```
import torch
import torchvision.transforms as transforms
from torchvision import datasets
from torch.utils.data import DataLoader, Subset
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
from tqdm import tqdm  # For progress bar


# Define the neural network model
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 10)

    def forward(self, x):
        x = x.view(-1, 28 * 28)  # Flatten the input
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
        return x



def train_model(model, train_loader, test_loader, num_epochs=20):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Training on {device}")

    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    train_losses, test_losses = [], []
    train_accuracies, test_accuracies = [], []

    for epoch in range(num_epochs):
        running_loss = 0.0
        correct = 0
        total = 0

        model.train()
        with tqdm(total=len(train_loader), desc=f"Epoch [{epoch+1}/{num_epochs}]", unit="batch") as pbar:
            for inputs, labels in train_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                optimizer.zero_grad()
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                loss.backward()
                optimizer.step()

                running_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                correct += (predicted == labels).sum().item()
                total += labels.size(0)

                pbar.set_postfix({"Loss": f"{running_loss / len(train_loader):.4f}"})
                pbar.update(1)

        train_loss = running_loss / len(train_loader)
        train_losses.append(train_loss)
        train_accuracy = correct / total
        train_accuracies.append(train_accuracy)

        # Evaluate on the test set
        model.eval()
        test_loss = 0.0
        correct = 0
        total = 0
        with torch.no_grad():
            for inputs, labels in test_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                test_loss += loss.item()

                _, predicted = torch.max(outputs, 1)
```

```
                correct += (predicted == labels).sum().item()
                total += labels.size(0)

        test_loss /= len(test_loader)
        test_losses.append(test_loss)
        test_accuracy = correct / total
        test_accuracies.append(test_accuracy)

        print(f"Epoch [{epoch+1}/{num_epochs}] - Train Loss: {train_loss:.4f}, Train Acc: {train_accuracy:.4f}, Test Loss: {

    # Plotting metrics
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(range(1, num_epochs + 1), train_losses, label='Train Loss')
    plt.plot(range(1, num_epochs + 1), test_losses, label='Test Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Loss Curve')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(range(1, num_epochs + 1), train_accuracies, label='Train Accuracy')
    plt.plot(range(1, num_epochs + 1), test_accuracies, label='Test Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Accuracy Curve')
    plt.legend()

    plt.tight_layout()
    plt.show()


def extract_and_save_samples(dataset, class_idx=3, num_samples=1000):
    # Extract samples of a specific class and save them to a file.
    class_samples_indices = [i for i in range(len(dataset.targets)) if dataset.targets[i] == class_idx]

    # Ensure we don't exceed the number of available samples.
    if len(class_samples_indices) < num_samples:
        raise ValueError(f"Not enough samples of class {class_idx} available in the dataset.")

    selected_indices = class_samples_indices[:num_samples]

    # Extract the data and targets.
    extracted_data = [dataset[i][0] for i in selected_indices]
    extracted_targets = [dataset[i][1] for i in selected_indices]

    # Save as tensors.
    extracted_data_tensor = torch.stack(extracted_data)  # Stack to create a single tensor.
    extracted_targets_tensor = torch.tensor(extracted_targets)  # Convert to tensor.

    # Save to file (you can choose .pt or .pth for PyTorch tensors).
    torch.save((extracted_data_tensor, extracted_targets_tensor), 'mnist_class_3_samples.pt')

    print(f"Extracted {num_samples} samples of class {class_idx} and saved to 'mnist_class_3_samples.pt'.")



# Define the transformations to apply to each image
transform = transforms.Compose([
    transforms.ToTensor(),  # Convert the images to tensors
    transforms.Normalize((0.5,), (0.5,))  # Normalize to [-1, 1] range
])


# Load the training and testing datasets from MNIST.
train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)


# Extract and save samples of class '3'.
extract_and_save_samples(train_dataset)
```

```
Extracted 1000 samples of class 3 and saved to 'mnist_class_3_samples.pt'.
