CNN

May 24, 2021

[1]:
```
%matplotlib inline

import torch
import torchvision
import torchvision.transforms as transforms

import matplotlib.pyplot as plt
import numpy as np

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import random
import numpy as np
import torch.cuda as cuda

import time


import os
```

[2]:
```
in_submission = os.path.exists('/flags/isgrader.flag')
perform_computation = not in_submission

if in_submission:
    assert os.path.exists('./cifar_net.pth'), 'The trained network for CIFAR␣
 ↪was not stored properly. ' + \
                                              'Please read and follow the␣
 ↪instructions/important notes.'

    assert os.path.exists('./mnist_net.pth'), 'The trained network for MNIST␣
 ↪was not stored properly. ' + \
                                              'Please read and follow the␣
 ↪instructions/important notes.'
```

1 *Assignment Summary

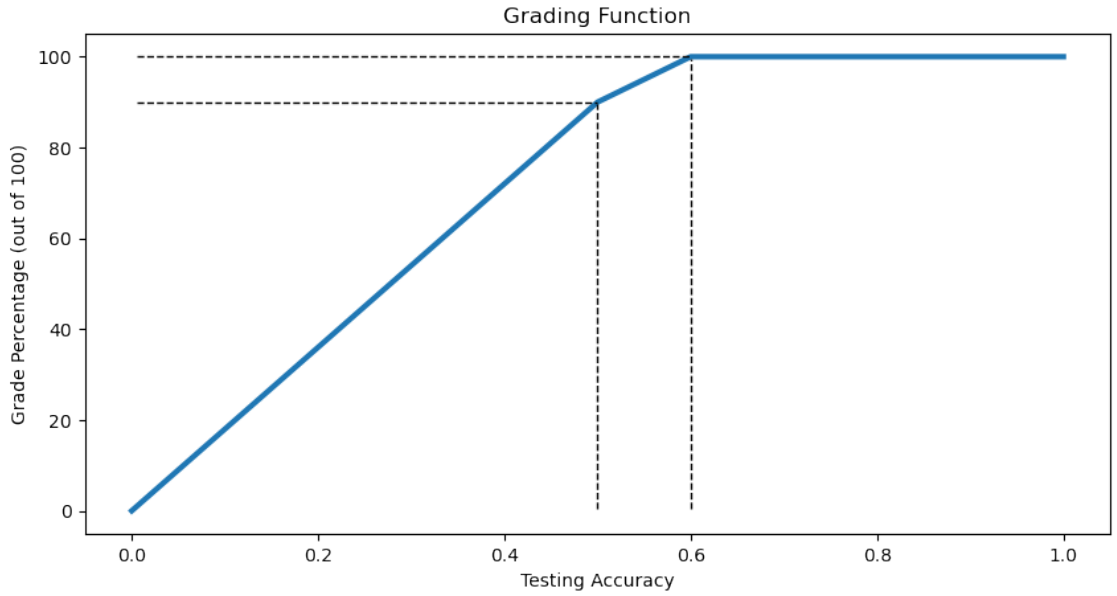
Go through the CIFAR-10 tutorial at https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html, and ensure you can run the code. Modify the architecture that is offered in the CIFAR-10 tutorial to get the best accuracy you can. Anything better than about 93.5% will be comparable with current research.

Redo the same efforts for the MNIST digit data set.

Procedural Instructions:

This assignment is less guided than the previous assignments. You are supposed to train a deep convolutional classifier, and store it in a file. The autograder will load the trained model, and test its accuracy on a hidden test data set. Your classifier's test accuracy will determine your grade for each part according to the following model.

[3]:
```
fig, ax = plt.subplots(1, 1, figsize=(10, 5), dpi=100)
ax.plot([0., 0.5, 0.6, 1.], [0., 90., 100., 100.], lw=3)
ax.axhline(y=90, xmin=0.05, xmax=.5, lw=1, ls='--', c='black')
ax.axvline(x=0.5, ymin=0.05, ymax=.86, lw=1, ls='--', c='black')
ax.axhline(y=100, xmin=0.05, xmax=.59, lw=1, ls='--', c='black')
ax.axvline(x=0.6, ymin=0.05, ymax=.95, lw=1, ls='--', c='black')
ax.set_xlabel('Testing Accuracy')
ax.set_ylabel('Grade Percentage (out of 100)')
ax.set_title('Grading Function')
None
```



2 Important Notes

You **should** read these notes before starting as these notes include crucial information about what is expected from you.

1. **Use Pytorch**: The autograder will only accept pytorch models.

- Pytorch's CIFAR-10 tutorial at https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html is the best starting point for this assignment. However, we will not prohibit using or learning from any other tutorial you may find online.

2. **No Downloads**: The coursera machines are disconnected from the internet. We already have downloaded the pytorch data files, and uploaded them for you. You will need to disable downloading the files if you're using data collector APIs such as `torchvision.datasets`.

- For the CIFAR data, you should provide the `root='/home/jovyan/work/course-lib/data_cifar', download=False` arguments to the `torchvision.datasets.CIFAR10` API.
- For the MNIST data, you should provide the `root='/home/jovyan/work/course-lib/data_mnist', download=False` arguments to the `torchvision.datasets.MNIST` API.

3. **Store the Trained Model**: The autograder can not and will not retrain your model. You are supposed to train your model, and then store your best model with the following names:

- The CIFAR classification model must be stored at `./cifar_net.pth`.
- The MNIST classification model must be stored at `./mnist_net.pth`.
- Do not place these file under any newly created directory.
- The trained model may **not exceed 1 MB** in size.

4. **Model Class Naming**: The neural models in the pytorch library are subclasses of the `torch.nn.Module` class. While you can define any architecture as you please, your `torch.nn.Module` must be named `Net` exactly. In other words, you are supposed to have the following lines somewhere in your network definition:

```
import torch.nn as nn
class Net(nn.Module):
    ...
```

5. **Grading Reference Pre-processing**: We will use a specific randomized transformation for grading that can be found in the `Autograding and Final Tests` section. Before training any model for long periods of time, you need to pay attention to the existence of such a testing pre-processing.

6. **Training Rules**: You are able to make the following decisions about your model:

- You **can** choose and change your architecture as you please.
  - You can have shallow networks, or deep ones.
  - You can customize the number of neural units in each layer and the depth of the network.
  - You are free to use convolutional, and non-convolutional layers.
  - You can employ batch normalization if you would like to.
  - You can use any type of non-linear layers as you please. `Tanh`, `Sigmoid`, and `ReLU` are some common activation functions.
  - You can use any kind of pooling layers you deem appropriate.
  - etc.

