors here are much higher. On visual comparison as shown in Figure 3.18, we observe that the predicted patterns emerging from the hole roughly extend over the correct regions, while its colour and form are inaccurate.

Compared with the numerical results of the FFNN model in Table 3.2, the RINN model shows a much smaller training error with similar levels of validation and testing errors. Visually speaking, the RINN's prediction in Figure 3.18 shows richer features than that of the FFNN model in Figure 3.10. Therefore, the RINN model is chosen for subsequent training on a larger dataset to reduce validation and testing errors.

4. Improved Dataset and Models

4.1. Expanded dataset

The dataset used to train the neural networks discussed in the previous section 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 reduce overfitting, hence performing better on the validation and testing sets.

About 80% of this dataset (5600 images) is used as the training set, and just over 10% (800 images) as the validation set. The remaining 656 images are set as test images, which includes the image in Figure 3.2. Generating all datasets took just over 100,000 seconds (close to 28 hours) of computation time without counting the time spent on user-intervention.

4.2. Convolutional Neural Network

With the enlarged dataset, the CNN is trained from scratch following the same processes applied in Section 3.3, and the hyperparameter processes are repeated in full again. The model configuration is shown in Table 4.1 along with its performance metrics. The output using the test example is shown in Figure 4.1. A low and nearly identical training and validation loss indicates that the model is neither underfitting nor overfitting. Again, it must be kept in mind that this is merely for the regeneration of the input image.

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

Table 4.1: Configuration and performance metrics of CNN model tuned and trained with expanded dataset

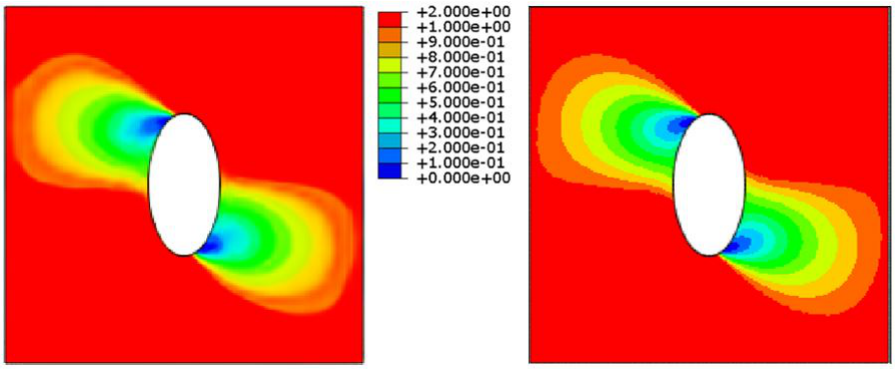


Figure 4.1: Output of CNN model tuned and trained with expanded dataset (left) compared to the reference (right)

4.3. Reduced-Image Neural Network

The RINN model proposed in Section 3.4 is retrained on the expanded dataset. The hyperparameter tuning process presented in Section 3.2.1 is followed here. The model configuration obtained with hyperparameter tuning, as well as the performance metrics from the optimized network are shown in Table 4.2. The output using the test example is shown in Figure 4.2.

As expected, the losses are greater than that in the CNN because generating the image is now a two-step process. The training and validation errors are now very close. The validation and testing errors are now an order of magnitude smaller than the previous ones in Table 3.5. Visually speaking, the model is able to reproduce the colours and patterns from the reference quite accurately. There is, however, some noise, as visible from the unexpected pattern towards the bottom-left of the plate. The predictions of the RINN model on additional reference images are shown in Appendix A.1. Although the first of those images, a relatively simple pattern, is accurately reproduced, the other two predictions do indicate the presence of noise near the domain boundary. We suspect that the reason for this noise is due to the fixed parameters in the deconvolution operations on the feature vector, which were previously optimized for the CNN rather than the current RINN.

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

Table 4.2: Configuration and performance metrics of RINN model

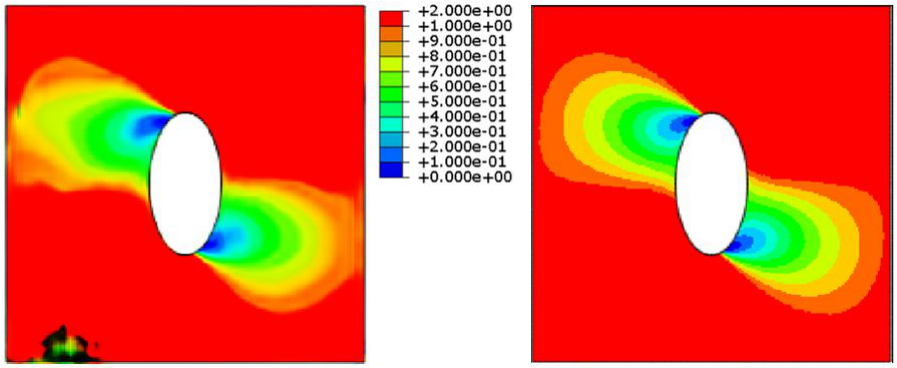


Figure 4.2: Output of RINN model tuned and trained with expanded dataset (left) compared to the reference (right)

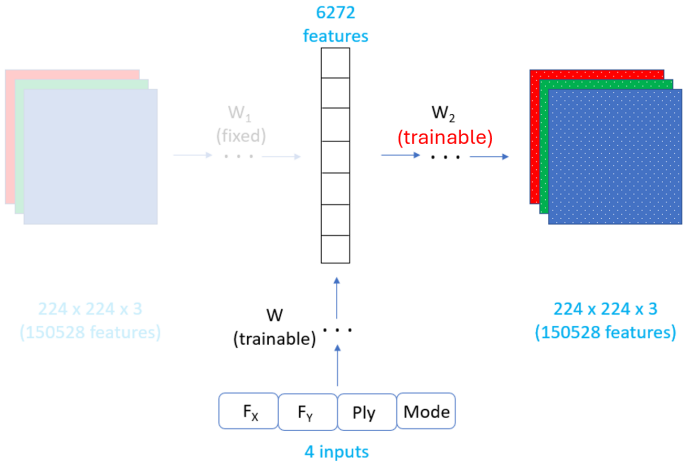


Figure 4.3: Architecture of the Hybrid Neural Network. The encoding part of the CNN is greyed out as it is not used.

In order to reduce the error of the network due to deviations in the predicted feature vector, weights used in the deconvolution process are changed from fixed to trainable, as shown in Figure 4.3. This makes the first half of the network, going from the inputs to 'feature vectors', a FFNN, and the second part, going from the feature vector to the image, a convolutional decoder, and is therefore termed a Hybrid Neural Network (HNN). The phrase *feature vectors* is within inverted commas because the network now learns their values afresh, which may not be the same as those learned in the CNN model. The CNN values now serve for initialization. Training of the HNN is carried out in a similar manner as before. Tuning of the hybrid network now includes the FFNN part, the latent feature space, and the convolutional decoder part. The optimized parameters obtained from the tuning process, apart from those pertaining to network architecture, are given in Table 4.3. The optimized architecture is tuned and evaluated for four different feature vector sizes. Their performance metrics are shown in Table 4.4. The predicted damage patterns of the four cases are shown in Figures 4.4 to 4.7.

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

Table 4.3: Model configuration (excluding architecture) for fine-tuning hybrid network

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

Table 4.4: Model configurations & performance metrics for four different sizes of latent feature space

