```
[notice] A new release of pip is available: 24.0 -> 25.2
[notice] To update, run: pip install --upgrade pip
```

In [ ]:

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, random_split
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5
train_data = datasets.MNIST(root='./data', train=True, download=True, transform=
test_data = datasets.MNIST(root='./data', train=False, download=True, transform=

train_size = int(0.9 * len(train_data))
val_size = len(train_data) - train_size
train_dataset, val_dataset = random_split(train_data, [train_size, val_size])

train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=128, shuffle=False)
test_loader = DataLoader(test_data, batch_size=128, shuffle=False)

class FeedForwardNN(nn.Module):
    def __init__(self):
        super(FeedForwardNN, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(28*28, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.log_softmax(self.fc3(x), dim=1)
        return x

model = FeedForwardNN()

criterion = nn.NLLLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

epochs = 5
for epoch in range(epochs):
    model.train()
    train_loss, correct = 0, 0
    for X, y in train_loader:
        optimizer.zero_grad()
        output = model(X)
        loss = criterion(output, y)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
        correct += (output.argmax(dim=1) == y).sum().item()
    train_acc = correct / len(train_loader.dataset)
    print(f"Epoch {epoch+1}/{epochs} - Loss: {train_loss:.4f}, Train Acc: {train
model.eval()
y_true, y_pred = [], []
with torch.no_grad():
    correct = 0
    for X, y in test_loader:
        output = model(X)
        pred = output.argmax(dim=1)
        y_true.extend(y.numpy())
        y_pred.extend(pred.numpy())
        correct += (pred == y).sum().item()

test_acc = correct / len(test_loader.dataset)
print(f"Test Accuracy: {test_acc:.4f}")

cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=range(10))
disp.plot(cmap='Blues')
plt.show()
```

```
Epoch 1/5 - Loss: 196.1358, Train Acc: 0.8664
Epoch 2/5 - Loss: 97.1451, Train Acc: 0.9304
Epoch 3/5 - Loss: 70.8888, Train Acc: 0.9497
Epoch 4/5 - Loss: 57.3038, Train Acc: 0.9595
Epoch 5/5 - Loss: 48.2227, Train Acc: 0.9648
```