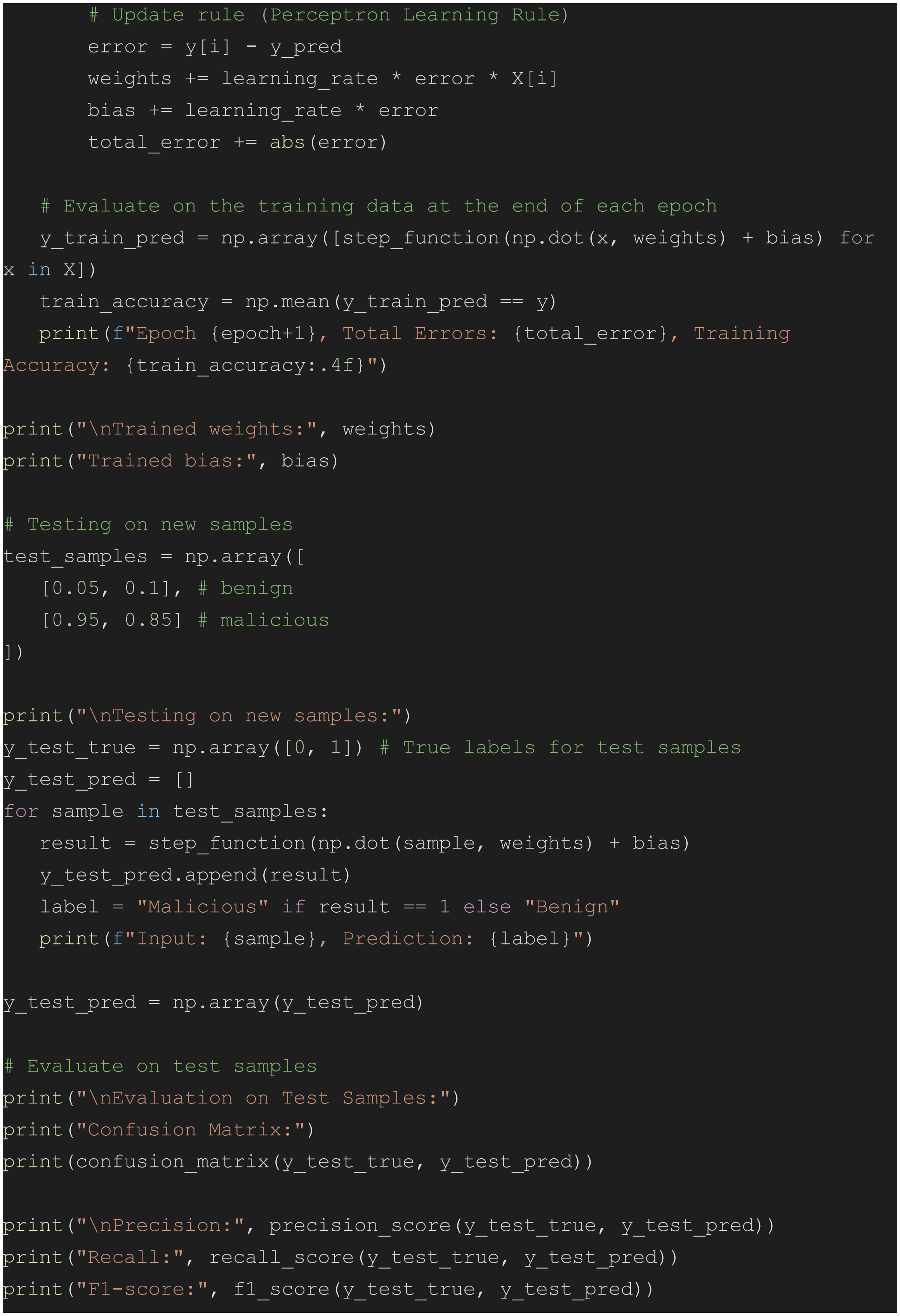
Praneesh R V
CB.SC.U4CYS23036

AINN Lab - 5 Traffic classification using single layer perceptron

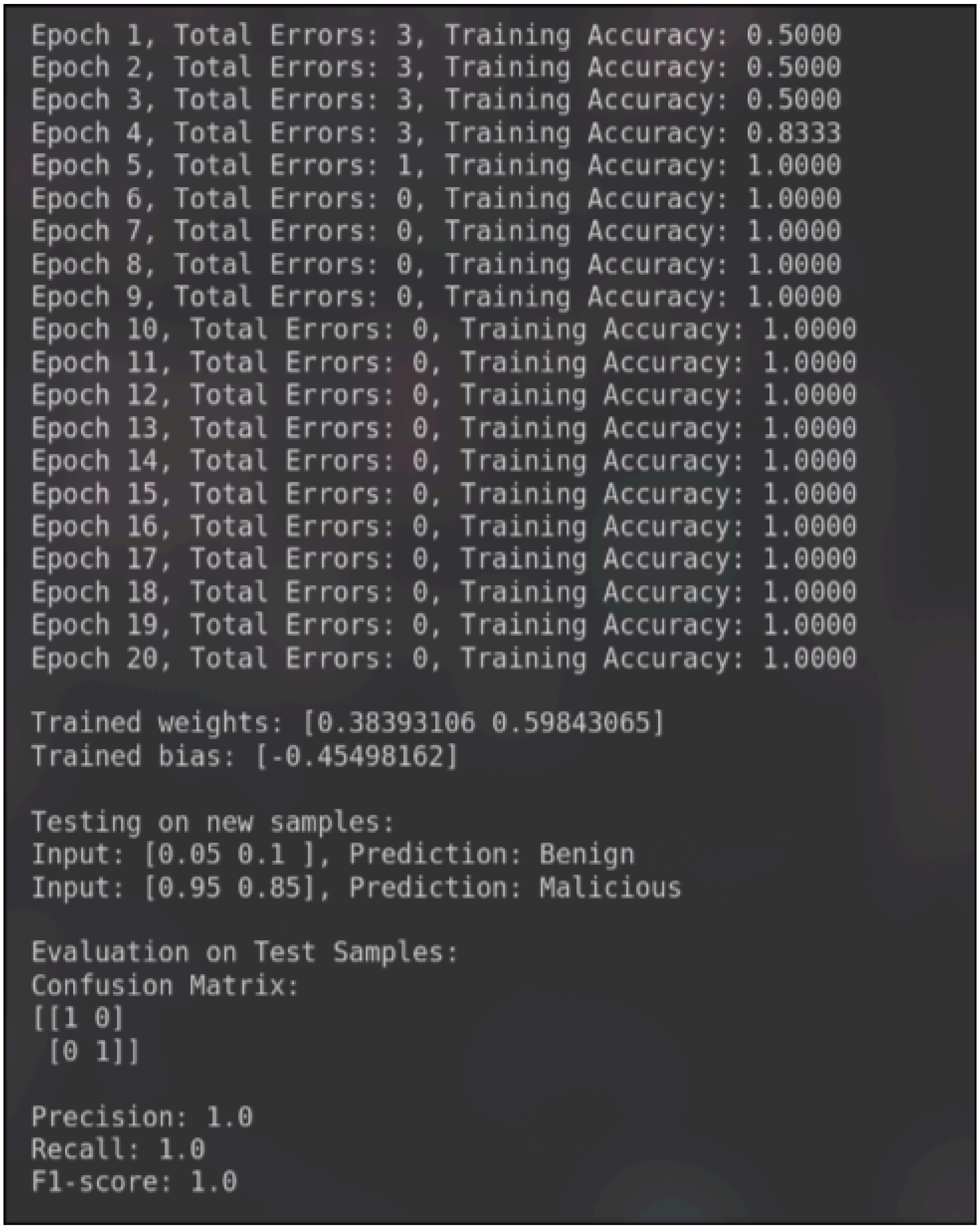
a) Code:

```
import numpy as np
from sklearn.metrics import confusion_matrix, precision_score,
recall_score, f1_score

X = np.array([
   [0.1, 0.2],  # benign
   [0.2, 0.1],  # benign
   [0.9, 0.8],  # malicious
   [0.8, 0.9],  # malicious
   [0.3, 0.4],  # benign
   [0.85, 0.7], # malicious
])

y = np.array([0, 0, 1, 1, 0, 1])

# Initialize weights and bias
weights = np.random.rand(2)  # 2 features
bias = np.random.rand(1)
learning_rate = 0.1
epochs = 20

# Activation function (Step function for perceptron)
def step_function(z):
   return 1 if z >= 0 else 0

# Training loop
for epoch in range(epochs):
   total_error = 0
   for i in range(len(X)):
       # Weighted sum
       linear_output = np.dot(X[i], weights) + bias
       y_pred = step_function(linear_output)
```



```
        # Update rule (Perceptron Learning Rule)
        error = y[i] - y_pred
        weights += learning_rate * error * X[i]
        bias += learning_rate * error
        total_error += abs(error)

    # Evaluate on the training data at the end of each epoch
    y_train_pred = np.array([step_function(np.dot(x, weights) + bias) for
x in X])
    train_accuracy = np.mean(y_train_pred == y)
    print(f"Epoch {epoch+1}, Total Errors: {total_error}, Training
Accuracy: {train_accuracy:.4f}")

print("\nTrained weights:", weights)
print("Trained bias:", bias)

# Testing on new samples
test_samples = np.array([
    [0.05, 0.1], # benign
    [0.95, 0.85] # malicious
])

print("\nTesting on new samples:")
y_test_true = np.array([0, 1])  # True labels for test samples
y_test_pred = []
for sample in test_samples:
    result = step_function(np.dot(sample, weights) + bias)
    y_test_pred.append(result)
    label = "Malicious" if result == 1 else "Benign"
    print(f"Input: {sample}, Prediction: {label}")

y_test_pred = np.array(y_test_pred)

# Evaluate on test samples
print("\nEvaluation on Test Samples:")
print("Confusion Matrix:")
print(confusion_matrix(y_test_true, y_test_pred))

print("\nPrecision:", precision_score(y_test_true, y_test_pred))
print("Recall:", recall_score(y_test_true, y_test_pred))
print("F1-score:", f1_score(y_test_true, y_test_pred))
```

Output:



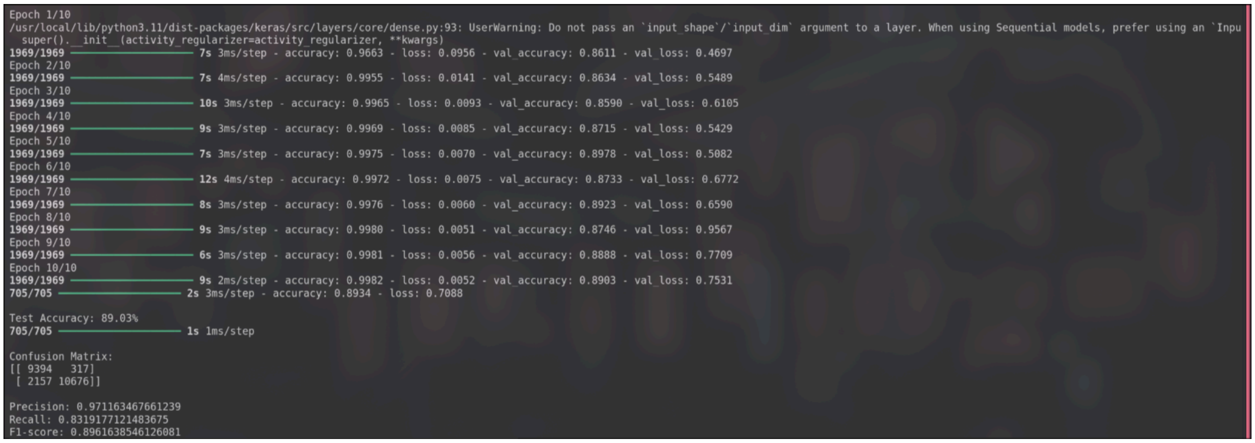
```
Epoch 1, Total Errors: 3, Training Accuracy: 0.5000
Epoch 2, Total Errors: 3, Training Accuracy: 0.5000
Epoch 3, Total Errors: 3, Training Accuracy: 0.5000
Epoch 4, Total Errors: 3, Training Accuracy: 0.8333
Epoch 5, Total Errors: 1, Training Accuracy: 1.0000
Epoch 6, Total Errors: 0, Training Accuracy: 1.0000
Epoch 7, Total Errors: 0, Training Accuracy: 1.0000
Epoch 8, Total Errors: 0, Training Accuracy: 1.0000
Epoch 9, Total Errors: 0, Training Accuracy: 1.0000
Epoch 10, Total Errors: 0, Training Accuracy: 1.0000
Epoch 11, Total Errors: 0, Training Accuracy: 1.0000
Epoch 12, Total Errors: 0, Training Accuracy: 1.0000
Epoch 13, Total Errors: 0, Training Accuracy: 1.0000
Epoch 14, Total Errors: 0, Training Accuracy: 1.0000
Epoch 15, Total Errors: 0, Training Accuracy: 1.0000
Epoch 16, Total Errors: 0, Training Accuracy: 1.0000
Epoch 17, Total Errors: 0, Training Accuracy: 1.0000
Epoch 18, Total Errors: 0, Training Accuracy: 1.0000
Epoch 19, Total Errors: 0, Training Accuracy: 1.0000
Epoch 20, Total Errors: 0, Training Accuracy: 1.0000

Trained weights: [0.38393106 0.59843065]
Trained bias: [-0.45498162]

Testing on new samples:
Input: [0.05 0.1 ], Prediction: Benign
Input: [0.95 0.85], Prediction: Malicious

Evaluation on Test Samples:
Confusion Matrix:
[[1 0]
 [0 1]]

Precision: 1.0
Recall: 1.0
F1-score: 1.0
```

b) Code:

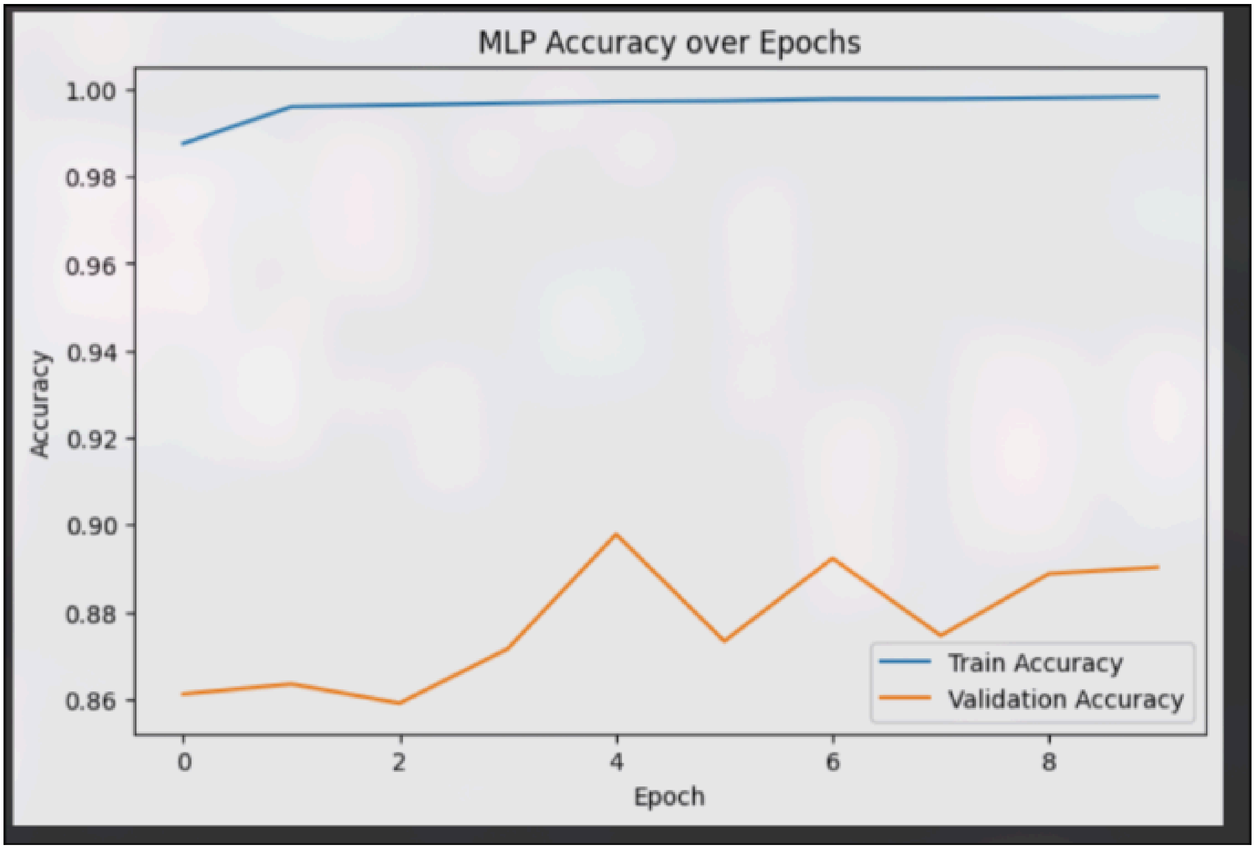
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import confusion_matrix, precision_score,
recall_score, f1_score
import urllib.request
import os

# 1. Download NSL-KDD Dataset

if not os.path.exists("KDDTrain+.txt"):
   urllib.request.urlretrieve(

"https://raw.githubusercontent.com/defcom17/NSL_KDD/master/KDDTrain+.txt"
,
       "KDDTrain+.txt"
   )
if not os.path.exists("KDDTest+.txt"):
   urllib.request.urlretrieve(

