Solution 1.1):

Co-adaptation refers to the phenomenon in deep learning where multiple components of a model become highly dependent on each other to achieve good performance, to the point where it becomes difficult to isolate the contributions of individual components. This can lead to overfitting and poor generalization when the model is applied to new data.

In deep learning, a model typically consists of many layers, each of which contains a large number of learnable parameters. These parameters are updated during training to optimize the model's performance on a training set. However, because the layers are highly interconnected, the optimization process can result in some layers becoming dependent on the output of others. This dependency can lead to a situation where the performance of the entire model is heavily dependent on the behavior of a few key components.

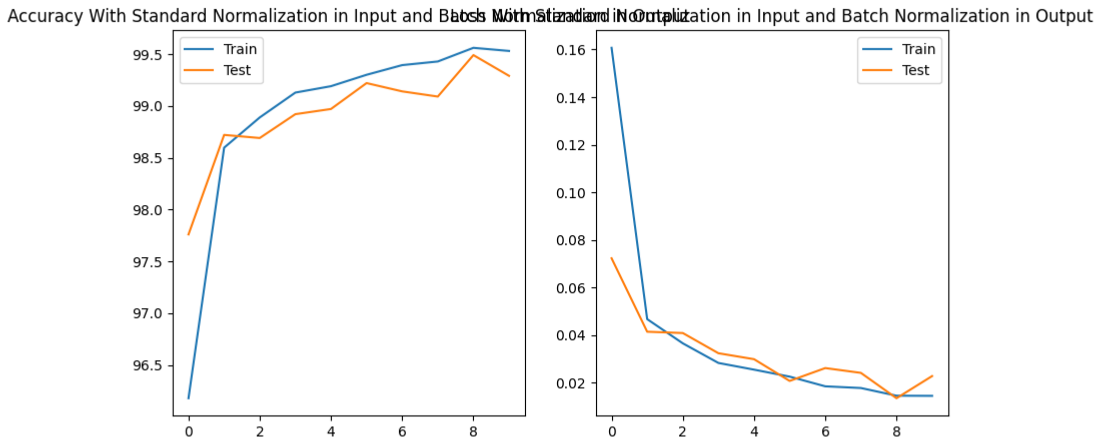
Internal covariance-shift, on the other hand, refers to a problem that can arise in machine learning algorithms when the statistical distribution of the input data changes between training and testing. This can happen, for example, if the training data is drawn from one population or environment, and the testing data is drawn from a different population or environment.

The problem with internal covariance-shift is that the model may not be able to generalize well to the testing data, because the statistical patterns that it learned from the training data no longer hold. This can lead to poor performance and low accuracy.

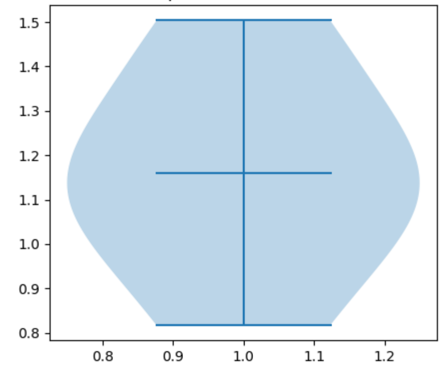
One way to mitigate the effects of internal covariance-shift is to use techniques such as domain adaptation or transfer learning, which aim to adapt the model to the new testing data by adjusting its internal representations or learning new representations from a related task. These techniques can help the model to generalize better to new environments and improve its accuracy.

In [ ]:
```python
#Solution 1.2):

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import numpy as np
import matplotlib.pyplot as plt

# Load the MNIST dataset
train_dataset = datasets.MNIST(root='./data', train=True, transform=transforms.Compose([
                                    transforms.ToTensor(),
                                    transforms.Normalize((0.131,), (0.302,))
                                ]), download=True)
test_dataset = datasets.MNIST(root='./data', train=False, transform=transforms.Compose([
                                    transforms.ToTensor(),
                                    transforms.Normalize((0.131,), (0.302,))
                                ]), download=True)

# Define the data loaders
batch_size = 64
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Define the LeNet-5 model with batch normalization for all layers
class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.bn1 = nn.BatchNorm2d(6)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.bn2 = nn.BatchNorm2d(16)
        self.fc1 = nn.Linear(16*4*4, 120)
        self.bn3 = nn.BatchNorm1d(120)
        self.fc2 = nn.Linear(120, 84)
        self.bn4 = nn.BatchNorm1d(84)
        self.fc3 = nn.Linear(84, 10)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        x = self.bn1(self.conv1(x))
        x = self.relu(x)
        x = self.maxpool(x)
        x = self.bn2(self.conv2(x))
        x = self.relu(x)
        x = self.maxpool(x)
        x = x.view(-1, 16*4*4)
        x = self.bn3(self.fc1(x))
        x = self.relu(x)
        x = self.bn4(self.fc2(x))
        x = self.relu(x)
        x = self.fc3(x)
        return x

# Initialize the model
model = LeNet5()

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Define the batch size and number of epochs
batch_size = 64
n_epochs = 10

# Create data loaders for the train and test datasets
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Train the model
train_loss_bn = []
train_acc_bn = []
test_loss_bn = []
test_acc_bn = []

for epoch in range(n_epochs):
    model.train()
    train_loss = 0.0
    train_total = 0
    train_correct = 0
    for i, (inputs, labels) in enumerate(train_loader):
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs.data, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()
        # Print training statistics
        if (i+1) % 100 == 0:
            print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'.format(epoch+1, n_epochs, i+1, len(train_loader), loss.item()))

    train_loss /= len(train_loader.dataset)
    train_acc = 100 * train_correct / train_total
    train_loss_bn.append(train_loss)
    train_acc_bn.append(train_acc)

    model.eval()
    test_loss = 0.0
    test_total = 0
    test_correct = 0
    for i, (inputs, labels) in enumerate(test_loader):
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        test_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs.data, 1)
        test_total += labels.size(0)
        test_correct += (predicted == labels).sum().item()

    test_loss /= len(test_loader.dataset)
    test_acc = 100 * test_correct / test_total
    test_loss_bn.append(test_loss)
    test_acc_bn.append(test_acc)

    print('Epoch [{}/{}], Train Loss: {:.4f}, Train Acc: {:.2f}, Test Loss: {:.4f}, Test Acc: {:.2f}'.format(epoch+1, n_epochs, train_loss, train_acc, test_loss, test_acc))

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(train_acc_bn, label='Train')
plt.plot(test_acc_bn, label='Test')
plt.title('Accuracy With Standard Normalization in Input and Batch Normalization in Output')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(train_loss_bn, label='Train')
plt.plot(test_loss_bn, label='Test')
plt.title('Loss With Standard Normalization in Input and Batch Normalization in Output')
plt.legend()
plt.show()

bn_params = []
for name, module in model.named_modules():
    if isinstance(module, nn.BatchNorm2d) or isinstance(module, nn.BatchNorm1d):
        bn_params.append(module.weight.data.cpu().numpy())

fig, axs = plt.subplots(len(bn_params), figsize=(5, 20))
for i, p in enumerate(bn_params):
    axs[i].violinplot(dataset=p, showmeans=True)
    axs[i].set_title(f'Standard Normalization in Input and Batch Normalization in Output Layer {i+1}')
plt.show()
```

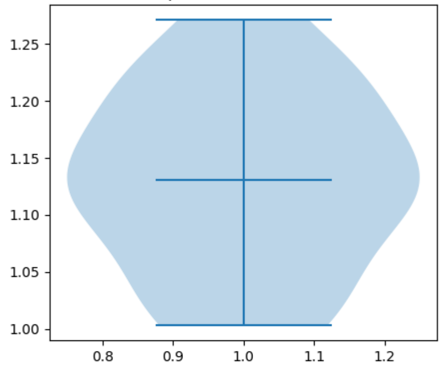
```
Epoch [1/10], Step [100/938], Loss: 0.2941
Epoch [1/10], Step [200/938], Loss: 0.0614
Epoch [1/10], Step [300/938], Loss: 0.0811
Epoch [1/10], Step [400/938], Loss: 0.2154
Epoch [1/10], Step [500/938], Loss: 0.1622
Epoch [1/10], Step [600/938], Loss: 0.0694
Epoch [1/10], Step [700/938], Loss: 0.0488
Epoch [1/10], Step [800/938], Loss: 0.2468
Epoch [1/10], Step [900/938], Loss: 0.0083
Epoch [1/10], Train Loss: 0.1607, Train Acc: 96.18, Test Loss: 0.0723, Test Acc: 97.76
Epoch [2/10], Step [100/938], Loss: 0.0553
Epoch [2/10], Step [200/938], Loss: 0.0417
Epoch [2/10], Step [300/938], Loss: 0.0132
Epoch [2/10], Step [400/938], Loss: 0.0263
Epoch [2/10], Step [500/938], Loss: 0.0233
Epoch [2/10], Step [600/938], Loss: 0.0176
Epoch [2/10], Step [700/938], Loss: 0.0196
Epoch [2/10], Step [800/938], Loss: 0.0874
Epoch [2/10], Step [900/938], Loss: 0.0115
Epoch [2/10], Train Loss: 0.0467, Train Acc: 98.60, Test Loss: 0.0414, Test Acc: 98.72
Epoch [3/10], Step [100/938], Loss: 0.0031
Epoch [3/10], Step [200/938], Loss: 0.0406
Epoch [3/10], Step [300/938], Loss: 0.0283
Epoch [3/10], Step [400/938], Loss: 0.1055
Epoch [3/10], Step [500/938], Loss: 0.0666
Epoch [3/10], Step [600/938], Loss: 0.0175
Epoch [3/10], Step [700/938], Loss: 0.0096
Epoch [3/10], Step [800/938], Loss: 0.0414
Epoch [3/10], Step [900/938], Loss: 0.0332
Epoch [3/10], Train Loss: 0.0365, Train Acc: 98.89, Test Loss: 0.0408, Test Acc: 98.69
Epoch [4/10], Step [100/938], Loss: 0.0264
Epoch [4/10], Step [200/938], Loss: 0.0016
Epoch [4/10], Step [300/938], Loss: 0.0021
Epoch [4/10], Step [400/938], Loss: 0.0271
Epoch [4/10], Step [500/938], Loss: 0.0073
Epoch [4/10], Step [600/938], Loss: 0.0045
Epoch [4/10], Step [700/938], Loss: 0.0188
Epoch [4/10], Step [800/938], Loss: 0.0101
Epoch [4/10], Step [900/938], Loss: 0.0118
Epoch [4/10], Train Loss: 0.0283, Train Acc: 99.13, Test Loss: 0.0324, Test Acc: 98.92
Epoch [5/10], Step [100/938], Loss: 0.0198
Epoch [5/10], Step [200/938], Loss: 0.0045
Epoch [5/10], Step [300/938], Loss: 0.0471
Epoch [5/10], Step [400/938], Loss: 0.0041
Epoch [5/10], Step [500/938], Loss: 0.0202
Epoch [5/10], Step [600/938], Loss: 0.0427
Epoch [5/10], Step [700/938], Loss: 0.0245
Epoch [5/10], Step [800/938], Loss: 0.0189
Epoch [5/10], Step [900/938], Loss: 0.0468
Epoch [5/10], Train Loss: 0.0255, Train Acc: 99.19, Test Loss: 0.0299, Test Acc: 98.97
Epoch [6/10], Step [100/938], Loss: 0.0024
Epoch [6/10], Step [200/938], Loss: 0.0768
Epoch [6/10], Step [300/938], Loss: 0.0051
Epoch [6/10], Step [400/938], Loss: 0.0006
Epoch [6/10], Step [500/938], Loss: 0.0019
Epoch [6/10], Step [600/938], Loss: 0.0059
Epoch [6/10], Step [700/938], Loss: 0.0028
Epoch [6/10], Step [800/938], Loss: 0.0056
Epoch [6/10], Step [900/938], Loss: 0.0037
Epoch [6/10], Train Loss: 0.0225, Train Acc: 99.30, Test Loss: 0.0207, Test Acc: 99.22
Epoch [7/10], Step [100/938], Loss: 0.0108
Epoch [7/10], Step [200/938], Loss: 0.0187
Epoch [7/10], Step [300/938], Loss: 0.0002
Epoch [7/10], Step [400/938], Loss: 0.0217
Epoch [7/10], Step [500/938], Loss: 0.0030
Epoch [7/10], Step [600/938], Loss: 0.0230
Epoch [7/10], Step [700/938], Loss: 0.0342
Epoch [7/10], Step [800/938], Loss: 0.0002
Epoch [7/10], Step [900/938], Loss: 0.0041
Epoch [7/10], Train Loss: 0.0185, Train Acc: 99.39, Test Loss: 0.0261, Test Acc: 99.14
Epoch [8/10], Step [100/938], Loss: 0.0020
Epoch [8/10], Step [200/938], Loss: 0.0065
Epoch [8/10], Step [300/938], Loss: 0.0011
Epoch [8/10], Step [400/938], Loss: 0.0233
Epoch [8/10], Step [500/938], Loss: 0.0014
Epoch [8/10], Step [600/938], Loss: 0.0024
Epoch [8/10], Step [700/938], Loss: 0.0035
Epoch [8/10], Step [800/938], Loss: 0.0128
Epoch [8/10], Step [900/938], Loss: 0.0003
Epoch [8/10], Train Loss: 0.0177, Train Acc: 99.43, Test Loss: 0.0241, Test Acc: 99.09
Epoch [9/10], Step [100/938], Loss: 0.0116
Epoch [9/10], Step [200/938], Loss: 0.0097
Epoch [9/10], Step [300/938], Loss: 0.0204
Epoch [9/10], Step [400/938], Loss: 0.0099
Epoch [9/10], Step [500/938], Loss: 0.0126
Epoch [9/10], Step [600/938], Loss: 0.0351
Epoch [9/10], Step [700/938], Loss: 0.0014
Epoch [9/10], Step [800/938], Loss: 0.0042
Epoch [9/10], Step [900/938], Loss: 0.0138
Epoch [9/10], Train Loss: 0.0145, Train Acc: 99.56, Test Loss: 0.0135, Test Acc: 99.49
Epoch [10/10], Step [100/938], Loss: 0.0005
Epoch [10/10], Step [200/938], Loss: 0.0053
Epoch [10/10], Step [300/938], Loss: 0.0290
Epoch [10/10], Step [400/938], Loss: 0.0130
Epoch [10/10], Step [500/938], Loss: 0.0003
Epoch [10/10], Step [600/938], Loss: 0.0009
Epoch [10/10], Step [700/938], Loss: 0.0013
Epoch [10/10], Step [800/938], Loss: 0.0015
Epoch [10/10], Step [900/938], Loss: 0.0050
Epoch [10/10], Train Loss: 0.0145, Train Acc: 99.53, Test Loss: 0.0228, Test Acc: 99.29
```


Accuracy With Standard Normalization in Input and Batch Normalization in Output
Loss With Standard Normalization in Input and Batch Normalization in Output
Train
Test
Train
Test

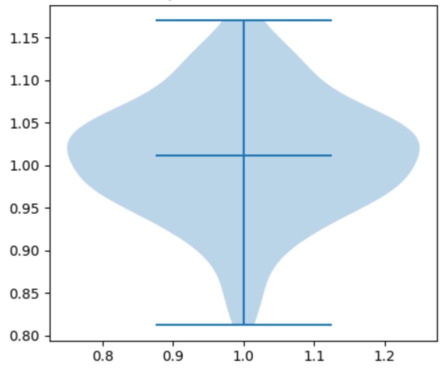
Standard Normalization in Input and Batch Normalization in Output Layer 1



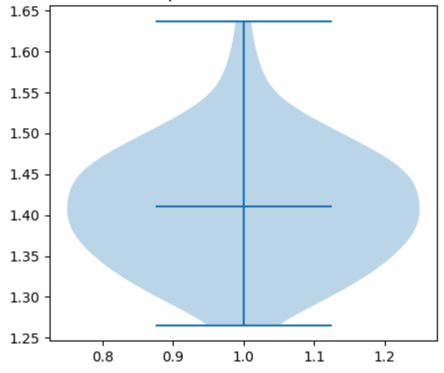
Standard Normalization in Input and Batch Normalization in Output Layer 2