- You **can** initialize your network using any of the methods described in `https://pytorch.org/docs/stable/nn.init.html`.
  - Some common layer initializations include the Xavier (a.k.a. Glorot), and orthogonal initializations.
  - You may want to avoid initializing your network with all zeros (think about the symmetry of the neural units, and how identical initialization may be a bad idea considering what happens during training).
- You **can** use and customize any kind of optimization methods you deem appropriate.
  - You can use any first order stochastic methods (i.e., Stochastic Gradient Descent variants) such as Vanilla SGD, Adam, RMSProp, Adagrad, etc.
  - You are also welcome to use second order optimization methods such as newton and quasi-newton methods. However, it may be expensive and difficult to make them work for this setting.
  - Zeroth order methods (i.e., Black Box methods) are also okay (although you may not find them very effective in this setting).
  - You can specify any learning rates first order stochastic methods. In fact, you can even customize your learning rate schedules.
  - You are free to use any mini-batch sizes for stochastic gradient computation.
  - etc.
- You **can** use any kind of loss function you deem effective.
  - You can add any kind of regularization to your loss.
  - You can pick any kind of classification loss functions such as the cross-entropy and the mean squared loss.
- You **cannot** warm-start your network (i.e., you **cannot** use a pre-trained network).
- You **may** use any kind of image pre-processing and transformations during training. However, for the same transformations to persist at grading time, you may need to apply such transformations within the neural network's `forward` function definition.
  - In other words, we will drop any `DataLoader` or transformations that your network may rely on to have good performance, and we will only load and use your neural network for grading.

3 1. Object Classification Using the CIFAR Data

3.1 1.1 Loading the Data

[4]:
```
message = 'You can implement the pre-processing transformations, data sets,␣
 ↪data loaders, etc. in this cell. \n'
message = message + '**Important Note**: Read the "Grading Reference␣
 ↪Pre-processing" bullet above, and look at the'
message = message + ' test pre-processing transformations in the "Autograding␣
 ↪and Final Tests" section before'
message = message + ' training models for long periods of time.'
print(message)
transformation_list = [transforms.RandomAffine(degrees=30, translate=(0.01, 0.
 ↪01), scale=(0.9, 1.1),
                                                shear=None, resample=0,␣
 ↪fillcolor=0),
```

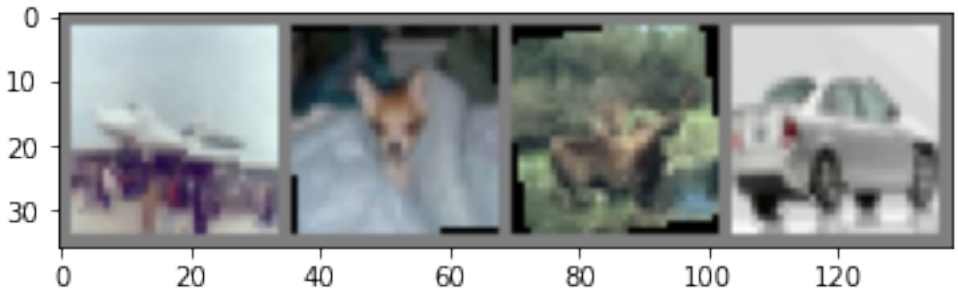
```
                        transforms.ToTensor(),
                        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]

test_pre_tranformation = transforms.Compose(transformation_list)
batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='/home/jovyan/work/course-lib/
 ↪data_cifar', train=True,
                                        download=False,␣
 ↪transform=test_pre_tranformation)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='/home/jovyan/work/course-lib/
 ↪data_cifar', train=False,
                                       download=False,␣
 ↪transform=test_pre_tranformation)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

You can implement the pre-processing transformations, data sets, data loaders, etc. in this cell.
Important Note: Read the "Grading Reference Pre-processing" bullet above, and look at the test pre-processing transformations in the "Autograding and Final Tests" section before training models for long periods of time.

[5]:
```
message = 'You can visualize some of the pre-processed images here (This is␣
 ↪optional and only for your own reference).'
print(message)

def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
```

```
print(' '.join('%5s' % classes[labels[j]] for j in range(batch_size)))
```

You can visualize some of the pre-processed images here (This is optional and only for your own reference).




```
plane   dog  deer   car
```

4 1.2 Defining the Model

Important Note: As mentioned above, make sure you name the neural module class as `Net`. In other words, you are supposed to have the following lines somewhere in your network definition:

```
import torch.nn as nn
class Net(nn.Module):
   ...
```

[6]:
```
message = 'You can define the neural architecture and instantiate it in this␣
 ↪cell.'
print(message)


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 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))
        x = self.fc3(x)
        return x
```

```
You can define the neural architecture and instantiate it in this cell.
```

5 1.3 Initializing the Neural Model

It may be a better idea to fully control the initialization process of the neural weights rather than leaving it to the default procedure chosen by pytorch.

Here is pytorch's documentation about different initialization methods: https://pytorch.org/docs/stable/nn.init.html

Some common layer initializations include the Xavier (a.k.a. Glorot), and orthogonal initializations.

[7]:
```
message = 'You can initialize the neural weights here, and not leave it to the␣
 ↪library default (this is optional).'
print(message)

net = Net()
```

```
You can initialize the neural weights here, and not leave it to the library
default (this is optional).
```

6 1.4 Defining The Loss Function and The Optimizer

[8]:
```
message = 'You can define the loss function and the optimizer of interest here.'
print(message)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.0003, momentum=0.9)
```

```
You can define the loss function and the optimizer of interest here.
```

7 1.5 Training the Model

Important Note: In order for the autograder not to time-out due to training during grading, please make sure you wrap your training code within the following conditional statement:

```
if perform_computation:
   # Place any computationally intensive training/optimization code here
```

[9]:
```
if perform_computation:
    message = 'You can define the training loop and forward-backward␣
 ↪propagation here.'
    print(message)
```

```
    for epoch in range(25):
      running_loss = 0.0
      for i, data in enumerate(trainloader, 0):
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:
          print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
          running_loss = 0.0
```

```
You can define the training loop and forward-backward propagation here.
[1,  2000] loss: 2.302
[1,  4000] loss: 2.280
[1,  6000] loss: 2.088
[1,  8000] loss: 1.940
[1, 10000] loss: 1.806
[1, 12000] loss: 1.760
[2,  2000] loss: 1.678
[2,  4000] loss: 1.628
[2,  6000] loss: 1.614
[2,  8000] loss: 1.591
[2, 10000] loss: 1.545
[2, 12000] loss: 1.549
[3,  2000] loss: 1.507
[3,  4000] loss: 1.482
[3,  6000] loss: 1.457
[3,  8000] loss: 1.459
[3, 10000] loss: 1.447
[3, 12000] loss: 1.421
[4,  2000] loss: 1.387
[4,  4000] loss: 1.394
[4,  6000] loss: 1.411
[4,  8000] loss: 1.384
[4, 10000] loss: 1.370
[4, 12000] loss: 1.349
[5,  2000] loss: 1.335
```

```
[5,  4000] loss: 1.346
[5,  6000] loss: 1.321
[5,  8000] loss: 1.335
[5, 10000] loss: 1.307
[5, 12000] loss: 1.335
[6,  2000] loss: 1.286
[6,  4000] loss: 1.303
[6,  6000] loss: 1.278
[6,  8000] loss: 1.286
[6, 10000] loss: 1.278
[6, 12000] loss: 1.284
[7,  2000] loss: 1.273
[7,  4000] loss: 1.245
[7,  6000] loss: 1.243
[7,  8000] loss: 1.242
[7, 10000] loss: 1.251
[7, 12000] loss: 1.258
[8,  2000] loss: 1.249
[8,  4000] loss: 1.230
[8,  6000] loss: 1.212
[8,  8000] loss: 1.240
[8, 10000] loss: 1.206
[8, 12000] loss: 1.213
[9,  2000] loss: 1.209
[9,  4000] loss: 1.210
[9,  6000] loss: 1.210
[9,  8000] loss: 1.187
[9, 10000] loss: 1.197
[9, 12000] loss: 1.169
[10,  2000] loss: 1.186
[10,  4000] loss: 1.167
[10,  6000] loss: 1.198
[10,  8000] loss: 1.171
[10, 10000] loss: 1.172
[10, 12000] loss: 1.170
[11,  2000] loss: 1.126
[11,  4000] loss: 1.152
[11,  6000] loss: 1.158
[11,  8000] loss: 1.144
[11, 10000] loss: 1.163
[11, 12000] loss: 1.158
[12,  2000] loss: 1.145
[12,  4000] loss: 1.126
[12,  6000] loss: 1.135
[12,  8000] loss: 1.135
[12, 10000] loss: 1.127
[12, 12000] loss: 1.148
[13,  2000] loss: 1.136
```

```
[13,  4000] loss: 1.131
[13,  6000] loss: 1.128
[13,  8000] loss: 1.113
[13, 10000] loss: 1.090
[13, 12000] loss: 1.108
[14,  2000] loss: 1.092
[14,  4000] loss: 1.086
[14,  6000] loss: 1.102
[14,  8000] loss: 1.094
[14, 10000] loss: 1.090
[14, 12000] loss: 1.101
[15,  2000] loss: 1.073
[15,  4000] loss: 1.089
[15,  6000] loss: 1.076
[15,  8000] loss: 1.081
[15, 10000] loss: 1.087
[15, 12000] loss: 1.087
[16,  2000] loss: 1.045
[16,  4000] loss: 1.071
[16,  6000] loss: 1.066
[16,  8000] loss: 1.066
[16, 10000] loss: 1.074
[16, 12000] loss: 1.092
[17,  2000] loss: 1.048
[17,  4000] loss: 1.042
[17,  6000] loss: 1.058
[17,  8000] loss: 1.077
[17, 10000] loss: 1.067
[17, 12000] loss: 1.058
[18,  2000] loss: 1.026
[18,  4000] loss: 1.051
[18,  6000] loss: 1.047
[18,  8000] loss: 1.036
[18, 10000] loss: 1.074
[18, 12000] loss: 1.052
[19,  2000] loss: 1.016
[19,  4000] loss: 1.039
[19,  6000] loss: 1.045
[19,  8000] loss: 1.041
[19, 10000] loss: 1.032
[19, 12000] loss: 1.039
[20,  2000] loss: 1.011
[20,  4000] loss: 1.021
[20,  6000] loss: 1.048
[20,  8000] loss: 0.998
[20, 10000] loss: 1.032
[20, 12000] loss: 1.035
[21,  2000] loss: 0.985
```

```
[21,  4000] loss: 1.043
[21,  6000] loss: 1.000
[21,  8000] loss: 1.014
[21, 10000] loss: 1.018
[21, 12000] loss: 1.010
[22,  2000] loss: 0.973
[22,  4000] loss: 1.010
[22,  6000] loss: 0.989
[22,  8000] loss: 1.037
[22, 10000] loss: 1.029
[22, 12000] loss: 1.009
[23,  2000] loss: 0.993
[23,  4000] loss: 0.985
[23,  6000] loss: 0.993
[23,  8000] loss: 1.012
[23, 10000] loss: 0.980
[23, 12000] loss: 1.004
[24,  2000] loss: 0.997
[24,  4000] loss: 0.987
[24,  6000] loss: 1.015
[24,  8000] loss: 0.992
[24, 10000] loss: 0.984
[24, 12000] loss: 0.981
[25,  2000] loss: 0.982
[25,  4000] loss: 0.968
[25,  6000] loss: 0.982
[25,  8000] loss: 0.985
[25, 10000] loss: 0.987
[25, 12000] loss: 0.991
```

8 1.6 Storing the Model

Important Note: In order for the autograder not to overwrite your model with empty (untrained) model, please make sure you wrap your code within the following conditional statement:

```
if perform_computation:
   # Save your trained model here
```

[10]:
```
message = 'Here you should store the model at "./cifar_net.pth" .'
print(message)

if perform_computation:
    # your code here
    PATH = './cifar_net.pth'
    torch.save(net.state_dict(), PATH)
```

```
Here you should store the model at "./cifar_net.pth" .
```

9 1.7 Evaluating the Trained Model

[11]:
```
message = 'Here you can visualize a bunch of examples and print the prediction␣
 ↪of the trained classifier (this is optional).'
print(message)

# your code here
```

```
Here you can visualize a bunch of examples and print the prediction of the
trained classifier (this is optional).
```

[12]:
```
message = 'Here you can evaluate the overall accuracy of the trained classifier␣
 ↪(this is optional).'
print(message)

# your code here
```

```
Here you can evaluate the overall accuracy of the trained classifier (this is
optional).
```

[13]:
```
message = 'Here you can evaluate the per-class accuracy of the trained␣
 ↪classifier (this is optional).'
print(message)
```

```
Here you can evaluate the per-class accuracy of the trained classifier (this is
optional).
```

9.1 1.8 Autograding and Final Tests

[14]:
```
assert 'Net' in globals().keys(), 'The Net class was not defined earlier. ' + \
                                  'Make sure you read and follow the␣
 ↪instructions provided as Important Notes' + \
                                  '(especially, the "Model Class Naming" part).'

cifar_net_path = './cifar_net.pth'

assert os.path.exists(cifar_net_path), 'You have not stored the trained model␣
 ↪properly. '+  \
                                       'Make sure you read and follow the␣
 ↪instructions provided as Important Notes.'

assert os.path.getsize(cifar_net_path) < 1000000, 'The size of your trained␣
 ↪model exceeds 1 MB.'



if 'net' in globals():
```

```
    del net
net = Net()
net.load_state_dict(torch.load(cifar_net_path))
net = net.eval()

# Disclaimer: Most of the following code was adopted from Pytorch's␣
 ↪Documentation and Examples
# https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

transformation_list = [transforms.RandomAffine(degrees=30, translate=(0.01, 0.
 ↪01), scale=(0.9, 1.1),
                                                shear=None, resample=0,␣
 ↪fillcolor=0),
                       transforms.ToTensor(),
                       transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]

test_pre_tranformation = transforms.Compose(transformation_list)

cifar_root = '/home/jovyan/work/course-lib/data_cifar'
testset = torchvision.datasets.CIFAR10(root=cifar_root, train=False,
                                       download=False,␣
 ↪transform=test_pre_tranformation)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=1)


class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
print('----------')
print(f'Overall Testing Accuracy: {100. * sum(class_correct) /␣
 ↪sum(class_total)} %%')
```

```
Accuracy of plane : 65 %
Accuracy of   car : 70 %
Accuracy of  bird : 46 %
Accuracy of   cat : 48 %
Accuracy of  deer : 48 %
Accuracy of   dog : 48 %
Accuracy of  frog : 72 %
Accuracy of horse : 74 %
Accuracy of  ship : 75 %
Accuracy of truck : 66 %
----------
Overall Testing Accuracy: 61.64 %%
```

[15]:
```
# "Object Classification Test: Checking the accuracy on the CIFAR Images"
```

10 2. Digit Recognition Using the MNIST Data

10.1 2.1 Loading the Data

[16]:
```
message = 'You can implement the pre-processing transformations, data sets,␣
 ↪data loaders, etc. in this cell. \n'
message = message + '**Important Note**: Read the "Grading Reference␣
 ↪Pre-processing" bullet, and look at the'
message = message + ' test pre-processing transformations in the "Autograding␣
 ↪and Final Tests" section before'
message = message + ' training models for long periods of time.'
print(message)

n_epochs = 7
batch_size_train = 64
batch_size_test = 1000
learning_rate = 0.01
momentum = 0.5
log_interval = 10

random_seed = 1
torch.backends.cudnn.enabled = False
torch.manual_seed(random_seed)
transformation_list = [transforms.RandomAffine(degrees=60, translate=(0.2, 0.
 ↪2), scale=(0.5, 2.),
                                               shear=None, resample=0,␣
 ↪fillcolor=0),
                       transforms.ToTensor(),
                       transforms.Normalize((0.5,), (0.5,))]
test_pre_tranformation = transforms.Compose(transformation_list)
```

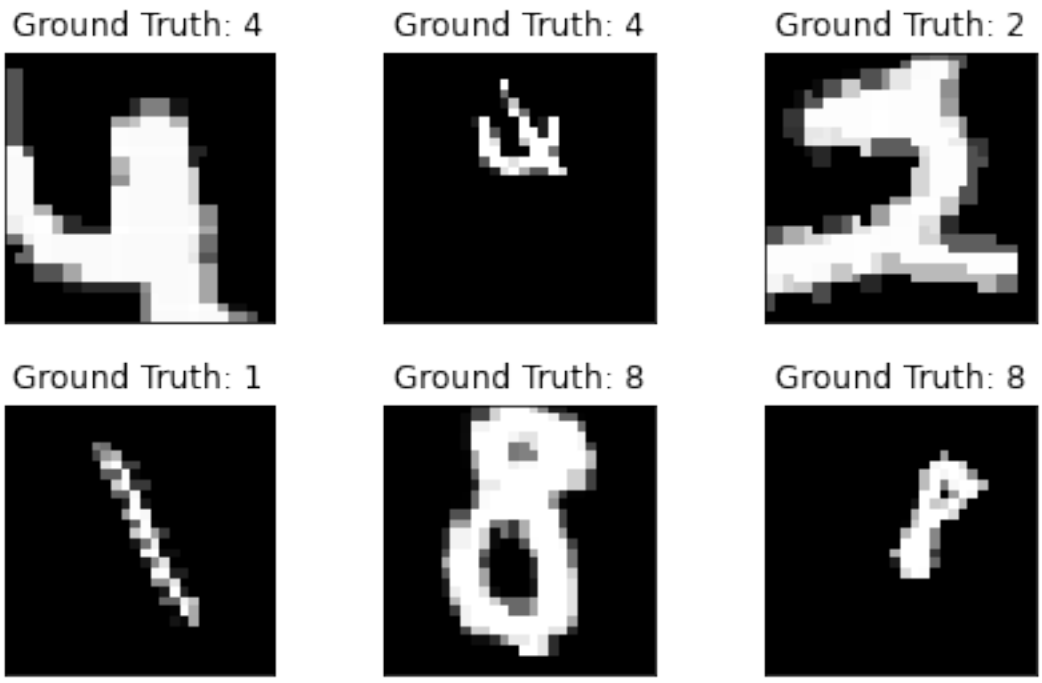
```
trainset = torchvision.datasets.MNIST(root='/home/jovyan/work/course-lib/
 ↪data_mnist', train=True,
                                       download=False,␣
 ↪transform=test_pre_tranformation)
train_loader = torch.utils.data.DataLoader(trainset,␣
 ↪batch_size=batch_size_train, shuffle=True, num_workers=2)
testset = torchvision.datasets.MNIST(root='/home/jovyan/work/course-lib/
 ↪data_mnist', train=False,
                                      download=False,␣
 ↪transform=test_pre_tranformation)
test_loader = torch.utils.data.DataLoader(testset, batch_size=batch_size_test,␣
 ↪shuffle=True, num_workers=2)
```

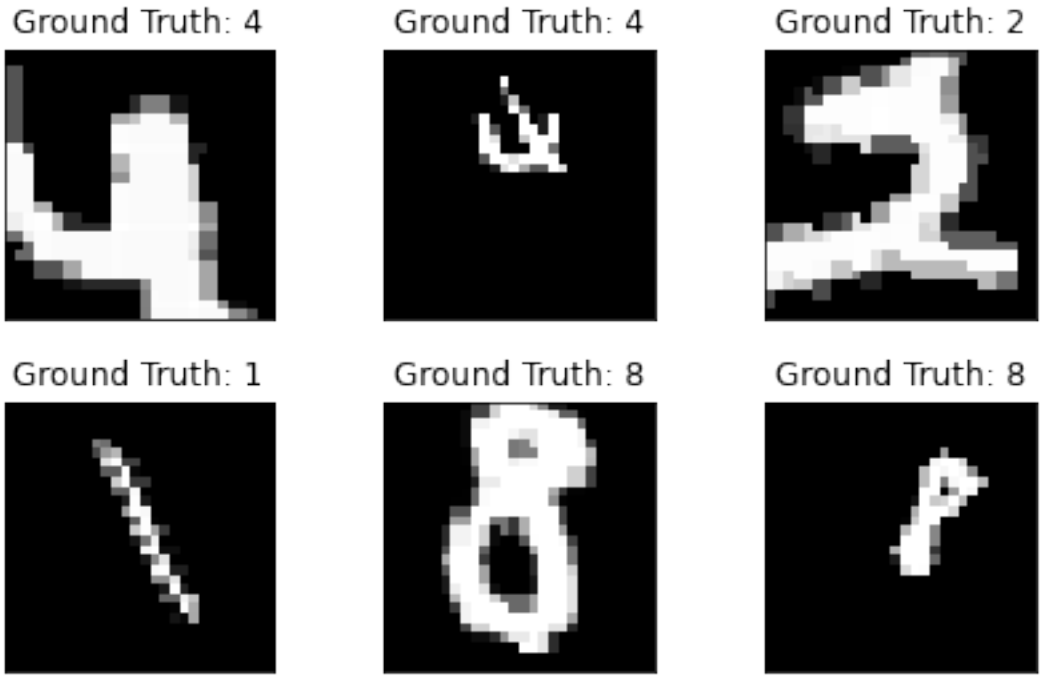
You can implement the pre-processing transformations, data sets, data loaders, etc. in this cell.
Important Note: Read the "Grading Reference Pre-processing" bullet, and look at the test pre-processing transformations in the "Autograding and Final Tests" section before training models for long periods of time.

[17]:
```
message = 'You can visualize some of the pre-processed images here (This is␣
 ↪optional and only for your own reference).'
print(message)

examples = enumerate(test_loader)
batch_idx, (example_data, example_targets) = next(examples)
example_data.shape
fig = plt.figure()
for i in range(6):
  plt.subplot(2,3,i+1)
  plt.tight_layout()
  plt.imshow(example_data[i][0], cmap='gray', interpolation='none')
  plt.title("Ground Truth: {}".format(example_targets[i]))
  plt.xticks([])
  plt.yticks([])
fig
```

You can visualize some of the pre-processed images here (This is optional and only for your own reference).

[17]:


Ground Truth: 4
Ground Truth: 4
Ground Truth: 2
Ground Truth: 1
Ground Truth: 8
Ground Truth: 8


Ground Truth: 4
Ground Truth: 4
Ground Truth: 2
Ground Truth: 1
Ground Truth: 8
Ground Truth: 8

11 2.2 Defining the Model

Important Note: As mentioned above, make sure you name the neural module class as Net. In other words, you are supposed to have the following lines somewhere in your network definition:

```
import torch.nn as nn
class Net(nn.Module):
    ...
```

[18]:
```
message = 'You can define the neural architecture and instantiate it in this␣
 ↪cell.'
print(message)


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)
```

```
You can define the neural architecture and instantiate it in this cell.
```

12 2.3 Initializing the Neural Model

It may be a better idea to fully control the initialization process of the neural weights rather than leaving it to the default procedure chosen by pytorch.

Here is pytorch's documentation about different initialization methods: https://pytorch.org/docs/stable/nn.init.html

Some common layer initializations include the Xavier (a.k.a. Glorot), and orthogonal initializations.

[19]:
```
message = 'You can initialize the neural weights here, and not leave it to the␣
 ↪library default (this is optional).'
print(message)
```

```
network = Net()
```

You can initialize the neural weights here, and not leave it to the library default (this is optional).

13 2.4 Defining The Loss Function and The Optimizer

[20]:
```
message = 'You can define the loss function and the optimizer of interest here.'
print(message)

optimizer = optim.SGD(network.parameters(), lr=learning_rate, momentum=momentum)
train_losses = []
train_counter = []
test_losses = []
test_counter = [i*len(train_loader.dataset) for i in range(n_epochs + 1)]
classes = ('0', '1', '2', '3',
           '4', '5', '6', '7', '8', '9')

def train(epoch):
  network.train()
  for batch_idx, (data, target) in enumerate(train_loader):
    optimizer.zero_grad()
    output = network(data)
    loss = F.nll_loss(output, target)
    loss.backward()
    optimizer.step()
    if batch_idx % log_interval == 0:
      print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
        epoch, batch_idx * len(data), len(train_loader.dataset),
        100. * batch_idx / len(train_loader), loss.item()))
      train_losses.append(loss.item())
      train_counter.append(
        (batch_idx*64) + ((epoch-1)*len(train_loader.dataset)))
      torch.save(network.state_dict(), './mnist_net.pth')
      torch.save(optimizer.state_dict(), './optimizer.pth')


def test():
  network.eval()
  test_loss = 0
  correct = 0
  with torch.no_grad():
    for data, target in test_loader:
      output = network(data)
      test_loss += F.nll_loss(output, target, size_average=False).item()
```

```
      pred = output.data.max(1, keepdim=True)[1]
      correct += pred.eq(target.data.view_as(pred)).sum()
  test_loss /= len(test_loader.dataset)
  test_losses.append(test_loss)
  print('\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
    test_loss, correct, len(test_loader.dataset),
    100. * correct / len(test_loader.dataset)))
```

You can define the loss function and the optimizer of interest here.

14 2.5 Training the Model

Important Note: In order for the autograder not to time-out due to training during grading, please make sure you wrap your training code within the following conditional statement:

```
if perform_computation:
   # Place any computationally intensive training/optimization code here
```

[21]:
```
if perform_computation:
    message = 'You can define the training loop and forward-backward␣
 ↪propagation here.'
    print(message)
    test()
    for epoch in range(1, n_epochs + 1):

        train(epoch)
        test()
```

```
You can define the training loop and forward-backward propagation here.

Test set: Avg. loss: 2.3048, Accuracy: 1040/10000 (10%)

Train Epoch: 1 [0/60000 (0%)]   Loss: 2.340464
Train Epoch: 1 [640/60000 (1%)] Loss: 2.313694
Train Epoch: 1 [1280/60000 (2%)]        Loss: 2.309328
Train Epoch: 1 [1920/60000 (3%)]        Loss: 2.300640
Train Epoch: 1 [2560/60000 (4%)]        Loss: 2.313391
Train Epoch: 1 [3200/60000 (5%)]        Loss: 2.303011
Train Epoch: 1 [3840/60000 (6%)]        Loss: 2.305618
Train Epoch: 1 [4480/60000 (7%)]        Loss: 2.300459
Train Epoch: 1 [5120/60000 (9%)]        Loss: 2.299018
Train Epoch: 1 [5760/60000 (10%)]       Loss: 2.297807
Train Epoch: 1 [6400/60000 (11%)]       Loss: 2.286046
Train Epoch: 1 [7040/60000 (12%)]       Loss: 2.316808
```

```
Train Epoch: 1 [7680/60000 (13%)]       Loss: 2.307629
Train Epoch: 1 [8320/60000 (14%)]       Loss: 2.286206
Train Epoch: 1 [8960/60000 (15%)]       Loss: 2.305111
Train Epoch: 1 [9600/60000 (16%)]       Loss: 2.305413
Train Epoch: 1 [10240/60000 (17%)]      Loss: 2.311998
Train Epoch: 1 [10880/60000 (18%)]      Loss: 2.306390
Train Epoch: 1 [11520/60000 (19%)]      Loss: 2.309137
Train Epoch: 1 [12160/60000 (20%)]      Loss: 2.307968
Train Epoch: 1 [12800/60000 (21%)]      Loss: 2.298977
Train Epoch: 1 [13440/60000 (22%)]      Loss: 2.295154
Train Epoch: 1 [14080/60000 (23%)]      Loss: 2.301936
Train Epoch: 1 [14720/60000 (25%)]      Loss: 2.305635
Train Epoch: 1 [15360/60000 (26%)]      Loss: 2.295557
Train Epoch: 1 [16000/60000 (27%)]      Loss: 2.297378
Train Epoch: 1 [16640/60000 (28%)]      Loss: 2.306149
Train Epoch: 1 [17280/60000 (29%)]      Loss: 2.285348
Train Epoch: 1 [17920/60000 (30%)]      Loss: 2.290962
Train Epoch: 1 [18560/60000 (31%)]      Loss: 2.299951
Train Epoch: 1 [19200/60000 (32%)]      Loss: 2.279544
Train Epoch: 1 [19840/60000 (33%)]      Loss: 2.306108
Train Epoch: 1 [20480/60000 (34%)]      Loss: 2.290363
Train Epoch: 1 [21120/60000 (35%)]      Loss: 2.290499
Train Epoch: 1 [21760/60000 (36%)]      Loss: 2.288279
Train Epoch: 1 [22400/60000 (37%)]      Loss: 2.273837
Train Epoch: 1 [23040/60000 (38%)]      Loss: 2.296584
Train Epoch: 1 [23680/60000 (39%)]      Loss: 2.284195
Train Epoch: 1 [24320/60000 (41%)]      Loss: 2.296991
Train Epoch: 1 [24960/60000 (42%)]      Loss: 2.281980
Train Epoch: 1 [25600/60000 (43%)]      Loss: 2.276244
Train Epoch: 1 [26240/60000 (44%)]      Loss: 2.277400
Train Epoch: 1 [26880/60000 (45%)]      Loss: 2.272495
Train Epoch: 1 [27520/60000 (46%)]      Loss: 2.290365
Train Epoch: 1 [28160/60000 (47%)]      Loss: 2.290165
Train Epoch: 1 [28800/60000 (48%)]      Loss: 2.281870
Train Epoch: 1 [29440/60000 (49%)]      Loss: 2.297097
Train Epoch: 1 [30080/60000 (50%)]      Loss: 2.292461
Train Epoch: 1 [30720/60000 (51%)]      Loss: 2.283561
Train Epoch: 1 [31360/60000 (52%)]      Loss: 2.267139
Train Epoch: 1 [32000/60000 (53%)]      Loss: 2.260638
Train Epoch: 1 [32640/60000 (54%)]      Loss: 2.292452
Train Epoch: 1 [33280/60000 (55%)]      Loss: 2.276723
Train Epoch: 1 [33920/60000 (57%)]      Loss: 2.287887
Train Epoch: 1 [34560/60000 (58%)]      Loss: 2.272947
Train Epoch: 1 [35200/60000 (59%)]      Loss: 2.307111
Train Epoch: 1 [35840/60000 (60%)]      Loss: 2.294042
Train Epoch: 1 [36480/60000 (61%)]      Loss: 2.292684
Train Epoch: 1 [37120/60000 (62%)]      Loss: 2.258988
Train Epoch: 1 [37760/60000 (63%)]      Loss: 2.294055
```

