Binary Classification Problem

The goal of this case study is to solve a binary classification problem using a fully connected neural network implemented from scratch in Java.
The task is to classify banknote data into genuine or forged based on numerical features extracted from images of banknotes.

Rationale

This problem was selected because:

- The dataset is clean, numerical, and suitable for testing a custom neural network.
- It allows evaluation of the full machine learning pipeline: preprocessing, training, optimization, and evaluation.
- It demonstrates the correctness and flexibility of the implemented neural network library

---

Dataset Description

The dataset consists of multiple samples, where:

- Each sample contains several numerical features (input vector).
- The target label is binary (0 or 1), representing the class of the banknote.

The dataset is stored in CSV format and loaded using a custom CSVLoader.

Preprocessing Steps

1. **CSV Loading**
    - Data is read from a CSV file.
    - Optional header handling is supported.
    - The last column is treated as the target label.
2. **Validation**
    - Input and target dimensions are validated using DatasetValidator.
3. **Normalization**
    - Min-Max normalization is applied to all input features:
    - Normalized_X = (x - min) / (max - min)
    - This improves training stability and convergence speed.
4. **Train-Test Split**
    - The dataset is split into training and testing sets using TrainTestSplit.
    - Optional shuffling is applied to avoid bias

Neural Network Architecture Choice and Justification

Architecture

- Fully connected feedforward neural network.
- Consists of:
  - Input layer matching feature size.
  - Two hidden layers (8, 16) neurons.
  - Output layer with a single neuron (binary classification).

Each layer is composed of multiple Neuron objects, each holding its own weights and bias.

- **Activation Functions**
  - Hidden layers: ReLU.
  - Output layer: Sigmoid.
- **Weight Initialization**
  - "He" Initialization supported for ReLU-based networks.
- **Loss Function**
  - Binary Cross-Entropy.
- **Optimizer**
  - SGD.

Justification:

- **Hidden Layers (8, 16 neurons):** Provides enough capacity to capture feature interactions.
- **Output Layer (1 neuron):** Matches binary classification output for **Genuine** vs **Counterfeit** banknotes.
- **Hidden Activation (ReLU):** Introduces non-linearity and enables faster convergence.
- **Output Activation (Sigmoid):** Produces probability between 0 and 1 for binary classification.
- **Weight Initialization (He):** Maintains stable variance with ReLU to prevent vanishing/exploding activations.
- **Loss Function (Binary Cross-Entropy):** Measures prediction error appropriately for probabilistic binary outputs.
- **Optimizer (SGD):** Provides simple and effective weight updates suitable for small datasets.

Final Training and Evaluation Results

Training Results

- Training performed for **10 epochs**.
- Batch training with gradient accumulation.
- Loss consistently decreased across epochs.

Example Training Output:

Epoch 1/10 - Loss: 0.6611466509938588

Epoch 2/10 - Loss: 0.6062601427302694

Epoch 3/10 - Loss: 0.5083842332919211

Epoch 4/10 - Loss: 0.3930244062543217

Epoch 5/10 - Loss: 0.3015299338253051

Epoch 6/10 - Loss: 0.22920740685463092

Epoch 7/10 - Loss: 0.16666239821395665

Epoch 8/10 - Loss: 0.12871045931572841

Epoch 9/10 - Loss: 0.10648518072620611

Epoch 10/10 - Loss: 0.08312142398214176

Evaluation Results

- Accuracy: **~96.8%**
- Confusion Matrix:
  - True Positives: 128
  - False Positives: 2
  - True Negatives: 142
  - False Negatives: 3

These results indicate strong generalization and correct implementation of training and backpropagation

---

Explanation of How the Custom Library Was Used

The neural network was built and trained using a **custom Java neural network library**, designed from scratch.

Key components used:

- Initializer interface (Xavier, He, RandomUniform).
- Activation functions (ReLU, Sigmoid, Tanh, Linear).
- Layer and Neuron classes for forward and backward propagation.

- Optimizers (SGD, SGDWithMomentum, RMSProp).
- Trainer class handling epochs, batching, shuffling, and metrics.
- Factory pattern (ActivationFactory, InitializerFactory, etc.) to build components from configuration files.