```

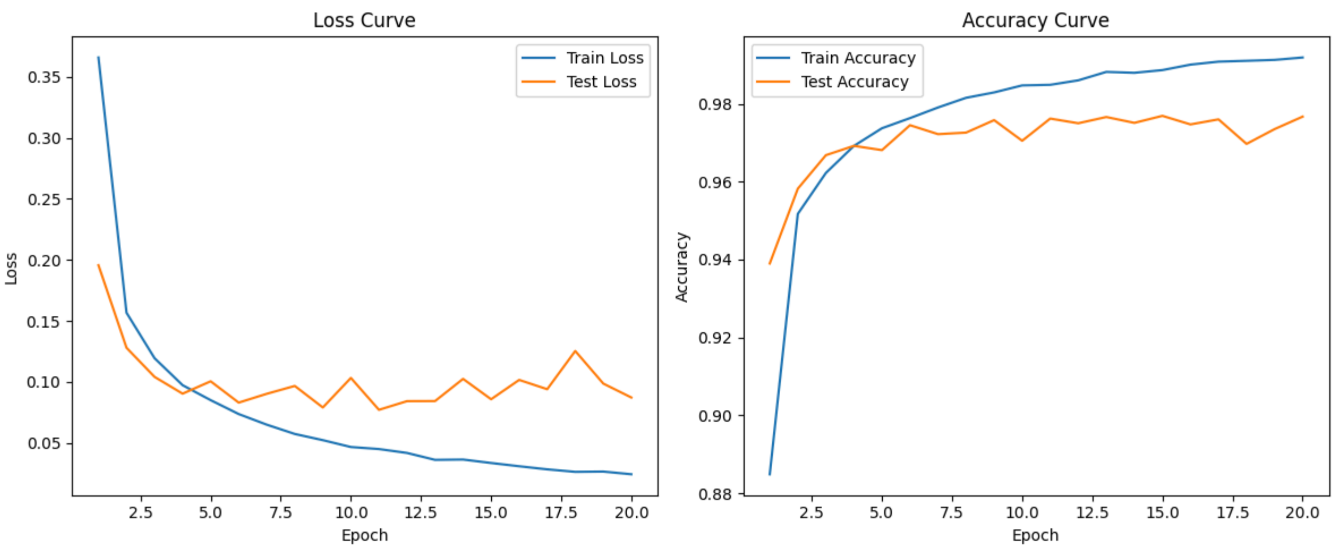
```
# Create DataLoader for training and testing.
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)



# Initialize and train the model.
model = SimpleNN()
```

```
train_model(model, train_loader, test_loader, num_epochs=20)
```

```
Training on cuda
Epoch [1/20]: 100%|██████████| 938/938 [00:18<00:00, 50.72batch/s, Loss=0.3659]
Epoch [1/20] - Train Loss: 0.3659, Train Acc: 0.8849, Test Loss: 0.1956, Test Acc: 0.9390
Epoch [2/20]: 100%|██████████| 938/938 [00:19<00:00, 49.09batch/s, Loss=0.1567]
Epoch [2/20] - Train Loss: 0.1567, Train Acc: 0.9517, Test Loss: 0.1280, Test Acc: 0.9582
Epoch [3/20]: 100%|██████████| 938/938 [00:19<00:00, 48.05batch/s, Loss=0.1194]
Epoch [3/20] - Train Loss: 0.1194, Train Acc: 0.9622, Test Loss: 0.1041, Test Acc: 0.9668
Epoch [4/20]: 100%|██████████| 938/938 [00:18<00:00, 50.37batch/s, Loss=0.0973]
Epoch [4/20] - Train Loss: 0.0973, Train Acc: 0.9692, Test Loss: 0.0903, Test Acc: 0.9692
Epoch [5/20]: 100%|██████████| 938/938 [00:19<00:00, 47.75batch/s, Loss=0.0850]
Epoch [5/20] - Train Loss: 0.0850, Train Acc: 0.9737, Test Loss: 0.1005, Test Acc: 0.9681
Epoch [6/20]: 100%|██████████| 938/938 [00:19<00:00, 49.10batch/s, Loss=0.0736]
Epoch [6/20] - Train Loss: 0.0736, Train Acc: 0.9763, Test Loss: 0.0830, Test Acc: 0.9745
Epoch [7/20]: 100%|██████████| 938/938 [00:18<00:00, 50.89batch/s, Loss=0.0649]
Epoch [7/20] - Train Loss: 0.0649, Train Acc: 0.9790, Test Loss: 0.0902, Test Acc: 0.9722
Epoch [8/20]: 100%|██████████| 938/938 [00:18<00:00, 49.63batch/s, Loss=0.0573]
Epoch [8/20] - Train Loss: 0.0573, Train Acc: 0.9815, Test Loss: 0.0967, Test Acc: 0.9726
Epoch [9/20]: 100%|██████████| 938/938 [00:19<00:00, 49.12batch/s, Loss=0.0522]
Epoch [9/20] - Train Loss: 0.0522, Train Acc: 0.9829, Test Loss: 0.0790, Test Acc: 0.9758
Epoch [10/20]: 100%|██████████| 938/938 [00:18<00:00, 50.84batch/s, Loss=0.0466]
Epoch [10/20] - Train Loss: 0.0466, Train Acc: 0.9847, Test Loss: 0.1033, Test Acc: 0.9705
Epoch [11/20]: 100%|██████████| 938/938 [00:19<00:00, 46.90batch/s, Loss=0.0450]
Epoch [11/20] - Train Loss: 0.0450, Train Acc: 0.9849, Test Loss: 0.0771, Test Acc: 0.9762
Epoch [12/20]: 100%|██████████| 938/938 [00:19<00:00, 47.09batch/s, Loss=0.0418]
Epoch [12/20] - Train Loss: 0.0418, Train Acc: 0.9860, Test Loss: 0.0842, Test Acc: 0.9750
Epoch [13/20]: 100%|██████████| 938/938 [00:18<00:00, 50.96batch/s, Loss=0.0361]
Epoch [13/20] - Train Loss: 0.0361, Train Acc: 0.9882, Test Loss: 0.0843, Test Acc: 0.9766
Epoch [14/20]: 100%|██████████| 938/938 [00:18<00:00, 50.64batch/s, Loss=0.0364]
Epoch [14/20] - Train Loss: 0.0364, Train Acc: 0.9879, Test Loss: 0.1025, Test Acc: 0.9751
Epoch [15/20]: 100%|██████████| 938/938 [00:19<00:00, 49.25batch/s, Loss=0.0335]
Epoch [15/20] - Train Loss: 0.0335, Train Acc: 0.9887, Test Loss: 0.0858, Test Acc: 0.9769
Epoch [16/20]: 100%|██████████| 938/938 [00:18<00:00, 49.82batch/s, Loss=0.0309]
Epoch [16/20] - Train Loss: 0.0309, Train Acc: 0.9900, Test Loss: 0.1016, Test Acc: 0.9747
Epoch [17/20]: 100%|██████████| 938/938 [00:18<00:00, 51.06batch/s, Loss=0.0283]
Epoch [17/20] - Train Loss: 0.0283, Train Acc: 0.9908, Test Loss: 0.0940, Test Acc: 0.9760
Epoch [18/20]: 100%|██████████| 938/938 [00:19<00:00, 48.71batch/s, Loss=0.0263]
Epoch [18/20] - Train Loss: 0.0263, Train Acc: 0.9910, Test Loss: 0.1253, Test Acc: 0.9697
Epoch [19/20]: 100%|██████████| 938/938 [00:19<00:00, 48.68batch/s, Loss=0.0265]
Epoch [19/20] - Train Loss: 0.0265, Train Acc: 0.9913, Test Loss: 0.0986, Test Acc: 0.9735
Epoch [20/20]: 100%|██████████| 938/938 [00:18<00:00, 51.20batch/s, Loss=0.0243]
Epoch [20/20] - Train Loss: 0.0243, Train Acc: 0.9919, Test Loss: 0.0872, Test Acc: 0.9767
```




```
torch.save(model.state_dict(), 'pretrained_model.pth')
```

Finetune class 3 on Pretrained model

```
model.load_state_dict(torch.load('pretrained_model.pth'))
model.eval()
```

```
<ipython-input-18-4e3a78d5b91c>:1: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defa
  model.load_state_dict(torch.load('pretrained_model.pth'))
SimpleNN(
  (fc1): Linear(in_features=784, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
```

```
    (fc3): Linear(in_features=128, out_features=64, bias=True)
    (fc4): Linear(in_features=64, out_features=10, bias=True)
  )
```

```
# Load class 3 samples
extracted_data_tensor, extracted_targets_tensor = torch.load('mnist_class_3_samples.pt')
```

```
