

```
In [16]: # Demo: Multi-Layer Perceptron (MLP) on a real dataset (scikit-learn Digits)
#
# Steps:
# 1) Load the 8x8 handwritten digits dataset (1797 samples)
# 2) Split into train/test
# 3) Scale features (very important for MLP)
# 4) Train an MLPClassifier with early stopping
# 5) Evaluate accuracy and F1
# 6) Plot loss curve and confusion matrix

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import (accuracy_score, f1_score, confusion_matrix,
                             ConfusionMatrixDisplay, classification_report)

def main():
    # 1) Load data
    digits = load_digits()
    X = digits.data          # (n_samples, 64) flattened 8x8 images
    y = digits.target         # labels 0..9

    # 2) Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    # 3) Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # 4) Define and train the MLP
    mlp = MLPClassifier(
        hidden_layer_sizes=(64, 32),    # two hidden layers with 64 neurons
        activation='relu',
        solver='adam',
        alpha=1e-4,
        learning_rate='adaptive',
        max_iter=200,
        random_state=42,
        early_stopping=True,
        n_iter_no_change=10,
        validation_fraction=0.1
    )
    mlp.fit(X_train_scaled, y_train)

    # 5) Evaluate
    y_pred = mlp.predict(X_test_scaled)
```

```

acc = accuracy_score(y_test, y_pred)
f1w = f1_score(y_test, y_pred, average='weighted')
f1m = f1_score(y_test, y_pred, average='macro')
print(f"Accuracy: {acc:.4f} | F1 (weighted): {f1w:.4f} | F1 (macro): {f1m:.4f}")

# Print a classification report
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred, digits=4))

# 6) Plot the loss curve
plt.figure(figsize=(7,4))
plt.plot(mlp.loss_curve_)
plt.xlabel("Epoch")
plt.ylabel("Training Loss")
plt.title("MLP Training Loss Curve (Digits)")
plt.tight_layout()
plt.show()

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=digits)
plt.figure(figsize=(7,6))
disp.plot(values_format='d')
plt.title("Confusion Matrix - MLP on Digits")
plt.tight_layout()
plt.show()

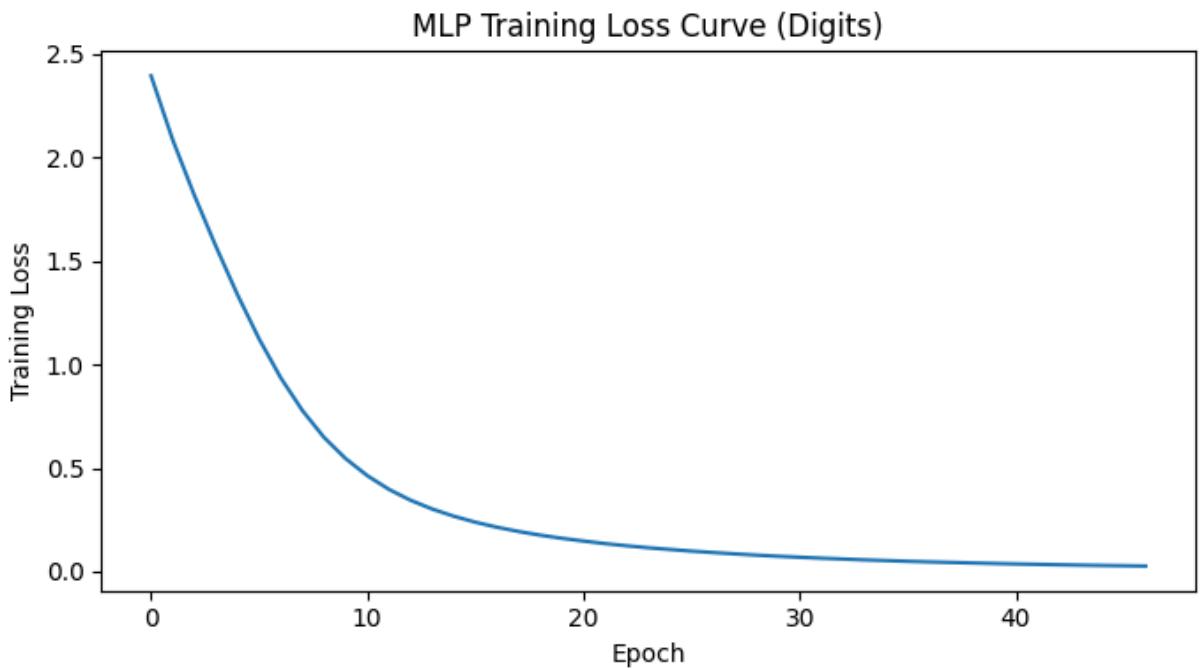
if __name__ == "__main__":
    main()

