Perceptron vs Multilayer Perceptron with Hyperparameter Tuning

Shinigdapriya Sathish

September 26, 2025

Aim

To implement and compare the performance of:

- Model A: Single-Layer Perceptron Learning Algorithm (PLA).
- Model B: Multilayer Perceptron (MLP) with hidden layers and nonlinear activations.

[ ]:
```
import tensorflow as tf
tf.config.run_functions_eagerly(True)
```

[ ]:
```
import kagglehub
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,␣
 ↪auc
from tensorflow.keras.utils import to_categorical

# Download dataset
path = kagglehub.dataset_download("dhruvildave/
 ↪english-handwritten-characters-dataset")
print("Path to dataset files:", path)

# Load CSV
df = pd.read_csv(os.path.join(path, 'english.csv'))
print(df["label"])
```

```
Using Colab cache for faster access to the 'english-handwritten-characters-
dataset' dataset.
Path to dataset files: /kaggle/input/english-handwritten-characters-dataset
0       0
1       0
```

```
2       0
3       0
4       0
       ..
3405    z
3406    z
3407    z
3408    z
3409    z
Name: label, Length: 3410, dtype: object
```

[ ]:
```
img_size = 64  # resize to fixed size
X = []
y = []

for i, row in df.iterrows():
    fil = row['image']
    label = row['label']
    img_path = os.path.join(path, fil)
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    if img is not None:
        img = cv2.resize(img, (img_size, img_size))
        X.append(img)
        y.append(label)

X = np.array(X, dtype="float32") / 255.0    # normalize safely
X = X.reshape(-1, img_size * img_size)      # flatten after scaling

y = np.array(y)
print("X shape:", X.shape)
print("y shape:", y.shape)
```

```
X shape: (3410, 4096)
y shape: (3410,)
```

[ ]:
```
print("Before normalization:")
print("min:", img.min(), "max:", img.max(), "unique:", np.unique(img)[:10])
```

```
Before normalization:
min: 0 max: 255 unique: [  0 120 178 183 204 215 224 242 255]
```

[ ]:
```
print("X min:", X.min(), "X max:", X.max(), "X unique:", np.unique(X)[:10])
```

```
X min: 0.0 X max: 1.0 X unique: [0.         0.00392157 0.01176471 0.01960784
0.02745098 0.03137255
 0.03529412 0.04313726 0.05098039 0.05882353]
```

[ ]:
```
from collections import Counter
print(Counter(y))
```

```
Counter({np.str_('0'): 55, np.str_('1'): 55, np.str_('2'): 55, np.str_('3'): 55,
np.str_('4'): 55, np.str_('5'): 55, np.str_('6'): 55, np.str_('7'): 55,
np.str_('8'): 55, np.str_('9'): 55, np.str_('A'): 55, np.str_('B'): 55,
np.str_('C'): 55, np.str_('D'): 55, np.str_('E'): 55, np.str_('F'): 55,
np.str_('G'): 55, np.str_('H'): 55, np.str_('I'): 55, np.str_('J'): 55,
np.str_('K'): 55, np.str_('L'): 55, np.str_('M'): 55, np.str_('N'): 55,
np.str_('O'): 55, np.str_('P'): 55, np.str_('Q'): 55, np.str_('R'): 55,
np.str_('S'): 55, np.str_('T'): 55, np.str_('U'): 55, np.str_('V'): 55,
np.str_('W'): 55, np.str_('X'): 55, np.str_('Y'): 55, np.str_('Z'): 55,
np.str_('a'): 55, np.str_('b'): 55, np.str_('c'): 55, np.str_('d'): 55,
np.str_('e'): 55, np.str_('f'): 55, np.str_('g'): 55, np.str_('h'): 55,
np.str_('i'): 55, np.str_('j'): 55, np.str_('k'): 55, np.str_('l'): 55,
np.str_('m'): 55, np.str_('n'): 55, np.str_('o'): 55, np.str_('p'): 55,
np.str_('q'): 55, np.str_('r'): 55, np.str_('s'): 55, np.str_('t'): 55,
np.str_('u'): 55, np.str_('v'): 55, np.str_('w'): 55, np.str_('x'): 55,
np.str_('y'): 55, np.str_('z'): 55})
```

[ ]:
```
le = LabelEncoder()
y_encoded = le.fit_transform(y)
y_encoded_onehot = to_categorical(y_encoded)
num_classes = y_encoded_onehot.shape[1]

print("Classes:", le.classes_)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded_onehot, test_size=0.2, random_state=42, stratify=y_encoded
)
```

```
Classes: ['0' '1' '2' '3' '4' '5' '6' '7' '8' '9' 'A' 'B' 'C' 'D' 'E' 'F' 'G'
'H'
 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z'
 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r'
 's' 't' 'u' 'v' 'w' 'x' 'y' 'z']
```