<ipython-input-19-555f51a1b071>:2: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defa
  extracted_data_tensor, extracted_targets_tensor = torch.load('mnist_class_3_samples.pt')
```

```
from torch.utils.data import DataLoader, TensorDataset

# Create a DataLoader for the class 3 samples
class_3_dataset = TensorDataset(extracted_data_tensor, extracted_targets_tensor)
class_3_loader = DataLoader(class_3_dataset, batch_size=64, shuffle=True)
```

```
# Fine-tuning function
def fine_tune_model(model, train_loader, test_loader, num_epochs=10):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Fine-tuning on {device}")

    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(num_epochs):
        # Training phase
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)

        train_loss = running_loss / len(train_loader)
        train_accuracy = correct / total

        # Evaluation phase
        model.eval()
        test_loss = 0.0
        correct_test = 0
        total_test = 0

        with torch.no_grad():
            for inputs, labels in test_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                test_loss += loss.item()

                _, predicted = torch.max(outputs, 1)
                correct_test += (predicted == labels).sum().item()
                total_test += labels.size(0)

        test_loss /= len(test_loader)
        test_accuracy = correct_test / total_test

        # Print results for this epoch
        print(f"Epoch [{epoch+1}/{num_epochs}] - "
              f"Train Loss: {train_loss:.4f}, Train Acc: {train_accuracy:.4f}, "
              f"Test Loss: {test_loss:.4f}, Test Acc: {test_accuracy:.4f}")
```

```
# Fine-tune the model on class 3 samples
fine_tune_model(model, train_loader, test_loader, num_epochs=10)
```

```
Fine-tuning on cuda
Epoch [1/10] - Train Loss: 0.0298, Train Acc: 0.9918, Test Loss: 0.1048, Test Acc: 0.9773
Epoch [2/10] - Train Loss: 0.0225, Train Acc: 0.9925, Test Loss: 0.0974, Test Acc: 0.9786
```

```
Epoch [3/10] - Train Loss: 0.0232, Train Acc: 0.9919, Test Loss: 0.0973, Test Acc: 0.9775
Epoch [4/10] - Train Loss: 0.0200, Train Acc: 0.9934, Test Loss: 0.1148, Test Acc: 0.9749
Epoch [5/10] - Train Loss: 0.0206, Train Acc: 0.9933, Test Loss: 0.0857, Test Acc: 0.9802
Epoch [6/10] - Train Loss: 0.0208, Train Acc: 0.9935, Test Loss: 0.0839, Test Acc: 0.9814
Epoch [7/10] - Train Loss: 0.0188, Train Acc: 0.9939, Test Loss: 0.1097, Test Acc: 0.9767
Epoch [8/10] - Train Loss: 0.0215, Train Acc: 0.9925, Test Loss: 0.1120, Test Acc: 0.9785
Epoch [9/10] - Train Loss: 0.0196, Train Acc: 0.9940, Test Loss: 0.1074, Test Acc: 0.9798
Epoch [10/10] - Train Loss: 0.0213, Train Acc: 0.9932, Test Loss: 0.1372, Test Acc: 0.9698
```

```
import numpy as np
import torch
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

def evaluate_model(model, test_loader):
    # Set the model to evaluation mode
    model.eval()
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Evaluating on {device}")

    # Move the model to the correct device
    model.to(device) # This line is added to move the model to the device

    # Initialize variables to track the predictions and true labels
    all_predictions = []
    all_labels = []

    # No gradients needed for evaluation
    with torch.no_grad():
        for inputs, labels in test_loader:
            # Move the data to the same device as the model (GPU or CPU)
            inputs, labels = inputs.to(device), labels.to(device)

            # Forward pass: Get model predictions
            outputs = model(inputs)

            # Get the predicted class by finding the class with the highest score
            _, predicted = torch.max(outputs, 1)

            # Store the predictions and true labels for metric calculation
            all_predictions.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    # Convert lists to numpy arrays
    all_predictions = np.array(all_predictions)
    all_labels = np.array(all_labels)

    # Generate classification report
    cr = classification_report(all_labels, all_predictions)
    print(f'Classification report for test data:\n{cr}')

    # Generate confusion matrix
    cm = confusion_matrix(all_labels, all_predictions)

    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=range(10), yticklabels=range(10))

    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

# Example usage after fine-tuning:
# evaluate_model(model, test_loader)
```

```
# Load the pre-trained model
pretrained_model = SimpleNN()
pretrained_model.load_state_dict(torch.load('pretrained_model.pth'))

# Load the fine-tuned model
fine_tuned_model = SimpleNN()
fine_tuned_model.load_state_dict(torch.load('fine_tuned_model_class_3.pth'))
```

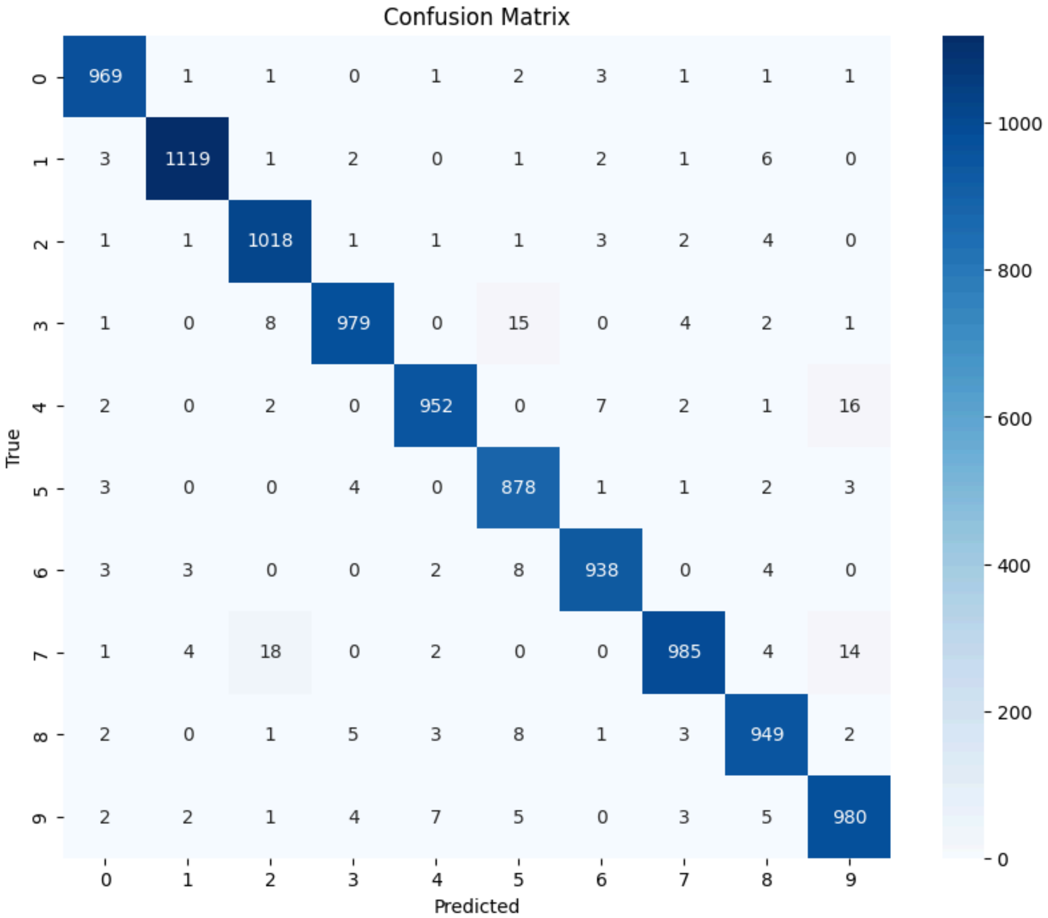
```
<ipython-input-34-9f8050fb60df>:3: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defa
  pretrained_model.load_state_dict(torch.load('pretrained_model.pth'))
<ipython-input-34-9f8050fb60df>:7: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defa
  fine_tuned_model.load_state_dict(torch.load('fine_tuned_model_class_3.pth'))
<All keys matched successfully>
```

```
evaluate_model(pretrained_model, test_loader)
```

```
Evaluating on cuda
Classification report for test data:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.97      0.99      0.98      1032
           3       0.98      0.97      0.98      1010
           4       0.98      0.97      0.98       982
           5       0.96      0.98      0.97       892
           6       0.98      0.98      0.98       958
           7       0.98      0.96      0.97      1028
           8       0.97      0.97      0.97       974
           9       0.96      0.97      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

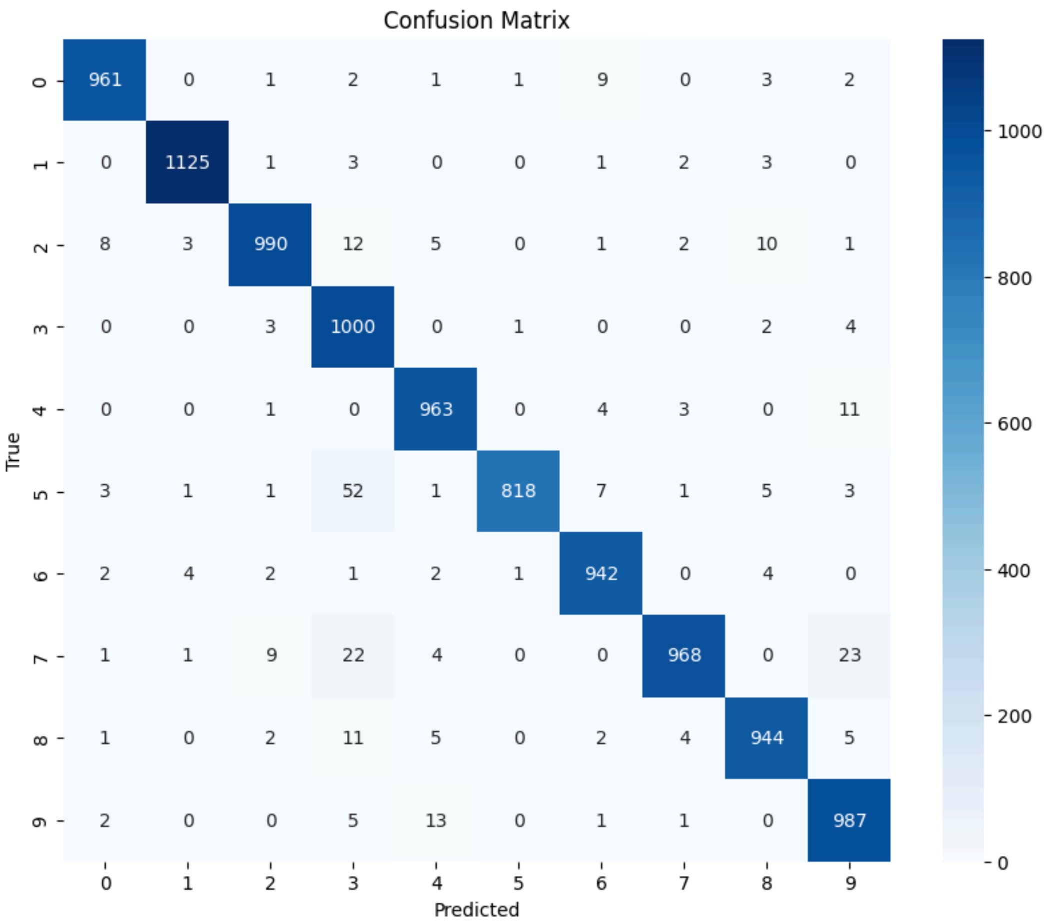



```
evaluate_model(fine_tuned_model, test_loader)
```

```
Evaluating on cuda
Classification report for test data:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.96      0.97      1032
           3       0.90      0.99      0.94      1010
           4       0.97      0.98      0.97       982
           5       1.00      0.92      0.96       892
           6       0.97      0.98      0.98       958
           7       0.99      0.94      0.96      1028
           8       0.97      0.97      0.97       974
           9       0.95      0.98      0.97      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```




```python
# Save the fine-tuned model (optional)
torch.save(model.state_dict(), 'fine_tuned_model_class_3.pth')
print("Fine-tuned model saved as 'fine_tuned_model_class_3.pth'.")
```

```
Fine-tuned model saved as 'fine_tuned_model_class_3.pth'.
```

Computing Task Vector

```python
# Compute the task vector by subtracting weights
task_vector = {}
for key in pretrained_model.state_dict():
    task_vector[key] = fine_tuned_model.state_dict()[key] - pretrained_model.state_dict()[key]
```

```python
task_vector
```

```
{'fc1.weight': tensor([[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [-0.0036, -0.0036, -0.0036,  ..., -0.0036, -0.0036, -0.0036],
         [ 0.0059,  0.0059,  0.0059,  ...,  0.0059,  0.0059,  0.0059],
         ...,
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
```

```
         [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000]]),
 'fc1.bias': tensor([ 0.0000e+00,  3.6207e-03, -5.8726e-03,  3.0456e-03,  1.4845e-03,
          8.8214e-03, -3.4774e-02,  0.0000e+00, -2.0420e-03, -5.8250e-03,
          0.0000e+00,  0.0000e+00,  4.3508e-03,  6.1808e-03,  0.0000e+00,
          9.6620e-03,  0.0000e+00,  6.9300e-03, -1.2320e-03,  0.0000e+00,
          9.0672e-04, -5.9627e-03, -2.3357e-03, -8.9148e-03, -5.2554e-03,
         -8.7910e-03,  0.0000e+00,  0.0000e+00,  0.0000e+00, -1.1272e-02,
         -5.9527e-03, -6.4054e-03,  0.0000e+00, -6.7748e-03,  7.0948e-04,
          0.0000e+00,  1.1651e-02, -5.1257e-04,  4.0067e-03,  0.0000e+00,
         -4.6074e-03,  0.0000e+00,  1.6545e-04, -3.3594e-03,  0.0000e+00,
         -7.8371e-03,  0.0000e+00,  0.0000e+00, -2.7559e-04,  0.0000e+00,
         -6.8689e-03,  0.0000e+00, -9.9365e-03,  5.4153e-04,  0.0000e+00,
          0.0000e+00, -4.5069e-03, -9.7297e-03,  0.0000e+00,  0.0000e+00,
         -3.7652e-03,  0.0000e+00,  3.8035e-03,  0.0000e+00, -2.7729e-03,
          2.1602e-02,  0.0000e+00,  0.0000e+00, -1.2536e-03,  6.0286e-03,
          5.6489e-03,  0.0000e+00,  0.0000e+00, -6.4178e-03,  0.0000e+00,
         -5.8806e-03, -1.3798e-02,  0.0000e+00,  0.0000e+00,  0.0000e+00,
          0.0000e+00, -8.4446e-03,  0.0000e+00, -1.1259e-02,  4.4170e-03,
          3.5190e-03,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
          0.0000e+00, -5.7200e-03,  2.7254e-03, -5.9838e-03,  0.0000e+00,
          0.0000e+00, -8.3311e-03, -1.6194e-02,  0.0000e+00,  0.0000e+00,
         -1.5793e-02,  0.0000e+00,  0.0000e+00,  0.0000e+00,  8.6636e-03,
         -9.9012e-04,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
          0.0000e+00,  0.0000e+00,  0.0000e+00, -3.2537e-03,  2.1158e-02,
         -7.9147e-03, -1.2230e-02,  0.0000e+00,  0.0000e+00,  0.0000e+00,
         -1.0061e-02,  0.0000e+00,  1.0134e-03, -7.3826e-03, -3.7873e-03,
          4.7684e-03, -1.3491e-02,  0.0000e+00, -2.2570e-03, -1.0157e-02,
         -1.1038e-02,  3.2829e-03, -8.2178e-04,  0.0000e+00,  7.2865e-04,
          4.2779e-05,  0.0000e+00,  2.0426e-03, -6.1435e-03,  3.0611e-03,
         -4.1246e-03, -1.8387e-02, -6.0286e-03, -1.1705e-02,  9.7691e-03,
          0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
          0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
          3.2175e-03, -1.1210e-04,  0.0000e+00,  0.0000e+00,  0.0000e+00,
          0.0000e+00,  1.0654e-02, -3.1298e-03,  0.0000e+00, -8.0806e-03,
          3.4253e-03,  8.7704e-03,  6.6218e-03,  0.0000e+00,  0.0000e+00,
         -7.4653e-03,  0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,
         -5.8894e-03, -1.4272e-02, -2.3097e-07, -4.6870e-03,  0.0000e+00,
         -6.7840e-03,  0.0000e+00,  0.0000e+00, -1.4182e-02, -1.3959e-02,
         -6.3678e-03,  0.0000e+00, -1.5090e-02,  0.0000e+00,  0.0000e+00,
         -1.1147e-02, -1.6695e-02,  0.0000e+00, -1.3176e-02,  0.0000e+00,
          0.0000e+00,  8.5976e-03,  1.3594e-02,  0.0000e+00, -2.6453e-02,
          0.0000e+00,  0.0000e+00,  4.6270e-03,  8.9977e-03,  5.8943e-03,
          0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00, -5.1944e-03,
          0.0000e+00, -6.1755e-03, -2.0492e-03, -6.1244e-03,  0.0000e+00,
          1.9795e-03,  1.6111e-03, -7.3105e-03,  6.4925e-03,  4.3189e-03,
         -8.6614e-03,  0.0000e+00,  0.0000e+00, -6.0743e-03,  0.0000e+00,
         -7.1512e-03,  0.0000e+00, -5.3036e-03, -1.0237e-02,  3.7253e-09,
          4.0862e-03,  6.8184e-03, -3.6388e-03, -1.7027e-02,  1.1231e-02,
          0.0000e+00,  0.0000e+00,  0.0000e+00,  0.0000e+00,  1.3331e-02,
          3.3401e-03,  5.8948e-04, -2.8619e-03,  2.0700e-02,  0.0000e+00,
          5.5243e-04,  0.0000e+00,  1.9641e-02, -1.9058e-02,  5.3048e-03,
```

```
# Save the task vector if needed
torch.save(task_vector, 'task_vector.pth')

print("Task vector computed and saved as 'task_vector.pth'.")
```

```
Task vector computed and saved as 'task_vector.pth'.
```

```
pip install notebook-as-pdf
```

```
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-as-pdf) (
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-as-pdf)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-as
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-
Requirement already satisfied: markupsafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-as-p
Requirement already satisfied: mistune<4,>=2.0.3 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-as
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-as-p
Requirement already satisfied: nbformat>=5.7 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-as-pdf
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-as-pdf) (2
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook
Requirement already satisfied: pygments>=2.4.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-as-p
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-as-pdf) (1.
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->notebook-as-pd
Collecting appdirs<2.0.0,>=1.4.3 (from pyppeteer->notebook-as-pdf)
  Downloading appdirs-1.4.4-py2.py3-none-any.whl.metadata (9.0 kB)
Requirement already satisfied: certifi>=2023 in /usr/local/lib/python3.10/dist-packages (from pyppeteer->notebook-as-pdf
Requirement already satisfied: importlib-metadata>=1.4 in /usr/local/lib/python3.10/dist-packages (from pyppeteer->noteb
Collecting pyee<12.0.0,>=11.0.0 (from pyppeteer->notebook-as-pdf)
```

```
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.7->nbc
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.10/dist-packages (from nbclient>=0.5.0->
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.10/dist-packages (from nbformat>=5.7->nbco
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat>=5.7->nbconvert
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from pyee<12.0.0,>=11.0.0->
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->nbconvert-
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jso
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbf
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-packages (from jupyter-client>=6.1.12->nbclie
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.10/dist-packages (from jupyter-client>=6.1
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.10/dist-packages (from jupyter-client>=6.1.12->nbc
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.1->jupyter-c
Downloading notebook_as_pdf-0.5.0-py3-none-any.whl (6.5 kB)
Downloading pypdf2-3.0.1-py3-none-any.whl (232 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.6/232.6 kB 11.9 MB/s eta 0:00:00
Downloading pyppeteer-2.0.0-py3-none-any.whl (82 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 82.9/82.9 kB 7.8 MB/s eta 0:00:00
Downloading appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Downloading pyee-11.1.1-py3-none-any.whl (15 kB)
Downloading urllib3-1.26.20-py2.py3-none-any.whl (144 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 144.2/144.2 kB 14.5 MB/s eta 0:00:00
Downloading websockets-10.4-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 106.8/106.8 kB 10.3 MB/s eta 0:00:00
Installing collected packages: appdirs, websockets, urllib3, PyPDF2, pyee, pyppeteer, notebook-as-pdf
  Attempting uninstall: urllib3
    Found existing installation: urllib3 2.2.3
    Uninstalling urllib3-2.2.3:
      Successfully uninstalled urllib3-2.2.3
Successfully installed PyPDF2-3.0.1 appdirs-1.4.4 notebook-as-pdf-0.5.0 pyee-11.1.1 pyppeteer-2.0.0 urllib3-1.26.20 webs
```

Start coding or generate with AI.

```
[NbConvertApp] WARNING | pattern 'task-vector.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include the error message in the cell outpu
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_dir
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_dir
--coalesce-streams
    Coalesce consecutive stdout and stderr outputs into one stream (within each cell).
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_dir
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports
```