DEEP LEARNING – CASE-STUDY

B. Harsha Vardhan – VU21CSEN0500003

G. Venkat Swaraj – VU21CSEN0500050

S. Kailash – VU21CSEN0500053

T.V.S. Sumanth – VU21CSEN0500074

Introduction:

a. Problem Description: Predict whether a loan will get approved using the Loan Prediction Dataset.

b. Literature Study: Explored existing loan prediction models done with feed-forward, their methodology, and the application of neural networks in enhancing accuracy and predictive capabilities.

Requirements Elicitation: Loan Prediction Dataset.

Problem Modelling:

Multilayer Perceptron:

- Using sklearn, we create a multi-layer perceptron classifier.
- We have selected MLP architecture with two hidden layers with 100 and 50 neurons, respectively.
- Activation Functions: The default activation function in MLPClassifier is ReLU. We have experimented with other activation functions like tanh, and logistic function, but the performance of the model did not improve.
- Without explicitly setting the alpha parameter, the MLPClassifier uses a default value for regularisation. This default regularisation helps prevent overfitting by penalising large weights in the network. ***Default value of alpha =0.0001***
- MLP Classifier is not performing well, so we used the LSTM model.

Long Short Term Memory:

- The architecture of the LSTM model consists of an LSTM layer, dropout regularisation, and a dense layer with sigmoid activation for binary classification.
- The model is compiled with the Adam optimiser and binary cross-entropy loss function.
- An 80-20 split divides the dataset into training and testing sets, with 80 going to training and 20 to testing.
- A 10% validation split is used to validate the LSTM model after it has been trained on the training data with a batch size of 32.
- Training progress is monitored over 50 epochs, and model performance metrics are recorded.
- The performance of the trained LSTM model is tested using the test data.
- Test accuracy, precision, recall, F1 score, and overall accuracy are some of the evaluation indicators.

System Design:

a. Import Libraries: Imported required libraries, including TensorFlow, pandas, and scikit-learn.

b. Load Dataset: Loaded the loan prediction dataset using pandas.

c. Preprocess Data:

- Identified the target variable (Loan_Status) and features (independent variables).
- Handled missing values and categorical variables by dropping and one hot encoding them.
- Split the dataset into features (X) and target variables (y).

d. Split Data: Split the dataset into training and testing sets using train_test_split from scikit-learn.

e. Standardise Features: Standardized the features using StandardScaler from scikit-learn to ensure all features have the same scale.

f. Build Neural Network Model:

- Utilising the sklearn, we created the MLPClassifier neural network.
- Input Layer: The number of columns in the feature matrix X_train determines the amount of input features.
- Hidden Layers: There are two hidden layers, each with fifty and one hundred neurons. By default, these layers make use of the ReLU activation function.
- Output Layer: One neuron makes up the output layer, which predicts the loan status (classification binary: 0 or 1). The output layer generates loan approval probability by utilising the sigmoid activation function.

g. Train Model:

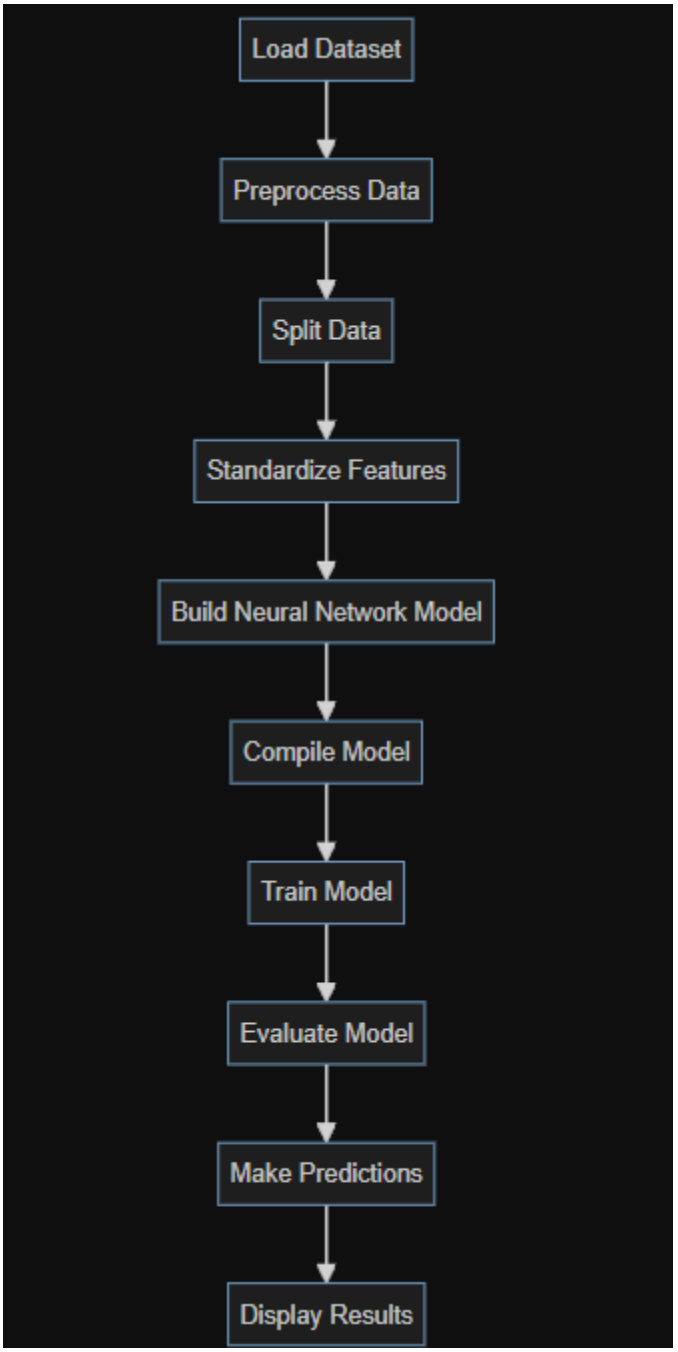
- Train the model on the training data using the fit method.
- We specified the max_iter as 140 and batch size as auto.

h. Evaluate Model:

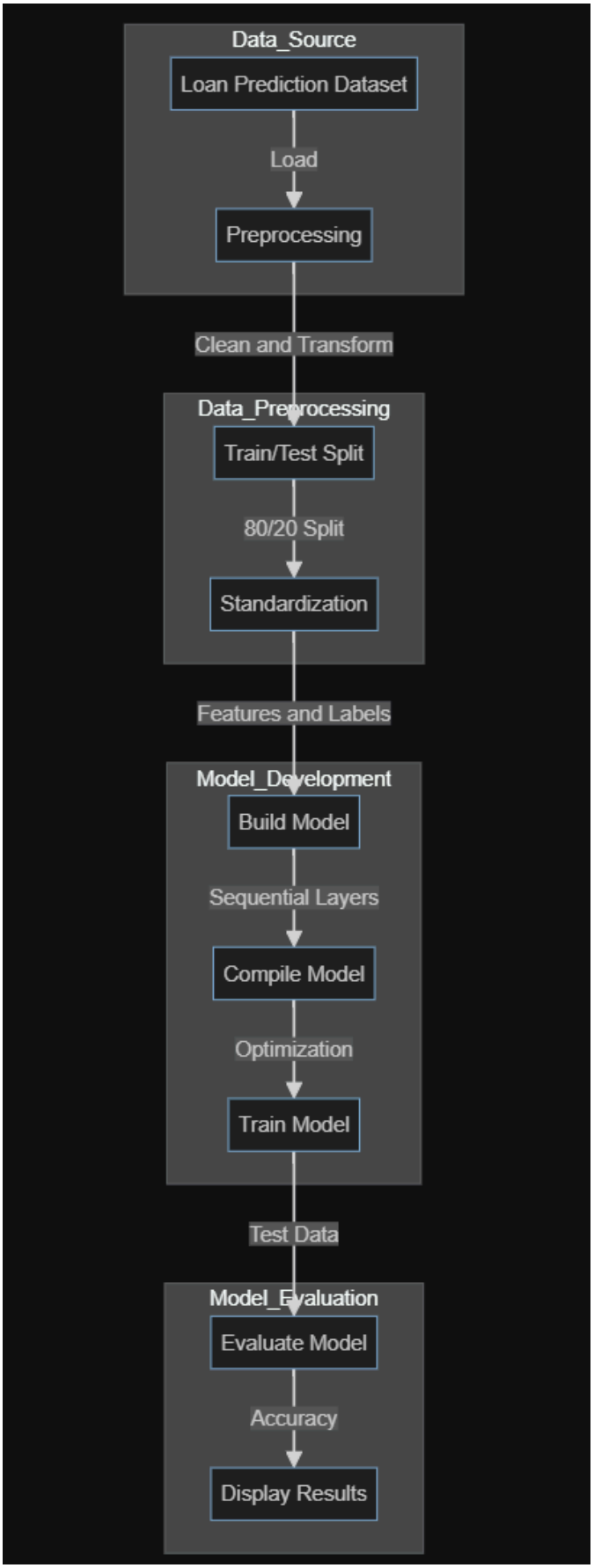
- Evaluate the model on the testing data to assess its performance.
- Use accuracy as the evaluation metric.
- We also used the F1 Score, precision, and recall to evaluate the model.