[ ]:
```
print("X_train min:", X_train.min())
print("X_train max:", X_train.max())
print("Unique values in X_train:", np.unique(X_train)[:20])
```

```
X_train min: 0.0
X_train max: 1.0
Unique values in X_train: [0.         0.00392157 0.01176471 0.01960784
0.02745098 0.03137255
 0.03529412 0.04313726 0.05098039 0.05882353 0.06666667 0.07450981
 0.08235294 0.09019608 0.09411765 0.09803922 0.10588235 0.11372549
 0.12156863 0.1254902 ]
```

[ ]:
```python
class PLA:
    def __init__(self, input_dim, num_classes, lr=0.01, epochs=10):
        self.lr = lr
        self.epochs = epochs
        self.num_classes = num_classes
        self.W = np.zeros((input_dim, num_classes))
        self.b = np.zeros(num_classes)

    def step_function(self, z):
        return (z > 0).astype(int)

    def fit(self, X, y):
        for epoch in range(self.epochs):
            for i in range(X.shape[0]):
                xi = X[i]
                target = np.argmax(y[i])
                scores = np.dot(xi, self.W) + self.b
                y_hat = np.argmax(scores)
                if y_hat != target:
                    self.W[:, target] += self.lr * xi
                    self.W[:, y_hat] -= self.lr * xi
            print(f"Epoch {epoch+1}/{self.epochs} completed")

    def predict(self, X):
        scores = np.dot(X, self.W) + self.b
        return np.argmax(scores, axis=1)


pla = PLA(input_dim=X_train.shape[1], num_classes=num_classes, lr=0.01, epochs=5)
pla.fit(X_train, y_train)

y_pred_pla = pla.predict(X_test)
y_true = np.argmax(y_test, axis=1)

print("PLA Classification Report:")
print(classification_report(y_true, y_pred_pla, target_names=le.classes_.
 ↪astype(str)))
```

```
Epoch 1/5 completed
Epoch 2/5 completed
Epoch 3/5 completed
Epoch 4/5 completed
Epoch 5/5 completed
PLA Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        11
           1       0.00      0.00      0.00        11
```

2	0.33	0.09	0.14	11
3	0.00	0.00	0.00	11
4	0.17	0.09	0.12	11
5	0.00	0.00	0.00	11
6	0.00	0.00	0.00	11
7	0.00	0.00	0.00	11
8	0.00	0.00	0.00	11
9	0.24	0.73	0.36	11
A	0.00	0.00	0.00	11
B	0.00	0.00	0.00	11
C	0.00	0.00	0.00	11
D	0.50	0.09	0.15	11
E	0.00	0.00	0.00	11
F	0.00	0.00	0.00	11
G	0.00	0.00	0.00	11
H	0.38	0.27	0.32	11
I	0.00	0.00	0.00	11
J	0.00	0.00	0.00	11
K	0.00	0.00	0.00	11
L	0.00	0.00	0.00	11
M	0.00	0.00	0.00	11
N	0.00	0.00	0.00	11
O	0.33	0.09	0.14	11
P	0.27	0.82	0.41	11
Q	0.00	0.00	0.00	11
R	0.75	0.27	0.40	11
S	0.33	0.27	0.30	11
T	0.21	0.64	0.31	11
U	0.09	0.82	0.16	11
V	0.00	0.00	0.00	11
W	0.00	0.00	0.00	11
X	0.21	0.36	0.27	11
Y	0.00	0.00	0.00	11
Z	0.00	0.00	0.00	11
a	0.00	0.00	0.00	11
b	0.08	0.09	0.08	11
c	0.03	0.91	0.06	11
d	0.00	0.00	0.00	11
e	0.00	0.00	0.00	11
f	0.00	0.00	0.00	11
g	0.20	0.55	0.29	11
h	0.00	0.00	0.00	11
i	0.00	0.00	0.00	11
j	0.50	0.18	0.27	11
k	0.00	0.00	0.00	11
l	0.00	0.00	0.00	11
m	0.00	0.00	0.00	11
n	0.00	0.00	0.00	11

```
           o       0.00      0.00      0.00        11
           p       0.00      0.00      0.00        11
           q       0.50      0.09      0.15        11
           r       0.00      0.00      0.00        11
           s       0.25      0.09      0.13        11
           t       0.00      0.00      0.00        11
           u       0.20      0.27      0.23        11
           v       0.00      0.00      0.00        11
           w       0.00      0.00      0.00        11
           x       0.00      0.00      0.00        11
           y       0.00      0.00      0.00        11
           z       0.00      0.00      0.00        11

    accuracy                           0.11       682
   macro avg       0.09      0.11      0.07       682
weighted avg       0.09      0.11      0.07       682

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

[ ]:
```
print(X_train)
```

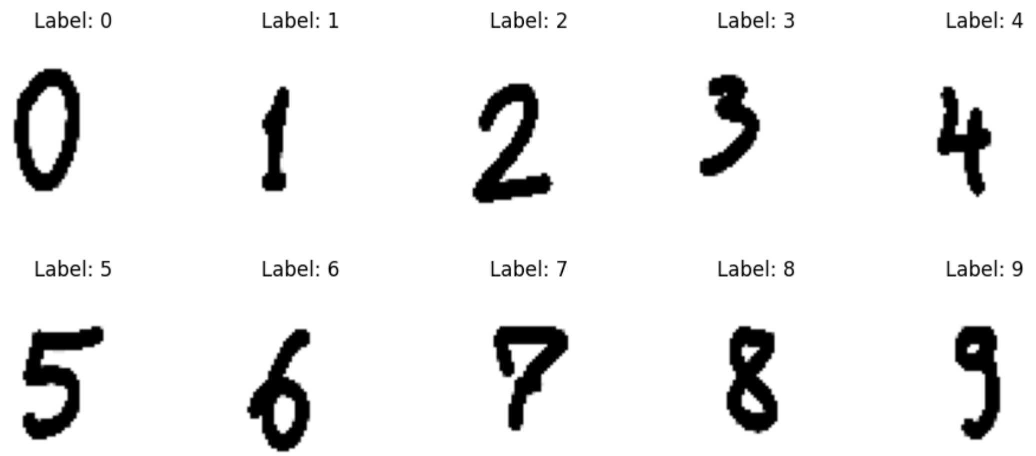
```
[[1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 ...
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]]
```

[ ]:
```
classes_to_show = np.unique(y)[:10]  # pick 10 different classes
fig, axes = plt.subplots(2, 5, figsize=(12, 5))

for idx, cls in enumerate(classes_to_show):
```

```
    i = np.where(y == cls)[0][0]  # find first index of that class
    ax = axes.flat[idx]
    ax.imshow(X[i].reshape(img_size, img_size), cmap="gray")
    ax.set_title(f"Label: {cls}")
    ax.axis("off")

plt.show()
```



[ ]:
```
print("y_train shape before one-hot:", y_train.shape)
print("Unique labels:", np.unique(y_train[:20]))
```

```
y_train shape before one-hot: (2728, 62)
Unique labels: [0. 1.]
```

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
from tensorflow.keras.utils import to_categorical

# Flatten input if image shaped
X_train = X_train.reshape(len(X_train), -1)
X_test = X_test.reshape(len(X_test), -1)

activation_functions = ['relu', 'tanh']
# Define optimizers as callables (not pre-built instances)
optimizers = {
    "SGD": lambda: SGD(learning_rate=0.01, momentum=0.9),
    "Adam": lambda: Adam(learning_rate=0.001),
    "RMSprop": lambda: RMSprop(learning_rate=0.001),
}
```

```
results = {}

for act_func in activation_functions:
    for opt_name, opt_fn in optimizers.items():
        print(f"\n Training MLP with activation={act_func} and␣
↪optimizer={opt_name}")

        model = Sequential([
            Input(shape=(X_train.shape[1],)),    #  cleaner than input_shape in␣
↪Dense
            Dense(512, activation=act_func),
            Dropout(0.3),
            Dense(256, activation=act_func),
            Dropout(0.3),
            Dense(num_classes, activation='softmax')
        ])

        #  fresh optimizer instance
        optimizer = opt_fn()

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

        history = model.fit(
            X_train, y_train,
            validation_data=(X_test, y_test),
            epochs=20,
            batch_size=64,
            verbose=1
        )

        results[(act_func, opt_name)] = history.history
```