Standard Normalization in Input and Batch Normalization in Output Layer 3

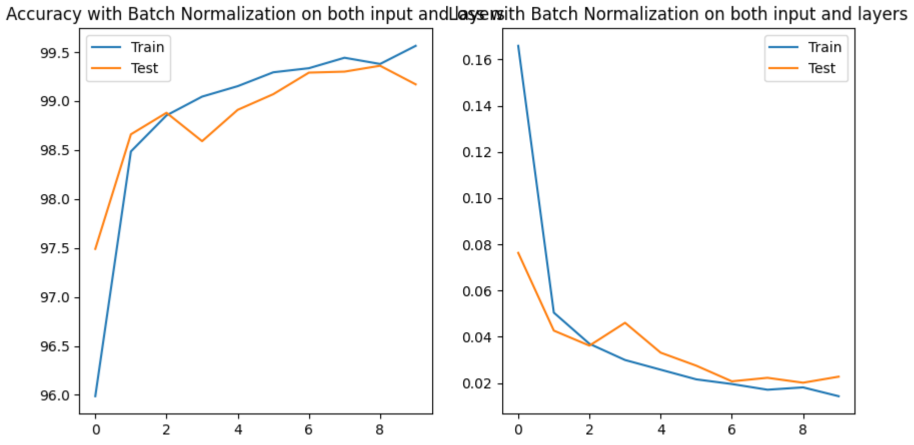


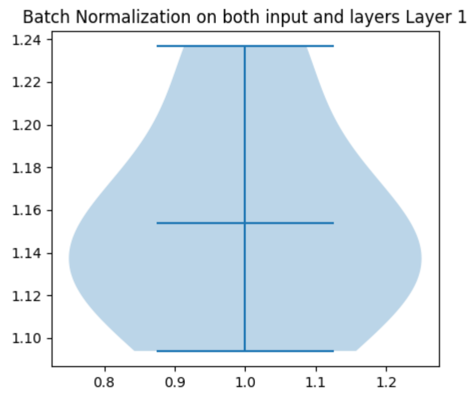
Standard Normalization in Input and Batch Normalization in Output Layer 4

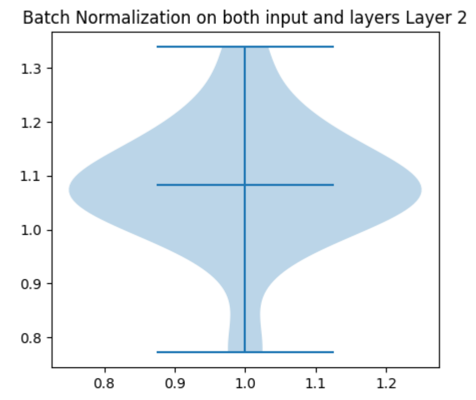


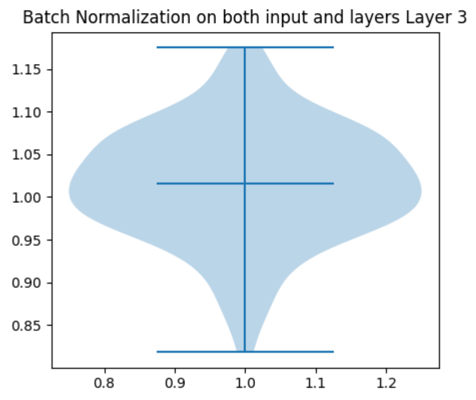
In [ ]:
```python
#Solution 1.3):

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import numpy as np
import matplotlib.pyplot as plt

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

# Define the LeNet-5 model with batch normalization for all layers
class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.bn1 = nn.BatchNorm2d(6)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.bn2 = nn.BatchNorm2d(16)
        self.fc1 = nn.Linear(16*4*4, 120)
        self.bn3 = nn.BatchNorm1d(120)
        self.fc2 = nn.Linear(120, 84)
        self.bn4 = nn.BatchNorm1d(84)
        self.fc3 = nn.Linear(84, 10)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        x = self.bn1(self.conv1(x))
        x = self.relu(x)
        x = self.maxpool(x)
        x = self.bn2(self.conv2(x))
        x = self.relu(x)
        x = self.maxpool(x)
        x = x.view(-1, 16*4*4)
        x = self.bn3(self.fc1(x))
        x = self.relu(x)
        x = self.bn4(self.fc2(x))
        x = self.relu(x)
        x = self.fc3(x)
        return x

# Initialize the model
model = LeNet5()

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Define the batch size and number of epochs
batch_size = 64
n_epochs = 10

# Create data loaders for the train and test datasets
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Train the model
train_loss_bn = []
train_acc_bn = []
test_loss_bn = []
test_acc_bn = []

for epoch in range(n_epochs):
    model.train()
    train_loss = 0.0
    train_total = 0
    train_correct = 0
    for i, (inputs, labels) in enumerate(train_loader):
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs.data, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()

        # Print training statistics
        if (i+1) % 100 == 0:
            print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'.format(epoch+1, n_epochs, i+1, len(train_loader), loss.item()))

    train_loss /= len(train_loader.dataset)
    train_acc = 100 * train_correct / train_total
    train_loss_bn.append(train_loss)
    train_acc_bn.append(train_acc)

    model.eval()
    test_loss = 0.0
    test_total = 0
    test_correct = 0
    for i, (inputs, labels) in enumerate(test_loader):
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        test_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs.data, 1)
        test_total += labels.size(0)
        test_correct += (predicted == labels).sum().item()

    test_loss /= len(test_loader.dataset)
    test_acc = 100 * test_correct / test_total
    test_loss_bn.append(test_loss)
    test_acc_bn.append(test_acc)

    print('Epoch [{}/{}], Train Loss: {:.4f}, Train Acc: {:.2f}, Test Loss: {:.4f}, Test Acc: {:.2f}'.format(epoch+1, n_epochs, train_loss, train_acc, test_loss, test_acc))

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(train_acc_bn, label='Train')
plt.plot(test_acc_bn, label='Test')
plt.title('Accuracy with Batch Normalization on both input and layers')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(train_loss_bn, label='Train')
plt.plot(test_loss_bn, label='Test')
plt.title('Loss with Batch Normalization on both input and layers')
plt.legend()
plt.show()

bn_params = []
for name, module in model.named_modules():
    if isinstance(module, nn.BatchNorm2d) or isinstance(module, nn.BatchNorm1d):
        bn_params.append(module.weight.data.cpu().numpy())

fig, axs = plt.subplots(len(bn_params), figsize=(5, 20))
for i, p in enumerate(bn_params):
    axs[i].violinplot(dataset=p, showmeans=True)
    axs[i].set_title(f'Batch Normalization on both input and layers Layer {i+1}')
plt.show()
```

