```
import torch.nn as nn
from torch.utils.data import DataLoader
import torch
import torchvision
import torchvision.transforms as transforms


# Instantiate model with BN and load trained parameters
class smallNetTrain(nn.Module) :
    # CIFAR-10 data is 32*32 images with 3 RGB channels
    def __init__(self, input_dim=3*32*32) :
        super().__init__()

        self.conv1 = nn.Sequential(
                        nn.Conv2d(3, 16, kernel_size=3, padding=1),
                        nn.BatchNorm2d(16),
                        nn.ReLU()
                        )
        self.conv2 = nn.Sequential(
                        nn.Conv2d(16, 16, kernel_size=3, padding=1),
                        nn.BatchNorm2d(16),
                        nn.ReLU()
                        )
        self.fc1 = nn.Sequential(
                        nn.Linear(16*32*32, 32*32),
                        nn.BatchNorm1d(32*32),
                        nn.ReLU()
                        )
        self.fc2 = nn.Sequential(
                        nn.Linear(32*32, 10),
                        nn.ReLU()
                        )
    def forward(self, x) :
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.float().view(-1, 16*32*32)
        x = self.fc1(x)
        x = self.fc2(x)

        return x

model = smallNetTrain()
model.load_state_dict(torch.load("./smallNetSaved",map_location=torch.device('cpu')))


# Instantiate model without BN
class smallNetTest(nn.Module) :
    # CIFAR-10 data is 32*32 images with 3 RGB channels
    def __init__(self, input_dim=3*32*32) :
        super().__init__()

        self.conv1 = nn.Sequential(
                        nn.Conv2d(3, 16, kernel_size=3, padding=1),
```

```
                    nn.ReLU()
                    )
        self.conv2 = nn.Sequential(
                    nn.Conv2d(16, 16, kernel_size=3, padding=1),
                    nn.ReLU()
                    )
        self.fc1 = nn.Sequential(
                    nn.Linear(16*32*32, 32*32),
                    nn.ReLU()
                    )
        self.fc2 = nn.Sequential(
                    nn.Linear(32*32, 10),
                    nn.ReLU()
                    )
    def forward(self, x) :
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.float().view(-1, 16*32*32)
        x = self.fc1(x)
        x = self.fc2(x)

        return x

model_test = smallNetTest()



# Initialize weights of model without BN

conv1_bn_beta, conv1_bn_gamma = model.conv1[1].bias, model.conv1[1].weight
conv1_bn_mean, conv1_bn_var = model.conv1[1].running_mean, model.conv1[1].running_var
conv2_bn_beta, conv2_bn_gamma = model.conv2[1].bias, model.conv2[1].weight
conv2_bn_mean, conv2_bn_var = model.conv2[1].running_mean, model.conv2[1].running_var
fc1_bn_beta, fc1_bn_gamma = model.fc1[1].bias, model.fc1[1].weight
fc1_bn_mean, fc1_bn_var = model.fc1[1].running_mean, model.fc1[1].running_var
eps = 1e-05



# Initialize the following parameters
model_test.conv1[0].weight.data = model.conv1[0].weight * (conv1_bn_gamma/torch.sqrt(conv1_bn_var
+ eps)).view(-1,1,1,1)
model_test.conv1[0].bias.data = conv1_bn_beta +
conv1_bn_gamma*(model.conv1[0].bias-conv1_bn_mean)/torch.sqrt(conv1_bn_var + eps)

model_test.conv2[0].weight.data = model.conv2[0].weight * (conv2_bn_gamma/torch.sqrt(conv2_bn_var
+ eps)).view(-1,1,1,1)
model_test.conv2[0].bias.data = conv2_bn_beta +
conv2_bn_gamma*(model.conv2[0].bias-conv2_bn_mean)/torch.sqrt(conv2_bn_var + eps)

model_test.fc1[0].weight.data = model.fc1[0].weight * (fc1_bn_gamma/torch.sqrt(fc1_bn_var +
eps)).view(-1,1)
model_test.fc1[0].bias.data = fc1_bn_beta +
```

```
fc1_bn_gamma*(model.fc1[0].bias-fc1_bn_mean)/torch.sqrt(fc1_bn_var + eps)

model_test.fc2[0].weight.data = model.fc2[0].weight
model_test.fc2[0].bias.data = model.fc2[0].bias




# Verify difference between model and model_test

model.eval()
# model_test.eval()  # not necessary since model_test has no BN or dropout


test_dataset = torchvision.datasets.CIFAR10(root='./cifar_10data/',
                                train=False,
                                transform=transforms.ToTensor(), download = True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=100, shuffle=False)

diff = []
with torch.no_grad():
    for images, _ in test_loader:
        diff.append(torch.norm(model(images) - model_test(images))**2)

print(max(diff)) # If less than 1e-08, you got the right answer.



'''
For debugging purposes, you may want to match the output of conv1 first before
moving on working on conv2. To do so, you can replace the forward-evaluation
functions of the two models with
def forward(self, x) :
    x = self.conv1(x)
    return x
'''
```

Result

Files already downloaded and verified

tensor(7.1059e-09)