The entire experiment is configurable via a JSON configuration file, making the system flexible and reusable

---

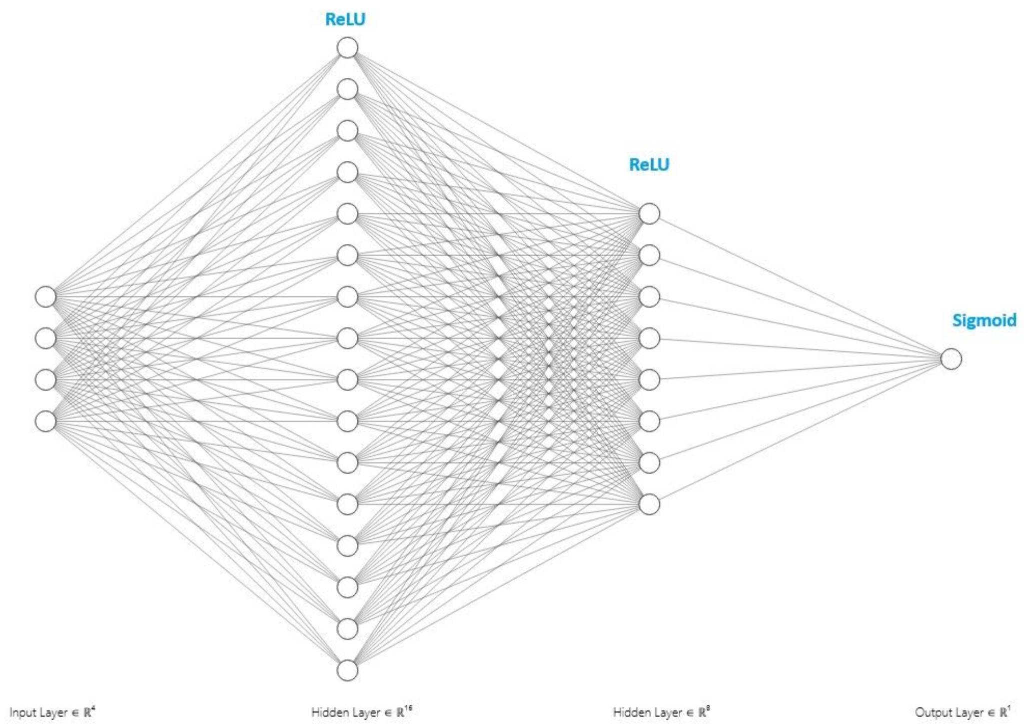
Tables:

Item	**Value**
Dataset	Banknote Authentication
Samples	1,372
Features	4
Classes	2 (Genuine / Counterfeit)
Train/Test Split	80% / 20%

Component	**Value**
Hidden Layers	2
Hidden Activation	ReLU
Output Activation	Sigmoid
Loss	Cross Entropy
Initializer	He
Optimizer	SGD
Learning Rate	0.005
Batch Size	16
Epochs	10

Metric	**Value**
Accuracy	96.8%
Best Accuracy (Epoch)	98.7% (Epoch 32)
Final Loss	0.083

Graphs:



ReLU
ReLU
Sigmoid
Input Layer ∈ ℝ⁴
Hidden Layer ∈ ℝ¹⁶
Hidden Layer ∈ ℝ⁸
Output Layer ∈ ℝ¹

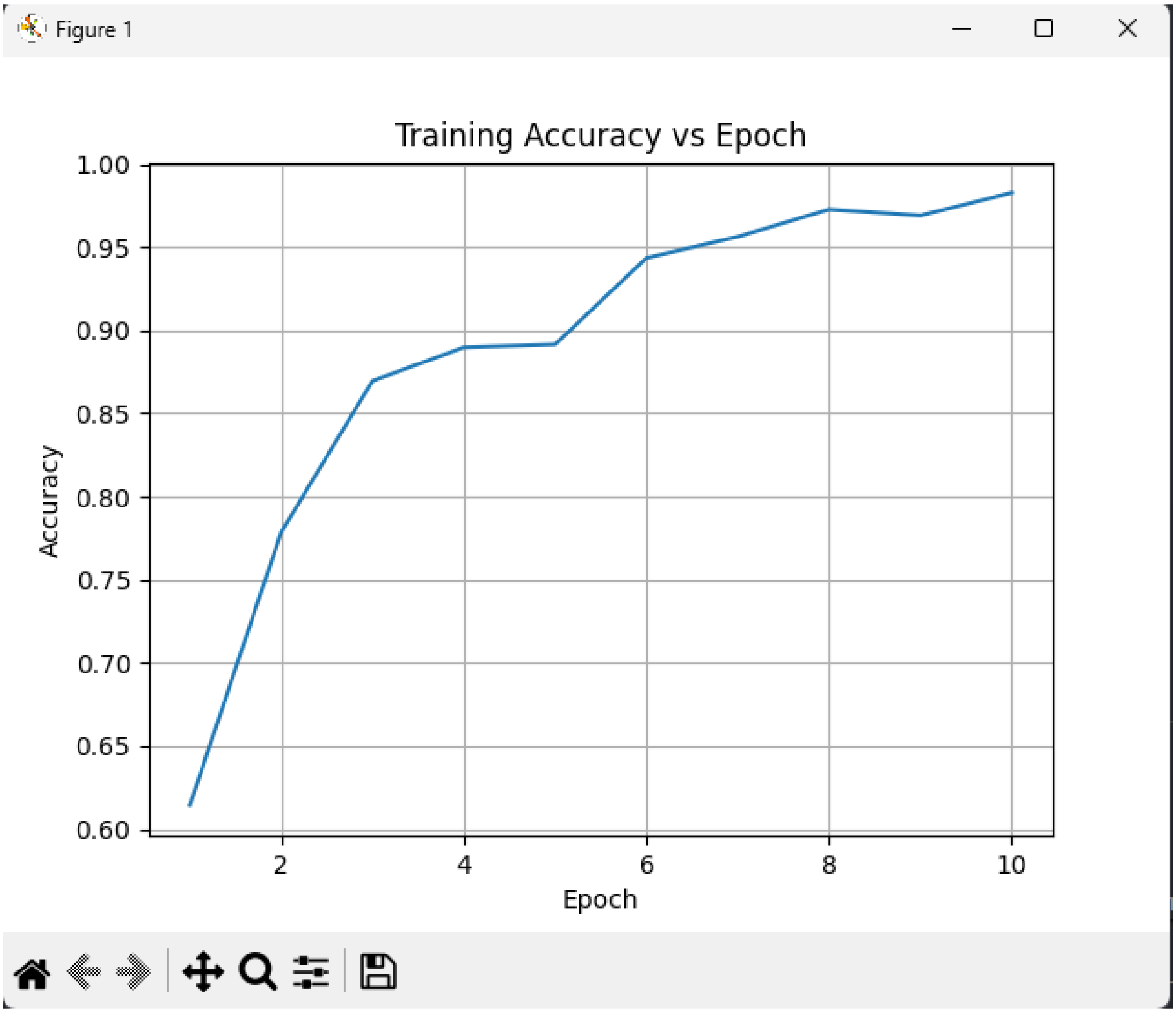


Figure 1
Training Accuracy vs Epoch
Accuracy
Epoch
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
2
4
6
8
10

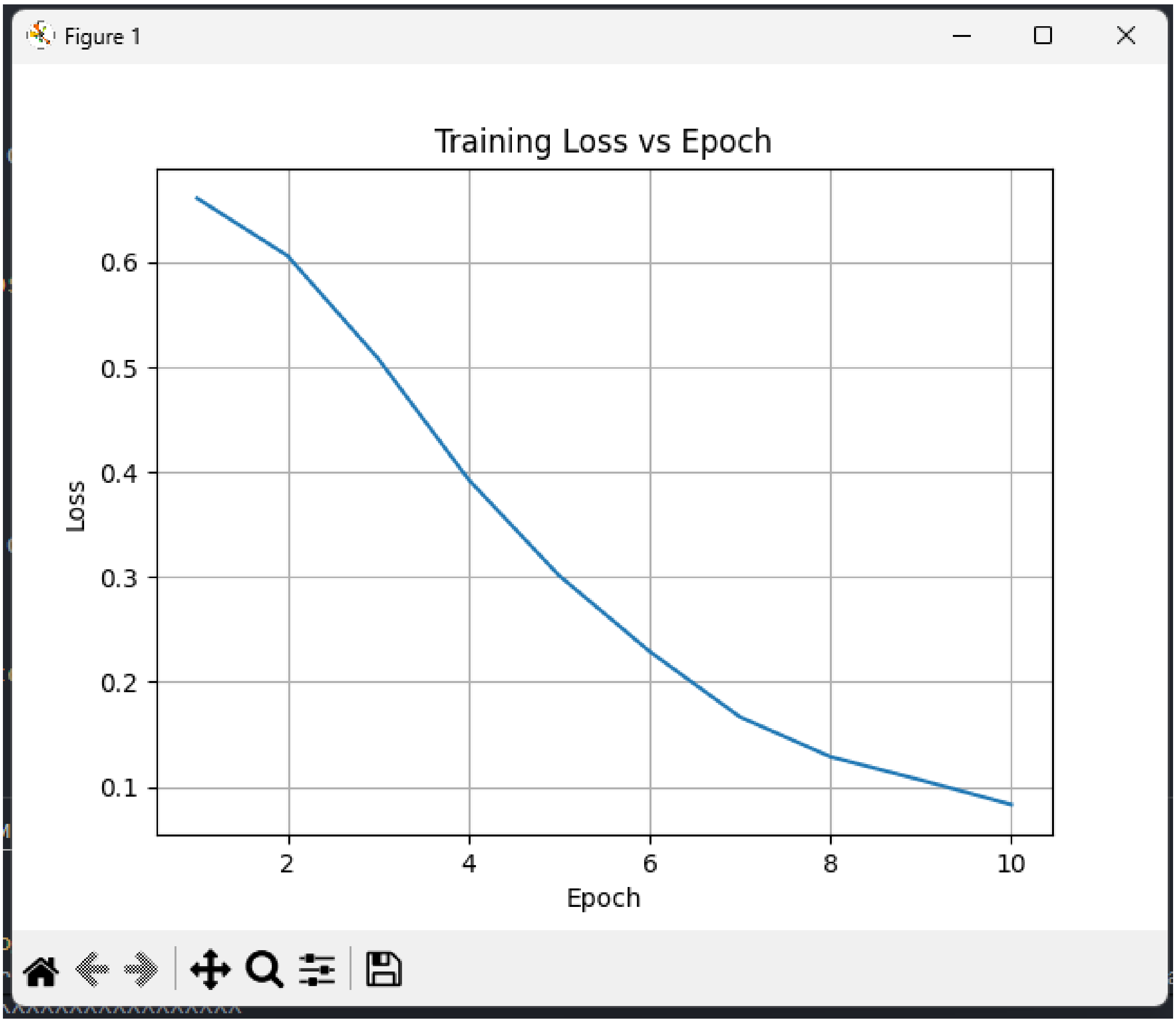


Figure 1
Training Loss vs Epoch
0.6
0.5
0.4
0.3
0.2
0.1
Loss
2
4
6
8
10
Epoch

Screenshots:

training_metrics.csv > data

```
epoch,loss,accuracy
1,0.6611466509938588,0.6144029170464904
2,0.6062601427302694,0.7784867821330902
3,0.5083842332919211,0.8696444849589791
4,0.3930244062543217,0.8896991795806746
5,0.3015299338253051,0.8915223336371924
6,0.22920740685463092,0.9434822242479489
7,0.16666239821395665,0.9562443026435734
8,0.12871045931572841,0.9726526891522334
9,0.10648518072620611,0.9690063810391978
10,0.08312142398214176,0.9826800364630811
```

```
Nour@LAPTOP-NNAA1CUS MINGW64 /d/programming/java/soft-computing-project/phase3/
$  cd d:\\programming\\java\\soft-computing-project\\phase3\\softComputingProje
win32-x86_64\\bin\\java.exe @C:\\Users\\Nour\\AppData\\Local\\Temp\\cp_9vsrptg4
Epoch 1/10 - Loss: 0.6611466509938588
Epoch 2/10 - Loss: 0.6062601427302694
Epoch 3/10 - Loss: 0.5083842332919211
Epoch 4/10 - Loss: 0.3930244062543217
Epoch 5/10 - Loss: 0.3015299338253051
Epoch 6/10 - Loss: 0.22920740685463092
Epoch 7/10 - Loss: 0.16666239821395665
Epoch 8/10 - Loss: 0.12871045931572841
Epoch 9/10 - Loss: 0.10648518072620611
Epoch 10/10 - Loss: 0.08312142398214176

=== Evaluation Results ===
Accuracy: 0.9818181818181818
TP: 128   FP: 2
TN: 142   FN: 3
```