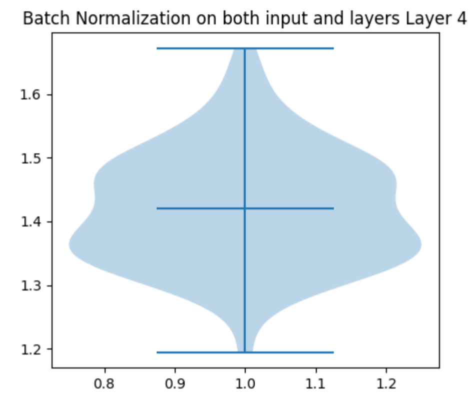
```
Epoch [1/10], Step [100/938], Loss: 0.3049
Epoch [1/10], Step [200/938], Loss: 0.2731
Epoch [1/10], Step [300/938], Loss: 0.0550
Epoch [1/10], Step [400/938], Loss: 0.0842
Epoch [1/10], Step [500/938], Loss: 0.0500
Epoch [1/10], Step [600/938], Loss: 0.0676
Epoch [1/10], Step [700/938], Loss: 0.0552
Epoch [1/10], Step [800/938], Loss: 0.0773
Epoch [1/10], Step [900/938], Loss: 0.0472
Epoch [1/10], Train Loss: 0.1659, Train Acc: 95.99, Test Loss: 0.0763, Test Acc: 97.49
Epoch [2/10], Step [100/938], Loss: 0.0335
Epoch [2/10], Step [200/938], Loss: 0.0354
Epoch [2/10], Step [300/938], Loss: 0.1013
Epoch [2/10], Step [400/938], Loss: 0.0679
Epoch [2/10], Step [500/938], Loss: 0.1258
Epoch [2/10], Step [600/938], Loss: 0.0113
Epoch [2/10], Step [700/938], Loss: 0.0941
Epoch [2/10], Step [800/938], Loss: 0.0423
Epoch [2/10], Step [900/938], Loss: 0.0226
Epoch [2/10], Train Loss: 0.0504, Train Acc: 98.48, Test Loss: 0.0426, Test Acc: 98.66
Epoch [3/10], Step [100/938], Loss: 0.0018
Epoch [3/10], Step [200/938], Loss: 0.1074
Epoch [3/10], Step [300/938], Loss: 0.0893
Epoch [3/10], Step [400/938], Loss: 0.0158
Epoch [3/10], Step [500/938], Loss: 0.0123
Epoch [3/10], Step [600/938], Loss: 0.0143
Epoch [3/10], Step [700/938], Loss: 0.0180
Epoch [3/10], Step [800/938], Loss: 0.0126
Epoch [3/10], Step [900/938], Loss: 0.0770
Epoch [3/10], Train Loss: 0.0369, Train Acc: 98.85, Test Loss: 0.0361, Test Acc: 98.88
Epoch [4/10], Step [100/938], Loss: 0.0076
Epoch [4/10], Step [200/938], Loss: 0.0528
Epoch [4/10], Step [300/938], Loss: 0.0206
Epoch [4/10], Step [400/938], Loss: 0.0393
Epoch [4/10], Step [500/938], Loss: 0.0433
Epoch [4/10], Step [600/938], Loss: 0.0172
Epoch [4/10], Step [700/938], Loss: 0.0065
Epoch [4/10], Step [800/938], Loss: 0.0089
Epoch [4/10], Step [900/938], Loss: 0.0316
Epoch [4/10], Train Loss: 0.0299, Train Acc: 99.05, Test Loss: 0.0460, Test Acc: 98.59
Epoch [5/10], Step [100/938], Loss: 0.0105
Epoch [5/10], Step [200/938], Loss: 0.0522
Epoch [5/10], Step [300/938], Loss: 0.0034
Epoch [5/10], Step [400/938], Loss: 0.0140
Epoch [5/10], Step [500/938], Loss: 0.0014
Epoch [5/10], Step [600/938], Loss: 0.0105
Epoch [5/10], Step [700/938], Loss: 0.0103
Epoch [5/10], Step [800/938], Loss: 0.1375
Epoch [5/10], Step [900/938], Loss: 0.0071
Epoch [5/10], Train Loss: 0.0257, Train Acc: 99.15, Test Loss: 0.0331, Test Acc: 98.91
Epoch [6/10], Step [100/938], Loss: 0.0278
Epoch [6/10], Step [200/938], Loss: 0.0595
Epoch [6/10], Step [300/938], Loss: 0.0068
Epoch [6/10], Step [400/938], Loss: 0.0056
Epoch [6/10], Step [500/938], Loss: 0.0286
Epoch [6/10], Step [600/938], Loss: 0.0067
Epoch [6/10], Step [700/938], Loss: 0.0701
Epoch [6/10], Step [800/938], Loss: 0.0054
Epoch [6/10], Step [900/938], Loss: 0.0988
Epoch [6/10], Train Loss: 0.0215, Train Acc: 99.29, Test Loss: 0.0275, Test Acc: 99.07
Epoch [7/10], Step [100/938], Loss: 0.0186
Epoch [7/10], Step [200/938], Loss: 0.0207
Epoch [7/10], Step [300/938], Loss: 0.0572
Epoch [7/10], Step [400/938], Loss: 0.0639
Epoch [7/10], Step [500/938], Loss: 0.0049
Epoch [7/10], Step [600/938], Loss: 0.0061
Epoch [7/10], Step [700/938], Loss: 0.0242
Epoch [7/10], Step [800/938], Loss: 0.0039
Epoch [7/10], Step [900/938], Loss: 0.0621
Epoch [7/10], Train Loss: 0.0195, Train Acc: 99.33, Test Loss: 0.0207, Test Acc: 99.29
Epoch [8/10], Step [100/938], Loss: 0.0008
Epoch [8/10], Step [200/938], Loss: 0.0040
Epoch [8/10], Step [300/938], Loss: 0.0059
Epoch [8/10], Step [400/938], Loss: 0.0494
Epoch [8/10], Step [500/938], Loss: 0.0005
Epoch [8/10], Step [600/938], Loss: 0.0012
Epoch [8/10], Step [700/938], Loss: 0.0065
Epoch [8/10], Step [800/938], Loss: 0.0041
Epoch [8/10], Step [900/938], Loss: 0.0165
Epoch [8/10], Train Loss: 0.0171, Train Acc: 99.44, Test Loss: 0.0222, Test Acc: 99.30
Epoch [9/10], Step [100/938], Loss: 0.0376
Epoch [9/10], Step [200/938], Loss: 0.0024
Epoch [9/10], Step [300/938], Loss: 0.0026
Epoch [9/10], Step [400/938], Loss: 0.0004
Epoch [9/10], Step [500/938], Loss: 0.0051
Epoch [9/10], Step [600/938], Loss: 0.0184
Epoch [9/10], Step [700/938], Loss: 0.0035
Epoch [9/10], Step [800/938], Loss: 0.0016
Epoch [9/10], Step [900/938], Loss: 0.0552
Epoch [9/10], Train Loss: 0.0181, Train Acc: 99.38, Test Loss: 0.0201, Test Acc: 99.36
Epoch [10/10], Step [100/938], Loss: 0.0005
Epoch [10/10], Step [200/938], Loss: 0.0011
Epoch [10/10], Step [300/938], Loss: 0.0054
Epoch [10/10], Step [400/938], Loss: 0.0445
Epoch [10/10], Step [500/938], Loss: 0.0022
Epoch [10/10], Step [600/938], Loss: 0.0043
Epoch [10/10], Step [700/938], Loss: 0.0094
Epoch [10/10], Step [800/938], Loss: 0.0021
Epoch [10/10], Step [900/938], Loss: 0.0012
Epoch [10/10], Train Loss: 0.0143, Train Acc: 99.56, Test Loss: 0.0227, Test Acc: 99.17
```


Accuracy with Batch Normalization on both input and layers
Train
Test
Loss with Batch Normalization on both input and layers
Train
Test





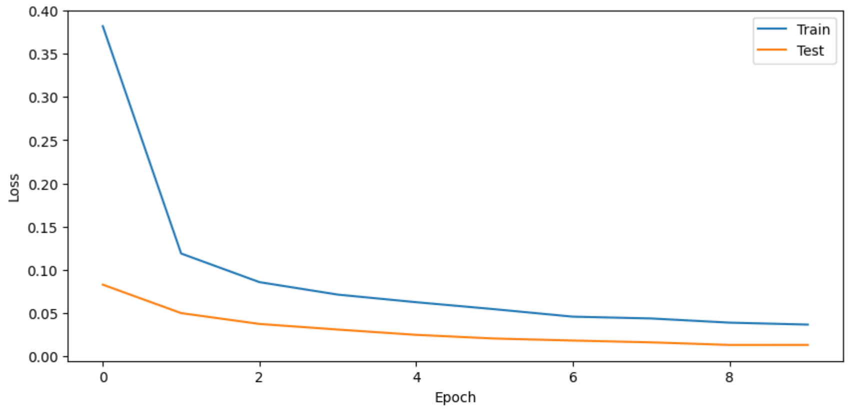


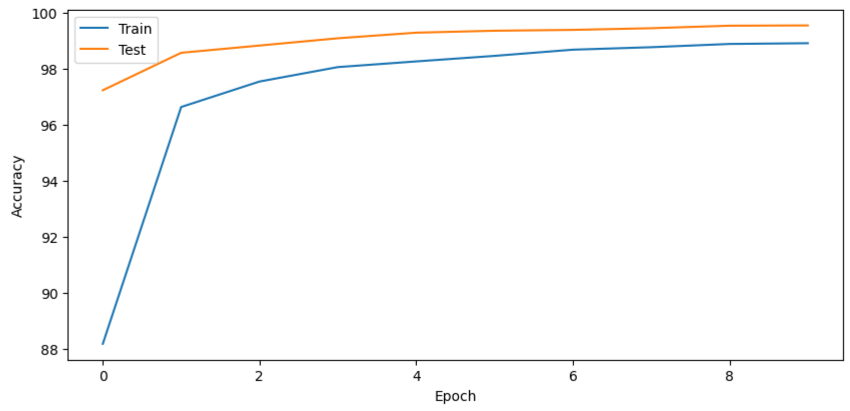


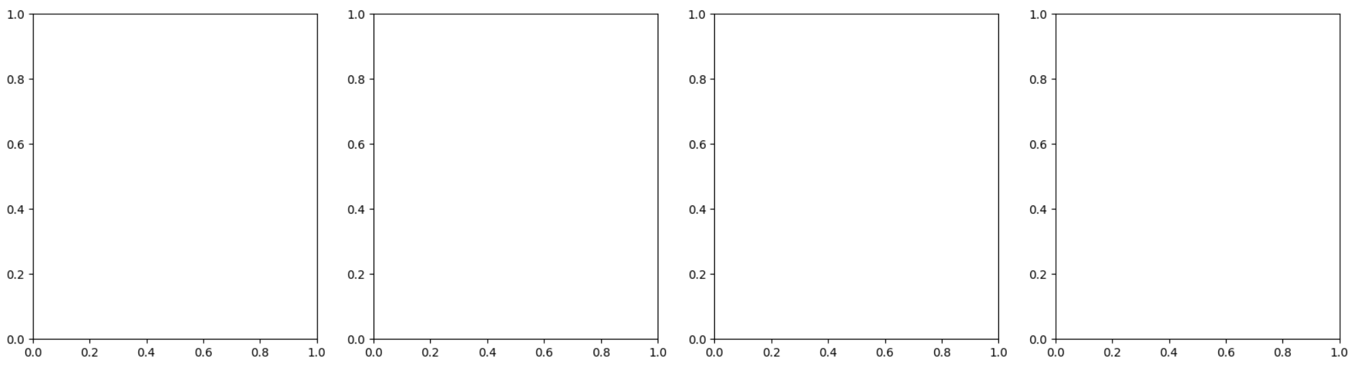
In [ ]:
```
#Solution 1.4):

#Drop out

import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import numpy as np
import matplotlib.pyplot as plt

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

# Define the LeNet-5 model with dropout for all layers
class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.fc1 = nn.Linear(16*4*4, 120)
        self.dropout1 = nn.Dropout(p=0.2)
        self.fc2 = nn.Linear(120, 84)
        self.dropout2 = nn.Dropout(p=0.5)
        self.fc3 = nn.Linear(84, 10)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.maxpool(x)
        x = self.conv2(x)
        x = self.relu(x)
        x = self.maxpool(x)
        x = x.view(-1, 16*4*4)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout1(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.dropout2(x)
        x = self.fc3(x)
        return x

# Initialize the model
model = LeNet5()

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())

# Define the batch size and number of epochs
batch_size = 64
n_epochs = 10

# Create data loaders for the train and test datasets
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Train the model
train_loss_bn = []
train_acc_bn = []
test_loss_bn = []
test_acc_bn = []

for epoch in range(n_epochs):
    model.train()
    train_loss = 0.0
    train_total = 0
    train_correct = 0
    for i, (inputs, labels) in enumerate(train_loader):
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs.data, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()

        # Print training statistics
        if (i+1) % 100 == 0:
            print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'.format(epoch+1, n_epochs, i+1, len(train_loader), loss.item()))

    train_loss /= len(train_loader.dataset)
    train_acc = 100 * train_correct / train_total
    train_loss_bn.append(train_loss)
    train_acc_bn.append(train_acc)

    model.eval()
    test_loss = 0.0
    test_total = 0
    test_correct = 0
    for i, (inputs, labels) in enumerate(test_loader):
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        test_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs.data, 1)
        test_total += labels.size(0)
        test_correct += (predicted == labels).sum().item()

    test_loss /= len(test_loader.dataset)
    test_acc = 100 * test_correct / test_total
    test_loss_bn.append(test_loss)
    test_acc_bn.append(test_acc)

    print('Epoch [{}/{}], Train Loss: {:.4f}, Train Acc: {:.2f}, Test Loss: {:.4f}, Test Acc: {:.2f}'.format(epoch+1, n_epochs, train_loss, train_acc, test_loss, test_acc))

# Plot the train/test loss and accuracy
fig, axs = plt.subplots(2, figsize=(10, 10))
axs[0].plot(train_loss_bn, label='Train')
axs[0].plot(test_loss_bn, label='Test')
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Loss')
axs[0].legend()
axs[1].plot(train_acc_bn, label='Train')
axs[1].plot(test_acc_bn, label='Test')
axs[1].set_xlabel('Epoch')
axs[1].set_ylabel('Accuracy')
axs[1].legend()

# Plot the distribution of learned dropout parameters for each layer
fig, axs = plt.subplots(1, 4, figsize=(20, 5))
dropout_params = []
for i, module in enumerate(model.modules()):
    if isinstance(module, nn.Dropout):
        axs[i].violinplot(module.p.cpu().detach().numpy())
        axs[i].set_xticks([])
        axs[i].set_title(f'Layer {i+1}')
        dropout_params.append(module.p.cpu().detach().numpy())
plt.show()
```

```
Epoch [1/10], Step [100/938], Loss: 0.7948
Epoch [1/10], Step [200/938], Loss: 0.4855
Epoch [1/10], Step [300/938], Loss: 0.1772
Epoch [1/10], Step [400/938], Loss: 0.2756
Epoch [1/10], Step [500/938], Loss: 0.0792
Epoch [1/10], Step [600/938], Loss: 0.1046
Epoch [1/10], Step [700/938], Loss: 0.2825
Epoch [1/10], Step [800/938], Loss: 0.0977
Epoch [1/10], Step [900/938], Loss: 0.1902
Epoch [1/10], Train Loss: 0.3818, Train Acc: 88.17, Test Loss: 0.0831, Test Acc: 97.24
Epoch [2/10], Step [100/938], Loss: 0.1191
Epoch [2/10], Step [200/938], Loss: 0.0554
Epoch [2/10], Step [300/938], Loss: 0.1783
Epoch [2/10], Step [400/938], Loss: 0.0300
Epoch [2/10], Step [500/938], Loss: 0.1074
Epoch [2/10], Step [600/938], Loss: 0.0890
Epoch [2/10], Step [700/938], Loss: 0.0720
Epoch [2/10], Step [800/938], Loss: 0.0665
Epoch [2/10], Step [900/938], Loss: 0.0707
Epoch [2/10], Train Loss: 0.1191, Train Acc: 96.64, Test Loss: 0.0502, Test Acc: 98.58
Epoch [3/10], Step [100/938], Loss: 0.1632
Epoch [3/10], Step [200/938], Loss: 0.2067
Epoch [3/10], Step [300/938], Loss: 0.0597
Epoch [3/10], Step [400/938], Loss: 0.0557
Epoch [3/10], Step [500/938], Loss: 0.0580
Epoch [3/10], Step [600/938], Loss: 0.0668
Epoch [3/10], Step [700/938], Loss: 0.0263
Epoch [3/10], Step [800/938], Loss: 0.1739
Epoch [3/10], Step [900/938], Loss: 0.0503
Epoch [3/10], Train Loss: 0.0860, Train Acc: 97.55, Test Loss: 0.0377, Test Acc: 98.84
Epoch [4/10], Step [100/938], Loss: 0.0473
Epoch [4/10], Step [200/938], Loss: 0.0680
Epoch [4/10], Step [300/938], Loss: 0.1888
Epoch [4/10], Step [400/938], Loss: 0.0284
Epoch [4/10], Step [500/938], Loss: 0.0325
Epoch [4/10], Step [600/938], Loss: 0.0340
Epoch [4/10], Step [700/938], Loss: 0.0809
Epoch [4/10], Step [800/938], Loss: 0.0621
Epoch [4/10], Step [900/938], Loss: 0.0681
Epoch [4/10], Train Loss: 0.0716, Train Acc: 98.07, Test Loss: 0.0312, Test Acc: 99.10
Epoch [5/10], Step [100/938], Loss: 0.0624
Epoch [5/10], Step [200/938], Loss: 0.1523
Epoch [5/10], Step [300/938], Loss: 0.1516
Epoch [5/10], Step [400/938], Loss: 0.0276
Epoch [5/10], Step [500/938], Loss: 0.0258
Epoch [5/10], Step [600/938], Loss: 0.0453
Epoch [5/10], Step [700/938], Loss: 0.2150
Epoch [5/10], Step [800/938], Loss: 0.0410
Epoch [5/10], Step [900/938], Loss: 0.0585
Epoch [5/10], Train Loss: 0.0628, Train Acc: 98.27, Test Loss: 0.0251, Test Acc: 99.30
Epoch [6/10], Step [100/938], Loss: 0.0099
Epoch [6/10], Step [200/938], Loss: 0.0283
Epoch [6/10], Step [300/938], Loss: 0.0566
Epoch [6/10], Step [400/938], Loss: 0.0504
Epoch [6/10], Step [500/938], Loss: 0.0686
Epoch [6/10], Step [600/938], Loss: 0.0408
Epoch [6/10], Step [700/938], Loss: 0.0369
Epoch [6/10], Step [800/938], Loss: 0.0395
Epoch [6/10], Step [900/938], Loss: 0.0323
Epoch [6/10], Train Loss: 0.0547, Train Acc: 98.47, Test Loss: 0.0208, Test Acc: 99.37
Epoch [7/10], Step [100/938], Loss: 0.0985
Epoch [7/10], Step [200/938], Loss: 0.1369
Epoch [7/10], Step [300/938], Loss: 0.0536
Epoch [7/10], Step [400/938], Loss: 0.0841
Epoch [7/10], Step [500/938], Loss: 0.0151
Epoch [7/10], Step [600/938], Loss: 0.0088
Epoch [7/10], Step [700/938], Loss: 0.0855
Epoch [7/10], Step [800/938], Loss: 0.0126
Epoch [7/10], Step [900/938], Loss: 0.0082
Epoch [7/10], Train Loss: 0.0461, Train Acc: 98.69, Test Loss: 0.0185, Test Acc: 99.40
Epoch [8/10], Step [100/938], Loss: 0.0016
Epoch [8/10], Step [200/938], Loss: 0.0212
Epoch [8/10], Step [300/938], Loss: 0.0315
Epoch [8/10], Step [400/938], Loss: 0.0169
Epoch [8/10], Step [500/938], Loss: 0.2024
Epoch [8/10], Step [600/938], Loss: 0.1852
Epoch [8/10], Step [700/938], Loss: 0.0311
Epoch [8/10], Step [800/938], Loss: 0.0656
Epoch [8/10], Step [900/938], Loss: 0.0361
Epoch [8/10], Train Loss: 0.0440, Train Acc: 98.78, Test Loss: 0.0164, Test Acc: 99.46
Epoch [9/10], Step [100/938], Loss: 0.0286
Epoch [9/10], Step [200/938], Loss: 0.0319
Epoch [9/10], Step [300/938], Loss: 0.2100
Epoch [9/10], Step [400/938], Loss: 0.1601
Epoch [9/10], Step [500/938], Loss: 0.0027
Epoch [9/10], Step [600/938], Loss: 0.0210
Epoch [9/10], Step [700/938], Loss: 0.0756
Epoch [9/10], Step [800/938], Loss: 0.0059
Epoch [9/10], Step [900/938], Loss: 0.0035
Epoch [9/10], Train Loss: 0.0392, Train Acc: 98.90, Test Loss: 0.0134, Test Acc: 99.55
Epoch [10/10], Step [100/938], Loss: 0.0305
Epoch [10/10], Step [200/938], Loss: 0.1178
Epoch [10/10], Step [300/938], Loss: 0.0148
Epoch [10/10], Step [400/938], Loss: 0.0070
Epoch [10/10], Step [500/938], Loss: 0.0105
Epoch [10/10], Step [600/938], Loss: 0.0290
Epoch [10/10], Step [700/938], Loss: 0.1500
Epoch [10/10], Step [800/938], Loss: 0.0116
Epoch [10/10], Step [900/938], Loss: 0.0887
Epoch [10/10], Train Loss: 0.0369, Train Acc: 98.92, Test Loss: 0.0134, Test Acc: 99.56
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/home/karanvora/Documents/New York University/Classes/Semester 2/Introdution to High-Performance Machine Learning/Assignments/Assignment 3/kv2154_Assignment3_Problem1.ipynb
Cell 4 in 1
    <a href='vscode-notebook-cell:/home/karanvora/Documents/New%20York%20University/Classes/Semester%202/Introdution%20to%20High-Performance%20Machine%20Learning/Assignment
s/Assignment%203/kv2154_Assignment3_Problem1.ipynb#W3sZmlsZQ%3D%3D?line=131'>132</a> for i, module in enumerate(model.modules()):
    <a href='vscode-notebook-cell:/home/karanvora/Documents/New%20York%20University/Classes/Semester%202/Introdution%20to%20High-Performance%20Machine%20Learning/Assignment
s/Assignment%203/kv2154_Assignment3_Problem1.ipynb#W3sZmlsZQ%3D%3D?line=132'>133</a>     if isinstance(module, nn.Dropout):
--> <a href='vscode-notebook-cell:/home/karanvora/Documents/New%20York%20University/Classes/Semester%202/Introdution%20to%20High-Performance%20Machine%20Learning/Assignment
s/Assignment%203/kv2154_Assignment3_Problem1.ipynb#W3sZmlsZQ%3D%3D?line=133'>134</a>         axs[i].violinplot(module.p.cpu().detach().numpy())
    <a href='vscode-notebook-cell:/home/karanvora/Documents/New%20York%20University/Classes/Semester%202/Introdution%20to%20High-Performance%20Machine%20Learning/Assignment
s/Assignment%203/kv2154_Assignment3_Problem1.ipynb#W3sZmlsZQ%3D%3D?line=134'>135</a>         axs[i].set_xticks([])
    <a href='vscode-notebook-cell:/home/karanvora/Documents/New%20York%20University/Classes/Semester%202/Introdution%20to%20High-Performance%20Machine%20Learning/Assignment
s/Assignment%203/kv2154_Assignment3_Problem1.ipynb#W3sZmlsZQ%3D%3D?line=135'>136</a>         axs[i].set_title(f'Layer {i+1}')

IndexError: index 4 is out of bounds for axis 0 with size 4
```