i. Make Predictions: The trained model takes input features (X_test) and returns the predicted labels.

Flowchart for FEED FORWARD NETWORK):



Data Flow Diagram for FEED FORWARD NETWORK:



Implementation of feed-forward network:

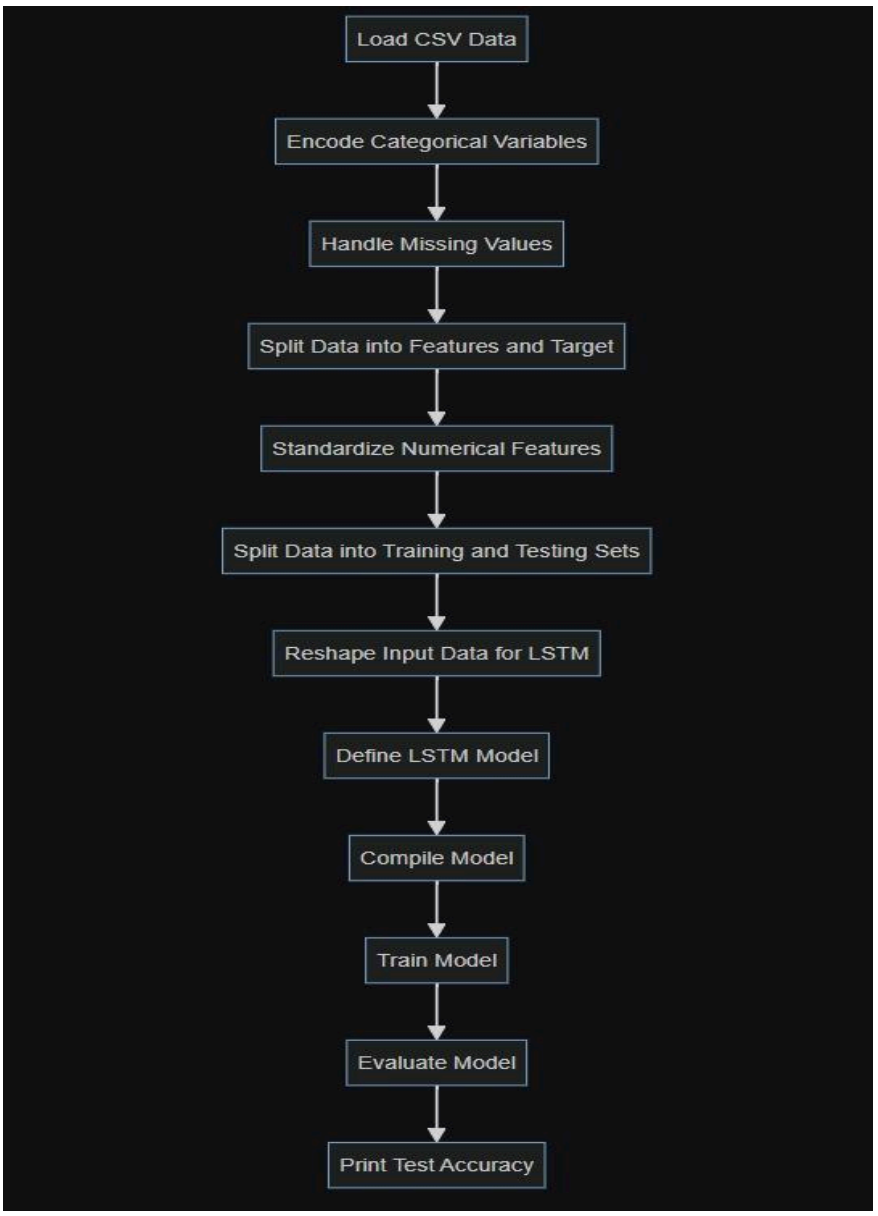
CODE:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
data = pd.read_csv('Loan_Dataa.csv')
data['Loan_Status'] = data['Loan_Status'].map({'Y': 1, 'N': 0})
data = pd.get_dummies(data, columns=['Gender', 'Married', 'Education',
'Self_Employed', 'Property_Area'])
scaler = StandardScaler()
numerical_cols = ['ApplicantIncome', 'CoapplicantIncome', 'Loan_Amount_Term',
'Credit_History']
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])
data=data.dropna()
y=data['Loan_Status']
X=data.drop(["Loan_ID","Loan_Status"], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=111)
mlp_clf = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=140,
random_state=111)
mlp_clf.fit(X_train, y_train)
mlp_predictions_train = mlp_clf.predict(X_train)
mlp_predictions_test = mlp_clf.predict(X_test)
test_accuracy_MLP=accuracy_score(y_test,mlp_predictions_test)
test_accuracy_MLP= test_accuracy_MLP*100
print("Test Accuracy of MLP  : ", test_accuracy_MLP)

accuracy = accuracy_score(y_test, mlp_predictions_test)
precision = precision_score(y_test, mlp_predictions_test)
recall = recall_score(y_test, mlp_predictions_test)
f1 = f1_score(y_test, mlp_predictions_test)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
```

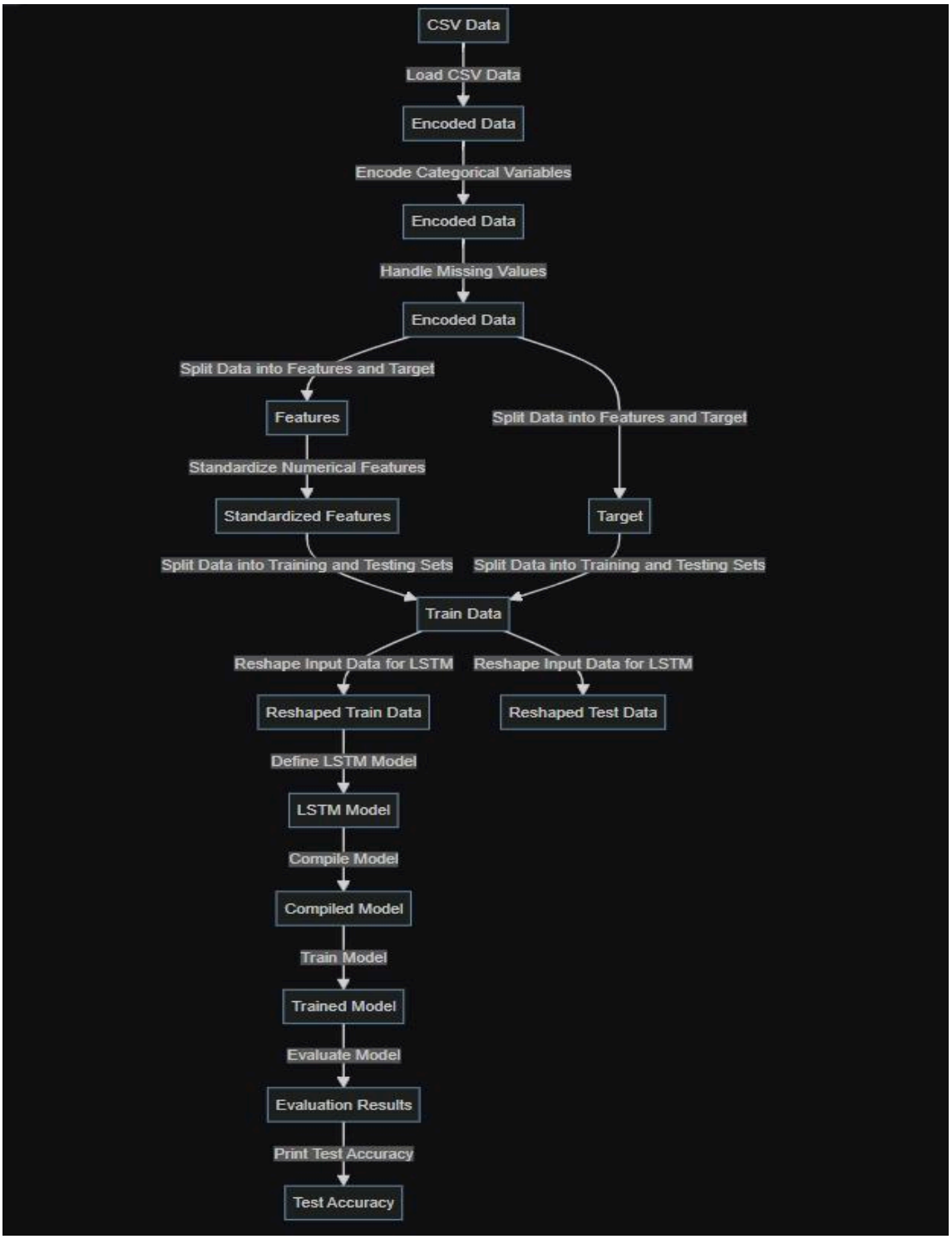
OUTPUT:

```
Test Accuracy of MLP  :  74.35897435897436
Accuracy: 0.7435897435897436
Precision: 0.7230769230769231
Recall: 0.9591836734693877
F1 Score: 0.8245614035087718
```

FLOW CHART(LSTM):



DATA FLOW DIAGRAM(LSTM):



Implementation of LSTM:

CODE:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import precision_score, recall_score, f1_score,
accuracy_score
import warnings
data = pd.read_csv('/content/drive/MyDrive/deep_learning/Loan_Data.csv')
label_encoder = LabelEncoder()
data['Gender'] = label_encoder.fit_transform(data['Gender'])
data['Married'] = label_encoder.fit_transform(data['Married'])
data['Education'] = label_encoder.fit_transform(data['Education'])
data['Self_Employed'] = label_encoder.fit_transform(data['Self_Employed'])
data['Property_Area'] = label_encoder.fit_transform(data['Property_Area'])
data['Loan_Status'] = label_encoder.fit_transform(data['Loan_Status'])
one_hot_cols = ['Dependents']
for col in one_hot_cols:
    one_hot_encoder = OneHotEncoder(sparse=False)
    encoded_cols = pd.DataFrame(one_hot_encoder.fit_transform(data[[col]]))
    encoded_cols.columns = [f'{col}_{i}' for i in
range(encoded_cols.shape[1])]
    data = pd.concat([data, encoded_cols], axis=1)
    data.drop(columns=[col], inplace=True)
data.dropna(inplace=True)
X = data.drop(columns=['Loan_ID', 'Loan_Status'])
y = data['Loan_Status']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
model = Sequential([
    LSTM(units=64, input_shape=(X_train.shape[1], X_train.shape[2])),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer=Adam(), loss='binary_crossentropy',
metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=50, batch_size=32,
validation_split=0.1)
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy}')
```

```
warnings.filterwarnings("ignore", category=FutureWarning)
y_pred = model.predict(X_test)
y_pred_classes = (y_pred > 0.5).astype(int)
precision = precision_score(y_test, y_pred_classes)
recall = recall_score(y_test, y_pred_classes)
f1 = f1_score(y_test, y_pred_classes)
accuracy = accuracy_score(y_test, y_pred_classes)
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
print(f'Accuracy: {accuracy}')
```

OUTPUT:

```
Epoch 1/50
12/12 [==============================] - 3s 58ms/step - loss: 0.6931 - accuracy: 0.5105 - val_loss: 0.6685 - val_accuracy: 0.7209
Epoch 2/50
12/12 [==============================] - 0s 7ms/step - loss: 0.6723 - accuracy: 0.6868 - val_loss: 0.6494 - val_accuracy: 0.7907
Epoch 3/50
12/12 [==============================] - 0s 7ms/step - loss: 0.6507 - accuracy: 0.7684 - val_loss: 0.6310 - val_accuracy: 0.8140
Epoch 4/50
12/12 [==============================] - 0s 6ms/step - loss: 0.6343 - accuracy: 0.7632 - val_loss: 0.6139 - val_accuracy: 0.8140
Epoch 5/50
12/12 [==============================] - 0s 6ms/step - loss: 0.6135 - accuracy: 0.7921 - val_loss: 0.5959 - val_accuracy: 0.8140
Epoch 6/50
12/12 [==============================] - 0s 6ms/step - loss: 0.5955 - accuracy: 0.7974 - val_loss: 0.5788 - val_accuracy: 0.8372
Epoch 7/50
12/12 [==============================] - 0s 6ms/step - loss: 0.5783 - accuracy: 0.8079 - val_loss: 0.5608 - val_accuracy: 0.8372
Epoch 8/50
12/12 [==============================] - 0s 9ms/step - loss: 0.5641 - accuracy: 0.8053 - val_loss: 0.5456 - val_accuracy: 0.8140
Epoch 9/50
12/12 [==============================] - 0s 7ms/step - loss: 0.5479 - accuracy: 0.8026 - val_loss: 0.5306 - val_accuracy: 0.8140
Epoch 10/50
12/12 [==============================] - 0s 7ms/step - loss: 0.5329 - accuracy: 0.8105 - val_loss: 0.5180 - val_accuracy: 0.8140
Epoch 11/50
12/12 [==============================] - 0s 6ms/step - loss: 0.5213 - accuracy: 0.8079 - val_loss: 0.5062 - val_accuracy: 0.8140
Epoch 12/50
12/12 [==============================] - 0s 7ms/step - loss: 0.5098 - accuracy: 0.8079 - val_loss: 0.4949 - val_accuracy: 0.8140
Epoch 13/50
12/12 [==============================] - 0s 8ms/step - loss: 0.5017 - accuracy: 0.8079 - val_loss: 0.4867 - val_accuracy: 0.8140
Epoch 14/50
12/12 [==============================] - 0s 7ms/step - loss: 0.4897 - accuracy: 0.8132 - val_loss: 0.4776 - val_accuracy: 0.8140
Epoch 15/50
12/12 [==============================] - 0s 6ms/step - loss: 0.4827 - accuracy: 0.8079 - val_loss: 0.4726 - val_accuracy: 0.8140
Epoch 16/50
12/12 [==============================] - 0s 6ms/step - loss: 0.4757 - accuracy: 0.8184 - val_loss: 0.4694 - val_accuracy: 0.8140
Epoch 17/50
```

System test and evaluation:

Comparing the LSTM with the feedforward network.

METRICS	FEED-FORWARD	LSTM
Precision	0.723	0.828
F1 Score	0.824	0.9
Recall	0.959	0.986
Accuracy	0.743	0.849

- ✔ The LSTM model performs better than the feed-forward network in all metrics.
- ✔ The LSTM model has a higher precision, recall 1-score and accuracy than the feed-forward network.

Conclusion:

Based on the above metrics, the LSTM model is better for loan prediction tasks than feed-forward networks. Therefore, the trained LSTM model better predicts loan approval decisions based on applicant attributes (columns or features).

Even if the dataset is not sequential or time series, LSTM (Long Short-Term Memory) networks can be used. While LSTMs are commonly employed in applications such as natural language processing, audio recognition, and time series prediction, they can also be applied in certain scenarios to non-sequential data.

Colab link for implementation of feed-forward and LSTM networks:

https://colab.research.google.com/drive/1EIP9iAqT7H9W6RuyLU9BPDrAzI20fe5i?usp=sharing

THANK YOU