"https://raw.githubusercontent.com/defcom17/NSL_KDD/master/KDDTest+.txt",
       "KDDTest+.txt"
   )

# Column names
col_names = [

"duration","protocol_type","service","flag","src_bytes","dst_bytes","land
",

"wrong_fragment","urgent","hot","num_failed_logins","logged_in","num_comp
romised",

"root_shell","su_attempted","num_root","num_file_creations","num_shells",

"num_access_files","num_outbound_cmds","is_host_login","is_guest_login","
count",

"srv_count","serror_rate","srv_serror_rate","rerror_rate","srv_rerror_rat
e",
   "same_srv_rate","diff_srv_rate","srv_diff_host_rate","dst_host_count",

"dst_host_srv_count","dst_host_same_srv_rate","dst_host_diff_srv_rate",
"dst_host_same_src_port_rate","dst_host_srv_diff_host_rate","dst_host_serror_rate",

"dst_host_srv_serror_rate","dst_host_rerror_rate","dst_host_srv_rerror_rate",
   "label","difficulty"
]

# 2. Load Data

train_df = pd.read_csv("KDDTrain+.txt", names=col_names)
test_df = pd.read_csv("KDDTest+.txt", names=col_names)

# 3. Convert Labels to Binary

train_df['label'] = train_df['label'].apply(lambda x: 0 if x == 'normal' else 1)
test_df['label'] = test_df['label'].apply(lambda x: 0 if x == 'normal' else 1)

# 4. Label Encode categorical cols BEFORE splitting

categorical_cols = ['protocol_type', 'service', 'flag']
for col in categorical_cols:
   le = LabelEncoder()
   le.fit(list(train_df[col]) + list(test_df[col]))  # fit on both sets
   train_df[col] = le.transform(train_df[col])
   test_df[col] = le.transform(test_df[col])

# 5. Split Features & Labels

x_train = train_df.drop('label', axis=1).values
y_train = train_df['label'].values
x_test = test_df.drop('label', axis=1).values
y_test = test_df['label'].values

# 6. Scale Features

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
# 7. Build MLP Model

model = Sequential()
model.add(Dense(32, input_dim=x_train.shape[1], activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Binary classification

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

# 8. Train Model

history = model.fit(x_train, y_train, epochs=10, batch_size=64,
validation_data=(x_test, y_test))

# 9. Evaluate Model and print detailed metrics

loss, acc = model.evaluate(x_test, y_test)
print(f"\nTest Accuracy: {acc*100:.2f}%")

y_pred_prob = model.predict(x_test)
y_pred = (y_pred_prob > 0.5).astype("int32")

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nPrecision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-score:", f1_score(y_test, y_pred))


# 10. Plot Accuracy

plt.figure(figsize=(8,5))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('MLP Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Output:



```
Epoch 1/10
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1969/1969 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.9663 - loss: 0.0956 - val_accuracy: 0.8611 - val_loss: 0.4697
Epoch 2/10
1969/1969 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9955 - loss: 0.0141 - val_accuracy: 0.8634 - val_loss: 0.5489
Epoch 3/10
1969/1969 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9965 - loss: 0.0093 - val_accuracy: 0.8590 - val_loss: 0.6105
Epoch 4/10
1969/1969 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9969 - loss: 0.0085 - val_accuracy: 0.8715 - val_loss: 0.5429
Epoch 5/10
1969/1969 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.9975 - loss: 0.0070 - val_accuracy: 0.8978 - val_loss: 0.5082
Epoch 6/10
1969/1969 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step - accuracy: 0.9972 - loss: 0.0075 - val_accuracy: 0.8733 - val_loss: 0.6772
Epoch 7/10
1969/1969 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.9976 - loss: 0.0060 - val_accuracy: 0.8923 - val_loss: 0.6590
Epoch 8/10
1969/1969 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9980 - loss: 0.0051 - val_accuracy: 0.8746 - val_loss: 0.9567
Epoch 9/10
1969/1969 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9981 - loss: 0.0056 - val_accuracy: 0.8888 - val_loss: 0.7709
Epoch 10/10
1969/1969 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.9982 - loss: 0.0052 - val_accuracy: 0.8903 - val_loss: 0.7531
705/705 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8934 - loss: 0.7088

Test Accuracy: 89.03%
705/705 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step

Confusion Matrix:
[[ 9394   317]
 [ 2157 10676]]

Precision: 0.971163467661239
Recall: 0.8319177121483675
F1-score: 0.8961638546126081
```


MLP Accuracy over Epochs
Accuracy
Epoch
Train Accuracy
Validation Accuracy