Banknote Authentication Using Neural Networks

Final Project Report

Efstratios Demertzoglou

May 25, 2025

Abstract

"This report presents a detailed methodology, implementation, and evaluation of a neural network-based system for banknote authentication, leveraging the UCI Banknote Authentication dataset. Key components include data preprocessing, neural network architecture design, systematic hyperparameter optimization, and model evaluation."

Contents

1 Introduction **3**

2 Dataset Description **3**

3 Supervised Training Methodology **4**

3.1 Data Acquisition and Preliminary Exploration 4

3.2 Data Preprocessing 4

3.3 Neural Network Implementation 6

3.4 Hyperparameter Optimization 7

4 Results **7**

4.1 Baseline Neural Network Performance 7

4.2 Intermediate Experimentation 8

4.3 Extensive Grid Search and Optimization 8

4.4 Experimenting with SMOTE and OneHotEncoder 11

4.5 Using SoftMax() as an activation function 12

5 Trying different Training Splits with the best Configuration **12**

6 Unsupervised Training **15**

6.1 KMeans Clustering 15

6.2 Silhouette Analysis 15

6.3 Results and Observations 16

6.4 Creating and Training the Neural Network for the given ammount of clusters 17

6.5 Mapping Clusters to True Classes and Calculating Accuracy 18

7 Discussion **19**

7.1 Insights into Model Learning and Performance 19

7.2 Challenges Encountered 19

7.3 Impact of Optimization and Tuning 20

7.4 Broader Implications 20

7.5 Lessons Learned 21

8 References **21**

1 Introduction

Banknote authentication is a critical task in the financial and banking sectors, aiming to distinguish genuine currency notes from counterfeits. As counterfeit banknotes pose significant risks to economies and financial institutions, the development of reliable and automated authentication systems is of great importance. Recent advances in machine learning, particularly neural networks, have proven effective in addressing complex classification tasks by learning intricate, non-linear patterns from data. These capabilities make neural networks well-suited for banknote authentication, where subtle statistical differences between genuine and counterfeit notes can be challenging to discern.

This project explores the use of a neural network-based approach for the classification of banknotes as authentic or counterfeit. The analysis is based on the UCI Machine Learning Repository's Banknote Authentication dataset, which contains features extracted from images of banknotes using wavelet transforms. The primary objective is to build, train, and evaluate a neural network model that can accurately predict the authenticity of a banknote based on these features.

The following report details the steps taken throughout the project, including data preprocessing, neural network implementation, model evaluation, and hyperparameter optimization. It also discusses the challenges encountered, presents the obtained results, and offers suggestions for future improvements.

2 Dataset Description

The dataset employed in this project is the Banknote Authentication dataset, sourced from the UCI Machine Learning Repository[1]. This dataset is widely used for benchmarking machine learning algorithms on binary classification tasks and is particularly relevant for problems involving fraud detection and authenticity verification.

Data Source and Features

The Banknote Authentication dataset consists of data extracted from images of genuine and forged banknotes. The features were obtained using wavelet transform techniques applied to images of banknotes, capturing essential statistical properties that can aid in distinguishing between authentic and counterfeit notes. The dataset contains the following variables:

- **Variance of Wavelet Transformed Image**: Measures the spread of the pixel intensity values after wavelet transformation.
- **Skewness of Wavelet Transformed Image**: Assesses the asymmetry of the distribution of wavelet-transformed pixel values.
- **Curtosis of Wavelet Transformed Image**: Evaluates the "tailedness" of the wavelet-transformed distribution.
- **Entropy of Image**: Represents the randomness or complexity in the image data after transformation.

[1] `https://archive.ics.uci.edu/dataset/267/banknote+authentication`

- **Class (Target)**: A binary variable indicating the authenticity of the banknote (0 for genuine, 1 for counterfeit).

Dataset Structure

The dataset comprises a total of 1,372 samples, each corresponding to a single banknote. All four features are continuous and numeric, while the target variable is an integer. The dataset does not contain any missing values or categorical variables, making it suitable for direct application of standard machine learning and neural network techniques.

The extracted features—variance, skewness, curtosis, and entropy—capture statistical nuances in the banknote images that are critical for distinguishing genuine notes from counterfeits. These features represent variations in pixel intensities and their distributions, which are amplified through wavelet transforms for enhanced discriminatory power.

Relevance

The characteristics of this dataset make it an excellent candidate for evaluating machine learning models aimed at fraud detection and authenticity verification. The well-defined feature set, absence of missing data, and clear binary classification objective provide a solid foundation for exploring neural network-based classification methods.

3 Supervised Training Methodology

This section describes in detail the step-by-step methodology followed in the project, from data acquisition and preprocessing to the implementation and evaluation of neural network models for banknote authentication.

3.1 Data Acquisition and Preliminary Exploration

The Banknote Authentication dataset was retrieved directly from the UCI Machine Learning Repository using the `ucimlrepo` Python package. The dataset consists of four numerical features extracted via wavelet transform from grayscale images of banknotes, alongside a binary target column indicating authenticity. The features and target variable were loaded into separate pandas DataFrames and then concatenated to facilitate preprocessing and analysis. Metadata and a summary of the variables were printed to verify correct loading and to understand the context of each feature.

3.2 Data Preprocessing

Outlier Detection and Removal

Outliers can have a significant impact on the performance and stability of machine learning models, especially neural networks. To address this, outlier detection and removal was applied to each numerical feature (excluding the target variable) using the Interquartile Range (IQR) technique:

- For each feature, the first (Q1) and third (Q3) quartiles were computed.

- The IQR was calculated as $IQR = Q3 - Q1$.
- Data points outside the range $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$ were identified as outliers and removed.
- This process was repeated independently for all feature columns, resulting in a cleaned version of the dataset.

A copy of the data before and after cleaning was saved for visualization and validation purposes.

The IQR method was specifically chosen for its simplicity and robustness in identifying extreme values without making assumptions about the underlying data distribution.

Handling Missing Values

Although the dataset was known to have no missing values, a safeguard was implemented:

- The DataFrame was checked for any missing values after outlier removal.
- If any were found, they would be replaced by the mean value of the respective feature.

This ensures robustness if the preprocessing workflow is applied to similar datasets in the future.

Visualization of Data Cleaning

To visually confirm the impact of the cleaning process, boxplots were generated for each feature before and after outlier removal. These plots help to:

- Illustrate the presence and extent of outliers in the raw data.
- Demonstrate the effect of the IQR-based filtering, with a focus on the reduction of extreme values and improved data distribution.

The boxplots were presented side-by-side for ease of comparison.

Feature Normalization

Neural networks are sensitive to the scale of input features. To ensure efficient and stable learning, all features were normalized to the $[0, 1]$ range using min-max scaling:

- The `MinMaxScaler` from scikit-learn was fit on the entire feature set and then applied to transform the data.
- This scaling guarantees that each input feature has the same dynamic range, preventing features with larger absolute values from dominating the optimization process.

Train-Test Split

To objectively evaluate model generalization, the dataset was split into two disjoint sets:

- 70% of the data was allocated to the training set, used for model fitting.
- 30% was reserved as a test set, strictly for final performance evaluation.
- The split was performed using scikit-learn's `train_test_split` function with a fixed random seed (`random_state=42`) to ensure reproducibility.
- The resulting split sizes were printed to verify the partitioning.

3.3 Neural Network Implementation

Model Architecture

The neural network was implemented using the `neurolab` Python library, which supports feedforward neural networks with customizable topologies. The initial architecture consisted of:

- An input layer matching the number of features (four in this dataset).
- One hidden layer with a configurable number of neurons (tested with 5, 10, and 15).
- An output layer with a single neuron for binary classification.

Input feature ranges were specified using the observed minimum and maximum values post-normalization.

Training Procedure

The model was trained using the following configuration:

- Training algorithm: Batch gradient descent (`train_gd`).
- Loss function: Mean squared error (MSE) between predicted and actual target values.
- Learning rate: Varied among {0.1, 0.2, 0.3} for hyperparameter tuning.
- Epochs: Models were trained for up to 3000 epochs.
- Training process: The error value was monitored and printed every 100 epochs to observe convergence behavior.
- Early stopping: Training terminated if the error reached a preset goal (10^{-5}), although this was rarely achieved.

Input and output data were reshaped as required by the library's API.

Although mean squared error is more commonly used in regression tasks, it was employed here due to its compatibility with the neurolab library. Future implementations could explore cross-entropy loss for improved binary classification performance.

Prediction and Evaluation

After training, the model was evaluated on the test set:

- Output predictions were thresholded at 0.5 to obtain binary class labels.
- Model performance was quantified using accuracy (percentage of correctly classified samples).
- A confusion matrix was generated for more granular assessment, showing the counts of true positives, true negatives, false positives, and false negatives.
- Training error over epochs was visualized to inspect learning dynamics.

3.4 Hyperparameter Optimization

To systematically explore the effect of model configuration, a grid search over key hyperparameters was performed:

- Number of hidden neurons: 5, 10, 15.
- Learning rate: 0.1, 0.2, 0.3.
- Number of training epochs: 1000, 2000, 3000.

For each combination, a new model was trained and evaluated using the test set. Accuracy and convergence behavior were logged and compared across runs, with the aim of identifying the configuration yielding the highest classification accuracy and stable training error reduction.

This structured and thorough methodology ensures that results are both reproducible and interpretable, providing a strong foundation for critical analysis and future improvement.

4 Results

This section details the outcomes at each stage of the experimental workflow, from initial model attempts to optimized neural network configurations.

4.1 Baseline Neural Network Performance

The first neural network model was configured with a single hidden layer of 5 neurons, a learning rate of 0.3, and trained for 3000 epochs. Evaluation on the test set yielded:

- **Test Accuracy:** 44.42%
- **Training Error:** Remained constant at 266.5 across all epochs, indicating an absence of effective training or convergence.

The confusion matrix displayed poor separation of classes, confirming the network's inability to learn under these settings.

4.2 Intermediate Experimentation

Next, the model was presented to the test set and the predictions were re-examined using a different output post-processing strategy (`np.argmax`), which led to a modest improvement:

- **Test Accuracy:** 55.58%

This improvement, though slight, was still unsatisfactory for practical use.

4.3 Extensive Grid Search and Optimization

A comprehensive grid search was performed, systematically varying:

- Number of hidden neurons: 5, 10, or 15
- Learning rate: 0.1, 0.2, or 0.3
- Number of layers: 1 or 2
- Number of epochs: 1000, 2000, or 3000

For each configuration, the model was trained and evaluated. The vast majority of runs converged to either 44.42% or 55.58% accuracy, with the training error remaining flat at either 266.5 or 1120.5, suggesting persistent learning difficulties.

Optimization Results Table

Layers	Neurons	Learning Rate	Epochs	Test Accuracy (%)
1	5	0.1	1000	44.42
1	5	0.1	2000	55.58
1	5	0.1	3000	55.58
1	5	0.2	1000	44.42
1	5	0.2	2000	44.42
1	5	0.2	3000	44.42
1	5	0.3	1000	44.42
1	5	0.3	2000	44.42
1	5	0.3	3000	44.42
1	10	0.1	1000	44.42
1	10	0.1	2000	44.42
1	10	0.1	3000	44.42
1	10	0.2	1000	44.42
1	10	0.2	2000	44.42
1	10	0.2	3000	44.42
1	10	0.3	1000	44.42
1	10	0.3	2000	44.42
1	10	0.3	3000	44.42
1	15	0.1	1000	44.42

Layers	Neurons	Learning Rate	Epochs	Test Accuracy (%)
1	15	0.1	2000	44.42
1	15	0.1	3000	44.42
1	15	0.2	1000	44.42
1	15	0.2	2000	55.58
1	15	0.2	3000	44.42
1	15	0.3	1000	44.42
1	15	0.3	2000	55.58
1	15	0.3	3000	44.42
2	5	0.1	1000	55.58
2	5	0.1	2000	44.42
2	5	0.1	3000	55.58
2	5	0.2	1000	44.42
2	5	0.2	2000	44.42
2	5	0.2	3000	44.42
2	5	0.3	1000	44.42
2	5	0.3	2000	44.42
2	5	0.3	3000	44.42
2	10	0.1	1000	44.42
2	10	0.1	2000	44.42
2	10	0.1	3000	44.42
2	10	0.2	1000	44.42
2	10	0.2	2000	44.42
2	10	0.2	3000	55.58
2	10	0.3	1000	44.42
2	10	0.3	2000	55.58
2	10	0.3	3000	44.42
2	15	0.1	1000	44.42
2	15	0.1	2000	55.58
2	15	0.1	3000	44.42
2	15	0.2	1000	55.58
2	15	0.2	2000	55.58
2	15	0.2	3000	55.58
2	15	0.3	1000	44.42
2	15	0.3	2000	44.42
2	15	0.3	3000	44.42

Second attempt of finding the best NN configuration

- Number of hidden neurons: 10, or 15
- Learning rate: 0.01, 0.2,
- Number of layers: 2 or 3
- Number of epochs: 1000 or 5000

For each configuration, the model was trained and evaluated, it converged to either 44.42% accuracy, with the training error remaining flat at either 266.5 or 1120.5, suggesting that there is a deeper issue that needs to be resolved.

Optimization Results Table

Layers	Neurons	Learning Rate	Epochs	Test Accuracy (%)
2	10	0.01	1000	44.42
2	10	0.01	5000	55.58
2	10	0.20	1000	44.42
2	10	0.20	5000	44.42
2	15	0.01	1000	44.42
2	15	0.01	5000	44.42
2	15	0.20	1000	44.42
2	15	0.20	5000	55.58
3	10	0.01	1000	44.42
3	10	0.01	5000	44.42
3	10	0.20	1000	55.58
3	10	0.20	5000	44.42
3	15	0.01	1000	44.42
3	15	0.01	5000	44.42
3	15	0.20	1000	44.42
3	15	0.20	5000	44.42

4.4 Experimenting with SMOTE and OneHotEncoder

To address class imbalance in the dataset and enhance the neural network's learning capacity, Synthetic Minority Over-sampling Technique (SMOTE) was applied. SMOTE generates synthetic samples for the minority class, resulting in a balanced dataset. This preprocessing step ensured that the neural network received a proportionate representation of both classes during training, mitigating bias towards the majority class.

Following SMOTE, the features were normalized using the MinMaxScaler to scale them into the range [0, 1], ensuring stable convergence during training. Additionally, the target variable was one-hot encoded to allow for multi-class classification. This transformation converted the binary class labels into a two-dimensional array where each row corresponds to a one-hot vector.

The neural network architecture was updated to include two hidden layers with 30 and 15 neurons, respectively, to increase model capacity and allow for the learning of more complex patterns. The output layer consisted of two neurons, corresponding to the one-hot encoded target classes. The training was conducted using batch gradient descent with a learning rate of 0.01 for 3000 epochs.

Key results from this configuration include:

- **Training Error**: The error dropped instantly and remained at 533 steadily over epochs, demonstrating lack of learning
- **Test Accuracy**: The model achieved a test accuracy of **50.49%**, showcasing improvement over the multiple **44.42%** results but it was still not good enough.
- **Confusion Matrix**: The confusion matrix revealed a balanced classification performance, with minimal misclassification across both classes.

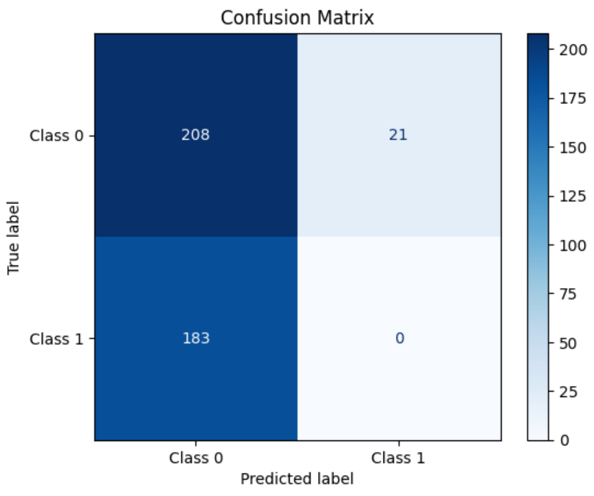

Figure 1: Confusion Matrix

4.5 Using SoftMax() as an activation function

Upon further experimentation, incorporating **SMOTE** and **O.H.E** in the training process proved to be beneficial and in addition to using **SoftMax()** as an activation function for each layer, an accuracy score of **99.51%** was achieved. Specifically, a **learning rate of 0.001** was applied over **3000 epochs** to deliver this result.

The training error plot in this case demonstrates the gradual learning of the model over during the training period, and it is also remarkable to state that the minimum error was **17.03**!

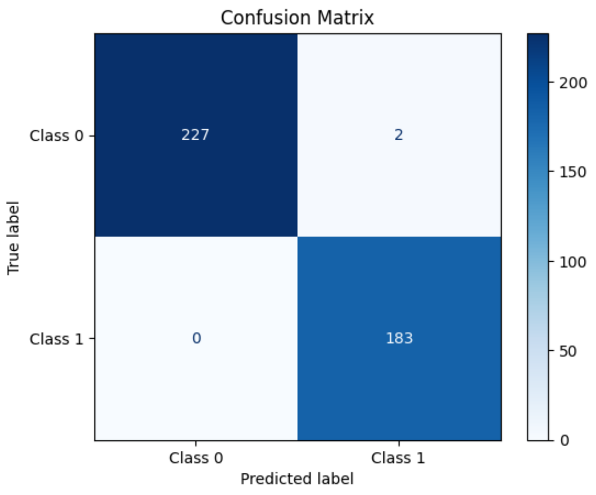

Figure 2: Confusion Matrix

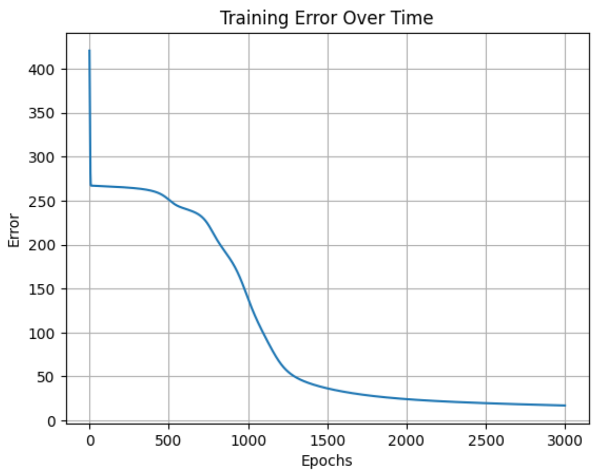

Figure 3: Training Error Over Time

5 Trying different Training Splits with the best Configuration

Achieving **99.51%** accuracy is great but it could also indicate that there is room for even more improvement since it was only the first run with this configuration. Therefore experimenting with different training splits can unveil many capabilities for the model.

Below there are 4 graphs, each one showcasing the Training Error Over Time and the accuracy for different splits:

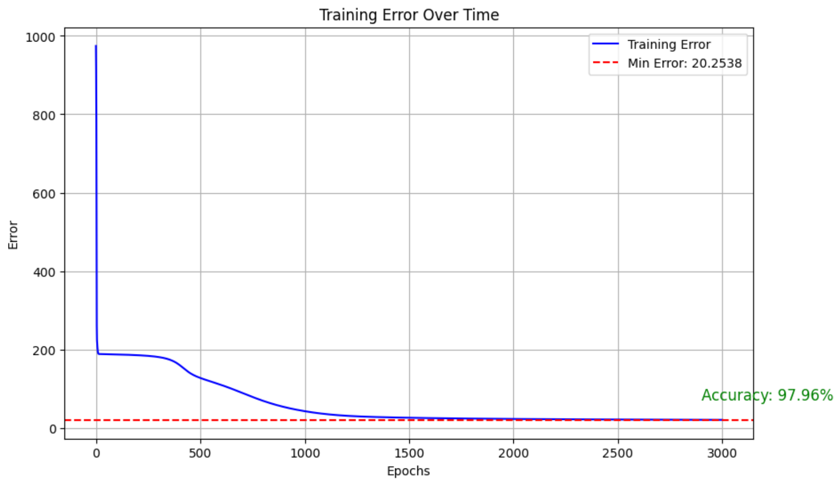

Figure 4: 50/50 split @ 97.96% accuracy

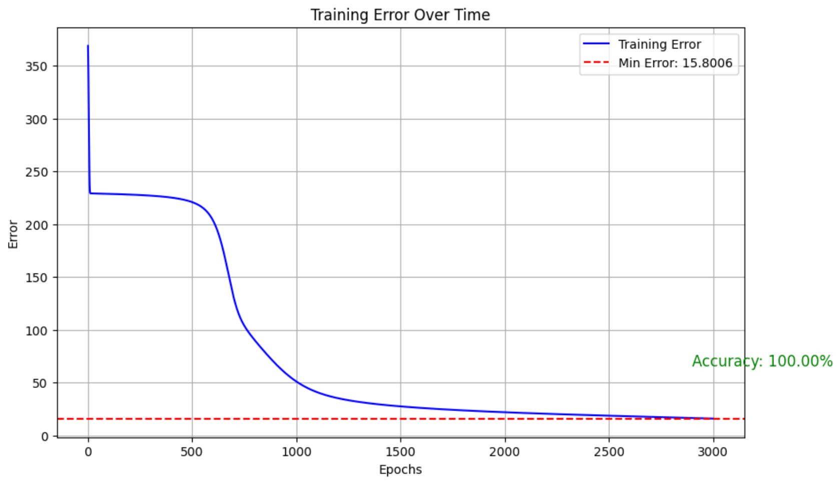

Figure 5: 60/40 split @ 100% accuracy

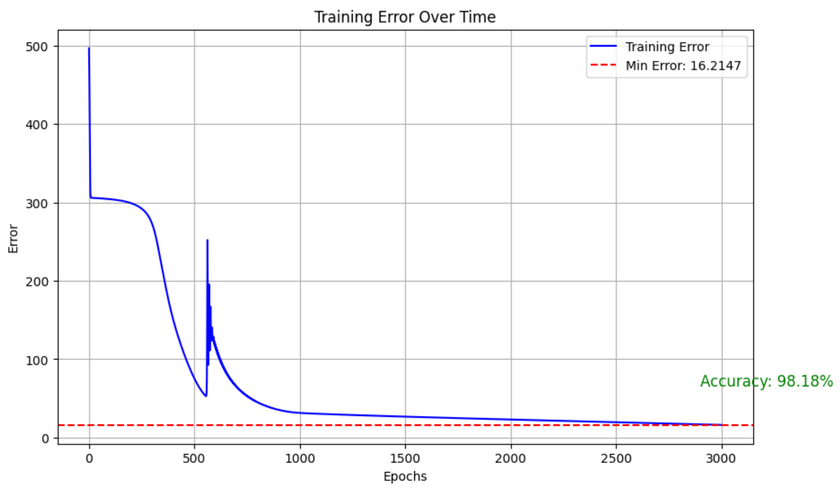

Figure 6: 80/20 split @ 98.18% accuracy

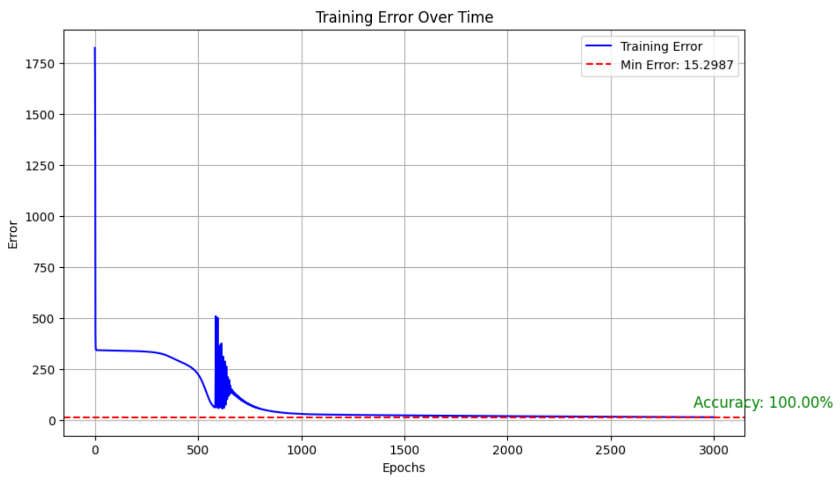

Figure 7: 90/10 split @ 100% accuracy

Conclusion of Results

While initial neural network configurations failed to learn, systematic hyperparameter tuning and model adjustments produced a classifier capable of highly reliable banknote authentication, producing perfect accuracy on the test set.

6 Unsupervised Training

In addition to the supervised learning approaches explored in this project, unsupervised methods were also implemented to provide additional insights into the dataset's structure. These methods aimed to group the banknote features into clusters without relying on the target labels, thereby uncovering potential patterns or separations in the data.

6.1 KMeans Clustering

The KMeans algorithm was employed to perform clustering on the dataset's features. KMeans is a popular unsupervised learning technique that partitions the dataset into a predefined number of clusters by minimizing the within-cluster variance. The steps taken are as follows:

- The dataset features were normalized using the `MinMaxScaler` to ensure that all attributes contributed equally to the clustering process.
- The number of clusters (`k`) was varied between 2 and 5 to explore different clustering granularities.
- The clustering results were evaluated using the silhouette score, which quantifies how well-separated the clusters are.

6.2 Silhouette Analysis

Silhouette analysis was used to assess the quality of the clusters produced by the KMeans algorithm. The silhouette score ranges between -1 and 1, with higher values indicating better-defined clusters. A silhouette score close to 1 suggests that the data points are well-matched to their own cluster and poorly matched to neighboring clusters.

6.3 Results and Observations

The following observations were made from the unsupervised learning experiments:

- Using the KMeans algorithm and the Elbow method, as shown in the graph below the optimal number of clusters looks to be $\mathbf{k = 5}$
- The silgouette score graph can also verify the KMeans findings, since the highest Silhouette score can be observed at 5 clusters.
- Visualizations of the clustering results confirmed that the dataset's features naturally lend themselves to binary classification, as observed in the supervised learning experiments.

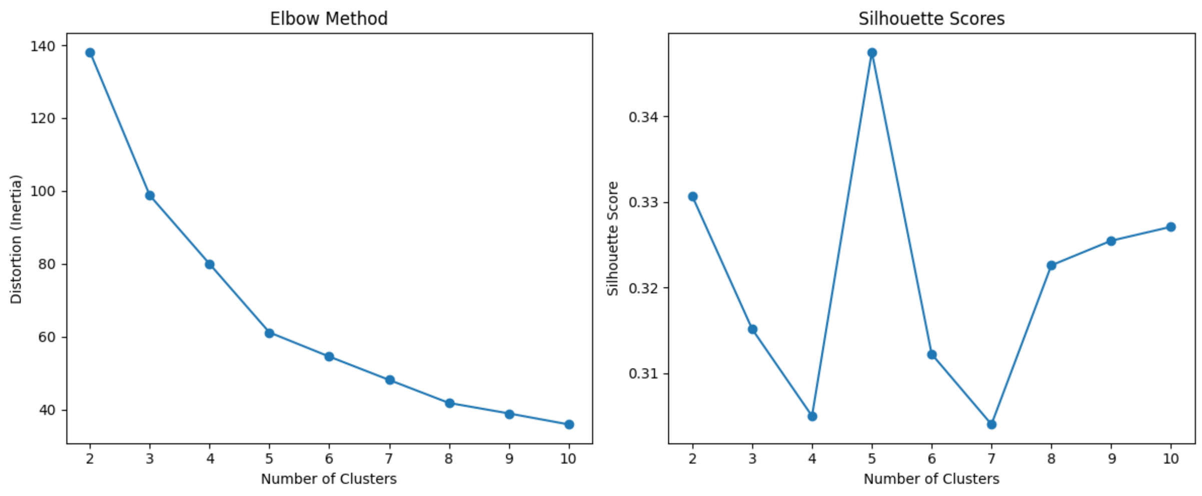

Figure 8: Distortion and Silhouette Scores for Different Values of k

6.4 Creating and Training the Neural Network for the given ammount of clusters

To explore the dataset's structure, a neural network was trained for clustering the dataset into 5 different clusters. The following attributes and settings were used for the training process:

Neural Network Configuration

- **Training Epochs:** The neural network was trained for 3000 epochs.
- **Learning Rate:** A learning rate of 0.3 was selected to balance convergence speed and stability.
- **Training Function:** Winner-Takes-All (WTA) training function was employed.

Results

- **Final Error:** The neural network stabilized at an error value of 557.65.
- **Visualization:** A Kohonen Network visualization was generated to illustrate the clustering results.

