Classify_image_dataset_p7

November 12, 2025

Write a program to enable pre-train models to classify a given image dataset

[1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

# Step 1: Transform (normalize MNIST)
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Step 2: Load MNIST dataset (28x28 grayscale)
trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True,␣
 ↪transform=transform)
testset = torchvision.datasets.MNIST(root='./data', train=False, download=True,␣
 ↪transform=transform)

trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testloader = DataLoader(testset, batch_size=1000, shuffle=False)

# Step 3: Define a simple CNN model
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, 3, 1)
        self.conv2 = nn.Conv2d(16, 32, 3, 1)
        self.fc1 = nn.Linear(5*5*32, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(2, 2)

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(-1, 5*5*32)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Step 4: Initialize model, loss, optimizer
model = SimpleCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Step 5: Train (only 1 epoch for demo)
for epoch in range(1):
    for images, labels in trainloader:
        outputs = model(images)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print(f"Epoch [{epoch+1}/1], Loss: {loss.item():.4f}")

# Step 6: Test
correct, total = 0, 0
model.eval()
with torch.no_grad():
    for images, labels in testloader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Accuracy on test set: {100 * correct / total:.2f}%")
```

```
Epoch [1/1], Loss: 0.0155
Accuracy on test set: 97.89%
```

[ ]: