## Submission format

Submit a PDF-version report for all problems, including source codes and print screen results. You can use any text/code editor; Google Colab is recommended.

Review lecture slide 7 and 8 and finish the following problem.

## Problem. Build LeNet for colorful image classification

LeNet is one of the earliest convolutional neural networks (CNNs), developed by Yann LeCun in the late 1980s and early 1990s. It was designed primarily for hand-written digit recognition, such as the MNIST dataset. You can refer to " LeNet-5" for more information about its CNN structure and implementation.

In this problem, you are asked to train and test a LeNet-5 for entire CIFAR-10 colorful image dataset. The CIFAR-10 dataset contains 60,000 32×32 color images in 10 different classes; please refer to its introduction for more information.

You need to implement a LeNet-5 to accommodate the CIFAR-10 dataset and fulfill the following requirements:

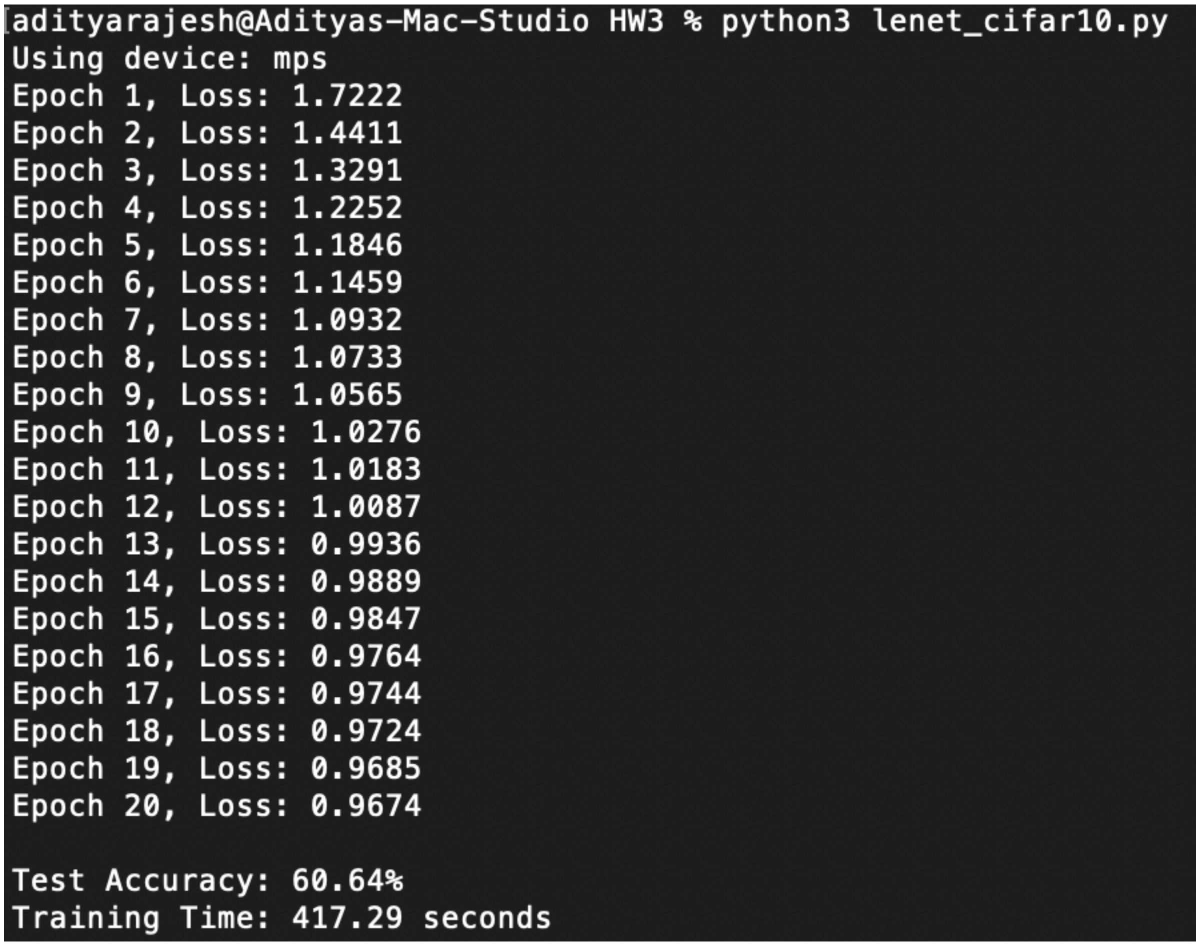
- Keep the basic network structure of LeNet-5 but you can change the hyperpa-rameters.
- You need to submit three results: 1) network without dropout/batch normal-ization, 2) network with one additional dropout layer and 3) network with one additional batch normalization. The test accuracy should all achieve above 50%.
- Submission should include your source codes and screen snapshot of your train and test accuracy, plus the training time.

Hints: you can use PyTorch torch.nn to find the packed Batch Normalization and Dropout layer if you would like to use. Also, you can load the CIFAR-10 dataset by torchvision.datasets.CIFAR10.

**Network without Dropout/Batch:**

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import time
import platform
import os
import sys
from contextlib import contextmanager

# Transform for CIFAR-10 (normalize RGB channels)
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

@contextmanager
def suppress_stdout():
    with open(os.devnull, 'w') as devnull:
        old_stdout = sys.stdout
        sys.stdout = devnull
        try:
            yield
        finally:
            sys.stdout = old_stdout

# CIFAR-10 dataset loading
with suppress_stdout():
    trainset = torchvision.datasets.CIFAR10(root='./data',
train=True, download=True, transform=transform)
    testset = torchvision.datasets.CIFAR10(root='./data',
train=False, download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset,
batch_size=64, shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(testset,
batch_size=1000, shuffle=False, num_workers=2)

# LeNet-5 model (no dropout / no batchnorm)
class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, kernel_size=5)
        self.pool = nn.AvgPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x)

# Train and evaluate function
def train_and_evaluate(model, trainloader, testloader, device):
    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.StepLR(optimizer,
step_size=3, gamma=0.5)

    start_time = time.time()
    model.train()
    for epoch in range(20):
        running_loss = 0.0
        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()
        scheduler.step()
        print(f"Epoch {epoch+1}, Loss: {running_loss /
len(trainloader):.4f}")
    train_time = time.time() - start_time

    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.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f"\nTest Accuracy: {accuracy:.2f}%")
    print(f"Training Time: {train_time:.2f} seconds")
    return accuracy, train_time

# Run
if __name__ == "__main__":
    if torch.backends.mps.is_available() and
platform.processor() == "arm":
        device = torch.device("mps")
    elif torch.cuda.is_available():
        device = torch.device("cuda")
    else:
        device = torch.device("cpu")

    print(f"Using device: {device}")
    model = LeNet5()
    train_and_evaluate(model, trainloader, testloader, device)
```

**Results:**



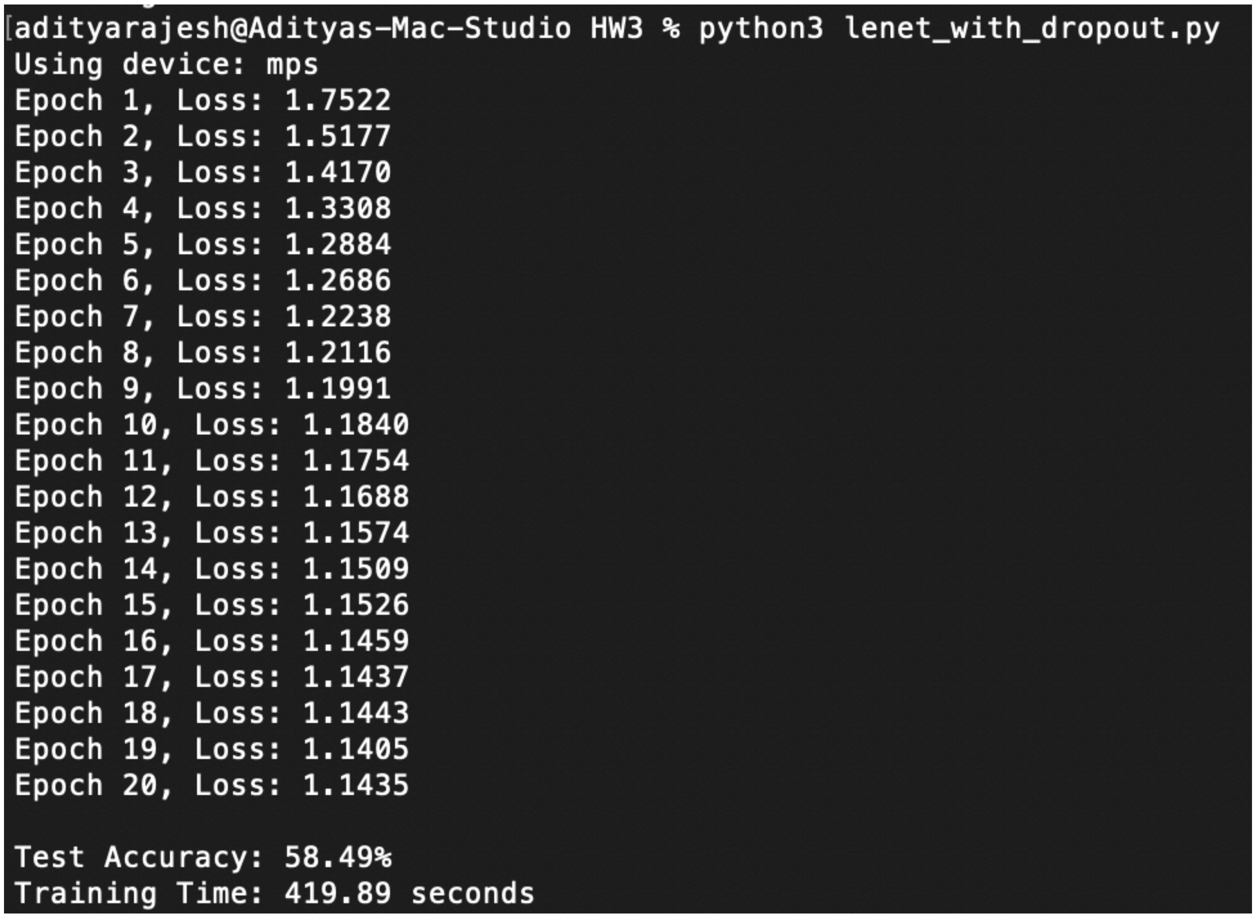
```
adityarajesh@Adityas-Mac-Studio HW3 % python3 lenet_cifar10.py
Using device: mps
Epoch 1, Loss: 1.7222
Epoch 2, Loss: 1.4411
Epoch 3, Loss: 1.3291
Epoch 4, Loss: 1.2252
Epoch 5, Loss: 1.1846
Epoch 6, Loss: 1.1459
Epoch 7, Loss: 1.0932
Epoch 8, Loss: 1.0733
Epoch 9, Loss: 1.0565
Epoch 10, Loss: 1.0276
Epoch 11, Loss: 1.0183
Epoch 12, Loss: 1.0087
Epoch 13, Loss: 0.9936
Epoch 14, Loss: 0.9889
Epoch 15, Loss: 0.9847
Epoch 16, Loss: 0.9764
Epoch 17, Loss: 0.9744
Epoch 18, Loss: 0.9724
Epoch 19, Loss: 0.9685
Epoch 20, Loss: 0.9674

Test Accuracy: 60.64%
Training Time: 417.29 seconds
```

**Network with exactly one additional dropout layer:**

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import time
import platform
import os
import sys
from contextlib import contextmanager

# Transform for CIFAR-10 (normalize RGB channels)
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

@contextmanager
```

```
def suppress_stdout():
    with open(os.devnull, 'w') as devnull:
        old_stdout = sys.stdout
        sys.stdout = devnull
        try:
            yield
        finally:
            sys.stdout = old_stdout

# CIFAR-10 dataset loading
with suppress_stdout():
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
download=True, transform=transform)
    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(testset, batch_size=1000,
shuffle=False, num_workers=2)

# LeNet-5 with one Dropout layer
class LeNet5Dropout(nn.Module):
    def __init__(self):
        super(LeNet5Dropout, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, kernel_size=5)
        self.pool = nn.AvgPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.dropout = nn.Dropout(p=0.3)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = self.dropout(F.relu(self.fc1(x)))
        x = F.relu(self.fc2(x))
        return self.fc3(x)

# Train and evaluate function
def train_and_evaluate(model, trainloader, testloader, device):
    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.5)

    start_time = time.time()
    model.train()
    for epoch in range(20):
        running_loss = 0.0
        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()
        scheduler.step()
        print(f"Epoch {epoch+1}, Loss: {running_loss /
len(trainloader):.4f}")
    train_time = time.time() - start_time

    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.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f"\nTest Accuracy: {accuracy:.2f}%")
    print(f"Training Time: {train_time:.2f} seconds")
    return accuracy, train_time

# Run
if __name__ == "__main__":
    if torch.backends.mps.is_available() and platform.processor() ==
"arm":
        device = torch.device("mps")
    elif torch.cuda.is_available():
        device = torch.device("cuda")
    else:
        device = torch.device("cpu")

    print(f"Using device: {device}")
    model = LeNet5Dropout()
    train_and_evaluate(model, trainloader, testloader, device)
```

**Results:**

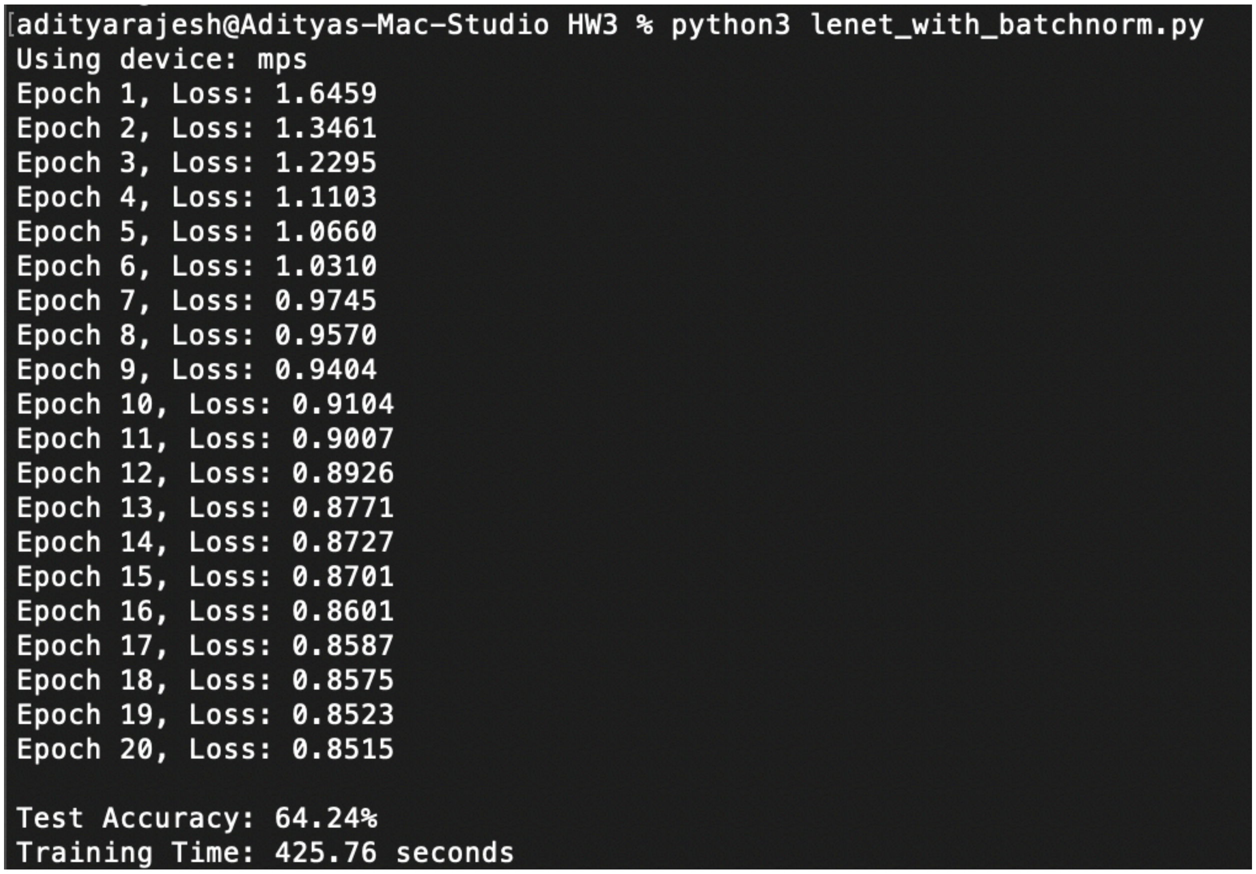

```
[adityarajesh@Adityas-Mac-Studio HW3 % python3 lenet_with_dropout.py
Using device: mps
Epoch 1, Loss: 1.7522
Epoch 2, Loss: 1.5177
Epoch 3, Loss: 1.4170
Epoch 4, Loss: 1.3308
Epoch 5, Loss: 1.2884
Epoch 6, Loss: 1.2686
Epoch 7, Loss: 1.2238
Epoch 8, Loss: 1.2116
Epoch 9, Loss: 1.1991
Epoch 10, Loss: 1.1840
Epoch 11, Loss: 1.1754
Epoch 12, Loss: 1.1688
Epoch 13, Loss: 1.1574
Epoch 14, Loss: 1.1509
Epoch 15, Loss: 1.1526
Epoch 16, Loss: 1.1459
Epoch 17, Loss: 1.1437
Epoch 18, Loss: 1.1443
Epoch 19, Loss: 1.1405
Epoch 20, Loss: 1.1435

Test Accuracy: 58.49%
Training Time: 419.89 seconds
```

**Network with exactly one additional batch normalization:**

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import time
import platform
import os
import sys
from contextlib import contextmanager

# Transform for CIFAR-10 (normalize RGB channels)
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

```
@contextmanager
def suppress_stdout():
    with open(os.devnull, 'w') as devnull:
        old_stdout = sys.stdout
        sys.stdout = devnull
        try:
            yield
        finally:
            sys.stdout = old_stdout

# CIFAR-10 dataset loading
with suppress_stdout():
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
download=True, transform=transform)
    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(testset, batch_size=1000,
shuffle=False, num_workers=2)

# LeNet-5 with one BatchNorm layer
class LeNet5BatchNorm(nn.Module):
    def __init__(self):
        super(LeNet5BatchNorm, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, kernel_size=5)
        self.bn1 = nn.BatchNorm2d(6)
        self.pool = nn.AvgPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x)

# Train and evaluate function
def train_and_evaluate(model, trainloader, testloader, device):
    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.5)

    start_time = time.time()
    model.train()
    for epoch in range(20):
        running_loss = 0.0
        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()
        scheduler.step()
        print(f"Epoch {epoch+1}, Loss: {running_loss /
len(trainloader):.4f}")
    train_time = time.time() - start_time

    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.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f"\nTest Accuracy: {accuracy:.2f}%")
    print(f"Training Time: {train_time:.2f} seconds")
    return accuracy, train_time

# Run
if __name__ == "__main__":
    if torch.backends.mps.is_available() and platform.processor() ==
"arm":
        device = torch.device("mps")
    elif torch.cuda.is_available():
        device = torch.device("cuda")
    else:
        device = torch.device("cpu")

    print(f"Using device: {device}")
    model = LeNet5BatchNorm()
    train_and_evaluate(model, trainloader, testloader, device)
```

**Results:**



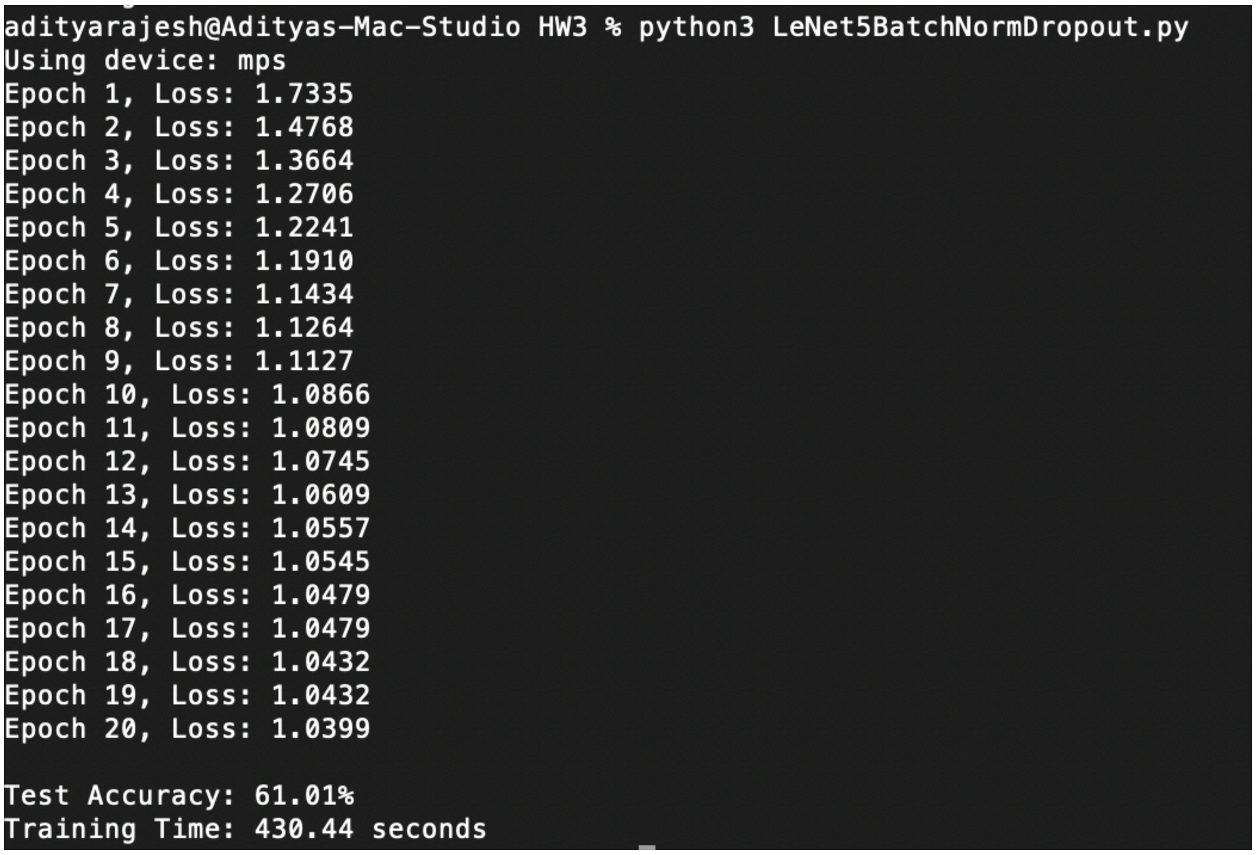
```
[adityarajesh@Adityas-Mac-Studio HW3 % python3 lenet_with_batchnorm.py
Using device: mps
Epoch 1, Loss: 1.6459
Epoch 2, Loss: 1.3461
Epoch 3, Loss: 1.2295
Epoch 4, Loss: 1.1103
Epoch 5, Loss: 1.0660
Epoch 6, Loss: 1.0310
Epoch 7, Loss: 0.9745
Epoch 8, Loss: 0.9570
Epoch 9, Loss: 0.9404
Epoch 10, Loss: 0.9104
Epoch 11, Loss: 0.9007
Epoch 12, Loss: 0.8926
Epoch 13, Loss: 0.8771
Epoch 14, Loss: 0.8727
Epoch 15, Loss: 0.8701
Epoch 16, Loss: 0.8601
Epoch 17, Loss: 0.8587
Epoch 18, Loss: 0.8575
Epoch 19, Loss: 0.8523
Epoch 20, Loss: 0.8515

Test Accuracy: 64.24%
Training Time: 425.76 seconds
```

**Network with exactly one additional dropout layer and exactly one addition batch normalization:**

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import time
import platform
import os
import sys
from contextlib import contextmanager
```

```
# Transform for CIFAR-10 (normalize RGB channels)
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

@contextmanager
def suppress_stdout():
    with open(os.devnull, 'w') as devnull:
        old_stdout = sys.stdout
        sys.stdout = devnull
        try:
            yield
        finally:
            sys.stdout = old_stdout

# CIFAR-10 dataset loading
with suppress_stdout():
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
download=True, transform=transform)
    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(testset, batch_size=1000,
shuffle=False, num_workers=2)

# LeNet-5 with BOTH BatchNorm and Dropout
class LeNet5BatchNormDropout(nn.Module):
    def __init__(self):
        super(LeNet5BatchNormDropout, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, kernel_size=5)
        self.bn1 = nn.BatchNorm2d(6)
        self.pool = nn.AvgPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.dropout = nn.Dropout(p=0.3)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = self.dropout(F.relu(self.fc1(x)))
        x = F.relu(self.fc2(x))
        return self.fc3(x)

# Train and evaluate function
def train_and_evaluate(model, trainloader, testloader, device):
    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3,
gamma=0.5)

    start_time = time.time()
    model.train()
    for epoch in range(20):
        running_loss = 0.0
        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()
        scheduler.step()
        print(f"Epoch {epoch+1}, Loss: {running_loss /
len(trainloader):.4f}")
    train_time = time.time() - start_time

    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.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f"\nTest Accuracy: {accuracy:.2f}%")
    print(f"Training Time: {train_time:.2f} seconds")
    return accuracy, train_time

# Run
if __name__ == "__main__":
    if torch.backends.mps.is_available() and platform.processor() ==
"arm":
        device = torch.device("mps")
    elif torch.cuda.is_available():
        device = torch.device("cuda")
    else:
        device = torch.device("cpu")

    print(f"Using device: {device}")
    model = LeNet5BatchNormDropout()
    train_and_evaluate(model, trainloader, testloader, device)
```

**Results:**



```
adityarajesh@Adityas-Mac-Studio HW3 % python3 LeNet5BatchNormDropout.py
Using device: mps
Epoch 1, Loss: 1.7335
Epoch 2, Loss: 1.4768
Epoch 3, Loss: 1.3664
Epoch 4, Loss: 1.2706
Epoch 5, Loss: 1.2241
Epoch 6, Loss: 1.1910
Epoch 7, Loss: 1.1434
Epoch 8, Loss: 1.1264
Epoch 9, Loss: 1.1127
Epoch 10, Loss: 1.0866
Epoch 11, Loss: 1.0809
Epoch 12, Loss: 1.0745
Epoch 13, Loss: 1.0609
Epoch 14, Loss: 1.0557
Epoch 15, Loss: 1.0545
Epoch 16, Loss: 1.0479
Epoch 17, Loss: 1.0479
Epoch 18, Loss: 1.0432
Epoch 19, Loss: 1.0432
Epoch 20, Loss: 1.0399

Test Accuracy: 61.01%
Training Time: 430.44 seconds
```