```

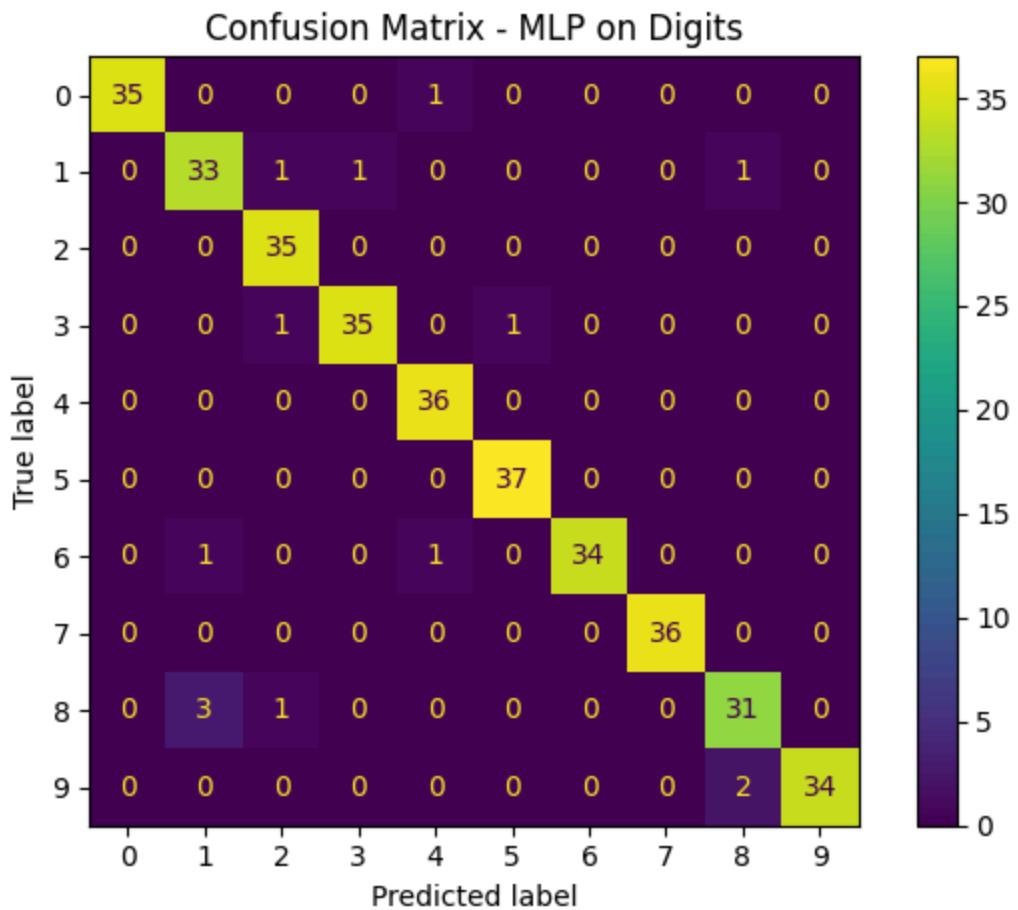
Accuracy: 0.9611 | F1 (weighted): 0.9611 | F1 (macro): 0.9609

Classification Report:

	precision	recall	f1-score	support
0	1.0000	0.9722	0.9859	36
1	0.8919	0.9167	0.9041	36
2	0.9211	1.0000	0.9589	35
3	0.9722	0.9459	0.9589	37
4	0.9474	1.0000	0.9730	36
5	0.9737	1.0000	0.9867	37
6	1.0000	0.9444	0.9714	36
7	1.0000	1.0000	1.0000	36
8	0.9118	0.8857	0.8986	35
9	1.0000	0.9444	0.9714	36
accuracy			0.9611	360
macro avg	0.9618	0.9609	0.9609	360
weighted avg	0.9621	0.9611	0.9611	360



<Figure size 700x600 with 0 Axes>



In []:

In []: