CO 543 Image Processing

Lab 07 – Report

Jayawardena H.D.N.S.

E/19/167

Convolutional Neural Networks

Task 1

Code

```python
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

# Define transformations for the training and testing data
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize to 224x224 for AlexNet compatibility
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

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

# Data loaders
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=1000, shuffle=False)

# Load pre-trained ResNet18 and modify the first and final layers
import torchvision.models as models

resnet18 = models.resnet18(pretrained=False)
resnet18.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3, bias=False)
resnet18.fc = nn.Linear(resnet18.fc.in_features, 10)

# Load pre-trained AlexNet and modify the first and final layers
alexnet = models.alexnet(pretrained=False)
alexnet.features[0] = nn.Conv2d(1, 64, kernel_size=11, stride=4, padding=2)
alexnet.classifier[6] = nn.Linear(alexnet.classifier[6].in_features, 10)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

resnet18 = resnet18.to(device)
alexnet = alexnet.to(device)
criterion = nn.CrossEntropyLoss()

# Optimizers
resnet18_optimizer = optim.Adam(resnet18.parameters(), lr=0.001)
alexnet_optimizer = optim.Adam(alexnet.parameters(), lr=0.001)
```

```python
def train_model(model, train_loader, criterion, optimizer, num_epochs=10):
    model.train()
    for epoch in range(num_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()

        epoch_loss = running_loss / len(train_loader)
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}')
    return model

def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 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.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    return accuracy

# Train ResNet18
print("Training ResNet18...")
resnet18 = train_model(resnet18, train_loader, criterion, resnet18_optimizer)

# Train AlexNet
print("Training AlexNet...")
alexnet = train_model(alexnet, train_loader, criterion, alexnet_optimizer)

# Evaluate the models
resnet18_accuracy = evaluate_model(resnet18, test_loader)
alexnet_accuracy = evaluate_model(alexnet, test_loader)
```

```python
print(f'ResNet18 Accuracy: {resnet18_accuracy:.2f}%')
print(f'AlexNet Accuracy: {alexnet_accuracy:.2f}%')
```

Output

```
Training ResNet18...
Epoch [1/10], Loss: 0.0977
Epoch [2/10], Loss: 0.0422
Epoch [3/10], Loss: 0.0316
Epoch [4/10], Loss: 0.0259
Epoch [5/10], Loss: 0.0239
Epoch [6/10], Loss: 0.0207
Epoch [7/10], Loss: 0.0176
Epoch [8/10], Loss: 0.0157
Epoch [9/10], Loss: 0.0134
Epoch [10/10], Loss: 0.0102
Training AlexNet...
/usr/local/lib/python3.10/dis
  return F.conv2d(input, weig
Epoch [1/10], Loss: 0.3748
Epoch [2/10], Loss: 0.0941
Epoch [3/10], Loss: 0.0792
Epoch [4/10], Loss: 0.0675
Epoch [5/10], Loss: 0.0637
Epoch [6/10], Loss: 0.0551
Epoch [7/10], Loss: 0.0556
Epoch [8/10], Loss: 0.0530
```

Task 2

1. **Training the models**

Codes

Step 1: Define Models A and B

```
] 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

  # Define Model A
  class ModelA(nn.Module):
      def __init__(self, activation_fn=nn.ReLU()):
          super(ModelA, self).__init__()
          self.conv1 = nn.Conv2d(in_channels=1, out_channels=8, kernel_size=3, padding=1)
          self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
          self.fc1 = nn.Linear(8 * 14 * 14, 64)
          self.fc2 = nn.Linear(64, 10)
          self.activation_fn = activation_fn

      def forward(self, x):
          x = self.pool(self.activation_fn(self.conv1(x)))
          x = x.view(-1, 8 * 14 * 14)
          x = self.activation_fn(self.fc1(x))
          x = self.fc2(x)
          return x

  # Define Model B
  class ModelB(nn.Module):
      def __init__(self):
          super(ModelB, self).__init__()
          self.conv1 = nn.Conv2d(in_channels=1, out_channels=8, kernel_size=3, padding=1)
          self.conv2 = nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3, padding=1)
          self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
          self.fc1 = nn.Linear(16 * 7 * 7, 64)  # Adjusted the dimensions here
          self.fc2 = nn.Linear(64, 10)
          self.relu = nn.ReLU()

      def forward(self, x):
          x = self.pool(self.relu(self.conv1(x)))
          x = self.pool(self.relu(self.conv2(x)))
          x = x.view(-1, 16 * 7 * 7)  # Adjusted the dimensions here
          x = self.relu(self.fc1(x))
          x = self.fc2(x)
          return x
```

Step 2: Prepare the MNIST Dataset

```
# Data loading and transformation
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

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

train_loader = DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=1000, shuffle=False)
```

Step 3: Define Training and Testing Functions

```
# Training and testing functions
def train(model, device, train_loader, optimizer, criterion, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print(f'Train Epoch: {epoch} [{batch_idx * len(data)}/{len(train_loader.dataset)} ({100. * batch_idx / len(train_loader):.0f}%)]\tLoss: {loss.item():.6f}')

def test(model, device, test_loader, criterion):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += criterion(output, target).item()
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    print(f'\nTest set: Average loss: {test_loss:.4f}, Accuracy: {correct}/{len(test_loader.dataset)} ({100. * correct / len(test_loader.dataset):.0f}%)\n')
```

Step 4: Train Model A and Model B

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Training Model A

print("Training Model A with ReLU")
modelA = ModelA(nn.ReLU()).to(device)
optimizerA = optim.Adam(modelA.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

for epoch in range(1, 11):
    train(modelA, device, train_loader, optimizerA, criterion, epoch)
    test(modelA, device, test_loader, criterion)

# Training Model B
print("\nTraining Model B with ReLU")
modelB = ModelB().to(device)
criterion = nn.CrossEntropyLoss()
optimizerB = optim.Adam(modelB.parameters(), lr=0.001)

# Training the model
for epoch in range(1, 11):
    train(modelB, device, train_loader, optimizerB, criterion, epoch)
    test(modelB, device, test_loader, criterion)
```

Output

```
Training Model A with ReLU
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.309469
Train Epoch: 1 [6400/60000 (11%)]       Loss: 0.496667
Train Epoch: 1 [12800/60000 (21%)]      Loss: 0.216169
Train Epoch: 1 [19200/60000 (32%)]      Loss: 0.155260
Train Epoch: 1 [25600/60000 (43%)]      Loss: 0.044719
Train Epoch: 1 [32000/60000 (53%)]      Loss: 0.103688
Train Epoch: 1 [38400/60000 (64%)]      Loss: 0.144899
Train Epoch: 1 [44800/60000 (75%)]      Loss: 0.216667
Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.266260
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.198395

Test set: Average loss: 0.0001, Accuracy: 9674/10000 (97%)

Train Epoch: 2 [0/60000 (0%)]   Loss: 0.096442
Train Epoch: 2 [6400/60000 (11%)]       Loss: 0.070526
Train Epoch: 2 [12800/60000 (21%)]      Loss: 0.081467
Train Epoch: 2 [19200/60000 (32%)]      Loss: 0.149162
Train Epoch: 2 [25600/60000 (43%)]      Loss: 0.116094
Train Epoch: 2 [32000/60000 (53%)]      Loss: 0.165182
Train Epoch: 2 [38400/60000 (64%)]      Loss: 0.153417
Train Epoch: 2 [44800/60000 (75%)]      Loss: 0.063115
Train Epoch: 2 [51200/60000 (85%)]      Loss: 0.118036
Train Epoch: 2 [57600/60000 (96%)]      Loss: 0.047883

Test set: Average loss: 0.0001, Accuracy: 9730/10000 (97%)

Train Epoch: 3 [0/60000 (0%)]   Loss: 0.019795
Train Epoch: 3 [6400/60000 (11%)]       Loss: 0.101508
Train Epoch: 3 [12800/60000 (21%)]      Loss: 0.047491
Train Epoch: 3 [19200/60000 (32%)]      Loss: 0.254441
Train Epoch: 3 [25600/60000 (43%)]      Loss: 0.132478
Train Epoch: 3 [32000/60000 (53%)]      Loss: 0.044903
Train Epoch: 3 [38400/60000 (64%)]      Loss: 0.084640
Train Epoch: 3 [44800/60000 (75%)]      Loss: 0.191737
Train Epoch: 3 [51200/60000 (85%)]      Loss: 0.018068
Train Epoch: 3 [57600/60000 (96%)]      Loss: 0.088544

Test set: Average loss: 0.0001, Accuracy: 9790/10000 (98%)

Train Epoch: 4 [0/60000 (0%)]   Loss: 0.020506
Train Epoch: 4 [6400/60000 (11%)]       Loss: 0.038153
Train Epoch: 4 [12800/60000 (21%)]      Loss: 0.086355
Train Epoch: 4 [19200/60000 (32%)]      Loss: 0.026429
Train Epoch: 4 [25600/60000 (43%)]      Loss: 0.007126
Train Epoch: 4 [32000/60000 (53%)]      Loss: 0.024345
Train Epoch: 4 [38400/60000 (64%)]      Loss: 0.021770
Train Epoch: 4 [44800/60000 (75%)]      Loss: 0.017982
```

2. Observations

Model A

Epoch no.	Initial training loss	Final training loss	Test accuracy
1	2.309469	0.198395	97%
2	0.096442	0.047883	97.3%
3	0.019795	0.088544	97.9%
4	0.020506	0.030304	97.95%
5	0.100589	0.130388	98.1%
6	0.027961	0.073530	98.04%
7	0.008292	0.005999	98.19%
8	0.073184	0.006126	98.19%
9	0.052302	0.012445	98.34%
10	0.001432	0.007731	98.43%

Model B

Epoch no.	Initial training loss	Final training loss	Test accuracy
1	2.287307	0.233523	97.35%
2	0.071309	0.173580	98.29%
3	0.016599	0.023260	98.31%
4	0.020217	0.016248	98.62%
5	0.011492	0.012736	98.57%
6	0.068760	0.001598	98.81%
7	0.017033	0.011569	98.49%
8	0.036058	0.025450	98.82%
9	0.004083	0.002773	98.59%
10	0.006944	0.004947	98.94%

Both models show a significant decrease in training loss from the initial to the final epoch, indicating effective learning. Model B generally shows slightly lower training loss towards the final epochs compared to Model A.

Both models achieved high accuracy on the test set, consistently above 97%. Model B shows a slight edge in final accuracy, reaching up to 98.94% in the final epoch, compared to Model A's 98.43%.

Both models performed well with the ReLU activation function. However, Model B demonstrated slightly better performance in terms of final test accuracy and consistently lower training loss towards the end of the training epochs. This indicates that Model B may have a slight advantage in terms of generalization and convergence efficiency compared to Model A.

3. **the reason for these observed differences in performance**

Additional Convolutional Layer in Model B:

- **Feature Extraction**: Model B has an additional convolutional layer (16 features) compared to Model A. This extra layer allows Model B to extract more complex and higher-level features from the input images. The first convolutional layer captures basic features like edges and textures, while the second convolutional layer can capture more complex patterns by combining these basic features.
- **Receptive Field**: With an extra convolutional layer, Model B has a larger receptive field, meaning each neuron in the deeper layers can "see" a larger portion of the input image. This helps the model learn more spatial hierarchies and intricate details, leading to better performance.

Increased Model Capacity:

- **More Parameters**: The additional convolutional layer in Model B increases the number of parameters, thus enhancing the model's capacity to learn and represent more complex functions. While more parameters increase the risk of overfitting, they also allow the model to fit the training data better if regularization is properly managed.
- **Better Representation**: The two convolutional layers in Model B allow it to represent the input data in a more nuanced manner before feeding it to the fully connected layers. This richer representation results in better classification accuracy.

Improved Generalization:

- **Learning More Robust Features**: The deeper architecture in Model B helps in learning more robust and invariant features, which improves its generalization performance on unseen test data. This is evident from the higher accuracy of Model B on the test set.

4. Explore the effect of different activation functions.

a. Train model A with all nonlinear activation functions set to ReLu
   ✓ Done in the first part

b. Train the model A with all nonlinear activation functions set to Sigmoid

Code

```
modelA_sigmoid = ModelA(activation_fn=nn.Sigmoid()).to(device)
optimizerA_sigmoid = optim.Adam(modelA_sigmoid.parameters(), lr=0.001)

for epoch in range(1, 11):
    train(modelA_sigmoid, device, train_loader, optimizerA_sigmoid, criterion, epoch)
    test(modelA_sigmoid, device, test_loader, criterion)
```

Output

```
Train Epoch: 5 [0/60000 (0%)]   Loss: 0.101456
Train Epoch: 5 [6400/60000 (11%)]       Loss: 0.187287
Train Epoch: 5 [12800/60000 (21%)]      Loss: 0.085459
Train Epoch: 5 [19200/60000 (32%)]      Loss: 0.143040
Train Epoch: 5 [25600/60000 (43%)]      Loss: 0.137267
Train Epoch: 5 [32000/60000 (53%)]      Loss: 0.173666
Train Epoch: 5 [38400/60000 (64%)]      Loss: 0.104995
Train Epoch: 5 [44800/60000 (75%)]      Loss: 0.154941
Train Epoch: 5 [51200/60000 (85%)]      Loss: 0.078908
Train Epoch: 5 [57600/60000 (96%)]      Loss: 0.122254

Test set: Average loss: 0.0001, Accuracy: 9612/10000 (96%)

Train Epoch: 6 [0/60000 (0%)]   Loss: 0.106562
Train Epoch: 6 [6400/60000 (11%)]       Loss: 0.045859
Train Epoch: 6 [12800/60000 (21%)]      Loss: 0.249199
Train Epoch: 6 [19200/60000 (32%)]      Loss: 0.101586
Train Epoch: 6 [25600/60000 (43%)]      Loss: 0.156552
Train Epoch: 6 [32000/60000 (53%)]      Loss: 0.120401
Train Epoch: 6 [38400/60000 (64%)]      Loss: 0.156034
Train Epoch: 6 [44800/60000 (75%)]      Loss: 0.157172
Train Epoch: 6 [51200/60000 (85%)]      Loss: 0.063193
Train Epoch: 6 [57600/60000 (96%)]      Loss: 0.102051

Test set: Average loss: 0.0001, Accuracy: 9628/10000 (96%)

Train Epoch: 7 [0/60000 (0%)]   Loss: 0.044460
Train Epoch: 7 [6400/60000 (11%)]       Loss: 0.152442
Train Epoch: 7 [12800/60000 (21%)]      Loss: 0.130897
Train Epoch: 7 [19200/60000 (32%)]      Loss: 0.120098
Train Epoch: 7 [25600/60000 (43%)]      Loss: 0.156440
Train Epoch: 7 [32000/60000 (53%)]      Loss: 0.054007
Train Epoch: 7 [38400/60000 (64%)]      Loss: 0.115444
Train Epoch: 7 [44800/60000 (75%)]      Loss: 0.095212
Train Epoch: 7 [51200/60000 (85%)]      Loss: 0.079576
Train Epoch: 7 [57600/60000 (96%)]      Loss: 0.116253
```

Observations

When Model A is trained with all nonlinear activation functions set to Sigmoid, we observe the following performance metrics:

1. **Initial Training Phase**:
    - **Epoch 1**: Rapid reduction in loss from 2.301739 to 0.182138, and test accuracy reaches 92%.
    - **Epoch 2**: Further improvement, with the loss continuing to decrease and test accuracy rising to 94%.
    - **Epoch 3-5**: Continuous improvement, reaching 96% accuracy by Epoch 5.
2. **Later Training Phase**:
    - **Epoch 6-10**: Gradual performance gains, with test accuracy peaking at 97% by Epoch 10.

Key Observations and Explanations

1. **Slower Convergence with Sigmoid**:
    - **Gradient Saturation**: Sigmoid functions suffer from gradient saturation (vanishing gradients) for very high or very low values, which can slow down training, especially in deeper networks. However, since Model A is relatively shallow, this issue is less pronounced but still affects convergence speed compared to ReLU.
2. **Smooth Training Progress**:
    - **Stability**: The training progress with sigmoid activations appears smooth, with consistent improvements in loss and accuracy across epochs. Sigmoid functions provide smooth gradients, leading to more stable updates during backpropagation.

c. Train the model A with all nonlinear activation functions set to tanh

Code

```
modelA_tanh = ModelA(activation_fn=nn.Tanh()).to(device)
optimizerA_tanh = optim.Adam(modelA_tanh.parameters(), lr=0.001)

for epoch in range(1, 11):
    train(modelA_tanh, device, train_loader, optimizerA_tanh, criterion, epoch)
    test(modelA_tanh, device, test_loader, criterion)
```

Output

```
Train Epoch: 6 [0/60000 (0%)]   Loss: 0.018372
Train Epoch: 6 [6400/60000 (11%)]       Loss: 0.024426
Train Epoch: 6 [12800/60000 (21%)]      Loss: 0.074415
Train Epoch: 6 [19200/60000 (32%)]      Loss: 0.019140
Train Epoch: 6 [25600/60000 (43%)]      Loss: 0.007493
Train Epoch: 6 [32000/60000 (53%)]      Loss: 0.047797
Train Epoch: 6 [38400/60000 (64%)]      Loss: 0.062198
Train Epoch: 6 [44800/60000 (75%)]      Loss: 0.017522
Train Epoch: 6 [51200/60000 (85%)]      Loss: 0.030699
Train Epoch: 6 [57600/60000 (96%)]      Loss: 0.014102

Test set: Average loss: 0.0001, Accuracy: 9789/10000 (98%)

Train Epoch: 7 [0/60000 (0%)]   Loss: 0.028182
Train Epoch: 7 [6400/60000 (11%)]       Loss: 0.027442
Train Epoch: 7 [12800/60000 (21%)]      Loss: 0.007551
Train Epoch: 7 [19200/60000 (32%)]      Loss: 0.023127
Train Epoch: 7 [25600/60000 (43%)]      Loss: 0.056503
Train Epoch: 7 [32000/60000 (53%)]      Loss: 0.014647
Train Epoch: 7 [38400/60000 (64%)]      Loss: 0.011827
Train Epoch: 7 [44800/60000 (75%)]      Loss: 0.006763
Train Epoch: 7 [51200/60000 (85%)]      Loss: 0.018571
Train Epoch: 7 [57600/60000 (96%)]      Loss: 0.016015

Test set: Average loss: 0.0001, Accuracy: 9810/10000 (98%)

Train Epoch: 8 [0/60000 (0%)]   Loss: 0.049304
Train Epoch: 8 [6400/60000 (11%)]       Loss: 0.014202
Train Epoch: 8 [12800/60000 (21%)]      Loss: 0.005823
Train Epoch: 8 [19200/60000 (32%)]      Loss: 0.003449
Train Epoch: 8 [25600/60000 (43%)]      Loss: 0.006950
Train Epoch: 8 [32000/60000 (53%)]      Loss: 0.009522
Train Epoch: 8 [38400/60000 (64%)]      Loss: 0.047987
Train Epoch: 8 [44800/60000 (75%)]      Loss: 0.003150
Train Epoch: 8 [51200/60000 (85%)]      Loss: 0.011382
Train Epoch: 8 [57600/60000 (96%)]      Loss: 0.036687

Test set: Average loss: 0.0001, Accuracy: 9812/10000 (98%)
```

d. Observations

Epoch-wise Detailed Loss Analysis:

- **Epoch 1:** Significant initial reduction in loss, suggesting the model quickly learns basic patterns.
- **Epoch 2 to Epoch 4:** Continued decrease in loss with high test accuracy (97-98%).
- **Epoch 5 to Epoch 10:** Loss values are very low, indicating fine-tuning of the model with minor improvements in accuracy.

Performance Stability:

- The test accuracy stabilizes at 98% from epoch 4 onward.
- The loss values remain low and consistent from epoch 5, suggesting the model has effectively learned the training data and is not overfitting.

Overall Model Behavior:

- The model with Tanh activation functions demonstrates rapid initial learning and reaches high accuracy quickly.
- Consistency in both training loss and test accuracy indicates a well-generalized model.

- The use of Tanh activation helps in maintaining a smooth and stable training process with effective gradient propagation.

Epoch no.	Sigmoid Activation		Tanh Activation	
	Loss	Test Acc.	Loss	Test acc.
1	2.301739 to 0.182138	92%	2.308387 to 0.158143	96%
2	0.403194 to 0.333733	94%	0.124535 to 0.092038	97%
3	0.339831 to 0.094472	95%	0.056472 to 0.062925	98%
4	0.197158 to 0.171716	95%	0.178942 to 0.036795	98%
5	0.101456 to 0.122254	96%	0.087920 to 0.004631	98%
6	0.106562 to 0.102051	96%	0.018372 to 0.014102	98%
7	0.044460 to 0.116253	96%	0.028182 to 0.016015	98%
8	0.087292 to 0.066062	96%	0.049304 to 0.036687	98%
9	0.033882 to 0.047635	97%	0.008944 to 0.019836	98%
10	0.040300 to 0.027556	97%	0.007862 to 0.004597	98%

Initial Convergence:

- Both models start with a high initial loss (~2.3).
- Tanh converges faster with a lower loss by the end of the first epoch compared to Sigmoid.

Test Accuracy:

- Tanh reaches 96% accuracy by the end of the first epoch, while Sigmoid reaches this accuracy by the fifth epoch.
- Both models end up with 98% accuracy by the tenth epoch, but Tanh reaches this level consistently earlier.

Loss Reduction:

- Tanh consistently shows lower loss values across epochs compared to Sigmoid.
- Tanh model's loss tends to be more stable and drops more consistently across epochs.

Using the Tanh activation function results in faster convergence and better performance in terms of loss reduction and test accuracy compared to the Sigmoid activation function. The Tanh activation function generally provides better and more consistent results for this specific model and dataset combination.

5. **Effect of the optimizer learning rate.**
    a. Trained the Model B on Adam optimizer with a learning rate of 0.1
    Code

```
modelB_lr_0_1 = ModelB().to(device)
optimizerB_lr_0_1 = optim.Adam(modelB_lr_0_1.parameters(), lr=0.1)

for epoch in range(1, 11):
    train(modelB_lr_0_1, device, train_loader, optimizerB_lr_0_1, criterion, epoch)
    test(modelB_lr_0_1, device, test_loader, criterion)
```

Output

```
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.309673
Train Epoch: 1 [6400/60000 (11%)]       Loss: 2.340111
Train Epoch: 1 [12800/60000 (21%)]      Loss: 2.309852
Train Epoch: 1 [19200/60000 (32%)]      Loss: 2.324872
Train Epoch: 1 [25600/60000 (43%)]      Loss: 2.293643
Train Epoch: 1 [32000/60000 (53%)]      Loss: 2.304275
Train Epoch: 1 [38400/60000 (64%)]      Loss: 2.312397
Train Epoch: 1 [44800/60000 (75%)]      Loss: 2.307401
Train Epoch: 1 [51200/60000 (85%)]      Loss: 2.296092
Train Epoch: 1 [57600/60000 (96%)]      Loss: 2.293898

Test set: Average loss: 0.0023, Accuracy: 1010/10000 (10%)

Train Epoch: 2 [0/60000 (0%)]   Loss: 2.314941
Train Epoch: 2 [6400/60000 (11%)]       Loss: 2.282449
Train Epoch: 2 [12800/60000 (21%)]      Loss: 2.282606
Train Epoch: 2 [19200/60000 (32%)]      Loss: 2.305658
Train Epoch: 2 [25600/60000 (43%)]      Loss: 2.311890
Train Epoch: 2 [32000/60000 (53%)]      Loss: 2.325495
Train Epoch: 2 [38400/60000 (64%)]      Loss: 2.319479
Train Epoch: 2 [44800/60000 (75%)]      Loss: 2.324875
Train Epoch: 2 [51200/60000 (85%)]      Loss: 2.353167
Train Epoch: 2 [57600/60000 (96%)]      Loss: 2.302382

Test set: Average loss: 0.0023, Accuracy: 1135/10000 (11%)

Train Epoch: 3 [0/60000 (0%)]   Loss: 2.303576
Train Epoch: 3 [6400/60000 (11%)]       Loss: 2.307500
Train Epoch: 3 [12800/60000 (21%)]      Loss: 2.291754
Train Epoch: 3 [19200/60000 (32%)]      Loss: 2.305777
Train Epoch: 3 [25600/60000 (43%)]      Loss: 2.312833
Train Epoch: 3 [32000/60000 (53%)]      Loss: 2.293413
Train Epoch: 3 [38400/60000 (64%)]      Loss: 2.286487
```

b. Trained the Model B on Adam optimizer with a learning rate of 0.01

Code

```
modelB_lr_0_01 = ModelB().to(device)
optimizerB_lr_0_01 = optim.Adam(modelB_lr_0_01.parameters(), lr=0.01)

for epoch in range(1, 11):
    train(modelB_lr_0_01, device, train_loader, optimizerB_lr_0_01, criterion, epoch)
    test(modelB_lr_0_01, device, test_loader, criterion)
```

Output

```
Test set: Average loss: 0.0001, Accuracy: 9776/10000 (98%)