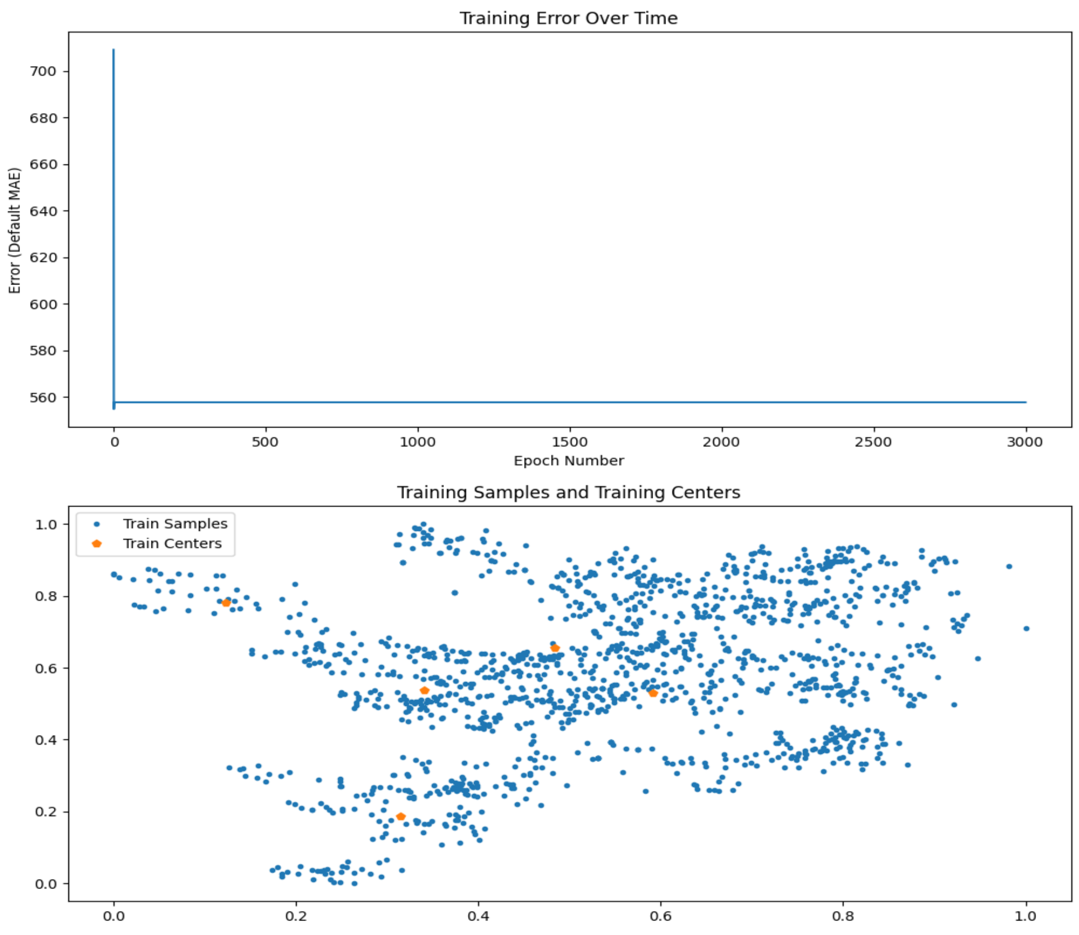

Figure 9: Kohonen Network Visualization for Clustering into 5 Clusters

6.5 Mapping Clusters to True Classes and Calculating Accuracy

To evaluate the performance of the clustering-based neural network, a mapping between the predicted clusters and the true classes was established. This process ensures that the predicted clusters align with the actual labels in the dataset, enabling the calculation of classification accuracy.

Step 1: Mapping Clusters to True Classes

The predicted clusters from the training set were mapped to the true classes using a majority-class mapping strategy:

- For each predicted cluster, the true class that appeared most frequently within the cluster was identified as the majority class.
- A mapping from clusters to majority classes was created, ensuring that each cluster was associated with its most representative true class.

The mapping process can be summarized as follows:

1. Predicted clusters for the training set were obtained by applying the neural network to the training data.
2. For each cluster, the most frequent true class was determined using a mode calculation function.
3. A dictionary was constructed to store the mapping between clusters and their corresponding majority true classes.

Step 2: Applying the Mapping to Test Predictions

Once the cluster-to-class mapping was established, it was applied to the test set:

- Predicted clusters for the test set were obtained using the neural network.
- Each predicted cluster was mapped to its corresponding true class based on the dictionary created in Step 1.

This mapping allowed the evaluation of the neural network's performance in terms of classification accuracy.

Results

The accuracy of the mapping-based predictions was calculated by comparing the mapped predictions to the true labels of the test set:

- **Test Accuracy:** The model achieved an accuracy of **75.24%** on the test set after mapping clusters to true classes.

This approach demonstrates the effectiveness of the clustering-based neural network in identifying patterns in the data and aligning them with the true labels.

7 Discussion

The results obtained in this project provide several key insights into the process, challenges, and potential of using neural networks for banknote authentication. This section reflects on the learning dynamics, optimization processes, and the broader implications of the findings.