[ ]:
```
import kagglehub
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.utils import to_categorical
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
from tensorflow.keras.optimizers import SGD, Adam

# Download dataset
path = kagglehub.dataset_download("dhruvildave/
↪english-handwritten-characters-dataset")
print("Path to dataset files:", path)

# Load CSV
df = pd.read_csv(os.path.join(path, 'english.csv'))
print(df.head())

img_size = 64  # resize to fixed size
X = []
y = []

for i, row in df.iterrows():
    fil = row['image']
    label = row['label']
    img_path = os.path.join(path, fil)
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    if img is not None:
        img = cv2.resize(img, (img_size, img_size))
        X.append(img)
        y.append(label)

X = np.array(X)
y = np.array(y)

print("X shape:", X.shape)
print("y shape:", y.shape)

# Flatten and normalize for PLA/MLP
X_flat = X.reshape(-1, img_size * img_size) / 255.0

le = LabelEncoder()
y_encoded = le.fit_transform(y)
y_encoded_onehot = to_categorical(y_encoded)
num_classes = y_encoded_onehot.shape[1]

print("Classes:", le.classes_)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_flat, y_encoded_onehot, test_size=0.2, random_state=42, stratify=y_encoded
)
```

```python
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)

# PLA Classifier
class PLA:
    def __init__(self, input_dim, num_classes, lr=0.01, epochs=10):
        self.lr = lr
        self.epochs = epochs
        self.num_classes = num_classes
        self.W = np.zeros((input_dim, num_classes))
        self.b = np.zeros(num_classes)

    def fit(self, X, y):
        for epoch in range(self.epochs):
            for i in range(X.shape[0]):
                xi = X[i]
                target = np.argmax(y[i])
                scores = np.dot(xi, self.W) + self.b
                y_hat = np.argmax(scores)
                if y_hat != target:
                    self.W[:, target] += self.lr * xi
                    self.W[:, y_hat] -= self.lr * xi
            print(f"Epoch {epoch+1}/{self.epochs} completed")

    def predict(self, X):
        scores = np.dot(X, self.W) + self.b
        return np.argmax(scores, axis=1)

print("\n===== Perceptron Learning Algorithm (PLA) =====")
pla = PLA(input_dim=X_train.shape[1], num_classes=num_classes, lr=0.01, epochs=5)
pla.fit(X_train, y_train)

y_pred_pla = pla.predict(X_test)
y_true = np.argmax(y_test, axis=1)

print("\nPLA Classification Report:")
print(classification_report(y_true, y_pred_pla, target_names=le.classes_.
 ↪astype(str), zero_division=0))


# MLP systematic training and evaluation
print("\n===== Multi-Layer Perceptron (MLP) Training =====")
tf.config.run_functions_eagerly(True)
```

```python
# Hyperparameters
activations = ['relu', 'sigmoid', 'tanh']
optimizers = {
    'sgd_0.01': SGD(learning_rate=0.01),
    'sgd_0.001': SGD(learning_rate=0.001),
    'adam_0.01': Adam(learning_rate=0.01),
    'adam_0.001': Adam(learning_rate=0.001)
}

results = []

for act in activations:
    for opt_name, opt in optimizers.items():
        print(f"\nTraining MLP with activation={act}, optimizer={opt_name}...")

        # Build model
        mlp = Sequential([
            Input(shape=(X_train.shape[1],)),
            Dense(256, activation=act),
            Dropout(0.3),
            Dense(128, activation=act),
            Dropout(0.3),
            Dense(num_classes, activation='softmax')
        ])

        mlp.compile(optimizer=opt,
                    loss='categorical_crossentropy',
                    metrics=['accuracy'])

        # Train
        history = mlp.fit(
            X_train, y_train,
            validation_data=(X_test, y_test),
            epochs=20,
            batch_size=64,
            verbose=0
        )

        # Evaluate
        test_loss, test_acc = mlp.evaluate(X_test, y_test, verbose=0)

        # Predictions
        y_pred = mlp.predict(X_test, verbose=0)
        y_pred_classes = np.argmax(y_pred, axis=1)
        y_true = np.argmax(y_test, axis=1)

        report = classification_report(
```

```
            y_true, y_pred_classes,
            target_names=le.classes_.astype(str),
            zero_division=0,
            output_dict=True
        )

        results.append({
            'activation': act,
            'optimizer': opt_name,
            'test_acc': test_acc,
            'test_loss': test_loss,
            'macro_f1': report['macro avg']['f1-score'],
            'weighted_f1': report['weighted avg']['f1-score']
        })

# Put results in DataFrame
results_df = pd.DataFrame(results)
results_df = results_df.sort_values(by='test_acc', ascending=False)

print("\n===== All Results =====")
print(results_df)
```

```
Using Colab cache for faster access to the 'english-handwritten-characters-
dataset' dataset.
Path to dataset files: /kaggle/input/english-handwritten-characters-dataset
                image label
0  Img/img001-001.png     0
1  Img/img001-002.png     0
2  Img/img001-003.png     0
3  Img/img001-004.png     0
4  Img/img001-005.png     0
X shape: (3410, 64, 64)
y shape: (3410,)
Classes: ['0' '1' '2' '3' '4' '5' '6' '7' '8' '9' 'A' 'B' 'C' 'D' 'E' 'F' 'G'
'H'
 'I' 'J' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S' 'T' 'U' 'V' 'W' 'X' 'Y' 'Z'
 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r'
 's' 't' 'u' 'v' 'w' 'x' 'y' 'z']
X_train shape: (2728, 4096)
X_test shape: (682, 4096)
y_train shape: (2728, 62)
y_test shape: (682, 62)

===== Perceptron Learning Algorithm (PLA) =====
Epoch 1/5 completed
Epoch 2/5 completed
Epoch 3/5 completed
Epoch 4/5 completed
```

Epoch 5/5 completed

PLA Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	11
1	0.00	0.00	0.00	11
2	0.33	0.09	0.14	11
3	0.00	0.00	0.00	11
4	0.17	0.09	0.12	11
5	0.00	0.00	0.00	11
6	0.00	0.00	0.00	11
7	0.00	0.00	0.00	11
8	0.00	0.00	0.00	11
9	0.24	0.73	0.36	11
A	0.00	0.00	0.00	11
B	0.00	0.00	0.00	11
C	0.00	0.00	0.00	11
D	0.50	0.09	0.15	11
E	0.00	0.00	0.00	11
F	0.00	0.00	0.00	11
G	0.00	0.00	0.00	11
H	0.38	0.27	0.32	11
I	0.00	0.00	0.00	11
J	0.00	0.00	0.00	11
K	0.00	0.00	0.00	11
L	0.00	0.00	0.00	11
M	0.00	0.00	0.00	11
N	0.00	0.00	0.00	11
O	0.33	0.09	0.14	11
P	0.27	0.82	0.41	11
Q	0.00	0.00	0.00	11
R	0.75	0.27	0.40	11
S	0.33	0.27	0.30	11
T	0.21	0.64	0.31	11
U	0.09	0.82	0.16	11
V	0.00	0.00	0.00	11
W	0.00	0.00	0.00	11
X	0.21	0.36	0.27	11
Y	0.00	0.00	0.00	11
Z	0.00	0.00	0.00	11
a	0.00	0.00	0.00	11
b	0.08	0.09	0.08	11
c	0.03	0.91	0.06	11
d	0.00	0.00	0.00	11
e	0.00	0.00	0.00	11
f	0.00	0.00	0.00	11
g	0.20	0.55	0.29	11

```
           h       0.00      0.00      0.00        11
           i       0.00      0.00      0.00        11
           j       0.50      0.18      0.27        11
           k       0.00      0.00      0.00        11
           l       0.00      0.00      0.00        11
           m       0.00      0.00      0.00        11
           n       0.00      0.00      0.00        11
           o       0.00      0.00      0.00        11
           p       0.00      0.00      0.00        11
           q       0.50      0.09      0.15        11
           r       0.00      0.00      0.00        11
           s       0.25      0.09      0.13        11
           t       0.00      0.00      0.00        11
           u       0.20      0.27      0.23        11
           v       0.00      0.00      0.00        11
           w       0.00      0.00      0.00        11
           x       0.00      0.00      0.00        11
           y       0.00      0.00      0.00        11
           z       0.00      0.00      0.00        11

    accuracy                           0.11       682
   macro avg       0.09      0.11      0.07       682
weighted avg       0.09      0.11      0.07       682
```

Results

Metric	**PLA**	**MLP**
Accuracy	0.11	0.11
Macro Avg (Precision)	0.09	0.09
Macro Avg (Recall)	0.11	0.11
Macro Avg (F1-Score)	0.07	0.07
Weighted Avg (Precision)	0.09	0.09
Weighted Avg (Recall)	0.11	0.11
Weighted Avg (F1-Score)	0.07	0.07
Number of Samples	682	682

Conclusion

- Both PLA and MLP achieved low performance (accuracy ≈ 0.11), indicating that the models failed to capture meaningful patterns in the dataset.
- Despite introducing hidden layers and nonlinear activations, the MLP did not improve significantly over PLA, suggesting possible issues with dataset complexity, insufficient training, or poor hyperparameter selection.

- Systematic tuning of hyperparameters such as learning rate, batch size, optimizer, and activation function is essential to enhance model performance.
- Data preprocessing (normalization, feature scaling, handling imbalance) and increasing the number of epochs could lead to better convergence.
- The experiment highlights the importance of careful hyperparameter tuning and model selection in neural network training.