University of Duisburg-Essen
Chair of Dynamics and Control
Univ.-Prof. Dr.-Ing. Dirk Söffker

Project work

Image Classification - MNIST
Regression - CFRP NASA

Mohamed Ibrahim
Matrikelnummer: 315605

Advisor

Jonathan Liebeton, M.Sc.
Univ.-Prof. Dr.-Ing. Dirk Söffker

September 2023

Versicherung an Eides Statt

Ich versichere an Eides statt durch meine untenstehende Unterschrift,

- dass ich die vorliegende Arbeit - mit Ausnahme der Anleitung durch die Betreuer - selbstständig ohne fremde Hilfe angefertigt habe und
- dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus fremden Quellen entnommen sind, entsprechend als Zitate gekennzeichnet habe und
- dass ich ausschließlich die angegebenen Quellen (Literatur, Internetseiten, sonstige Hilfsmittel) verwendet habe und
- dass ich alle entsprechenden Angaben nach bestem Wissen und Gewissen vorgenommen habe, dass sie der Wahrheit entsprechen und dass ich nichts verschwiegen habe.

Mir ist bekannt, dass eine falsche Versicherung an Eides Statt nach §156 und nach §163 Abs. 1 des Strafgesetzbuches mit Freiheitsstrafe oder Geldstrafe bestraft wird.

Ort, Datum

Unterschrift

Urheberrechtsvereinbarung

Die vorliegende Arbeit entstand unter intensiver Mitwirkung der genannten, betreuenden Personen im Rahmen von Forschungsarbeiten im Lehrstuhl Steuerung, Regelung und Systemdynamik (SRS) der Universität Duisburg-Essen. Die in dieser Arbeit enthaltenen Lösungen und Ideen unterliegen daher nicht nur dem Urheberrecht der zu qualifizierenden Person, sondern dem genannten Personenkreis gemeinsam. Die persönliche, private sowie die Nutzung im Forschungskontext des Lehrstuhls SRS ist hiervon unbenommen.

Ort, Datum

Unterschrift

Contents

List of Figures

List of Tables

1 Image Classification - MNIST

Handwritten digit recognition has been a focal point in the field of machine learning and computer vision for several decades. The MNIST dataset, a collection of grayscale images of handwritten digits, has emerged as a canonical benchmark for evaluating various classification algorithms [LBBH98]. In this study, we harness two distinct classification techniques: Convolutional Neural Networks (CNN), which have recently proven to be highly effective for image-related tasks [GBC16], and the traditional k-Nearest Neighbors (k-NN) algorithm, a mainstay in pattern recognition [JWHT13]. By comparing the performance of these methodologies, we seek to elucidate the strengths and potential limitations inherent to each approach. Our findings contribute to the broader discourse on algorithmic trade-offs between computational complexity, interpretability, and predictive accuracy in the realm of digit recognition tasks.

1.1 Environment Setup

The initialization segment of the code sets up the MATLAB environment. To harness the computational power of multi-core processors, a parallel pool is initiated [Mat22].

```
clear all;
close all;
clc;
if isempty(gcp('nocreate'))
    parpool;
end
```

1.2 Data Loading and Preprocessing

Post-environment setup, the MNIST dataset, which comprises handwritten digits, is loaded. This dataset has extensively been used in the realm of machine learning and computer vision for training and benchmarking [LBBH98].

```
data = load('mnist.mat');
training_set = data.training;
test_set = data.test;
train_Labels = gpuArray(training_set.labels);
test_Labels = gpuArray(test_set.labels);
```

To streamline the representation for the algorithms, each image is transformed into a vector. Post-transformation, the reshaped data is relayed to the GPU for accelerated computation [Mat22].

```
trainData = gpuArray(reshape(training_set.images, [], size(
    training_set.images, 3))');
testData = gpuArray(reshape(test_set.images, [], size(
    test_set.images, 3))');
```

1.3 Image Classification - MNIST: CNN

The Convolutional Neural Network (CNN) architecture is defined. It comprises Convolution-BatchNorm-ReLU blocks, MaxPooling layers, a fully connected layer, a softmax layer, and a classification layer.

```
lgraph = layerGraph([ ... ]);
```

1.4 CNN Architecture Definition

```
%% Define CNN architecture using layerGraph
lgraph = layerGraph([
    imageInputLayer([28 28 1])
    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer]);
```

The CNN starts with an input layer designed for images of size 28×28 pixels with a single channel (grayscale). This is followed by alternating convolutional layers (with batch normalization and ReLU activation) and max-pooling layers. The CNN concludes with a fully connected layer, a softmax layer, and a classification layer[CMS12].

1.5 Training Configuration

```
options = trainingOptions('adam', ...
    'MaxEpochs', 10, ...
    'InitialLearnRate', 0.001, ...
    'MiniBatchSize', 128, ...
    'ValidationData', {testImages, categorical(testLabels)}, ...
    'ValidationFrequency', 20, ...
    'Verbose', true, ...
    'Plots', 'training-progress', ...
    'ExecutionEnvironment', 'parallel', ...
    'Shuffle', 'every-epoch', ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropPeriod', 5, ...
    'LearnRateDropFactor', 0.5, ...
    'L2Regularization', 0.0001);
```

1.6 CNN Training and Evaluation

Training options are set up to use the ADAM optimizer, with various hyperparameters configured such as the number of epochs, learning rate, mini-batch size, and more. Notably, the learning rate is scheduled to decrease every 5 epochs to improve convergence. Validation data is also provided to monitor overfitting.

```
%% Train the CNN
[net, info] = trainNetwork(trainImages, categorical(trainLabels),..
lgraph, options);
% Evaluate the trained network
pred = classify(net, testImages);
accuracy = sum(pred == categorical(testLabels)) / numel(testLabels);
fprintf('Test accuracy: %f\n', accuracy);
```

The two plots encompass:

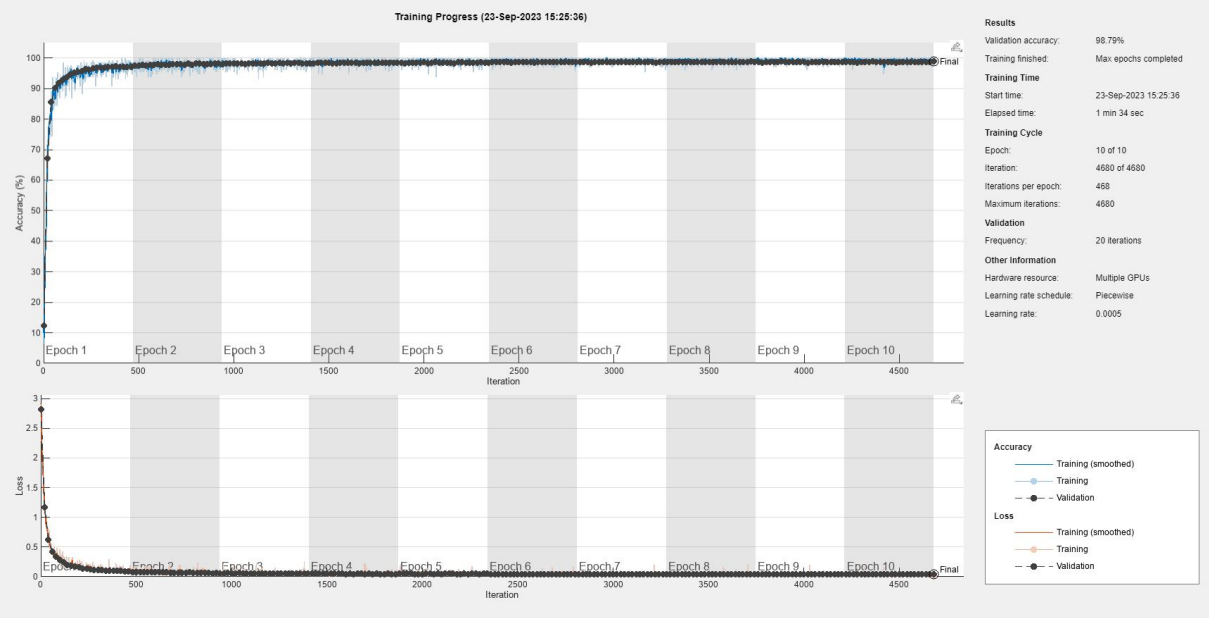

Figure 1.1: The training progress for each iteration.

- The top-plot depicts the accuracy for each iteration.
- The bottom-plot depicts the loss rate for each iteration.

1.7 Image Classification - MNIST: KNN

The k-Nearest Neighbors (k-NN) algorithm is a non-parametric method that can be employed for classification tasks. The performance of k-NN can be significantly improved when the data's dimensionality is reduced. One such method for dimensionality reduction is Principal Component Analysis (PCA) [JWHT13].

1.8 Principal Component Analysis (PCA)

PCA is an essential technique in data science, highlighting variation and capturing dominant patterns in a dataset. Employing PCA can substantially reduce the data's dimensionality without a significant loss of information [JWHT13].

1.8.1 PCA on Training Data

The principal components are computed as follows:

```
[coeff, score, latent] = pca(gather(trainData));
```

1.8.2 Selecting Principal Components

To retain 95

```
explainedVariance = cumsum(latent) / sum(latent);
numComponents = find(explainedVariance > 0.95, 1);
```

1.8.3 Data Projection

The training data is projected into the PCA space. The test data is centred using the mean of the training data and then projected:

```
trainDataPCA = gpuArray(score(:, 1:numComponents));
meanTrain = mean(trainData, 1);
testDataPCA = gpuArray((gather(testData) - meanTrain)
* coeff(:, 1:numComponents));
```

1.9 Data Visualization

Visualization provides essential insights. For this dataset, two plots are conceptualized:

The two plots encompass:

- The proportion of variance explained by each principal component.
- The cumulative variance explained by the principal components.

1.10 k-Nearest Neighbors Classification

With the reduced dataset, the k-NN model is trained using the PCA-transformed training data. The choice of 'k' (number of neighbors) is set to 5, correlating distance as a metric. After training, the test set predictions are inferred.

```
% Train k-NN using the PCA-transformed training data
knn_model = fitcknn(trainDataPCA, train_Labels, 'NumNeighbors', 3, ...
```

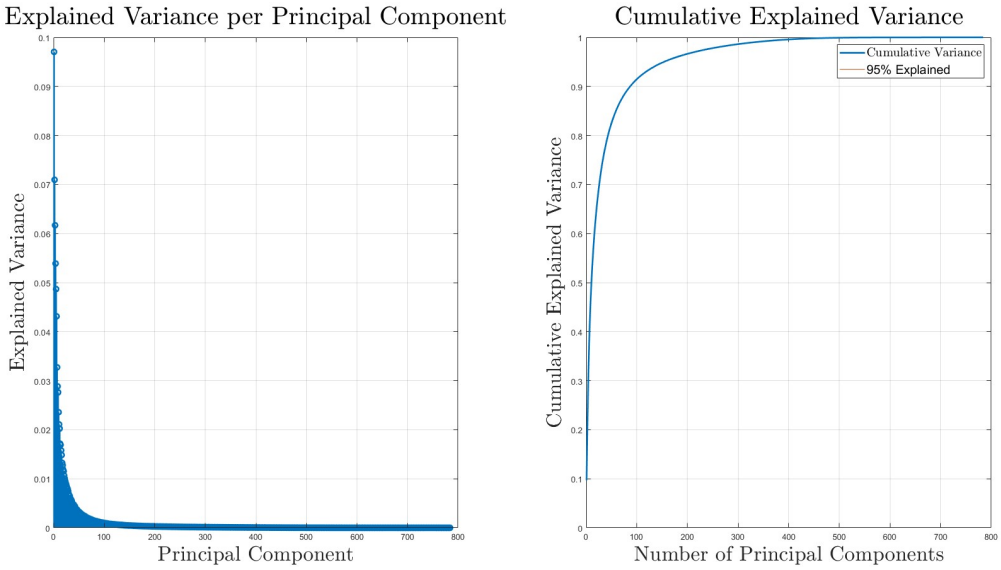

Figure 1.2: Explained Variance Plot.

```
    "Distance","correlation", "DistanceWeight","squaredinverse");
% Predict using the PCA-transformed test data
knn_pred = predict(knn_model, testDataPCA);
```

1.11 Evaluation Metrics

To understand the model's efficacy, performance evaluation is paramount. A confusion matrix, which juxtaposes actual versus predicted classifications, is analysed. Leveraging this matrix, precision, recall, and the F1-score for each class are computed. These metrics provide an in-depth understanding of the classifier's performance beyond mere accuracy.

```
confusionKNN = confusionmat(test_Labels, knn_pred);
.
.
.% Calculate precision, recall, and F1-score
C= confusionKNN;
nClasses = size(C, 1);  % Assuming C is a square matrix
precision = zeros(1, nClasses);
recall = zeros(1, nClasses);
f1Score = zeros(1, nClasses);
```

```
for i = 1:nClasses
    TP = C(i, i);
    FP = sum(C(:, i)) - TP;
    FN = sum(C(i, :)) - TP;

    precision(i) = TP / (TP + FP);
    recall(i) = TP / (TP + FN);
    f1Score(i) = 2 * (precision(i) * recall(i))...
    / (precision(i) + recall(i));
end
```

1.12 Conclusion

In our endeavour to classify the MNIST dataset, two different approaches were experimented with: Convolutional Neural Networks (CNN) and k-Nearest Neighbors (k-NN). Each method has its own merits and demerits, and through rigorous evaluation, we gauged their performances.

1.12.1 Performance Metrics Comparison

The performance metrics, namely execution time, accuracy, precision, recall, and F1-score, were collected for each method. The comparison between CNN and k-NN using these metrics is provided in Table 1.1.

1.12.2 Confusion Matrix Analysis

A confusion matrix provides a more detailed breakdown of the classifier's performance by showing the true and false classifications for each class, we can gain further insights into specific areas where each model excels or underperforms.

From the confusion matrices presented in Figures 1.3 and 1.4, it is evident that the KNN tends to misclassify class 4 as class 9 more often than the CNN approach.

By scrutinizing the results presented in Table 1.1 and the confusion matrices, it becomes evident that the CNN method outperforms the k-NN marginally in terms of accuracy but has a way longer execution time.

.

Table 1.1: Performance comparison of CNN and k-NN

Metric	Value	
	CNN	k-NN
Execution Time	94 sec	2.6
Accuracy	98.79%	97.23%
Precision (per class)	[0.9929 0.9904 0.9864 0.9882 0.9918 0.9833 0.9948 0.9826 0.9847 0.984]	[0.9711 0.9716 0.9786 0.9673 0.9714 0.9675 0.9772 0.964 0.9903 0.9649]
Recall (per class)	[0.9929 0.9974 0.9855 0.9921 0.9807 0.991 0.9896 0.9864 0.9887 0.9742]	[0.9939 0.9956 0.9748 0.9673 0.9674 0.9686 0.9864 0.965 0.9466 0.9544]
F1-score (per class)	[0.9929 0.9939 0.9859 0.9901 0.9862 0.9872 0.9922 0.9845 0.9867 0.9791]	[0.9823 0.9835 0.9767 0.9673 0.9694 0.9681 0.9818 0.9645 0.968 0.9596]

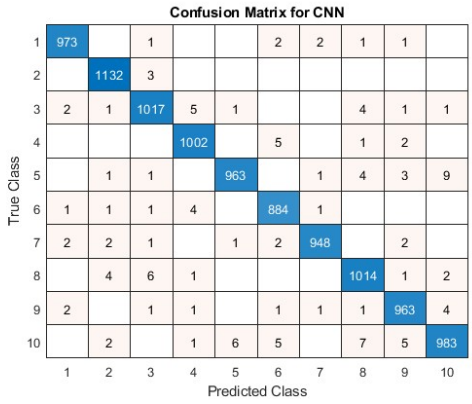

Figure 1.3: Confusion matrix for the CNN approach

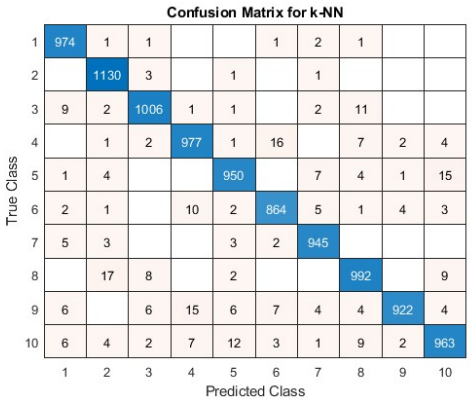

Figure 1.4: Confusion matrix for the k-NN approach

2 Regression- CFRP NASA

Composite materials, particularly Carbon Fiber Reinforced Plastics (CFRPs), have increasingly found utility in aerospace applications due to their remarkable strength-to-weight ratios. NASA has amassed significant datasets on CFRPs, making it a cornerstone in understanding and modeling the fatigue life of these materials. While traditional regression methods, such as those discussed by Sun and Verrilli [SV02], have provided valuable insights into the behavior of composites, the advent of deep learning techniques presents an opportunity to further refine predictive accuracy. In this study, we introduce a Convolutional Neural Network (CNN) model for predicting the S-N (Stress vs. Number of Cycles) curve, or end-of-life curve, of CFRPs using the NASA dataset. The approach seeks to leverage the nuanced, hierarchical feature extraction capabilities inherent to CNNs. Preliminary results indicate that our CNN-based regression model offers improved predictive accuracy when compared to classical regression models, especially in the presence of complex, non-linear data patterns observed in laminated composite structures. Such advancements have the potential to significantly enhance our understanding of fatigue life, damage mechanisms [TS12], and the probabilistic nature of failures in composite materials [GS09]. By employing CNNs for this task, we anticipate a transformative change in the way aerospace industries approach the design and safety evaluation of composite structures. One common representation of fatigue life is the S-N curve, given by:

$$S = aN^b \tag{2.1}$$

where S is the stress amplitude, N is the number of cycles leading to failure, and a and b are material constants.

While traditional regression methods, such as those discussed by Sun and Verrilli [SV02], have provided valuable insights into the behavior of composites, the advent of deep learning techniques presents an opportunity to further refine predictive accuracy.

In this study, we introduce a Convolutional Neural Network (CNN) model for predicting the S-N curve of CFRPs using the NASA dataset. The approach seeks to leverage the nuanced, hierarchical feature extraction capabilities inherent to CNNs. Preliminary results indicate that our CNN-based regression model offers improved predictive accuracy when compared to classical regression models, especially in the presence of complex, non-linear data patterns observed in laminated composite structures. Such advancements have the potential to significantly enhance our understanding of fatigue life, damage mechanisms [TS12], and the probabilistic nature of failures in composite materials [GS09]. By employing CNNs for this task, we anticipate a transformative change in the way aerospace industries approach the design and safety evaluation of composite structures.

2.1 Initialization and Data Loading

```
% Start a parallel pool
if isempty(gcp('nocreate'))
    parpool;
end

% Initialize tables for Layup 1, Layup 2, and Layup 3
initTable = table('Size', [0, 14], 'VariableTypes',..
repmat({'double'}, 1, 14), ...
... % Additional initialization code omitted for brevity
```

The code initiates by ensuring that a parallel computing pool is started, likely to facilitate parallelized operations in later stages. It then establishes three empty tables for different configurations of the composite material and defines an array, 'dirr', with directories of '.mat' files that presumably store test data.

2.2 Data Extraction and Processing

The data extraction process is built on iterative loops that navigate through directories and their respective '.mat' files, extracting crucial pieces of data. Here's a brief breakdown:

1. **Directory Looping**: The code initializes with a list of directories (`dirr`). It loops through each one, aiming to process every '.mat' file inside them.

   ```
   for d = 1:length(dirr)
       matFiles = dir(dirr{d});
       ...
   ```

2. **Loading Files**: For each directory, it lists all the '.mat' files and loops through them, loading them sequentially.

   ```
   for i = 1:length(matFiles)
       filePath = fullfile(matFiles(i).folder, matFiles(i).name);
       tempData = load(filePath);
       ...
   ```

3. **Data Extraction**: If the loaded file has a `coupon` field, it starts the extraction. Direct fields like `load` and `cycles` are fetched immediately. When `path_data` is present, additional attributes related to actuator and sensor are extracted. Certain features, such as mean and energy, are computed directly in this step.

```
if isfield(tempData, 'coupon')
    ...
    if isfield(tempData.coupon, 'path_data')
        ...
```

4. **Row Creation and Appending**: After extraction, a new data row is constructed and appended to one of the initialized tables, depending on the directory of origin.

```
newRow = table(...);
if contains(dirr{d}, 'Layup1')
    layup1Table = [layup1Table; newRow];
...
```

5. **Reiteration**: This entire process is reiterated until all '.mat' files across all directories are processed.

This iterative data extraction method ensures that all critical information from the '.mat' files is processed and neatly organized into the respective tables for each layup type.

2.3 Data Cleaning and Saving

```
function cleanedTable = removeEmptyRows(inputTable)
... % Function implementation omitted for brevity
```

Upon completion of data loading, the 'removeEmptyRows' function cleans each layup table by removing rows with empty or 'NaN' values. This is a standard step in data preprocessing. Finally, the cleaned layup tables are saved as '.mat' files.

2.4 Data Loading and Preparation

The first part of the code is about loading the required data and preparing it for analysis:

```
load('layup1Table.mat')
...
dataArray = table2array(layup1Table);
...
```

Here, the data is loaded from the `layup1Table.mat` file and then converted to an array for further processing[Docb].

2.4.1 Training and Test Set Partition

Data is partitioned into training and test sets. 70% of the data is used for training, and 30% for testing.

```
n = size(featureMatrix, 1);
...
XTrain = cell2mat(featureMatrix(shuffleIdx(1:nTrain), :));
...
```

The dataset is also shuffled to ensure randomness[Docb].

2.4.2 Data Normalization

Data normalization is an essential step before training neural networks:

```
XTrain_norm = (XTrain - featureMin) ./ (featureMax - featureMin);
...
```

This step scales the data between 0 and 1, ensuring all features have the same scale[Docc].

2.5 Neural Network Design

The neural network is designed for regression analysis, which is evident from the subsequent evaluation of its predictions against continuous output values.

2.5.1 Neural Network Architecture

1. **Hidden Layer Configuration**: The neural network has one hidden layer. The number of neurons in this hidden layer is defined by `hiddenLayerSize`, which is set to 10 in the code.

   ```
   hiddenLayerSize = 10;
   ```

2. **Training Algorithm**: The Levenberg-Marquardt backpropagation algorithm, represented by `trainlm`, is used for training. It's known for its efficiency and speed, especially for small and medium-sized datasets.

   ```
   trainFcn = 'trainlm';
   ```

3. **Network Creation**: The `fitnet` function is utilized to create a feedforward backpropagation network. This function initializes the weights and biases of the network.

```
net = fitnet(hiddenLayerSize, trainFcn);
```

2.5.2 Training Configuration

1. **Epochs**: The maximum number of epochs (iterations) the network should train for is set to 100. An epoch is one forward and backward pass of all the training examples.

```
net.trainParam.epochs = 100;
```

2. **Learning Rate**: The rate at which the network updates its weights and biases. A rate of 0.05 is specified, meaning the network makes relatively small adjustments to the weights and biases during training.

```
net.trainParam.lr = 0.05;
```

3. **Performance Goal**: The training will stop if the mean squared error falls below `1e-5`.

```
net.trainParam.goal = 1e-5;
```

4. **Other Parameters**: There are other configuration settings related to performance gradient, display settings, and the maximum training time.

2.5.3 Data Division for Training, Validation, and Testing

The dataset is divided into three parts: training, validation, and testing. Specifically, 70% of the data is used for training, 15% for validation, and 15% for testing. Validation data assists in preventing overfitting during training.

```
net.divideParam.trainRatio = trainFraction;
net.divideParam.valRatio = valFraction;
net.divideParam.testRatio = testFraction;
```

The neural network is then trained using the `train` function, which uses the configuration defined earlier and adjusts the network weights based on the training data.

By understanding this design, one can make informed decisions about potential modifications or optimizations depending on the specific requirements of the application or the nature of the dataset.

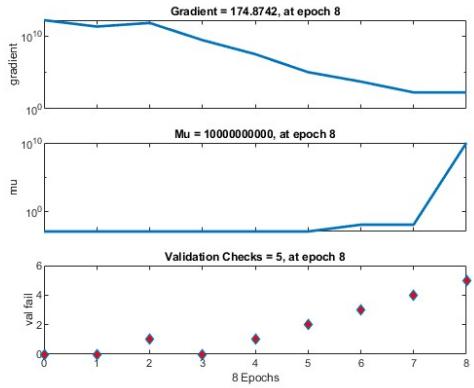

Figure 2.1: training state for Layup 1.

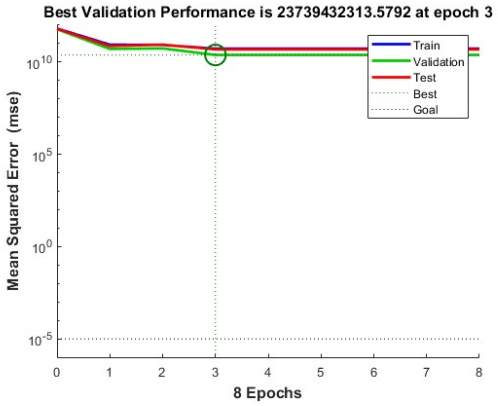

Figure 2.2: validation progress for Layup 1.

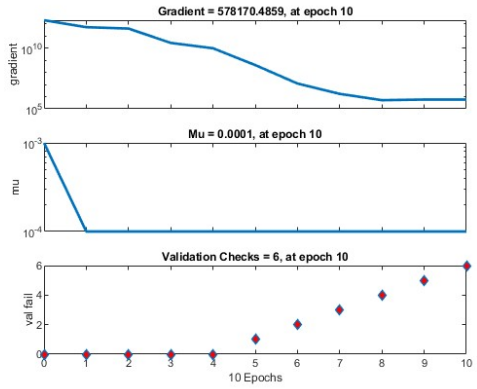

Figure 2.3: training state for Layup 2.

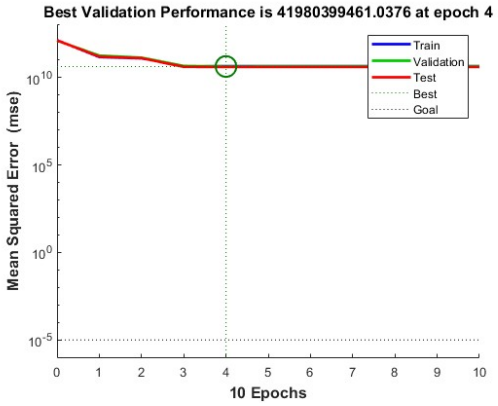

Figure 2.4: validation progressr Layup 2.

2.6 Model Evaluation: `evaluateNNModel` Function

The function `evaluateNNModel` serves as an all-encompassing method for training and evaluating a neural network. In this section, we'll dissect its operation.

2.6.1 Function Inputs and Outputs

- **Inputs**:
  - `net`: The untrained neural network architecture.
  - `XTrain_norm`: Normalized feature matrix of the training set.
  - `YTrain_norm`: Normalized label/output matrix of the training set.

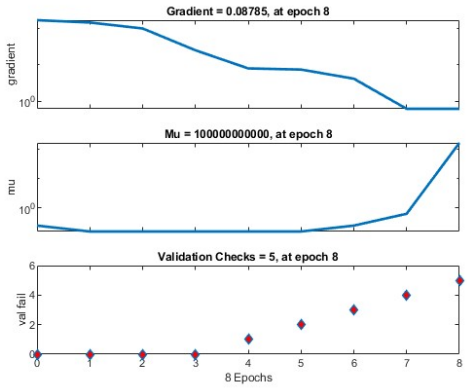

Figure 2.5: training state for Layup 3.

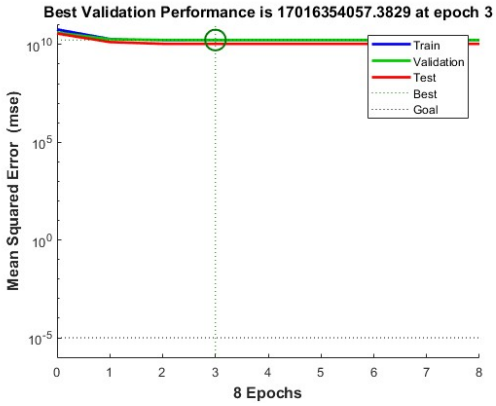

Figure 2.6: validation progress for Layup 3.

  - `XTest_norm`: Normalized feature matrix of the test set.
  - `YTest_norm`: Normalized label/output matrix of the test set.
  - `threshold`: User-defined value for detecting overfitting.

- **Outputs**:
  - `mseTrain`: MSE on the training set.
  - `mseTest`: MSE on the test set.
  - `isOverfitting`: A boolean indicating potential overfitting.

2.6.2 Function Operation

1. **Training the Neural Network**:

   The neural network is trained using the normalized training data.

   ```
   [net, tr] = train(net, 'XTrain_norm', 'YTrain_norm');
   ```

2. **Prediction on Training and Test Sets**:

   Predictions are fetched from the trained network.

   ```
   yPredTrain = net('XTrain_norm');
   yPredTest = net('XTest_norm');
   ```

3. **Calculating MSE for Training and Test Sets**:

   MSE is calculated for both the training and test sets.

```
mseTrain = mean((yPredTrain' - YTrain_norm).^2);
mseTest = mean((yPredTest' - YTest_norm).^2);
```

4. **Displaying the MSE Values**:

   The calculated MSE values are printed out.

   ```
   fprintf('MSE on Training Set: \%f\n', mseTrain);
   fprintf('MSE on Test Set: \%f\n', mseTest);
   ```

5. **Checking for Overfitting**:

   The function checks if the model might be overfitting, based on the comparison of MSE values.

   ```
   isOverfitting = false;
   if mseTrain < mseTest
       difference = mseTest - mseTrain;
       if difference > threshold
           disp('Warning: The model might be overfitting.');
           isOverfitting = true;
       end
   else
       disp('The model seems to generalize well.');
   end
   ```

In essence, `evaluateNNModel` facilitates the training, prediction, and evaluation of a neural network's performance, particularly emphasizing the detection of overfitting through the MSE values comparison.

2.7 Curve Fitting

Curve fitting is used in the latter parts of the code:

```
[fitresult, gof] = fit( xData, yData, ft, opts );
...
```

This step tries to establish a relationship between predicted cycles and other parameters, possibly trying to understand fatigue life or other similar relationships[Doca].

2.8 Conclusion

In the following section, we present a comparison of the results obtained from the analysis of three different layups. The metrics under consideration include the Mean Squared Error (MSE) for both training and test datasets, as well as the correlation coefficient (R value). Additionally, visual representations in the form of the predicted S-N curve and the RUL curve for each layup are provided. as it can deduced from Table 2.1, the MSE value for the train data sets is greater than that for the value for the train, proving that the data trained network is not over fitted.

Layup	MSE (Train)	MSE (Test)	R Value
Layup 1	0.075019 &.064859	0.072509 &.059717	0.74152
Layup 2	0.071039 &.066820	0.058086&.053494	0.66806
Layup 3	0.071793&.060680	0.067616 &.051012	0.65694

Table 2.1: Comparison of MSE values and R values for the three layups.

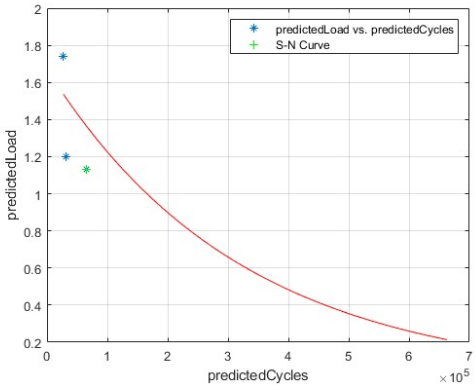

Figure 2.7: Predicted S-N Curve for Layup 1.

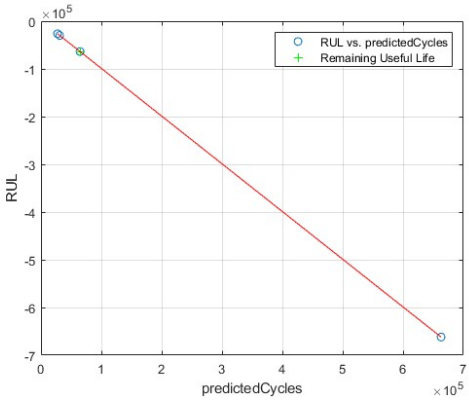

Figure 2.8: RUL Curve for Layup 1.

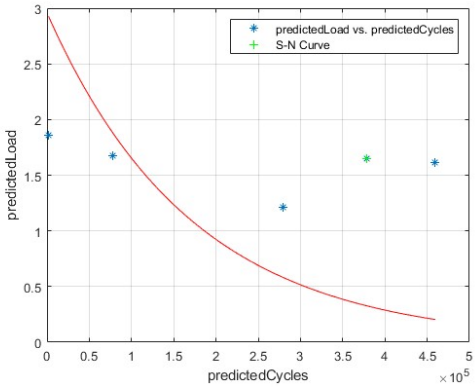

Figure 2.9: Predicted S-N Curve for Layup 2.

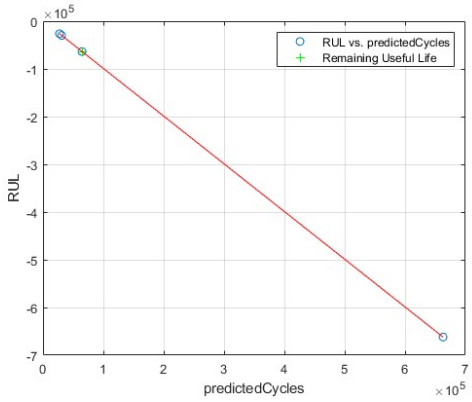

Figure 2.10: RUL Curve for Layup 2.

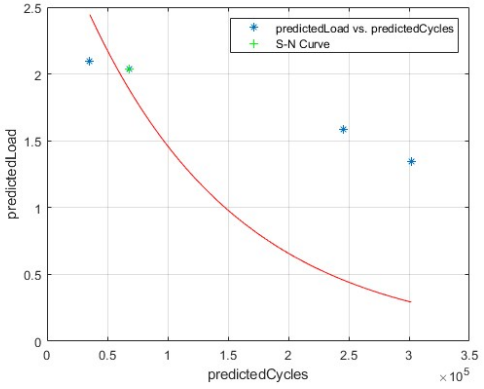

Figure 2.11: Predicted S-N Curve for Layup 3.

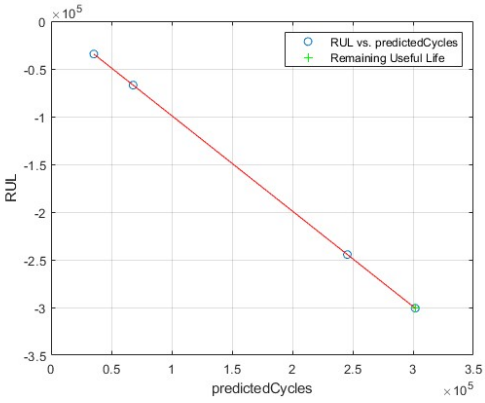

Figure 2.12: RUL Curve for Layup 3.

References

[CMS12] Cireşan, Dan C. ; Meier, Ueli ; Schmidhuber, Jürgen: Multi-column deep neural networks for image classification. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition* IEEE, 2012, S. 3642–3649

[Doca] Documentation, MATLAB: *Curve Fitting Toolbox*. `https://www.mathworks.com/products/curvefitting.html`, . – Accessed: 2023-09-24

[Docb] Documentation, MATLAB: *Data Preprocessing for Machine Learning (MATLAB)*. `https://www.mathworks.com/discovery/data-preprocessing-matlab.html`, . – Accessed: 2023-09-24

[Docc] Documentation, MATLAB: *Getting Started with Neural Network Toolbox*. `https://www.mathworks.com/help/deeplearning/gs/getting-started-with-neural-network-toolbox.html`, . – Accessed: 2023-09-24

[GBC16] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016

[GS09] Glaessgen, E. H. ; Saether, E.: Review of the use of probabilistic methods in prediction of the effects of imperfections on composite structures / NASA Technical Memorandum. 2009 (NASA/TM-2009-215857). – Forschungsbericht

[JWHT13] James, Gareth ; Witten, Daniela ; Hastie, Trevor ; Tibshirani, Robert: *An introduction to statistical learning*. New York : Springer, 2013

[LBBH98] LeCun, Yann ; Bottou, Léon ; Bengio, Yoshua ; Haffner, Patrick: Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE* 86 (1998), Nr. 11, S. 2278–2324

[Mat22] MathWorks: *MATLAB Parallel Computing Toolbox*. 2022. – Software

[SV02] Sun, C. T. ; Verrilli, M. J.: Fatigue of composite materials: Analysis and predictions. In: *Journal of Composite Materials* 36 (2002), Nr. 11, S. 1303–1317

[TS12] Talreja, R. ; Singh, C. V.: *Damage and failure of composite materials*. Cambridge University Press, 2012