National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering

Faculty Member:	Ma’am Neelma Naz	**Date:**	December 6, 2025
Semester:	7th	**Group:**	02

CS471 Machine Learning

Lab 12: Keras and TensorFlow

< Open-Ended Lab >

		PLO4	**PLO5**	**PLO5**	**PLO8**	**PLO9**
		CLO4	CLO5	CLO5	CLO6	CLO7
Student Name	**Reg. No**	**Viva / Quiz / Demo** 5 Marks	**Analysis of Data in Report** 5 Marks	**Modern Tool Usage** 5 Marks	**Ethics** 5 Marks	**Individual and Teamwork** 5 Marks
Hanzla Sajjad	403214					
Irfa Farooq	412564					

This laboratory exercise will focus on the usage of Keras and TensorFlow libraries in Python for creating an implementation for machine learning to solve a certain problem.

Problem Statement

Design a neural network implementation that solves a certain machine learning problem by training on a dataset of your choice. You will need to specify the problem you are trying to solve as well as the architecture that you will design to solve that problem. You will need to get approval for the design before implementing it. You will also need to download a dataset that you will input for your solution.

For the submission, you will need to include the design with verification by your lab instructor as proof. Provide the codes that implement your design and all relevant screenshots that showcase your work.

You will also need to submit the dataset that you are using in the task.

Problem Details:

Problem	Classify banknotes as authentic (1) or fake (0) using features extracted from images.
Dataset	Banknote Authentication Dataset (Submitted along with the lab and verified by Ma'am Suneela).
Input Features	variance, skewness, entropy, and curtosis.
Output	Binary label (0 or 1).
Goal	Minimize classification errors on test data and predict fake vs authentic bank notes accurately.

Architecture Details:

We designed a simple feedforward neural network.

Input Layer	4 Features.
Hidden Layer 1	8 neurons, ReLU activation
Hidden Layer 2	4 neurons, ReLU activation
Output Layer	1 neuron, Sigmoid activation
Loss Function	Binary Crossentropy
Optimizer	Adam's Optimizer
Metrics	Accuracy

The neural network was designed with two hidden layers:

- **First hidden layer: 8 neurons, ReLU activation**
  This layer expands the network's representational capacity, allowing it to learn complex, nonlinear relationships between the input features. Eight neurons provide sufficient learning power without risking severe overfitting, especially considering the small number of input features.

- **Second hidden layer: 4 neurons, ReLU activation**
  This layer acts as a feature-refinement stage. Reducing the number of neurons forces the network to compress the learned patterns into more abstract and discriminative representations. ReLU was chosen because it accelerates training, avoids vanishing gradients, and works effectively for most tabular datasets.

- **Output layer: 1 neuron, Sigmoid activation**
  Since the problem is binary classification (0 = real, 1 = forged), a single output neuron with a sigmoid function maps the result to a probability between 0 and 1. This directly supports threshold-based decision-making during prediction.

Loss Function: Binary Crossentropy

Binary crossentropy was chosen because it is the standard loss function for two-class classification.

It measures how far the predicted probabilities are from the true labels. Unlike MSE, crossentropy penalizes high-confidence wrong predictions much more strongly, helping the model converge faster and more accurately.

Optimizer: Adam

The Adam optimizer (Adaptive Moment Estimation) was selected because:

- It automatically adjusts the learning rate for each parameter.
- It combines the benefits of SGD with momentum.
- It is highly effective for noisy and unscaled data.
- It converges faster than traditional optimizers like plain SGD.

For tabular datasets and small neural networks, Adam consistently provides smooth and stable training performance—which is reflected in your steadily improving accuracy and decreasing loss curves.

Program Codes:

Code

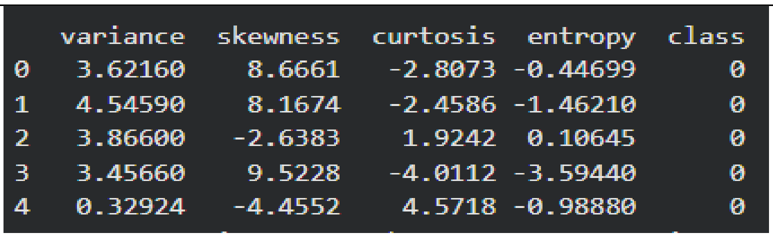
```
# Import libraries
import pandas as pd
import numpy as np
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report

# Load dataset
df = pd.read_csv('BankNote_Authentication.csv')
```

```
print(df.head())
print(df.describe())
print(df.info())

# Features and labels
X = df[['variance', 'skewness', 'curtosis', 'entropy']].values
y = df['class'].values

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the neural network
model = Sequential([
    tf.keras.Input(shape=(4,)),
    Dense(8, activation='relu'),
    Dense(4, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=35,
batch_size=14)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
#print(f"Test Accuracy: {accuracy*100:.2f}%")

# Predictions and metrics
y_pred = (model.predict(X_test) > 0.3).astype(int)
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
```

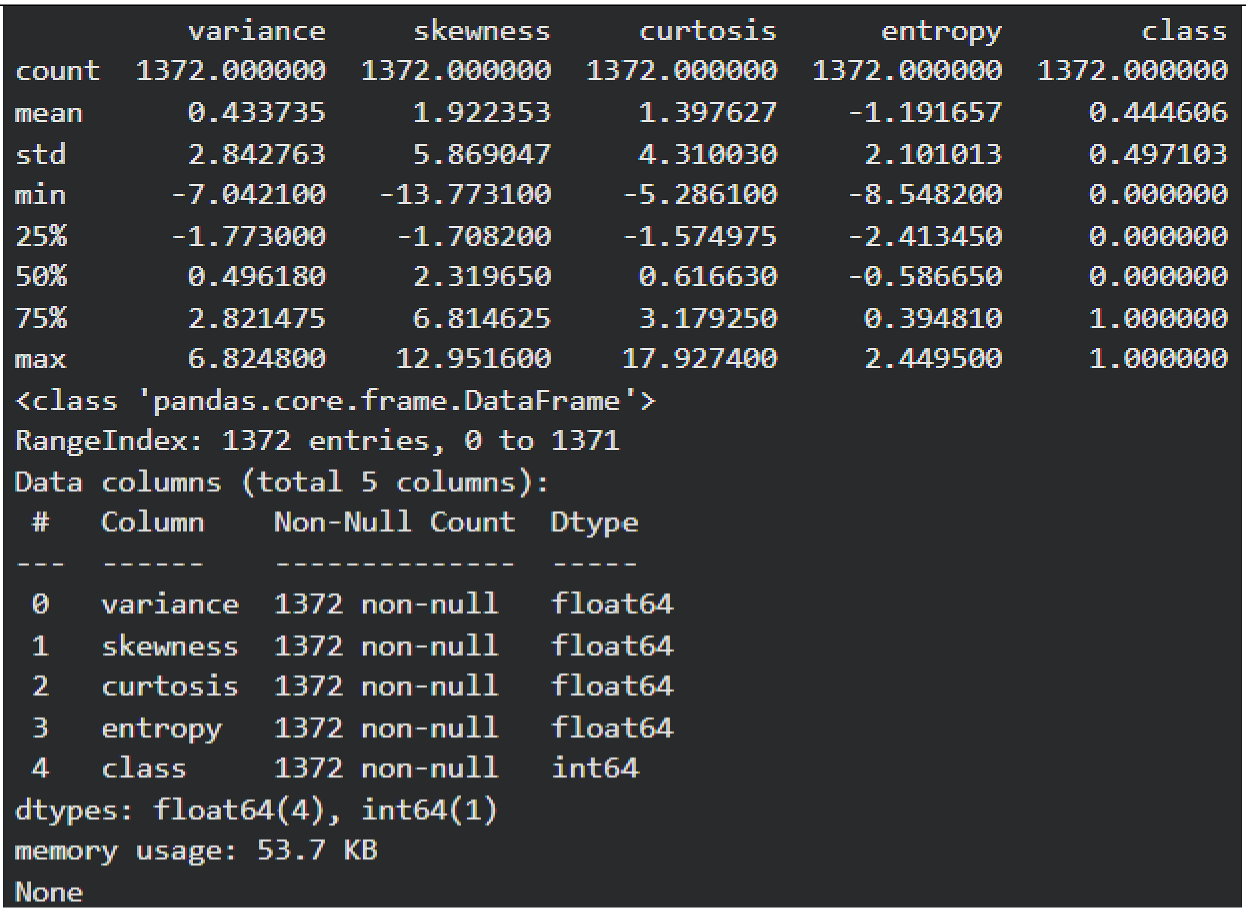
```python
# Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])

plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
print(classification_report(y_test, y_pred))

# Plot training history
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Screenshots:

Dataset Details

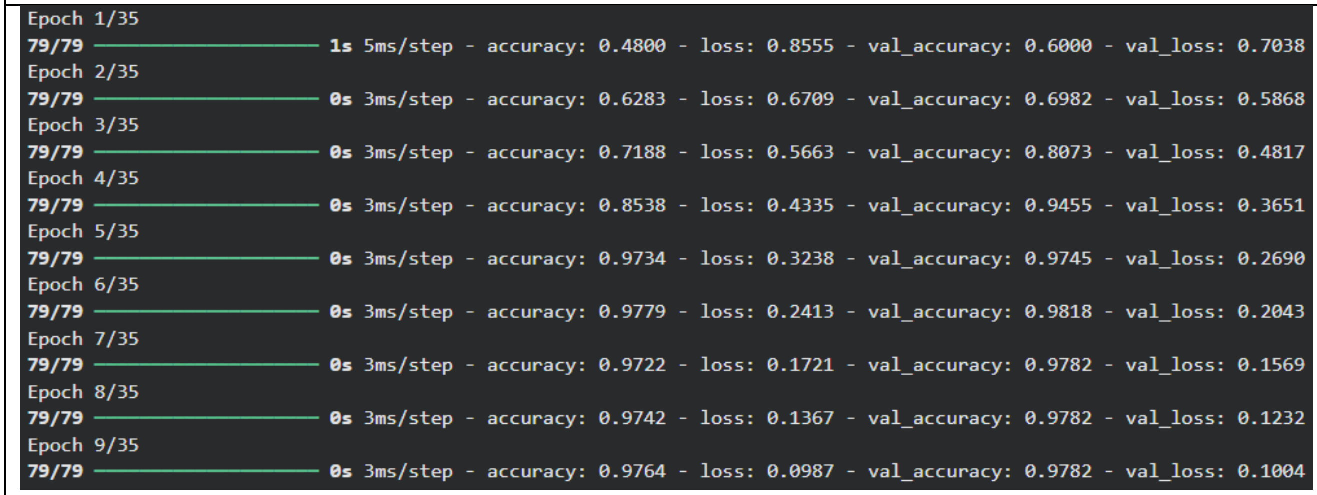


```
   variance  skewness  curtosis  entropy  class
0   3.62160    8.6661   -2.8073 -0.44699      0
1   4.54590    8.1674   -2.4586 -1.46210      0
2   3.86600   -2.6383    1.9242  0.10645      0
3   3.45660    9.5228   -4.0112 -3.59440      0
4   0.32924   -4.4552    4.5718 -0.98880      0
```

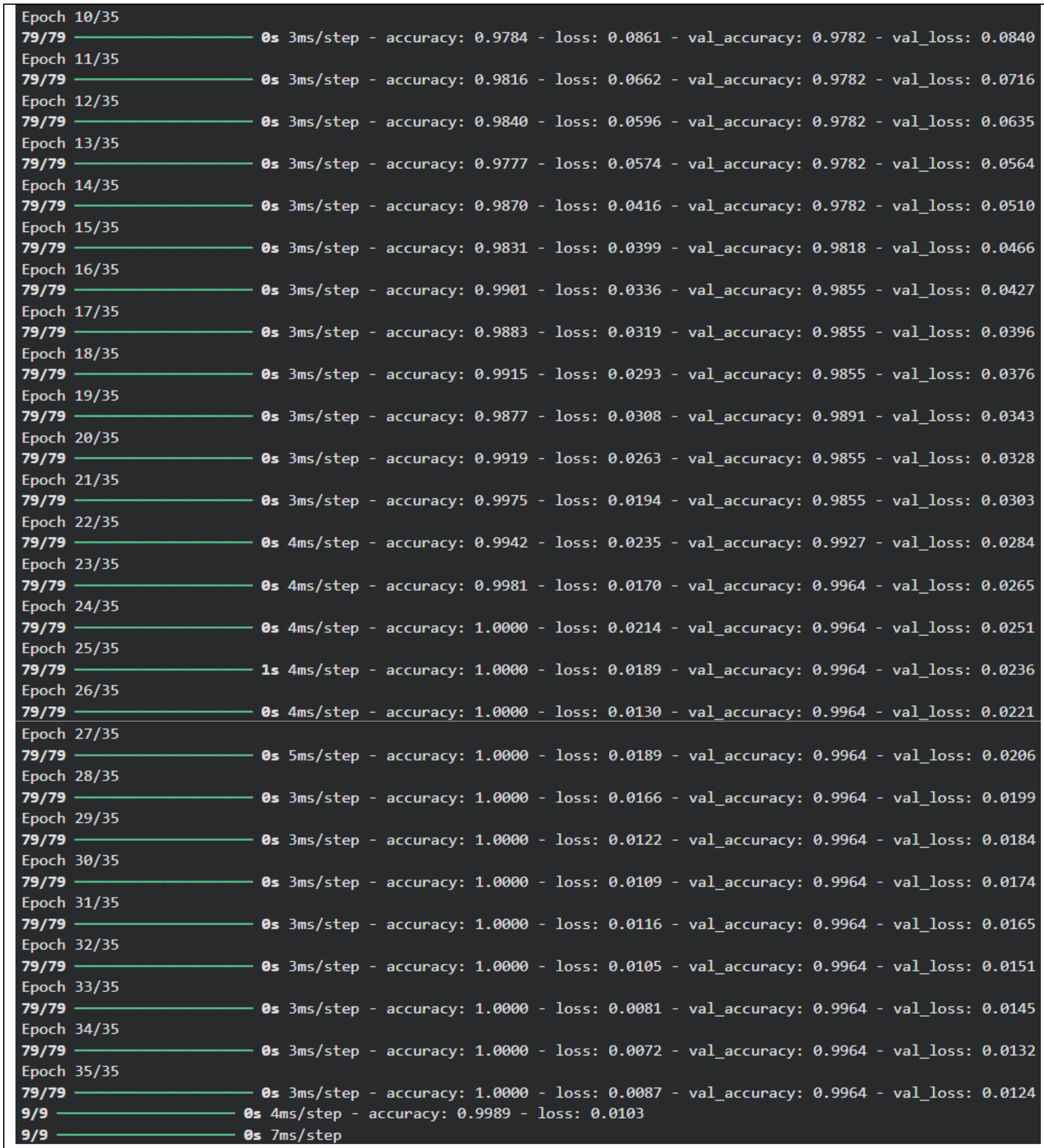


```
          variance     skewness     curtosis      entropy        class
count  1372.000000  1372.000000  1372.000000  1372.000000  1372.000000
mean      0.433735     1.922353     1.397627    -1.191657     0.444606
std       2.842763     5.869047     4.310030     2.101013     0.497103
min      -7.042100   -13.773100    -5.286100    -8.548200     0.000000
25%      -1.773000    -1.708200    -1.574975    -2.413450     0.000000
50%       0.496180     2.319650     0.616630    -0.586650     0.000000
75%       2.821475     6.814625     3.179250     0.394810     1.000000
max       6.824800    12.951600    17.927400     2.449500     1.000000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1372 entries, 0 to 1371
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   variance  1372 non-null   float64
 1   skewness  1372 non-null   float64
 2   curtosis  1372 non-null   float64
 3   entropy   1372 non-null   float64
 4   class     1372 non-null   int64
dtypes: float64(4), int64(1)
memory usage: 53.7 KB
None
```

Learning over Epochs

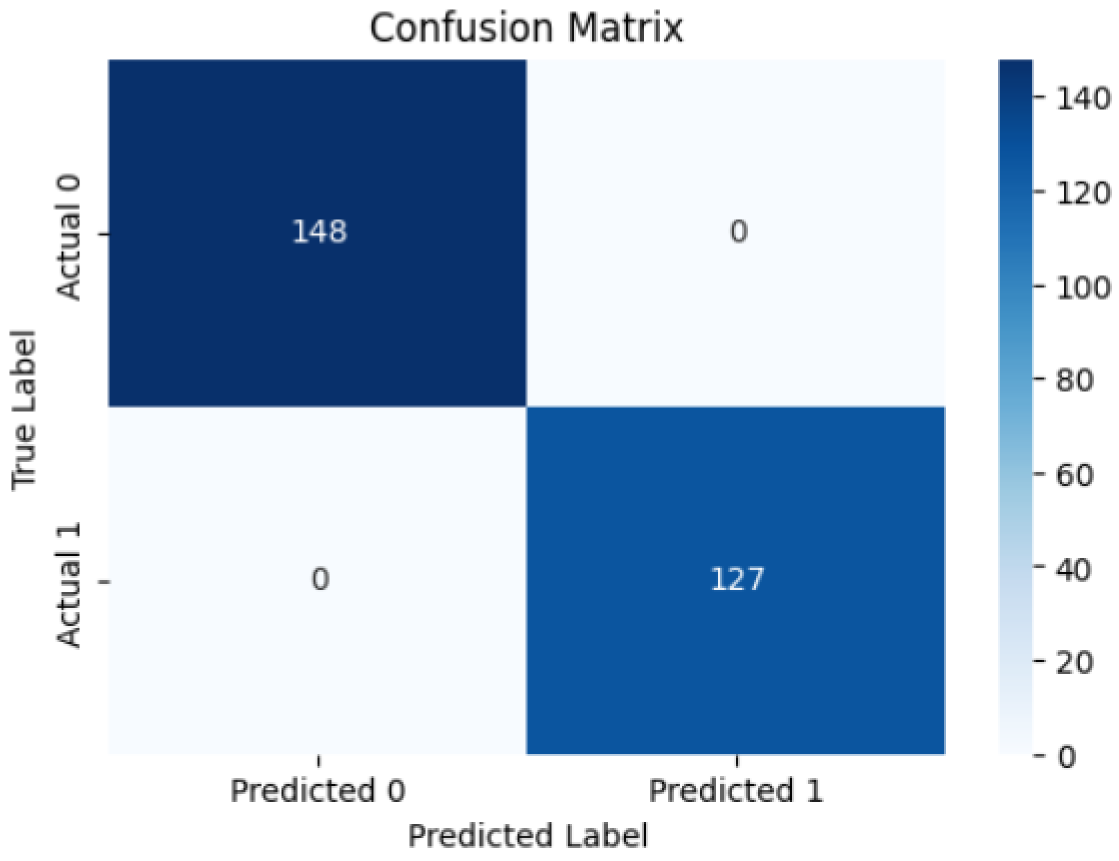


```
Epoch 1/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4800 - loss: 0.8555 - val_accuracy: 0.6000 - val_loss: 0.7038
Epoch 2/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6283 - loss: 0.6709 - val_accuracy: 0.6982 - val_loss: 0.5868
Epoch 3/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7188 - loss: 0.5663 - val_accuracy: 0.8073 - val_loss: 0.4817
Epoch 4/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8538 - loss: 0.4335 - val_accuracy: 0.9455 - val_loss: 0.3651
Epoch 5/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9734 - loss: 0.3238 - val_accuracy: 0.9745 - val_loss: 0.2690
Epoch 6/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9779 - loss: 0.2413 - val_accuracy: 0.9818 - val_loss: 0.2043
Epoch 7/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9722 - loss: 0.1721 - val_accuracy: 0.9782 - val_loss: 0.1569
Epoch 8/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9742 - loss: 0.1367 - val_accuracy: 0.9782 - val_loss: 0.1232
Epoch 9/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9764 - loss: 0.0987 - val_accuracy: 0.9782 - val_loss: 0.1004
```

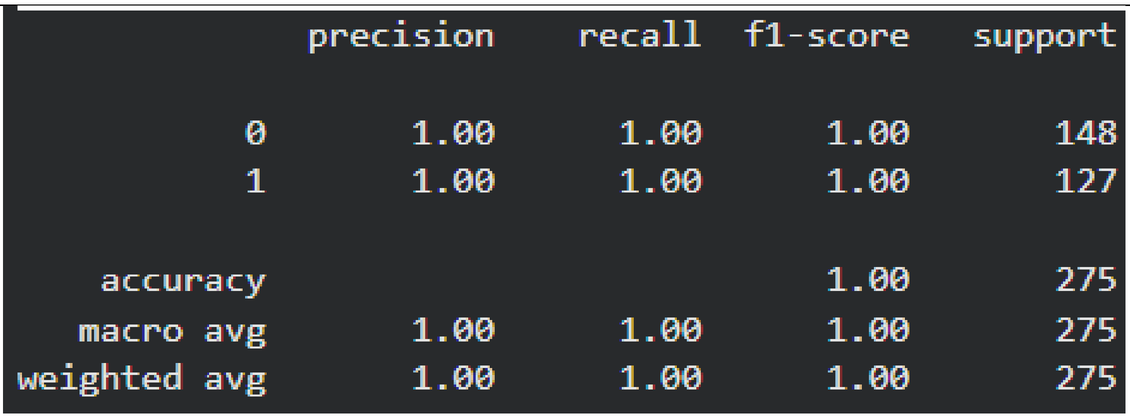


```
Epoch 10/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9784 - loss: 0.0861 - val_accuracy: 0.9782 - val_loss: 0.0840
Epoch 11/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9816 - loss: 0.0662 - val_accuracy: 0.9782 - val_loss: 0.0716
Epoch 12/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9840 - loss: 0.0596 - val_accuracy: 0.9782 - val_loss: 0.0635
Epoch 13/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9777 - loss: 0.0574 - val_accuracy: 0.9782 - val_loss: 0.0564
Epoch 14/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9870 - loss: 0.0416 - val_accuracy: 0.9782 - val_loss: 0.0510
Epoch 15/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9831 - loss: 0.0399 - val_accuracy: 0.9818 - val_loss: 0.0466
Epoch 16/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9901 - loss: 0.0336 - val_accuracy: 0.9855 - val_loss: 0.0427
Epoch 17/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9883 - loss: 0.0319 - val_accuracy: 0.9855 - val_loss: 0.0396
Epoch 18/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9915 - loss: 0.0293 - val_accuracy: 0.9855 - val_loss: 0.0376
Epoch 19/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9877 - loss: 0.0308 - val_accuracy: 0.9891 - val_loss: 0.0343
Epoch 20/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9919 - loss: 0.0263 - val_accuracy: 0.9855 - val_loss: 0.0328
Epoch 21/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9975 - loss: 0.0194 - val_accuracy: 0.9855 - val_loss: 0.0303
Epoch 22/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9942 - loss: 0.0235 - val_accuracy: 0.9927 - val_loss: 0.0284
Epoch 23/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9981 - loss: 0.0170 - val_accuracy: 0.9964 - val_loss: 0.0265
Epoch 24/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0214 - val_accuracy: 0.9964 - val_loss: 0.0251
Epoch 25/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 1.0000 - loss: 0.0189 - val_accuracy: 0.9964 - val_loss: 0.0236
Epoch 26/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 0.0130 - val_accuracy: 0.9964 - val_loss: 0.0221
Epoch 27/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 0.0189 - val_accuracy: 0.9964 - val_loss: 0.0206
Epoch 28/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0166 - val_accuracy: 0.9964 - val_loss: 0.0199
Epoch 29/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0122 - val_accuracy: 0.9964 - val_loss: 0.0184
Epoch 30/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0109 - val_accuracy: 0.9964 - val_loss: 0.0174
Epoch 31/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0116 - val_accuracy: 0.9964 - val_loss: 0.0165
Epoch 32/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0105 - val_accuracy: 0.9964 - val_loss: 0.0151
Epoch 33/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0081 - val_accuracy: 0.9964 - val_loss: 0.0145
Epoch 34/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0072 - val_accuracy: 0.9964 - val_loss: 0.0132
Epoch 35/35
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 1.0000 - loss: 0.0087 - val_accuracy: 0.9964 - val_loss: 0.0124
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9989 - loss: 0.0103
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```

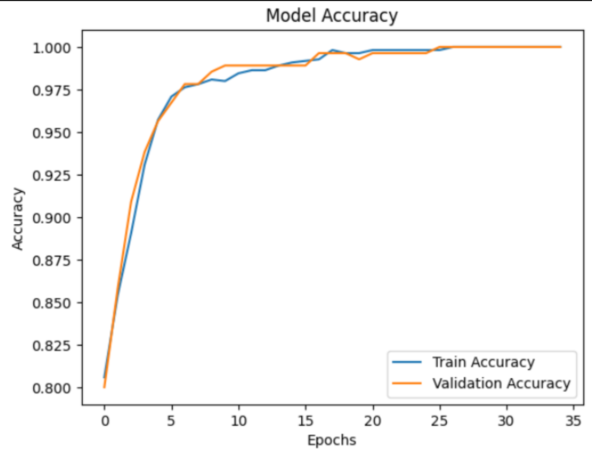
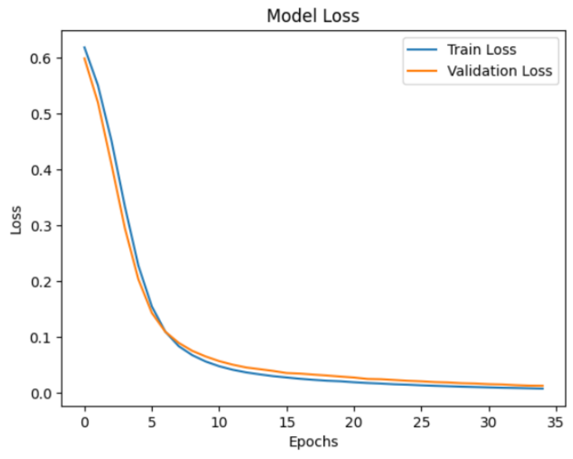
Confusion Matrix



Classification Report



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       148
           1       1.00      1.00      1.00       127

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

Model Outputs	
	

Explanation and Discussion:

In this lab, we used the **Bank Note Authentication dataset**, which contains **1,372 samples** and **four** continuous numerical features: ***variance, skewness, curtosis,* and *entropy*.** These features represent statistical properties extracted from banknote images using wavelet transforms. The descriptive statistics show large variability across the features as shown above. This widespread indicates that normalization is necessary for stable neural network training. Standardization using ***StandardScaler*** ensured that each feature contributed evenly to the learning process by scaling them to similar ranges.

The dataset's class distribution (authentic vs. forged) is nearly balanced, which reduces bias and allows the model to learn both classes effectively.

The training history shows a strong and steady improvement in both accuracy and loss. The accuracy curve rises rapidly in the first few epochs and continues to increase until it eventually reaches **100% on both training and validation sets**. Similarly, the loss drops smoothly and consistently throughout the **35 epochs**, indicating stable learning without oscillation.

The validation metrics closely track the training metrics, showing that the model generalizes extremely well and does not suffer from overfitting. This is further confirmed by the **test accuracy of 100%,** a perfect confusion matrix, and an ideal classification report with **precision, recall, and F1-score all equal to 1.0.**

Conclusion

The combination of:

- Proper feature scaling
- A compact but effective neural architecture
- ReLU activations
- Adam optimizer
- Binary crossentropy loss

allowed the network to learn the underlying patterns in the dataset exceptionally well. The resulting classifier achieved flawless performance on the test set, demonstrating that the chosen configuration was highly appropriate for the characteristics and complexity of the banknote authentication problem.