Train Epoch: 3 [0/60000 (0%)]   Loss: 0.152649
Train Epoch: 3 [6400/60000 (11%)]       Loss: 0.009970
Train Epoch: 3 [12800/60000 (21%)]      Loss: 0.023960
Train Epoch: 3 [19200/60000 (32%)]      Loss: 0.168762
Train Epoch: 3 [25600/60000 (43%)]      Loss: 0.038707
Train Epoch: 3 [32000/60000 (53%)]      Loss: 0.051820
Train Epoch: 3 [38400/60000 (64%)]      Loss: 0.064354
Train Epoch: 3 [44800/60000 (75%)]      Loss: 0.016617
Train Epoch: 3 [51200/60000 (85%)]      Loss: 0.029943
Train Epoch: 3 [57600/60000 (96%)]      Loss: 0.013641

Test set: Average loss: 0.0000, Accuracy: 9847/10000 (98%)

Train Epoch: 4 [0/60000 (0%)]   Loss: 0.026389
Train Epoch: 4 [6400/60000 (11%)]       Loss: 0.063817
Train Epoch: 4 [12800/60000 (21%)]      Loss: 0.007593
Train Epoch: 4 [19200/60000 (32%)]      Loss: 0.081765
Train Epoch: 4 [25600/60000 (43%)]      Loss: 0.006554
Train Epoch: 4 [32000/60000 (53%)]      Loss: 0.001759
Train Epoch: 4 [38400/60000 (64%)]      Loss: 0.091514
Train Epoch: 4 [44800/60000 (75%)]      Loss: 0.009879
Train Epoch: 4 [51200/60000 (85%)]      Loss: 0.170409
Train Epoch: 4 [57600/60000 (96%)]      Loss: 0.000633

Test set: Average loss: 0.0001, Accuracy: 9818/10000 (98%)

Train Epoch: 5 [0/60000 (0%)]   Loss: 0.070052
Train Epoch: 5 [6400/60000 (11%)]       Loss: 0.140680
Train Epoch: 5 [12800/60000 (21%)]      Loss: 0.067581
Train Epoch: 5 [19200/60000 (32%)]      Loss: 0.011321
Train Epoch: 5 [25600/60000 (43%)]      Loss: 0.006023
```

c. Trained the Model B on Adam optimizer with a learning rate of 0.001
✓ Completed in part 1.
d. Observe and discuss the effect of learning rate on model performance

Model B trained on Adam optimizer with learning rate of 0.1

1. **Training Loss Progression:**
    - Initial loss starts high around 2.3096, typical for a neural network starting its training process.
    - Loss fluctuates slightly during training, indicating instability.
    - Loss values hover around the same range throughout epochs, showing no significant reduction.
2. **Test Performance:**
    - The test accuracy remains very low, starting at 10% and slightly increasing to 11% by the 10th epoch.
    - The average loss on the test set stays around 0.0023.
3. **General Observation:**
    - The high learning rate (0.1) seems too aggressive, preventing the model from converging.
    - The model does not show signs of effective learning or improvement over epochs.

Model B trained on Adam optimizer with learning rate of 0.01

1. **Training Loss Progression:**
    - Initial loss decreases rapidly from 2.3201 to lower values within the first epoch, indicating the model quickly learning the initial patterns.
    - Loss values significantly drop after the first epoch, reaching very low values in subsequent epochs.
2. **Test Performance:**
    - The test accuracy drastically improves, starting at 98% after the first epoch and maintaining high accuracy around 98% throughout the training process.
    - The average loss on the test set remains very low (around 0.0001).
3. **General Observation:**
    - The learning rate of 0.01 allows the model to converge effectively.
    - Model performance is robust and consistent, demonstrating good generalization on the test set.

Model B trained on Adam optimizer with learning rate of 0.001

1. **Training Loss Progression:**
    - Initial loss starts high and decreases steadily.
    - The loss reduction is more gradual compared to the learning rate of 0.01 but consistent.
2. **Test Performance:**
    - The test accuracy improves significantly, reaching around 98% after several epochs.
    - The average loss on the test set remains low (around 0.0001).

3. **General Observation:**
    - The learning rate of 0.001 also enables effective training, though the convergence is slower compared to 0.01.
    - Model performance remains robust, demonstrating good generalization.

Summary

- **High Learning Rate (0.1):** The model fails to converge, showing instability and poor generalization, with minimal improvement in accuracy.
- **Moderate Learning Rate (0.01):** The model converges quickly and performs very well, maintaining high accuracy and low loss.
- **Low Learning Rate (0.001):** The model converges more slowly but still achieves high accuracy and low loss, similar to the moderate learning rate but with a longer training duration.

Overall, the learning rate of 0.01 appears to be the most effective for training Model B.