Computer Intelligence Lab - CS354N

Weekly Assignment Report

Training 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

from torch.optim.lr_scheduler import StepLR


# Data augmentation and normalization

transform_train = transforms.Compose([

    transforms.RandomCrop(32, padding=4),

    transforms.RandomHorizontalFlip(),

    transforms.RandomRotation(15),

    transforms.ToTensor(),

    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

])


transform_test = transforms.Compose([

    transforms.ToTensor(),
```

```
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Load CIFAR-10 dataset
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
download=True, transform=transform_train)
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform_test)

trainloader = DataLoader(trainset, batch_size=256, shuffle=True, num_workers=2)
testloader = DataLoader(testset, batch_size=256, shuffle=False, num_workers=2)

# CNN model with normalization, pooling and convolution layers
class CNN_Model(nn.Module):
    def __init__(self):
        super(CNN_Model, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.conv4 = nn.Conv2d(256, 512, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(512 * 2 * 2, 1024)
        self.fc2 = nn.Linear(1024, 10)
        self.pool = nn.MaxPool2d(2, 2)
```

```
        self.dropout = nn.Dropout(0.5)
        self.batch_norm1 = nn.BatchNorm2d(64)
        self.batch_norm2 = nn.BatchNorm2d(128)
        self.batch_norm3 = nn.BatchNorm2d(256)
        self.batch_norm4 = nn.BatchNorm2d(512)

    def forward(self, x):
        x = self.pool(torch.relu(self.batch_norm1(self.conv1(x))))
        x = self.pool(torch.relu(self.batch_norm2(self.conv2(x))))
        x = self.pool(torch.relu(self.batch_norm3(self.conv3(x))))
        x = self.pool(torch.relu(self.batch_norm4(self.conv4(x))))
        x = x.view(-1, 512 * 2 * 2)
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

# Initialize the model, loss function, optimizer, and learning rate scheduler
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = CNN_Model().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
scheduler = StepLR(optimizer, step_size=30, gamma=0.1)
```

```
# Training loop
num_epochs = 30
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for inputs, labels in trainloader:
        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)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    scheduler.step()
    train_accuracy = 100 * correct / total
```

```
    print(f'Epoch {epoch+1}/{num_epochs}, Loss:
{running_loss/len(trainloader):.4f}, Accuracy: {train_accuracy:.2f}%')


# Testing loop
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in testloader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()


test_accuracy = 100 * correct / total
print(f'Test Accuracy: {test_accuracy:.2f}%')
```

Output:

Epoch 1/30, Loss: 1.7033, Accuracy: 37.45%

Epoch 2/30, Loss: 1.2892, Accuracy: 53.03%

Epoch 3/30, Loss: 1.1124, Accuracy: 60.24%

Epoch 4/30, Loss: 0.9813, Accuracy: 65.28%

Epoch 5/30, Loss: 0.9020, Accuracy: 68.36%

Epoch 6/30, Loss: 0.8505, Accuracy: 70.40%

Epoch 7/30, Loss: 0.7947, Accuracy: 72.47%

Epoch 8/30, Loss: 0.7466, Accuracy: 74.28%

Epoch 9/30, Loss: 0.7229, Accuracy: 74.92%

Epoch 10/30, Loss: 0.6923, Accuracy: 76.23%

Epoch 11/30, Loss: 0.6657, Accuracy: 77.07%

Epoch 12/30, Loss: 0.6430, Accuracy: 77.82%

Epoch 13/30, Loss: 0.6194, Accuracy: 78.92%

Epoch 14/30, Loss: 0.5912, Accuracy: 79.73%

Epoch 15/30, Loss: 0.5766, Accuracy: 80.24%

Epoch 16/30, Loss: 0.5542, Accuracy: 80.83%

Epoch 17/30, Loss: 0.5422, Accuracy: 81.40%

Epoch 18/30, Loss: 0.5275, Accuracy: 81.86%

Epoch 19/30, Loss: 0.5108, Accuracy: 82.41%

Epoch 20/30, Loss: 0.4951, Accuracy: 82.89%

Epoch 21/30, Loss: 0.4828, Accuracy: 83.44%

Epoch 22/30, Loss: 0.4741, Accuracy: 83.75%

Epoch 23/30, Loss: 0.4590, Accuracy: 84.10%

Epoch 24/30, Loss: 0.4429, Accuracy: 84.82%

Epoch 25/30, Loss: 0.4362, Accuracy: 85.04%

Epoch 26/30, Loss: 0.4242, Accuracy: 85.51%

Epoch 27/30, Loss: 0.4175, Accuracy: 85.54%

Epoch 28/30, Loss: 0.4048, Accuracy: 86.13%

Epoch 29/30, Loss: 0.3925, Accuracy: 86.54%

Epoch 30/30, Loss: 0.3923, Accuracy: 86.40%

Test Accuracy: 86.29%