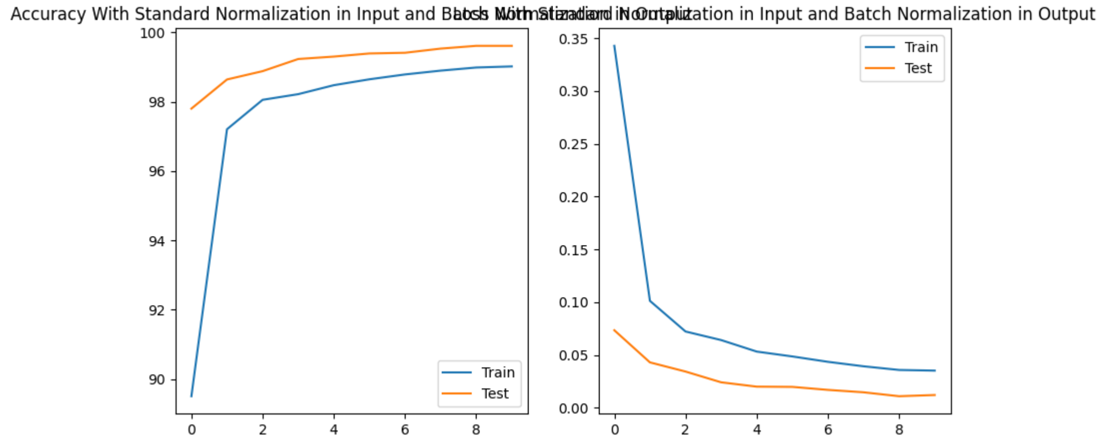




```
In [ ]: #Solution 1.5

        #Solution 1.2):

        import torch
        import torch.nn as nn
        import torch.optim as optim
        import torchvision.datasets as datasets
        import torchvision.transforms as transforms
        import numpy as np
        import matplotlib.pyplot as plt

        # Load the MNIST dataset
        train_dataset = datasets.MNIST(root='./data', train=True, transform=transforms.Compose([
                                            transforms.ToTensor(),
                                            transforms.Normalize((0.5,), (0.5,))
                                        ]), download=True)
        test_dataset = datasets.MNIST(root='./data', train=False, transform=transforms.Compose([
                                            transforms.ToTensor(),
                                            transforms.Normalize((0.5,), (0.5,))
                                        ]), download=True)

        # Define the data loaders
        batch_size = 64
        train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
        test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

```python
# Define the LeNet-5 model with batch normalization for all layers
class LeNet5BN(nn.Module):
    def __init__(self, num_classes=10):
        super(LeNet5BN, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5, stride=1)
        self.bn1 = nn.BatchNorm2d(6)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5, stride=1)
        self.bn2 = nn.BatchNorm2d(16)
        self.fc1 = nn.Linear(16*4*4, 120)
        self.bn3 = nn.BatchNorm1d(120)
        self.fc2 = nn.Linear(120, 84)
        self.bn4 = nn.BatchNorm1d(84)
        self.fc3 = nn.Linear(84, num_classes)
        self.dropout = nn.Dropout(p=0.5)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(self.bn1(x))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = self.conv2(x)
        x = F.relu(self.bn2(x))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = x.view(-1, 16*4*4)
        x = F.relu(self.bn3(self.fc1(x)))
        x = F.relu(self.bn4(self.fc2(x)))
        x = self.dropout(x)
        x = self.fc3(x)
        return x

# Initialize the model
model = LeNet5()

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Define the batch size and number of epochs
batch_size = 64
n_epochs = 10

# Create data loaders for the train and test datasets
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Train the model
train_loss_bn = []
train_acc_bn = []
test_loss_bn = []
test_acc_bn = []

for epoch in range(n_epochs):
    model.train()
    train_loss = 0.0
    train_total = 0
    train_correct = 0
    for i, (inputs, labels) in enumerate(train_loader):
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs.data, 1)
        train_total += labels.size(0)
        train_correct += (predicted == labels).sum().item()

        # Print training statistics
        if (i+1) % 100 == 0:
            print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'.format(epoch+1, n_epochs, i+1, len(train_loader), loss.item()))

    train_loss /= len(train_loader.dataset)
    train_acc = 100 * train_correct / train_total
    train_loss_bn.append(train_loss)
    train_acc_bn.append(train_acc)

    model.eval()
    test_loss = 0.0
    test_total = 0
    test_correct = 0
    for i, (inputs, labels) in enumerate(test_loader):
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        test_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs.data, 1)
        test_total += labels.size(0)
        test_correct += (predicted == labels).sum().item()

    test_loss /= len(test_loader.dataset)
    test_acc = 100 * test_correct / test_total
    test_loss_bn.append(test_loss)
    test_acc_bn.append(test_acc)

    print('Epoch [{}/{}], Train Loss: {:.4f}, Train Acc: {:.2f}, Test Loss: {:.4f}, Test Acc: {:.2f}'.format(epoch+1, n_epochs, train_loss, train_acc, test_loss, test_acc))

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(train_acc_bn, label='Train')
plt.plot(test_acc_bn, label='Test')
plt.title('Accuracy With Standard Normalization in Input and Batch Normalization in Output')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(train_loss_bn, label='Train')
plt.plot(test_loss_bn, label='Test')
plt.title('Loss With Standard Normalization in Input and Batch Normalization in Output')
plt.legend()
plt.show()
```

```
Epoch [1/10], Step [100/938], Loss: 0.6428
Epoch [1/10], Step [200/938], Loss: 0.1904
Epoch [1/10], Step [300/938], Loss: 0.2616
Epoch [1/10], Step [400/938], Loss: 0.2062
Epoch [1/10], Step [500/938], Loss: 0.0988
Epoch [1/10], Step [600/938], Loss: 0.1413
Epoch [1/10], Step [700/938], Loss: 0.0973
Epoch [1/10], Step [800/938], Loss: 0.1326
Epoch [1/10], Step [900/938], Loss: 0.2702
Epoch [1/10], Train Loss: 0.3428, Train Acc: 89.51, Test Loss: 0.0735, Test Acc: 97.80
Epoch [2/10], Step [100/938], Loss: 0.1540
Epoch [2/10], Step [200/938], Loss: 0.4550
Epoch [2/10], Step [300/938], Loss: 0.1096
Epoch [2/10], Step [400/938], Loss: 0.0733
Epoch [2/10], Step [500/938], Loss: 0.0988
Epoch [2/10], Step [600/938], Loss: 0.0900
Epoch [2/10], Step [700/938], Loss: 0.0786
Epoch [2/10], Step [800/938], Loss: 0.0221
Epoch [2/10], Step [900/938], Loss: 0.0706
Epoch [2/10], Train Loss: 0.1013, Train Acc: 97.20, Test Loss: 0.0431, Test Acc: 98.64
Epoch [3/10], Step [100/938], Loss: 0.0635
Epoch [3/10], Step [200/938], Loss: 0.1027
Epoch [3/10], Step [300/938], Loss: 0.0484
Epoch [3/10], Step [400/938], Loss: 0.1167
Epoch [3/10], Step [500/938], Loss: 0.0220
Epoch [3/10], Step [600/938], Loss: 0.0942
Epoch [3/10], Step [700/938], Loss: 0.1368
Epoch [3/10], Step [800/938], Loss: 0.0892
Epoch [3/10], Step [900/938], Loss: 0.0178
Epoch [3/10], Train Loss: 0.0723, Train Acc: 98.05, Test Loss: 0.0344, Test Acc: 98.88
Epoch [4/10], Step [100/938], Loss: 0.0315
Epoch [4/10], Step [200/938], Loss: 0.0512
Epoch [4/10], Step [300/938], Loss: 0.0458
Epoch [4/10], Step [400/938], Loss: 0.0847
Epoch [4/10], Step [500/938], Loss: 0.0328
Epoch [4/10], Step [600/938], Loss: 0.0369
Epoch [4/10], Step [700/938], Loss: 0.0266
Epoch [4/10], Step [800/938], Loss: 0.0252
Epoch [4/10], Step [900/938], Loss: 0.0774
Epoch [4/10], Train Loss: 0.0642, Train Acc: 98.22, Test Loss: 0.0242, Test Acc: 99.23
Epoch [5/10], Step [100/938], Loss: 0.0196
Epoch [5/10], Step [200/938], Loss: 0.0713
Epoch [5/10], Step [300/938], Loss: 0.0727
Epoch [5/10], Step [400/938], Loss: 0.0074
Epoch [5/10], Step [500/938], Loss: 0.2092
Epoch [5/10], Step [600/938], Loss: 0.0414
Epoch [5/10], Step [700/938], Loss: 0.0326
Epoch [5/10], Step [800/938], Loss: 0.0227
Epoch [5/10], Step [900/938], Loss: 0.0463
Epoch [5/10], Train Loss: 0.0533, Train Acc: 98.47, Test Loss: 0.0201, Test Acc: 99.30
Epoch [6/10], Step [100/938], Loss: 0.0145
Epoch [6/10], Step [200/938], Loss: 0.0320
Epoch [6/10], Step [300/938], Loss: 0.0150
Epoch [6/10], Step [400/938], Loss: 0.0228
Epoch [6/10], Step [500/938], Loss: 0.1365
Epoch [6/10], Step [600/938], Loss: 0.2312
Epoch [6/10], Step [700/938], Loss: 0.0144
Epoch [6/10], Step [800/938], Loss: 0.0183
Epoch [6/10], Step [900/938], Loss: 0.1622
Epoch [6/10], Train Loss: 0.0487, Train Acc: 98.64, Test Loss: 0.0199, Test Acc: 99.39
Epoch [7/10], Step [100/938], Loss: 0.0121
Epoch [7/10], Step [200/938], Loss: 0.0111
Epoch [7/10], Step [300/938], Loss: 0.0261
Epoch [7/10], Step [400/938], Loss: 0.0068
Epoch [7/10], Step [500/938], Loss: 0.0059
Epoch [7/10], Step [600/938], Loss: 0.0685
Epoch [7/10], Step [700/938], Loss: 0.0133
Epoch [7/10], Step [800/938], Loss: 0.0636
Epoch [7/10], Step [900/938], Loss: 0.0264
Epoch [7/10], Train Loss: 0.0436, Train Acc: 98.78, Test Loss: 0.0170, Test Acc: 99.41
Epoch [8/10], Step [100/938], Loss: 0.0266
Epoch [8/10], Step [200/938], Loss: 0.0381
Epoch [8/10], Step [300/938], Loss: 0.0589
Epoch [8/10], Step [400/938], Loss: 0.0236
Epoch [8/10], Step [500/938], Loss: 0.0520
Epoch [8/10], Step [600/938], Loss: 0.0138
Epoch [8/10], Step [700/938], Loss: 0.0013
Epoch [8/10], Step [800/938], Loss: 0.0370
Epoch [8/10], Step [900/938], Loss: 0.0008
Epoch [8/10], Train Loss: 0.0394, Train Acc: 98.89, Test Loss: 0.0148, Test Acc: 99.53
Epoch [9/10], Step [100/938], Loss: 0.0046
Epoch [9/10], Step [200/938], Loss: 0.0331
Epoch [9/10], Step [300/938], Loss: 0.0077
Epoch [9/10], Step [400/938], Loss: 0.0112
Epoch [9/10], Step [500/938], Loss: 0.0601
Epoch [9/10], Step [600/938], Loss: 0.0107
Epoch [9/10], Step [700/938], Loss: 0.0278
Epoch [9/10], Step [800/938], Loss: 0.0028
Epoch [9/10], Step [900/938], Loss: 0.0166
Epoch [9/10], Train Loss: 0.0359, Train Acc: 98.98, Test Loss: 0.0110, Test Acc: 99.61
Epoch [10/10], Step [100/938], Loss: 0.0008
Epoch [10/10], Step [200/938], Loss: 0.0011
Epoch [10/10], Step [300/938], Loss: 0.0248
Epoch [10/10], Step [400/938], Loss: 0.0269
Epoch [10/10], Step [500/938], Loss: 0.0175
Epoch [10/10], Step [600/938], Loss: 0.0076
Epoch [10/10], Step [700/938], Loss: 0.0062
Epoch [10/10], Step [800/938], Loss: 0.0579
Epoch [10/10], Step [900/938], Loss: 0.1801
Epoch [10/10], Train Loss: 0.0353, Train Acc: 99.02, Test Loss: 0.0122, Test Acc: 99.61
```