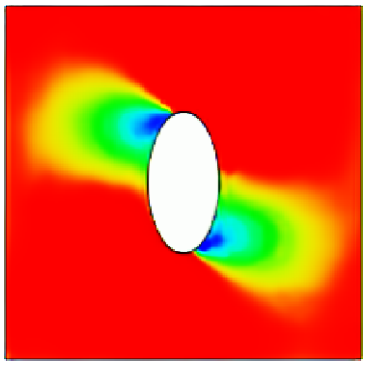


Figure 4.4: Hybrid network output for feature vector size 6272

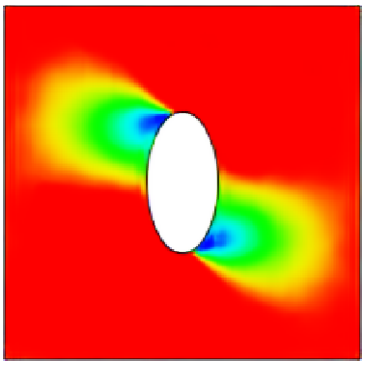


Figure 4.5: Hybrid network output for feature vector size 12544

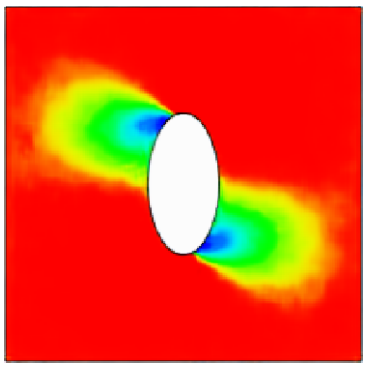


Figure 4.6: Hybrid network output for feature vector size 25088

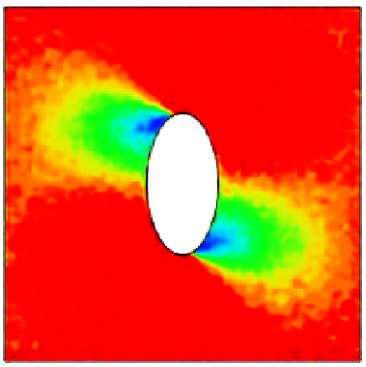


Figure 4.7: Hybrid network output for feature vector size 50176

Similar to what we have observed before in Section 3.3, the performance of the network reaches a peak and subsequently drops again while increasing the feature vector size. A visual inspection also reveals that a feature vector size of 12544 yields the best results. Further damage pattern predictions from this model are provided in Appendix A.2. The performance metrics show that the HNNs on all four configurations are predicting more accurately than the RINN model.

Now we have obtained a trained HNN model which can predict damage patterns with a high degree of accuracy measured by the MSE loss. However, MSE is not the only metric for assessing image quality. Different metrics could be needed for different purposes. The HNN could then be trained on these different metrics to give predictions for different purposes. In the following section, another metric is used on the HNN to test its performance in this respect.

4.5. Hybrid Neural Network with Structural Similarity Index

4.5.1. Structural Similarity Index (SSIM))

SSIM is sometimes preferred over MSE for assessing 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 by comparing features of the image locally [32] - this is carried out by incorporating the luminance l, contrast c and structure comparison s to give the SSIM value S as:

$$S(x, y) = f(l(x, y), c(x, y), s(x, y)) \tag{4.1}$$

x and y here refer to the two images that are to be compared. The result 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 4.4 to Figure 4.7, 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 in this case. The SSIM is implemented in the next subsection.

4.5.2. Training HNN using SSIM

The HNN is retrained as before, except that the loss metric is now SSIM instead of MSE. Since a higher SSIM value is desirable, $(1 - \text{SSIM})$ is used as the cost function to be minimized.

The training curve and the test output are shown in Figure 4.8. Clearly, these results suggest that the model is not learning anything. There is an initial drop in the first epoch due to the gradient associated with random initialization, 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 when hyperparameters are tuned on a wide range of values.

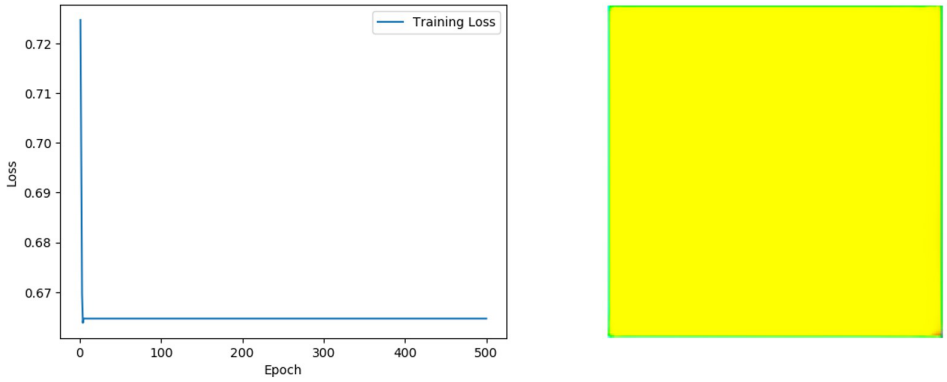


Figure 4.8: Learning curve (left), and output image (right) upon training with SSIM

In order to give a chance for gradient descent to focus sufficiently on all three components of SSIM, a weight initialization that allows the process to start with a relatively low loss could prove useful. In order to do this, the hybrid model is initialized with the weights obtained from the earlier model when trained using MSE. In this sense, we are refining an MSE-based pre-trained model using SSIM, while keeping the network architecture fixed. The model configuration and the SSIM values are shown in Table 4.5. The output generated with this model is shown in Figure 4.9.

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
Optimization 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 4.5: Configuration and performance metrics of SSIM-based tuned hybrid neural network

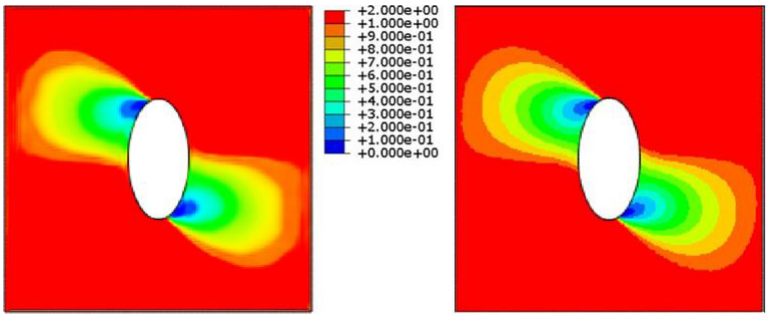


Figure 4.9: Output of SSIM-tuned tuned hybrid network (left) compared to the reference (right)

Initializing the SSIM-based models with pre-trained weights leads the image to converge towards a much-improved result as compared to that obtained through random initialization. As expected, using SSIM as a training metric results in an output with improved visual sharpness - notice how the test image has its SSIM value improve from 0.9039 to 0.9418. Unsurprisingly, with the objective of maximize SSIM, the focus has been taken away from MSE, leading to an increase in MSE loss values, as shown in Table 4.6. Similar performance is observed on the additional test cases shown in Appendix A.

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

Table 4.6: MSEs on model trained using SSIM

4.6. Reliability of Trained Model

The measure of the performance of the model should be obtained using a completely independent dataset that played no role in the training process, which discounts even the validation set. The chosen test image used throughout the study is one such 'set' that fits this criterion. As introduced in Section 4.1, there are 655 additional test images which can be used to evaluate the performance of the model. The mean MSE and SSIM values on this expanded testing set are reported in Table 4.7.

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

Table 4.7: Mean MSE and SSIM values of the Hybrid NN model evaluated on an independent dataset of 655 images

The distribution of errors are as important as the mean - nearly 300 of the 655 examples yield an error of less than 0.0002, while the maximum error is 0.0112. This is represented in the cumulative density plot shown in the right half of Figure 4.10 - 100% of the values lie below 0.0112, while approximately 95% of

the errors are within the 0.0040 mark. This is also shown for SSIM in the left half - all images have an SSIM value greater than 0.8500, and approximately 95% of them are over 0.9100. These results confirm the performance and reliability of the model on unseen examples.

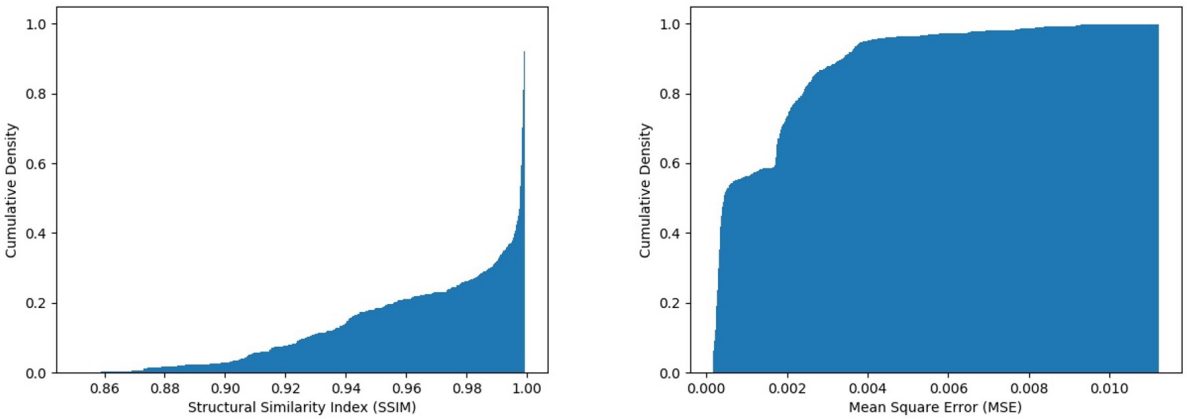


Figure 4.10: Cumulative densities of SSIM values (left), and that of MSE values (right)

4.7. Computation Costs

The neural network models developed in this work were optimized purely on the basis of performance metric values. However, one of the major motivating factors for building a ML surrogate was to explore the computational speed-up over the high-fidelity FEM model when a prediction is generated. The prediction time on a complete analysis using the HNN, ie, for the generation of 16 damage patterns, is found to be 6.6 seconds on average with negligible spread. This last detail suggests that the complexity of the underlying FEM model is not a factor that influences the prediction time of the surrogate. This is unsurprising as every input is processed in exactly the same way in the neural network. In FEM models, more iterations are required to reach convergence on more complex problems. The FEM models in Sections 2 and 4 took an average of 227 seconds to complete, making it 34 times longer than the HNN model. In fact, more complex cases took even longer, such as the problem analysed in Section 1, where the computation took 2927 seconds, or 443 times longer than the HNN model.

With all the gains in prediction times, it must be recognised that the network needs to be trained for 76225 seconds, or just over 21 hours. The generation of data requires another 28 hours. This is a worthwhile investment only when the model is reused a substantial number of times, e.g., the model represents a modular substructure used extensively in higher-level designs. 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 return on investment of building a surrogate model.

5. Conclusions

In this work, we formed and tested different neural networks for the surrogate modelling of an open-hole composite plate. To the authors' best knowledge, this is the first work where ML surrogate models are built to output the damage patterns of composite laminates. Data were generated through nonlinear FEM models for composites damage as implemented in Abaqus. A smaller dataset was firstly generated to test the performance of different neural networks, namely the standard FFNN and the RINN which connects layers of FFNN with the decoder part of a pretrained convolutional autoencoder. The RINN model was shown to have smaller training errors and give richer predictions of damage patterns than the FFNN model. The RINN model was then trained on a larger dataset, showing noticeable improvements on prediction accuracy, but suffered from noises in the predictions near the outer boundary of the domain. To further increase its accuracy, a HNN model has been proposed. Similarly to RINN, the HNN also connects layers of FFNN with the decoder part of a pretrained convolutional autoencoder. However, when training the HNN, the decoder is retrained instead of being fixed as in the RINN model. It was shown that this retraining of the decoder substantially improved both the training and testing performance. The HNN has been shown to accurately and efficiently predict the damage patterns of the open-hole laminate, thereby constituting a promising candidate for the surrogate modelling of open-hole composite panels.

To quantify the resemblance between the predicted and actual outputs in terms of colours and contours, different performance metrics have been explored. The use of the Structural Similarity Index (SSIM), in addition to the standard MSE, was explored. The model was 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 minimum during training, the model's weights need to be pre-trained first with MSE loss. With both MSE and SSIM losses, the HNN's performance on a large test set shows a high level of accuracy with small scatter.

The HNN surrogate is on average 34 times faster than the underlying FEM model. On the more complex problems, the speed-up can be 443 times. However, the data generation process took 28 hours and the network training took 21 hours. Therefore, a cost-benefit analysis should be carried out to determine whether such an investment of resources would be beneficial. It is likely to be so when the trained surrogate model will be used frequently, e.g., when it represents a modular unit repeated extensively in a larger system. Last but not least, the trained model should be employed within the training boundaries for inputs to avoid the dangers of extrapolation.

Appendix A. Model predictions on enlarged datasets

Appendix A.1. Reduced Image Neural Network

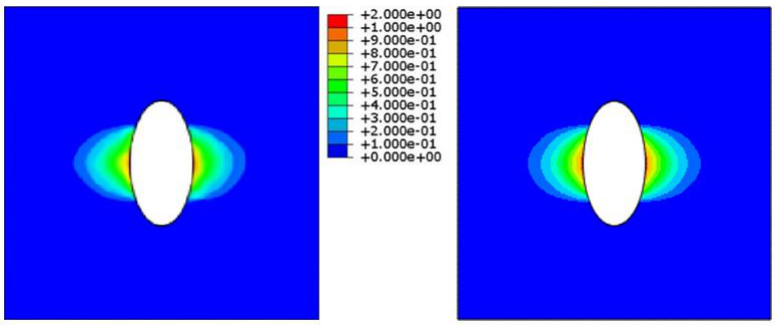


Figure A.1: Output of RINN model for an input (0.75, 1.00, 4, 1) (left) compared to the true damage pattern (right)

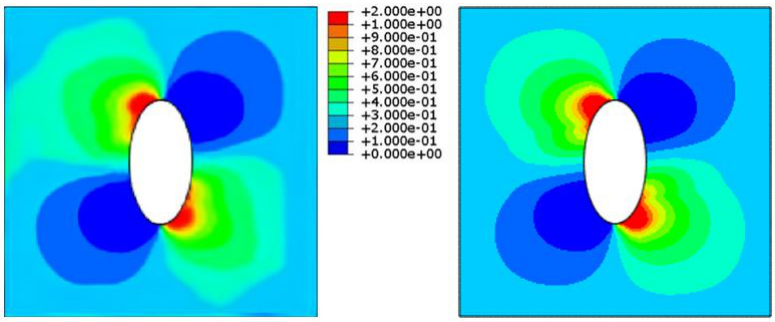


Figure A.2: Output of RINN model for an input (0.25, 0.75, 1, 2) (left) compared to the true damage pattern (right)

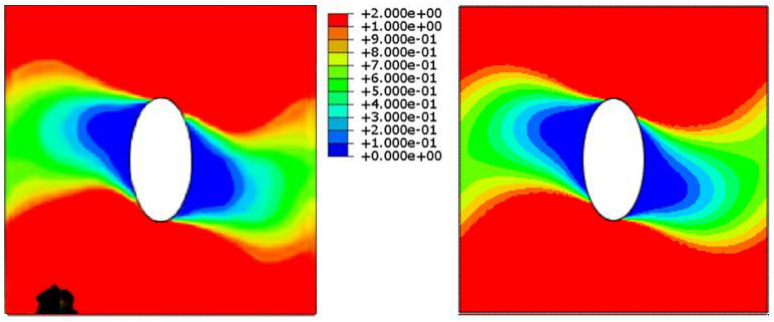


Figure A.3: Output of RINN model for an input (0.50, 0.00, 1, 4) (left) compared to the true damage pattern (right)

Appendix A.2. Hybrid Neural Network

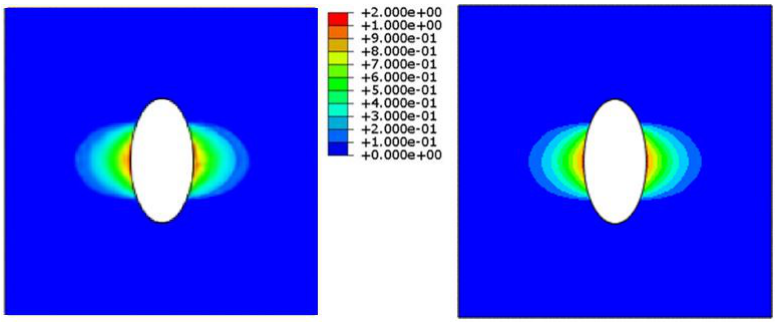


Figure A.4: Output of tuned hybrid network for an input (0.75, 1.00, 4, 1) (left) compared to the true damage pattern (right)

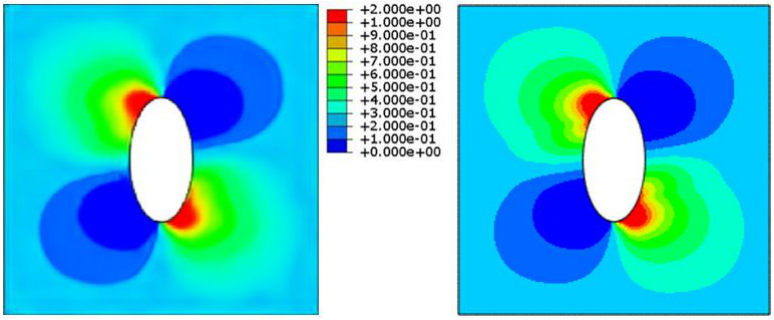


Figure A.5: Output of tuned hybrid network for an input (0.25, 0.75, 1, 2) (left) compared to the true damage pattern (right)

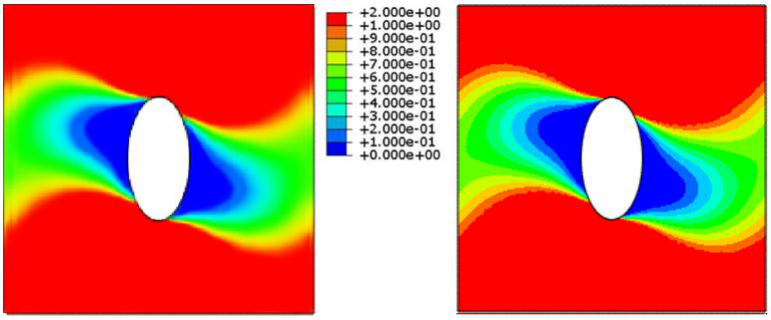


Figure A.6: Output of tuned hybrid network for an input (0.50, 0.00, 1, 4) (left) compared to the true damage pattern (right)

Appendix A.3. Hybrid Neural Network Trained with SSIM

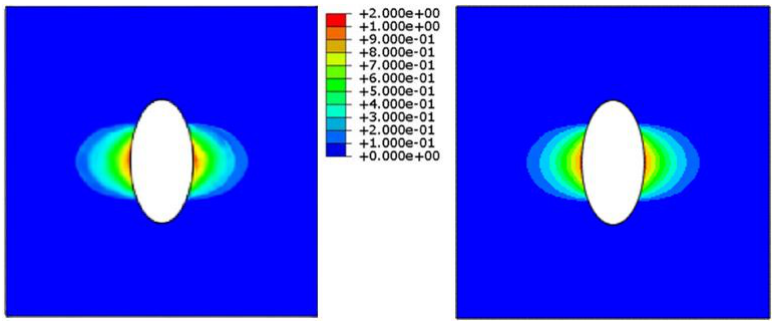


Figure A.7: Output of SSIM-tuned hybrid network for an input (0.75, 1.00, 4, 1) (left) compared to the true damage pattern (right)

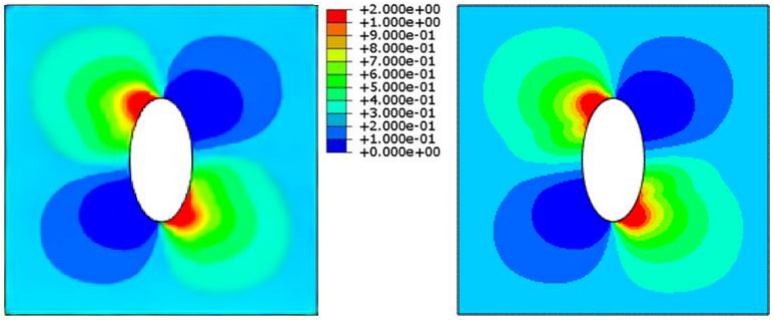


Figure A.8: Output of SSIM-tuned hybrid network for an input (0.25, 0.75, 1, 2) (left) compared to the true damage pattern (right)

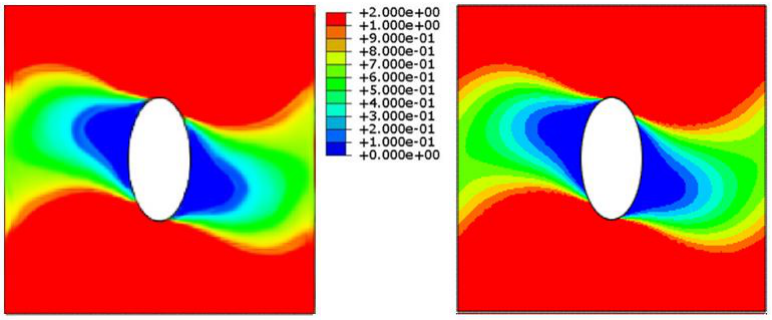


Figure A.9: Output of SSIM-tuned hybrid network for an input (0.50, 0.00, 1, 4) (left) compared to the true damage pattern (right)

Appendix B. Data availability statement

The data that support the findings of this study are openly available at `https://github.com/k-venkatesan/composite-damage-prediction`

References

[1] S. Hallett, W. Jiang, M. Wisnom, An experimental and numerical investigation into the damage mechanisms in notched composites, Composites Part A: Applied Science and Manufacturing 40 (5) (2009) 613–624.

[2] F. Van der Meer, L. Sluys, S. Hallett, M. Wisnom, Computational modeling of complex failure mechanisms in laminates, Journal of Composite Materials 46 (5) (2012) 603–623.

[3] M. J. Swindeman, E. V. Iarve, R. A. Brockman, D. H. Mollenhauer, S. R. Hallett, Strength prediction in open hole composite laminates by using discrete damage modeling, AIAA journal 51 (4) (2013) 936–945.

[4] Q. Yang, D. Schesser, M. Niess, P. Wright, M. Mavrogordato, I. Sinclair, S. Spearing, B. Cox, On crack initiation in notched, cross-plied polymer matrix composites, Journal of the Mechanics and Physics of Solids 78 (2015) 314–332.

[5] B. Chen, T. Tay, S. Pinho, Modelling the tensile failure of composites with the floating node method, Computer Methods in Applied Mechanics and Engineering 308 (2016) 414–442.

[6] O. Falcó, R. Ávila, B. Tijs, C. Lopes, Modelling and simulation methodology for unidirectional composite laminates in a virtual test lab framework, Composite Structures 190 (2018) 137–159.

[7] S. Yan, X. Zou, M. Ilkhani, A. Jones, An efficient multiscale surrogate modelling framework for composite materials considering progressive damage based on artificial neural networks, Composites Part B: Engineering 194 (2020) 108014.

[8] I. B. Rocha, P. Kerfriden, F. van Der Meer, On-the-fly construction of surrogate constitutive models for concurrent multiscale mechanical analysis through probabilistic machine learning, Journal of Computational Physics: X 9 (2021) 100083.

[9] B. El Said, Predicting the non-linear response of composite materials using deep recurrent convolutional neural networks, International Journal of Solids and Structures 276 (2023) 112334.

[10] T. Gulikers, An integrated machine learning and finite element analysis framework, applied to composite substructures including damage, Master's thesis, Delft University of Technology (2018).

[11] C. Furtado, L. Pereira, R. P. Tavares, M. Salgado, F. Otero, G. Catalanotti, A. Arteiro, M. A. Bessa, P. P. Camanho, A methodology to generate design allowables of composite laminates using machine learning, International Journal of Solids and Structures 233 (2021) 111095.

[12] Y. Li, H. Qin, V. Tan, L. Jia, Y. Liu, A deep transfer learning approach to construct the allowable load space of notched composite laminates, Composites Science and Technology 247 (2024) 110432.

[13] O. A. Imran Azeem, S. T. Pinho, A physics-informed machine learning model for global-local stress prediction of open holes with finite-width effects in composite structures, Journal of Composite Materials 58 (23) (2024) 2501–2514.

[14] G. T. Balducci, B. Chen, M. Möller, M. Gerritsma, R. De Breuker, Predicting open-hole laminates failure using support vector machines with classical and quantum kernels, arXiv preprint arXiv:2405.02903 (2024).

[15] X. Liu, S. Tian, F. Tao, W. Yu, A review of artificial neural networks in the constitutive modeling of composite materials, Composites Part B: Engineering 224 (2021) 109152.

[16] A. Sharma, T. Mukhopadhyay, S. M. Rangappa, S. Siengchin, V. Kushvaha, Advances in computational intelligence of polymer composite materials: machine learning assisted modeling, analysis and design, Archives of Computational Methods in Engineering 29 (5) (2022) 3341–3385.

[17] C. Nelon, O. Myers, A. Hall, The intersection of damage evaluation of fiber-reinforced composite materials with machine learning: A review, Journal of Composite Materials 56 (9) (2022) 1417–1452.

[18] Y. Wang, K. Wang, C. Zhang, Applications of artificial intelligence/machine learning to high-performance composites, Composites Part B: Engineering (2024) 111740.

[19] M. Liu, H. Li, H. Zhou, H. Zhang, G. Huang, Development of machine learning methods for mechanical problems associated with fibre composite materials: A review, Composites Communications (2024) 101988.

[20] J. Loh, K. Yeoh, K. Raju, V. Pham, V. Tan, T. Tay, A review of machine learning for progressive damage modelling of fiber-reinforced composites, Applied Composite Materials (2024) 1–38.

[21] R. F. Ribeiro Junior, G. F. Gomes, On the use of machine learning for damage assessment in composite structures: a review, Applied Composite Materials 31 (1) (2024) 1–37.

[22] S. S. Sorour, C. A. Saleh, M. Shazly, A review on machine learning implementation for predicting and optimizing the mechanical behaviour of laminated fiber-reinforced polymer composites, Heliyon (2024).

[23] R. Sepasdar, A. Karpatne, M. Shakiba, A data-driven approach to full-field nonlinear stress distribution and failure pattern prediction in composites using deep learning, Computer Methods in Applied Mechanics and Engineering 397 (2022) 115126.

[24] D. P. Mandic, Y. Bengio, A generalized normalized gradient descent algorithm, IEEE Signal Processing Letters 11 (2) (2004) 115–118. doi:10.1109/LSP.2003.821649.

[25] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, COLT 2010 - The 23rd Conference on Learning Theory (2010) 257–269.

[26] M. D. Zeiler, Adadelta: An adaptive learning rate method, Tech. Rep. Available at: https://arxiv.org/abs/1212.5701 (2012).

[27] Y. Nesterov, A method for unconstrained convex minimization problem with the rate of convergence o(1/k2), Doklady ANSSR 27 (2) (1983) 543–547.

[28] TensorFlow, Tensorflow guide, Tech. Rep. Available at: https://www.tensorflow.org/guide/.

[29] S. Maleki, Y. Gao, M. J. Garzarán, T. Wong, D. A. Padua, An evaluation of vectorizing compilers, Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT (2011) 372–382doi:10.1109/PACT.2011.68.

[30] J. Gauthier, Conditional generative adversarial nets for convolutional face generation, Tech. Rep. Available at: http://cs231n.stanford.edu/reports/2015/pdfs/jgauthie_final_report.pdf (2014).

[31] D. Ciresan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2012). doi:10.1109/CVPR.2012.6248110.

[32] Z. Wang, A. Bovik, H. Sheikh, E. Simoncelli, Image quality assessment: From error measurement to structural similarity, IEEE transactions on image processing 13 (4) (2004) 600–612.