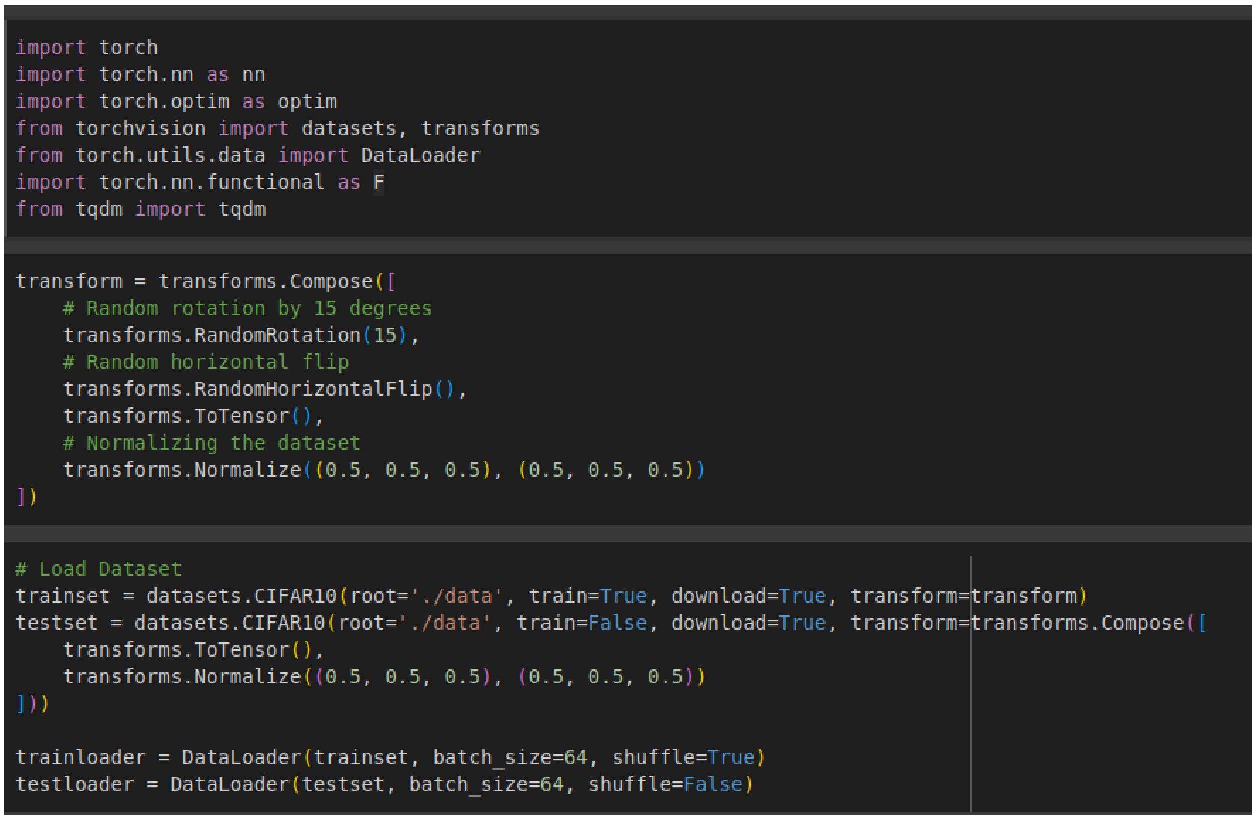
CSE354N - ASSIGNMENT 11

220002018 - Arnav Jain

Code Snippet:



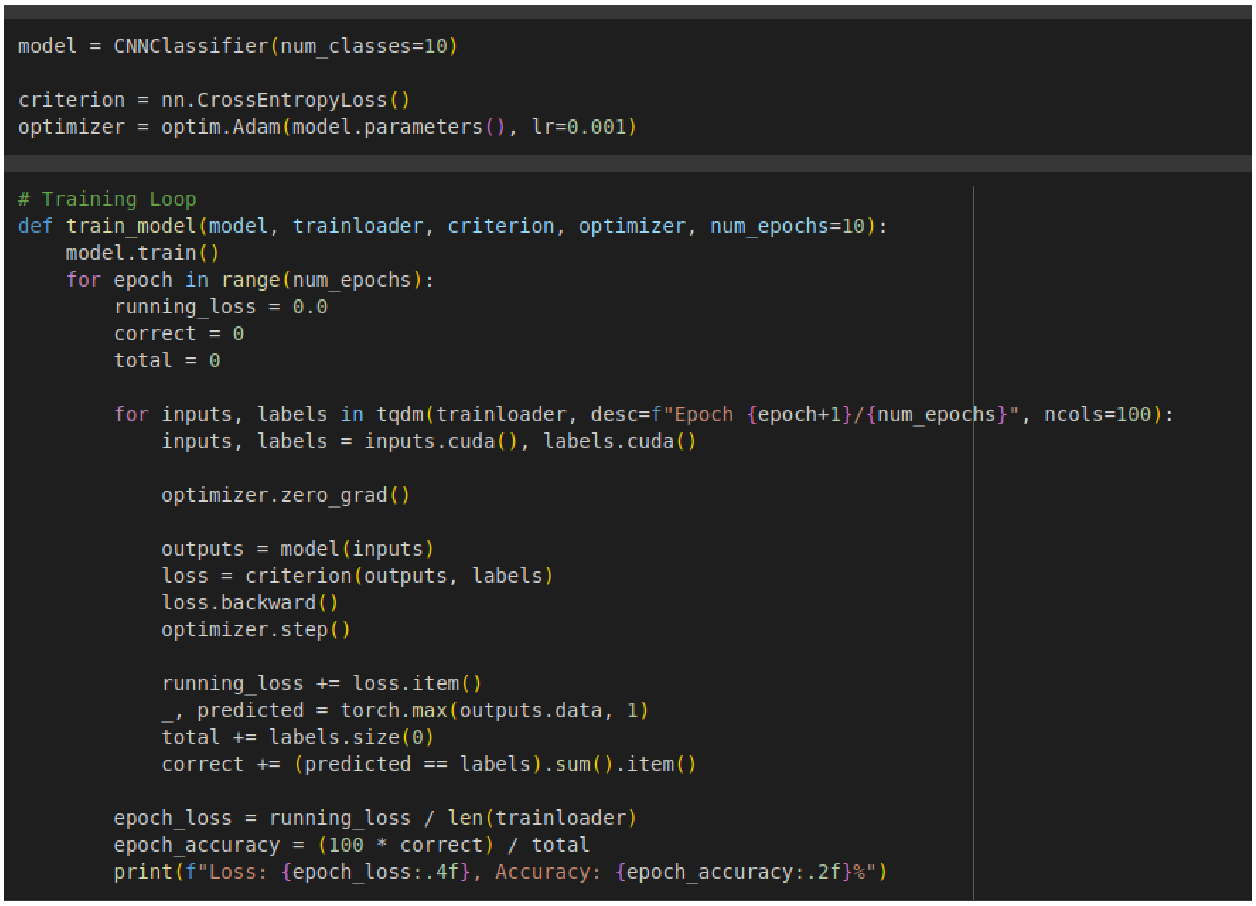
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torch.nn.functional as F
from tqdm import tqdm
```

```
transform = transforms.Compose([
    # Random rotation by 15 degrees
    transforms.RandomRotation(15),
    # Random horizontal flip
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    # Normalizing the dataset
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

```
# Load Dataset
trainset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
testset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
]))

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

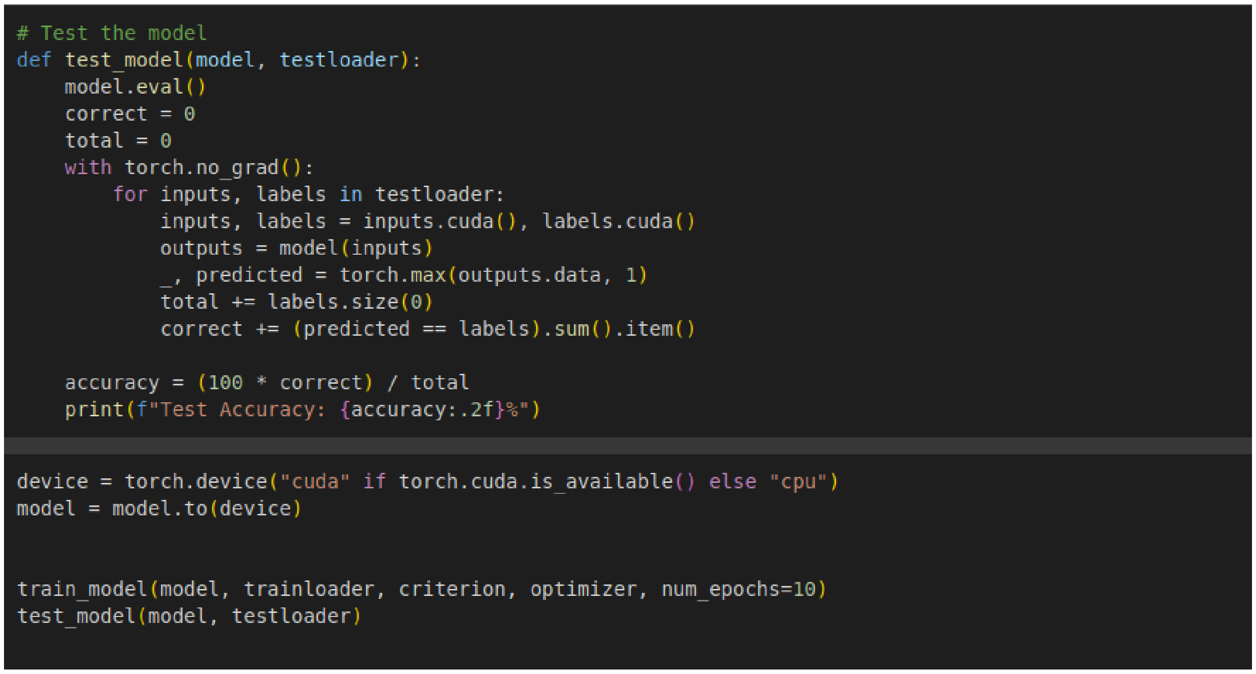
```python
class CNNClassifier(nn.Module):
    def __init__(self, num_classes=10):
        super(CNNClassifier, self).__init__()

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.pool = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(128)

        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(256)

        self.conv4 = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
        self.bn4 = nn.BatchNorm2d(512)

        self.fc1 = nn.Linear(2048, 1024)
        self.fc2 = nn.Linear(1024, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        x = self.pool(F.relu(self.bn4(self.conv4(x))))

        x = torch.flatten(x, 1)

        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```python
model = CNNClassifier(num_classes=10)

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



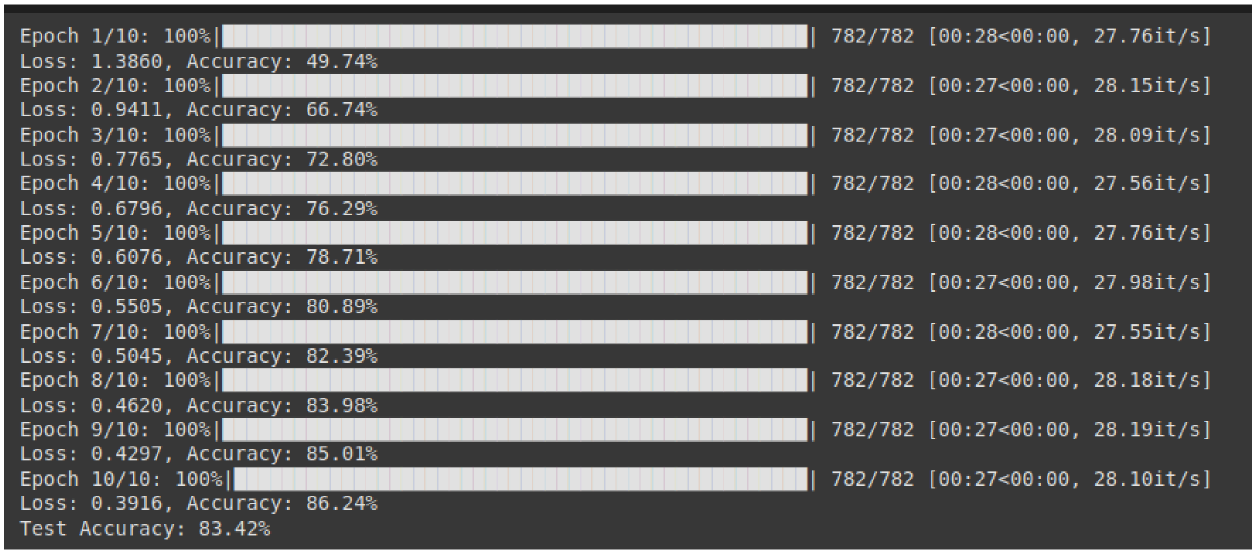
```python
model = CNNClassifier(num_classes=10)

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

```python
# Training Loop
def train_model(model, trainloader, criterion, optimizer, num_epochs=10):
    model.train()
    for epoch in range(num_epochs):
        running_loss = 0.0
        correct = 0
        total = 0

        for inputs, labels in tqdm(trainloader, desc=f"Epoch {epoch+1}/{num_epochs}", ncols=100):
            inputs, labels = inputs.cuda(), labels.cuda()

            optimizer.zero_grad()

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

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

        epoch_loss = running_loss / len(trainloader)
        epoch_accuracy = (100 * correct) / total
        print(f"Loss: {epoch_loss:.4f}, Accuracy: {epoch_accuracy:.2f}%")
```



```python
# Test the model
def test_model(model, testloader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in testloader:
            inputs, labels = inputs.cuda(), labels.cuda()
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

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

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


train_model(model, trainloader, criterion, optimizer, num_epochs=10)
test_model(model, testloader)
```



```
Epoch 1/10: 100%|██████████████████████████████████████████████████| 782/782 [00:28<00:00, 27.76it/s]
Loss: 1.3860, Accuracy: 49.74%
Epoch 2/10: 100%|██████████████████████████████████████████████████| 782/782 [00:27<00:00, 28.15it/s]
Loss: 0.9411, Accuracy: 66.74%
Epoch 3/10: 100%|██████████████████████████████████████████████████| 782/782 [00:27<00:00, 28.09it/s]
Loss: 0.7765, Accuracy: 72.80%
Epoch 4/10: 100%|██████████████████████████████████████████████████| 782/782 [00:28<00:00, 27.56it/s]
Loss: 0.6796, Accuracy: 76.29%
Epoch 5/10: 100%|██████████████████████████████████████████████████| 782/782 [00:28<00:00, 27.76it/s]
Loss: 0.6076, Accuracy: 78.71%
Epoch 6/10: 100%|██████████████████████████████████████████████████| 782/782 [00:27<00:00, 27.98it/s]
Loss: 0.5505, Accuracy: 80.89%
Epoch 7/10: 100%|██████████████████████████████████████████████████| 782/782 [00:28<00:00, 27.55it/s]
Loss: 0.5045, Accuracy: 82.39%
Epoch 8/10: 100%|██████████████████████████████████████████████████| 782/782 [00:27<00:00, 28.18it/s]
Loss: 0.4620, Accuracy: 83.98%
Epoch 9/10: 100%|██████████████████████████████████████████████████| 782/782 [00:27<00:00, 28.19it/s]
Loss: 0.4297, Accuracy: 85.01%
Epoch 10/10: 100%|█████████████████████████████████████████████████| 782/782 [00:27<00:00, 28.10it/s]
Loss: 0.3916, Accuracy: 86.24%
Test Accuracy: 83.42%
```

GitHub

https://github.com/arnavjain2710/Computational-Intelligence-Lab-CS354N/tree/main/LAB%2011