```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/home/karanvora/Documents/New York University/Classes/Semester 2/Introdution to High-Performance Machine Learning/Assignments/Assignment 3/kv2154_Assignment3_Problem1.ipynb Cell 5 in 1
    <a href='vscode-notebook-cell:/home/karanvora/Documents/New%20York%20University/Classes/Semester%202/Introdution%20to%20High-Performance%20Machine%20Learning/Assignments/Assignment%203/kv2154_Assignment3_Problem1.ipynb#W6sZmlsZQ%3D%3D?line=139'>140</a>     if isinstance(module, nn.BatchNorm2d) or isinstance(module, nn.BatchNorm1d):
    <a href='vscode-notebook-cell:/home/karanvora/Documents/New%20York%20University/Classes/Semester%202/Introdution%20to%20High-Performance%20Machine%20Learning/Assignments/Assignment%203/kv2154_Assignment3_Problem1.ipynb#W6sZmlsZQ%3D%3D?line=140'>141</a>         bn_params.append(module.weight.data.cpu().numpy())
--> <a href='vscode-notebook-cell:/home/karanvora/Documents/New%20York%20University/Classes/Semester%202/Introdution%20to%20High-Performance%20Machine%20Learning/Assignments/Assignment%203/kv2154_Assignment3_Problem1.ipynb#W6sZmlsZQ%3D%3D?line=142'>143</a> fig, axs = plt.subplots(len(bn_params), figsize=(5, 20))
    <a href='vscode-notebook-cell:/home/karanvora/Documents/New%20York%20University/Classes/Semester%202/Introdution%20to%20High-Performance%20Machine%20Learning/Assignments/Assignment%203/kv2154_Assignment3_Problem1.ipynb#W6sZmlsZQ%3D%3D?line=143'>144</a> for i, p in enumerate(bn_params):
    <a href='vscode-notebook-cell:/home/karanvora/Documents/New%20York%20University/Classes/Semester%202/Introdution%20to%20High-Performance%20Machine%20Learning/Assignments/Assignment%203/kv2154_Assignment3_Problem1.ipynb#W6sZmlsZQ%3D%3D?line=144'>145</a>     axs[i].violinplot(dataset=p, showmeans=True)

File ~/.local/lib/python3.10/site-packages/matplotlib/pyplot.py:1454, in subplots(nrows, ncols, sharex, sharey, squeeze, subplot_kw, gridspec_kw, **fig_kw)
   1321 """
   1322 Create a figure and a set of subplots.
   1323 
   (...)
   1451 
   1452 """
   1453 fig = figure(**fig_kw)
-> 1454 axs = fig.subplots(nrows=nrows, ncols=ncols, sharex=sharex, sharey=sharey,
   1455                    squeeze=squeeze, subplot_kw=subplot_kw,
   1456                    gridspec_kw=gridspec_kw)
   1457 return fig, axs

File ~/.local/lib/python3.10/site-packages/matplotlib/figure.py:896, in FigureBase.subplots(self, nrows, ncols, sharex, sharey, squeeze, subplot_kw, gridspec_kw)
    894 if gridspec_kw is None:
    895     gridspec_kw = {}
--> 896 gs = self.add_gridspec(nrows, ncols, figure=self, **gridspec_kw)
    897 axs = gs.subplots(sharex=sharex, sharey=sharey, squeeze=squeeze,
    898                   subplot_kw=subplot_kw)
    899 return axs

File ~/.local/lib/python3.10/site-packages/matplotlib/figure.py:1447, in FigureBase.add_gridspec(self, nrows, ncols, **kwargs)
   1408 """
   1409 Return a `.GridSpec` that has this figure as a parent.  This allows
   1410 complex layout of Axes in the figure.
   (...)
   1443 
   1444 """
   1446 _ = kwargs.pop('figure', None)  # pop in case user has added this...
-> 1447 gs = GridSpec(nrows=nrows, ncols=ncols, figure=self, **kwargs)
   1448 self._gridspecs.append(gs)
   1449 return gs

File ~/.local/lib/python3.10/site-packages/matplotlib/gridspec.py:385, in GridSpec.__init__(self, nrows, ncols, figure, left, bottom, right, top, wspace, hspace, width_ratios, height_ratios)
    382 self.hspace = hspace
    383 self.figure = figure
--> 385 super().__init__(nrows, ncols,
    386                  width_ratios=width_ratios,
    387                  height_ratios=height_ratios)

File ~/.local/lib/python3.10/site-packages/matplotlib/gridspec.py:49, in GridSpecBase.__init__(self, nrows, ncols, height_ratios, width_ratios)
     34 """
     35 Parameters
     36 ----------
   (...)
     46     If not given, all rows will have the same height.
     47 """
     48 if not isinstance(nrows, Integral) or nrows <= 0:
---> 49     raise ValueError(
     50         f"Number of rows must be a positive integer, not {nrows!r}")
     51 if not isinstance(ncols, Integral) or ncols <= 0:
     52     raise ValueError(
     53         f"Number of columns must be a positive integer, not {ncols!r}")

ValueError: Number of rows must be a positive integer, not 0
<Figure size 500x2000 with 0 Axes>
```