CNN_for_classification_of_image_dataset_p3

October 22, 2025

Desing and implement a Convolutional Neural Network(CNN) for classification of image dataset

[1]:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms

# ------------------------------
# 1. Load Dataset (MNIST)
# ------------------------------
transform = transforms.Compose([transforms.ToTensor()])

train_dataset = torchvision.datasets.MNIST(root='./data', train=True,
                                           download=True, transform=transform)
test_dataset = torchvision.datasets.MNIST(root='./data', train=False,
                                          download=True, transform=transform)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=64, shuffle=False)

# ------------------------------
# 2. Define CNN Model
# ------------------------------
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, padding=1)   # 1→16
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)  # 16→32
        self.fc1 = nn.Linear(32 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)  # 10 classes (digits 0-9)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))   # [1,28,28] → [16,14,14]
        x = self.pool(F.relu(self.conv2(x)))   # [16,14,14] → [32,7,7]
        x = x.view(-1, 32 * 7 * 7)             # flatten
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# ------------------------------
# 3. Initialize Model
# ------------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# ------------------------------
# 4. Training
# ------------------------------
epochs = 5
for epoch in range(epochs):
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss/len(train_loader):.
 ↪4f}")

# ------------------------------
# 5. Evaluation
# ------------------------------
correct, total = 0, 0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"\nTest Accuracy: {100 * correct / total:.2f}%")
```

```
100%|
  | 9.91M/9.91M [00:15<00:00, 645kB/s]
100%|
  | 28.9k/28.9k [00:00<00:00, 124kB/s]
100%|
  | 1.65M/1.65M [00:04<00:00, 359kB/s]
100%|
      | 4.54k/4.54k [00:00<?, ?B/s]

Epoch 1/5, Loss: 0.2247
Epoch 2/5, Loss: 0.0601
Epoch 3/5, Loss: 0.0425
Epoch 4/5, Loss: 0.0320
Epoch 5/5, Loss: 0.0247

Test Accuracy: 99.05%
```

```
[ ]:
```