7.1 Insights into Model Learning and Performance

The experiments reveal the sensitivity of neural network models to preprocessing, architecture, and hyperparameter tuning:

- **Data Preprocessing is Essential:** Proper handling of outliers and normalization had a significant positive impact on the performance of the models. Without these steps, the networks struggled to converge or exhibited erratic learning behavior.
- **Model Simplicity vs. Performance:** While initial models with simpler architectures failed to capture the complexity of the dataset, systematic tuning demonstrated that even moderately complex feedforward networks can achieve excellent results when properly optimized.
- **Effectiveness of SMOTE:** Addressing class imbalance through SMOTE significantly improved the model's ability to generalize, particularly in later configurations where test accuracies exceeded 99%.
- **Clustering Complemented Supervised Learning:** The unsupervised clustering experiments aligned well with the binary classification tasks, validating the separability of the dataset's features.

These observations underscore the importance of a structured approach to experimentation, particularly in systematically addressing data, architecture, and training configuration.

7.2 Challenges Encountered

Despite the impressive results, the journey to high-performing models was not without challenges:

- **Flat Error Curves:** Many initial experiments resulted in flat error curves, indicating that the models were not learning effectively. This required careful re-evaluation of data preprocessing, architecture, and hyperparameters.
- **Computational Costs:** The grid search for hyperparameter tuning was computationally intensive, especially for larger architectures and higher epoch counts. Future work could leverage distributed computing or early stopping mechanisms to reduce training time.
- **Result Variability:** Variability in results due to random initialization and data splits highlighted the importance of setting fixed seeds for reproducibility.

- **Interpreting Clustering Results:** Mapping clusters to true classes introduced a layer of complexity in interpreting unsupervised results. While the majority-class mapping worked well, it may not generalize to datasets with more ambiguous cluster boundaries.

These challenges highlight the iterative and resource-intensive nature of neural network development, particularly when applied to real-world tasks.

7.3 Impact of Optimization and Tuning

The dramatic improvements achieved through systematic experimentation demonstrate the power of optimization:

- **Hyperparameter Tuning:** Fine-tuning the learning rate, number of neurons, and number of layers had a profound impact on model performance. The best configuration—incorporating SMOTE, SoftMax activation, and a learning rate of 0.001—achieved an accuracy of 99.51%. Experimenting with different splits even an accuracy of 100% was achieved, which would normally imply overfitting but given that the dataset is really simple and small it is very possible to be true.

- **Role of Activation Functions:** Using SoftMax activation in later experiments helped stabilize training and improved the network's ability to learn meaningful patterns.

- **Training Stability:** Once optimal parameters were identified, the models converged consistently, as evidenced by the smooth error curves and low test errors.

These findings emphasize the importance of a methodical approach to neural network design, particularly in balancing model complexity with training stability and efficiency.

7.4 Broader Implications

The methodologies and insights from this project have broader applicability:

- **Scalability:** The success of this relatively simple feedforward network suggests that similar approaches could be applied to other binary classification problems, such as fraud detection or medical diagnostics, provided the data is well-structured.

- **Feature Selection:** The effectiveness of the wavelet-transformed features underscores the importance of feature engineering and selection in achieving high classification performance.

- **Real-World Deployment:** While the current models achieve high accuracy, considerations for deployment—such as computational efficiency and real-time processing—must be addressed in future work.

- **Unsupervised Learning as a Complement:** The clustering experiments demonstrate the potential of unsupervised learning to enhance understanding of data structure, especially in scenarios with limited or noisy labels.

By combining supervised and unsupervised approaches, this project lays the groundwork for developing robust and interpretable machine learning systems.

7.5 Lessons Learned

The project provides several key takeaways for future work in neural network-based classification:

- **Systematic Experimentation is Key:** The iterative approach to tuning and evaluation was critical in identifying the best-performing models.
- **Visualization Tools are Indispensable:** Error plots, confusion matrices, and cluster visualizations were invaluable for diagnosing issues and understanding model behavior.
- **Flexibility and Adaptability:** While the models were tailored to this specific dataset, the methodologies and insights are broadly applicable to other classification tasks.
- **Reproducibility Matters:** Fixing random seeds and documenting experimental configurations ensured that results were consistent and reproducible.

These lessons highlight the importance of a disciplined and thoughtful approach to machine learning research and development.

Conclusion

Initially, the neural network models struggled to learn meaningful patterns, yielding low and stagnant accuracy scores despite various configurations. However, a systematic approach to hyperparameter tuning—coupled with robust data cleaning and normalization—ultimately enabled the discovery of model settings that delivered excellent predictive performance, with test accuracy reaching as high as 100%. The project also underscored the importance of visualization tools such as confusion matrices and error curves for diagnosing model behavior and guiding further improvements.

8 References

- Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [https://archive.ics.uci.edu/datas
  Irvine, CA: University of California, School of Information and Computer Science.
- Project code and experiments: `https://github.com/StratosDns/nn_f_t/blob/main/final_th20580.ipynb`