Neural Networks Project – Case Study Report

Banknote Authentication Using a Custom Neural Network Library

---

1. Introduction

In this phase of the project, we designed and implemented a **generic, reusable Neural Network (NN) library in Java**, independent of any specific problem. After completing the core library, we applied it to a **real-world classification problem** to validate its correctness and usability.

The selected case study is the **Banknote Authentication problem**, a binary classification task that determines whether a banknote is **genuine** or **counterfeit** based on numerical features extracted from wavelet transforms.

This case study demonstrates:

- Proper usage of the implemented NN library
- Clean separation between library code and application logic
- Configurable architecture and hyperparameters
- End-to-end training, evaluation, and prediction pipeline

---

2. Problem Description

Problem Statement

The goal is to classify banknotes as:

- **0 → Genuine**
- **1 → Counterfeit**

based on a set of numerical features derived from digital images of banknotes.

Why Neural Networks?

- The relationship between the features and the authenticity of banknotes is **non-linear**
- Neural Networks are well-suited for learning complex decision boundaries
- The dataset size is moderate, making it ideal for a fully connected feedforward network

---

3. Dataset Description

The **Banknote Authentication Dataset** contains numerical attributes extracted from wavelet-transformed images of banknotes.

Features

Each sample contains **4 input features**:

1. Variance of the wavelet transform
2. Skewness of the wavelet transform
3. Kurtosis of the wavelet transform
4. Entropy of image

Target

- Binary label:
    - `0` → Genuine banknote
    - `1` → Counterfeit banknote

Dataset Size

- Total samples: **1372**
- Input dimension: **4**
- Output dimension: **1**

The dataset is provided as a CSV file and included with the project submission.

4. Data Preprocessing

To ensure stable and efficient training, the following preprocessing steps were applied:

4.1 Normalization

- **Min–Max normalization** was applied to the input features
- This scales all features to the range **[0, 1]**
- Prevents features with large numeric ranges from dominating learning

4.2 Train/Test Split

- Dataset split into:
    - **80% training**
    - **20% testing**
- Optional shuffling was enabled to ensure random sampling

4.3 Input Validation

To handle invalid or missing inputs gracefully:

- Structural validation ensures correct dimensions
- Numerical validation replaces `NaN` or `Infinity` values with default values
- This prevents runtime failures and unstable gradients

---

5. Neural Network Architecture

The neural network architecture is **fully configurable via an external JSON configuration file**, eliminating the need for modifications to the source code.

Final Architecture Used

Layer	Neuros	Activation
Input Layer	4	—
Hidden Layer	16	ReLU
Hidden Layer	8	Tanh
Hidden Layer	4	ReLU
Hidden Layer	2	Tanh
Output Layer	1	Sigmoid

Design Justification

- **Sigmoid** output produces probabilities suitable for binary classification
- A single hidden layer is sufficient for this dataset

---

6. Training Configuration

The training process is controlled through `ModelConfig`.

Hyperparameter	Value
Learning Rate	0.01
Epochs	10
Batch Size	16
Optimizer	Stochastic Gradient Descent(SGD)
Loss Function	Binary Cross-Entropy
Weight Initialization	Xavier Initialization

Loss–Activation Pairing

- **Sigmoid + Cross-Entropy** was used
- This pairing provides stable gradients and faster convergence

---

7. Library Usage

The case study **does not hardcode problem-specific logic inside the library**.

Instead:

- The neural network is built dynamically using configuration files
- Layers, activations, and hyperparameters are injected externally
- Training and evaluation are handled through reusable components (`Trainer`, `Metrics`, `Utils`)

This demonstrates that the library can be reused for **any supervised learning task** with minimal changes.

---

8. Training and Evaluation Results

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

Evaluation Metrics

- **Accuracy**
- **True Positives (TP)**
- **True Negatives (TN)**
- **False Positives (FP)**
- **False Negatives (FN)**

9. Handling Invalid or Missing Inputs

The system gracefully handles invalid or missing inputs using:

- Dataset validation utilities
- Input sanitization for single samples and batches
- Default replacement for invalid numerical values

This design ensures:

- Stable training and inference
- Clear error reporting for structural issues
- Robustness against noisy data

---

10. Runnable Demo

A runnable demo is provided via:

- A `main` class (`BanknoteCaseStudy`)
- External configuration file
- Dataset included in the project

The demo can be executed using standard Maven commands as described in the README.