```
Train Epoch: 1 [38400/60000 (64%)]      Loss: 2.231907
Train Epoch: 1 [39040/60000 (65%)]      Loss: 2.269340
Train Epoch: 1 [39680/60000 (66%)]      Loss: 2.230595
Train Epoch: 1 [40320/60000 (67%)]      Loss: 2.248391
Train Epoch: 1 [40960/60000 (68%)]      Loss: 2.197743
Train Epoch: 1 [41600/60000 (69%)]      Loss: 2.267940
Train Epoch: 1 [42240/60000 (70%)]      Loss: 2.244492
Train Epoch: 1 [42880/60000 (71%)]      Loss: 2.251605
Train Epoch: 1 [43520/60000 (72%)]      Loss: 2.269162
Train Epoch: 1 [44160/60000 (74%)]      Loss: 2.213498
Train Epoch: 1 [44800/60000 (75%)]      Loss: 2.266531
Train Epoch: 1 [45440/60000 (76%)]      Loss: 2.193966
Train Epoch: 1 [46080/60000 (77%)]      Loss: 2.258895
Train Epoch: 1 [46720/60000 (78%)]      Loss: 2.250482
Train Epoch: 1 [47360/60000 (79%)]      Loss: 2.238697
Train Epoch: 1 [48000/60000 (80%)]      Loss: 2.287396
Train Epoch: 1 [48640/60000 (81%)]      Loss: 2.207901
Train Epoch: 1 [49280/60000 (82%)]      Loss: 2.298186
Train Epoch: 1 [49920/60000 (83%)]      Loss: 2.281384
Train Epoch: 1 [50560/60000 (84%)]      Loss: 2.293426
Train Epoch: 1 [51200/60000 (85%)]      Loss: 2.223303
Train Epoch: 1 [51840/60000 (86%)]      Loss: 2.227564
Train Epoch: 1 [52480/60000 (87%)]      Loss: 2.244966
Train Epoch: 1 [53120/60000 (88%)]      Loss: 2.223181
Train Epoch: 1 [53760/60000 (90%)]      Loss: 2.195858
Train Epoch: 1 [54400/60000 (91%)]      Loss: 2.210400
Train Epoch: 1 [55040/60000 (92%)]      Loss: 2.175919
Train Epoch: 1 [55680/60000 (93%)]      Loss: 2.260529
Train Epoch: 1 [56320/60000 (94%)]      Loss: 2.224595
Train Epoch: 1 [56960/60000 (95%)]      Loss: 2.246071
Train Epoch: 1 [57600/60000 (96%)]      Loss: 2.280429
Train Epoch: 1 [58240/60000 (97%)]      Loss: 2.258635
Train Epoch: 1 [58880/60000 (98%)]      Loss: 2.271589
Train Epoch: 1 [59520/60000 (99%)]      Loss: 2.242103

Test set: Avg. loss: 2.1743, Accuracy: 2413/10000 (24%)

Train Epoch: 2 [0/60000 (0%)]   Loss: 2.236088
Train Epoch: 2 [640/60000 (1%)] Loss: 2.275695
Train Epoch: 2 [1280/60000 (2%)]        Loss: 2.181202
Train Epoch: 2 [1920/60000 (3%)]        Loss: 2.139921
Train Epoch: 2 [2560/60000 (4%)]        Loss: 2.193906
Train Epoch: 2 [3200/60000 (5%)]        Loss: 2.170912
Train Epoch: 2 [3840/60000 (6%)]        Loss: 2.171501
Train Epoch: 2 [4480/60000 (7%)]        Loss: 2.207087
Train Epoch: 2 [5120/60000 (9%)]        Loss: 2.240713
Train Epoch: 2 [5760/60000 (10%)]       Loss: 2.212960
Train Epoch: 2 [6400/60000 (11%)]       Loss: 2.208942
```

```
Train Epoch: 2 [7040/60000 (12%)]       Loss: 2.208517
Train Epoch: 2 [7680/60000 (13%)]       Loss: 2.182835
Train Epoch: 2 [8320/60000 (14%)]       Loss: 2.170204
Train Epoch: 2 [8960/60000 (15%)]       Loss: 2.110485
Train Epoch: 2 [9600/60000 (16%)]       Loss: 2.212475
Train Epoch: 2 [10240/60000 (17%)]      Loss: 2.154203
Train Epoch: 2 [10880/60000 (18%)]      Loss: 2.187783
Train Epoch: 2 [11520/60000 (19%)]      Loss: 2.135276
Train Epoch: 2 [12160/60000 (20%)]      Loss: 2.208395
Train Epoch: 2 [12800/60000 (21%)]      Loss: 2.240384
Train Epoch: 2 [13440/60000 (22%)]      Loss: 2.175734
Train Epoch: 2 [14080/60000 (23%)]      Loss: 2.105631
Train Epoch: 2 [14720/60000 (25%)]      Loss: 2.167714
Train Epoch: 2 [15360/60000 (26%)]      Loss: 2.171093
Train Epoch: 2 [16000/60000 (27%)]      Loss: 2.075268
Train Epoch: 2 [16640/60000 (28%)]      Loss: 2.084757
Train Epoch: 2 [17280/60000 (29%)]      Loss: 2.150020
Train Epoch: 2 [17920/60000 (30%)]      Loss: 2.182360
Train Epoch: 2 [18560/60000 (31%)]      Loss: 2.162079
Train Epoch: 2 [19200/60000 (32%)]      Loss: 2.117469
Train Epoch: 2 [19840/60000 (33%)]      Loss: 2.032832
Train Epoch: 2 [20480/60000 (34%)]      Loss: 2.016612
Train Epoch: 2 [21120/60000 (35%)]      Loss: 2.173625
Train Epoch: 2 [21760/60000 (36%)]      Loss: 2.173103
Train Epoch: 2 [22400/60000 (37%)]      Loss: 2.111854
Train Epoch: 2 [23040/60000 (38%)]      Loss: 2.113996
Train Epoch: 2 [23680/60000 (39%)]      Loss: 2.096831
Train Epoch: 2 [24320/60000 (41%)]      Loss: 2.076144
Train Epoch: 2 [24960/60000 (42%)]      Loss: 2.125189
Train Epoch: 2 [25600/60000 (43%)]      Loss: 2.215479
Train Epoch: 2 [26240/60000 (44%)]      Loss: 2.128596
Train Epoch: 2 [26880/60000 (45%)]      Loss: 2.255390
Train Epoch: 2 [27520/60000 (46%)]      Loss: 2.218870
Train Epoch: 2 [28160/60000 (47%)]      Loss: 2.039286
Train Epoch: 2 [28800/60000 (48%)]      Loss: 2.182580
Train Epoch: 2 [29440/60000 (49%)]      Loss: 2.218952
Train Epoch: 2 [30080/60000 (50%)]      Loss: 2.173742
Train Epoch: 2 [30720/60000 (51%)]      Loss: 2.046681
Train Epoch: 2 [31360/60000 (52%)]      Loss: 2.106340
Train Epoch: 2 [32000/60000 (53%)]      Loss: 2.130632
Train Epoch: 2 [32640/60000 (54%)]      Loss: 2.166447
Train Epoch: 2 [33280/60000 (55%)]      Loss: 2.118722
Train Epoch: 2 [33920/60000 (57%)]      Loss: 2.177624
Train Epoch: 2 [34560/60000 (58%)]      Loss: 2.157123
Train Epoch: 2 [35200/60000 (59%)]      Loss: 2.085031
Train Epoch: 2 [35840/60000 (60%)]      Loss: 2.130616
Train Epoch: 2 [36480/60000 (61%)]      Loss: 2.074472
Train Epoch: 2 [37120/60000 (62%)]      Loss: 1.998199
```

```
Train Epoch: 2 [37760/60000 (63%)]      Loss: 2.127239
Train Epoch: 2 [38400/60000 (64%)]      Loss: 2.114364
Train Epoch: 2 [39040/60000 (65%)]      Loss: 2.005909
Train Epoch: 2 [39680/60000 (66%)]      Loss: 2.032242
Train Epoch: 2 [40320/60000 (67%)]      Loss: 2.115446
Train Epoch: 2 [40960/60000 (68%)]      Loss: 2.038192
Train Epoch: 2 [41600/60000 (69%)]      Loss: 2.115020
Train Epoch: 2 [42240/60000 (70%)]      Loss: 2.101193
Train Epoch: 2 [42880/60000 (71%)]      Loss: 2.183818
Train Epoch: 2 [43520/60000 (72%)]      Loss: 2.092688
Train Epoch: 2 [44160/60000 (74%)]      Loss: 2.229033
Train Epoch: 2 [44800/60000 (75%)]      Loss: 2.071849
Train Epoch: 2 [45440/60000 (76%)]      Loss: 2.182327
Train Epoch: 2 [46080/60000 (77%)]      Loss: 2.075303
Train Epoch: 2 [46720/60000 (78%)]      Loss: 2.255790
Train Epoch: 2 [47360/60000 (79%)]      Loss: 2.046139
Train Epoch: 2 [48000/60000 (80%)]      Loss: 2.134438
Train Epoch: 2 [48640/60000 (81%)]      Loss: 2.047319
Train Epoch: 2 [49280/60000 (82%)]      Loss: 2.026389
Train Epoch: 2 [49920/60000 (83%)]      Loss: 2.031892
Train Epoch: 2 [50560/60000 (84%)]      Loss: 2.078555
Train Epoch: 2 [51200/60000 (85%)]      Loss: 1.919534
Train Epoch: 2 [51840/60000 (86%)]      Loss: 1.970591
Train Epoch: 2 [52480/60000 (87%)]      Loss: 2.037223
Train Epoch: 2 [53120/60000 (88%)]      Loss: 2.010153
Train Epoch: 2 [53760/60000 (90%)]      Loss: 2.198880
Train Epoch: 2 [54400/60000 (91%)]      Loss: 1.923458
Train Epoch: 2 [55040/60000 (92%)]      Loss: 2.066138
Train Epoch: 2 [55680/60000 (93%)]      Loss: 2.111316
Train Epoch: 2 [56320/60000 (94%)]      Loss: 1.996965
Train Epoch: 2 [56960/60000 (95%)]      Loss: 1.942556
Train Epoch: 2 [57600/60000 (96%)]      Loss: 2.096126
Train Epoch: 2 [58240/60000 (97%)]      Loss: 2.051365
Train Epoch: 2 [58880/60000 (98%)]      Loss: 1.885957
Train Epoch: 2 [59520/60000 (99%)]      Loss: 2.062863

Test set: Avg. loss: 1.8958, Accuracy: 3580/10000 (36%)

Train Epoch: 3 [0/60000 (0%)]   Loss: 1.889464
Train Epoch: 3 [640/60000 (1%)] Loss: 2.012615
Train Epoch: 3 [1280/60000 (2%)]        Loss: 2.090328
Train Epoch: 3 [1920/60000 (3%)]        Loss: 2.063469
Train Epoch: 3 [2560/60000 (4%)]        Loss: 2.209372
Train Epoch: 3 [3200/60000 (5%)]        Loss: 2.033462
Train Epoch: 3 [3840/60000 (6%)]        Loss: 1.977486
Train Epoch: 3 [4480/60000 (7%)]        Loss: 1.757457
Train Epoch: 3 [5120/60000 (9%)]        Loss: 1.958113
Train Epoch: 3 [5760/60000 (10%)]       Loss: 2.079346
```

```
Train Epoch: 3 [6400/60000 (11%)]       Loss: 1.972608
Train Epoch: 3 [7040/60000 (12%)]       Loss: 2.004066
Train Epoch: 3 [7680/60000 (13%)]       Loss: 1.913987
Train Epoch: 3 [8320/60000 (14%)]       Loss: 1.986359
Train Epoch: 3 [8960/60000 (15%)]       Loss: 1.888766
Train Epoch: 3 [9600/60000 (16%)]       Loss: 1.909665
Train Epoch: 3 [10240/60000 (17%)]      Loss: 1.821487
Train Epoch: 3 [10880/60000 (18%)]      Loss: 2.018204
Train Epoch: 3 [11520/60000 (19%)]      Loss: 1.980446
Train Epoch: 3 [12160/60000 (20%)]      Loss: 2.149694
Train Epoch: 3 [12800/60000 (21%)]      Loss: 2.007029
Train Epoch: 3 [13440/60000 (22%)]      Loss: 2.124921
Train Epoch: 3 [14080/60000 (23%)]      Loss: 1.959986
Train Epoch: 3 [14720/60000 (25%)]      Loss: 1.976332
Train Epoch: 3 [15360/60000 (26%)]      Loss: 1.919504
Train Epoch: 3 [16000/60000 (27%)]      Loss: 1.897773
Train Epoch: 3 [16640/60000 (28%)]      Loss: 1.948774
Train Epoch: 3 [17280/60000 (29%)]      Loss: 2.052310
Train Epoch: 3 [17920/60000 (30%)]      Loss: 1.945293
Train Epoch: 3 [18560/60000 (31%)]      Loss: 2.019271
Train Epoch: 3 [19200/60000 (32%)]      Loss: 2.077594
Train Epoch: 3 [19840/60000 (33%)]      Loss: 1.990933
Train Epoch: 3 [20480/60000 (34%)]      Loss: 2.083562
Train Epoch: 3 [21120/60000 (35%)]      Loss: 1.985242
Train Epoch: 3 [21760/60000 (36%)]      Loss: 2.028002
Train Epoch: 3 [22400/60000 (37%)]      Loss: 1.900296
Train Epoch: 3 [23040/60000 (38%)]      Loss: 2.211195
Train Epoch: 3 [23680/60000 (39%)]      Loss: 1.962946
Train Epoch: 3 [24320/60000 (41%)]      Loss: 1.994033
Train Epoch: 3 [24960/60000 (42%)]      Loss: 1.933843
Train Epoch: 3 [25600/60000 (43%)]      Loss: 2.010466
Train Epoch: 3 [26240/60000 (44%)]      Loss: 1.958573
Train Epoch: 3 [26880/60000 (45%)]      Loss: 1.768713
Train Epoch: 3 [27520/60000 (46%)]      Loss: 1.729929
Train Epoch: 3 [28160/60000 (47%)]      Loss: 2.012455
Train Epoch: 3 [28800/60000 (48%)]      Loss: 1.945344
Train Epoch: 3 [29440/60000 (49%)]      Loss: 1.813397
Train Epoch: 3 [30080/60000 (50%)]      Loss: 1.964776
Train Epoch: 3 [30720/60000 (51%)]      Loss: 1.738745
Train Epoch: 3 [31360/60000 (52%)]      Loss: 2.064827
Train Epoch: 3 [32000/60000 (53%)]      Loss: 1.841727
Train Epoch: 3 [32640/60000 (54%)]      Loss: 1.924854
Train Epoch: 3 [33280/60000 (55%)]      Loss: 1.827437
Train Epoch: 3 [33920/60000 (57%)]      Loss: 1.897057
Train Epoch: 3 [34560/60000 (58%)]      Loss: 1.841273
Train Epoch: 3 [35200/60000 (59%)]      Loss: 1.785028
Train Epoch: 3 [35840/60000 (60%)]      Loss: 2.061403
Train Epoch: 3 [36480/60000 (61%)]      Loss: 1.888680
```

```
Train Epoch: 3 [37120/60000 (62%)]      Loss: 1.824024
Train Epoch: 3 [37760/60000 (63%)]      Loss: 1.912232
Train Epoch: 3 [38400/60000 (64%)]      Loss: 1.954671
Train Epoch: 3 [39040/60000 (65%)]      Loss: 1.957512
Train Epoch: 3 [39680/60000 (66%)]      Loss: 1.851861
Train Epoch: 3 [40320/60000 (67%)]      Loss: 1.949636
Train Epoch: 3 [40960/60000 (68%)]      Loss: 1.891635
Train Epoch: 3 [41600/60000 (69%)]      Loss: 1.829808
Train Epoch: 3 [42240/60000 (70%)]      Loss: 1.965059
Train Epoch: 3 [42880/60000 (71%)]      Loss: 1.798096
Train Epoch: 3 [43520/60000 (72%)]      Loss: 1.870126
Train Epoch: 3 [44160/60000 (74%)]      Loss: 1.765537
Train Epoch: 3 [44800/60000 (75%)]      Loss: 1.985361
Train Epoch: 3 [45440/60000 (76%)]      Loss: 2.043460
Train Epoch: 3 [46080/60000 (77%)]      Loss: 1.790845
Train Epoch: 3 [46720/60000 (78%)]      Loss: 2.014141
Train Epoch: 3 [47360/60000 (79%)]      Loss: 1.923309
Train Epoch: 3 [48000/60000 (80%)]      Loss: 1.823502
Train Epoch: 3 [48640/60000 (81%)]      Loss: 2.031925
Train Epoch: 3 [49280/60000 (82%)]      Loss: 1.815407
Train Epoch: 3 [49920/60000 (83%)]      Loss: 1.735101
Train Epoch: 3 [50560/60000 (84%)]      Loss: 1.864389
Train Epoch: 3 [51200/60000 (85%)]      Loss: 1.933601
Train Epoch: 3 [51840/60000 (86%)]      Loss: 1.819651
Train Epoch: 3 [52480/60000 (87%)]      Loss: 1.770111
Train Epoch: 3 [53120/60000 (88%)]      Loss: 2.194273
Train Epoch: 3 [53760/60000 (90%)]      Loss: 1.967424
Train Epoch: 3 [54400/60000 (91%)]      Loss: 1.939107
Train Epoch: 3 [55040/60000 (92%)]      Loss: 1.815243
Train Epoch: 3 [55680/60000 (93%)]      Loss: 2.099025
Train Epoch: 3 [56320/60000 (94%)]      Loss: 1.692834
Train Epoch: 3 [56960/60000 (95%)]      Loss: 1.901822
Train Epoch: 3 [57600/60000 (96%)]      Loss: 1.708660
Train Epoch: 3 [58240/60000 (97%)]      Loss: 1.527393
Train Epoch: 3 [58880/60000 (98%)]      Loss: 1.879482
Train Epoch: 3 [59520/60000 (99%)]      Loss: 1.877594

Test set: Avg. loss: 1.6368, Accuracy: 4649/10000 (46%)

Train Epoch: 4 [0/60000 (0%)]   Loss: 1.925924
Train Epoch: 4 [640/60000 (1%)] Loss: 1.799835
Train Epoch: 4 [1280/60000 (2%)]        Loss: 1.977253
Train Epoch: 4 [1920/60000 (3%)]        Loss: 1.764698
Train Epoch: 4 [2560/60000 (4%)]        Loss: 1.887839
Train Epoch: 4 [3200/60000 (5%)]        Loss: 1.770017
Train Epoch: 4 [3840/60000 (6%)]        Loss: 1.782968
Train Epoch: 4 [4480/60000 (7%)]        Loss: 1.894584
Train Epoch: 4 [5120/60000 (9%)]        Loss: 1.674726
```

```
Train Epoch: 4 [5760/60000 (10%)]       Loss: 1.733268
Train Epoch: 4 [6400/60000 (11%)]       Loss: 1.903388
Train Epoch: 4 [7040/60000 (12%)]       Loss: 1.958595
Train Epoch: 4 [7680/60000 (13%)]       Loss: 1.967858
Train Epoch: 4 [8320/60000 (14%)]       Loss: 2.149697
Train Epoch: 4 [8960/60000 (15%)]       Loss: 1.753634
Train Epoch: 4 [9600/60000 (16%)]       Loss: 1.945041
Train Epoch: 4 [10240/60000 (17%)]      Loss: 2.003852
Train Epoch: 4 [10880/60000 (18%)]      Loss: 1.860577
Train Epoch: 4 [11520/60000 (19%)]      Loss: 1.803828
Train Epoch: 4 [12160/60000 (20%)]      Loss: 1.801020
Train Epoch: 4 [12800/60000 (21%)]      Loss: 1.750473
Train Epoch: 4 [13440/60000 (22%)]      Loss: 1.998679
Train Epoch: 4 [14080/60000 (23%)]      Loss: 1.740216
Train Epoch: 4 [14720/60000 (25%)]      Loss: 1.879247
Train Epoch: 4 [15360/60000 (26%)]      Loss: 1.792052
Train Epoch: 4 [16000/60000 (27%)]      Loss: 1.972993
Train Epoch: 4 [16640/60000 (28%)]      Loss: 1.693432
Train Epoch: 4 [17280/60000 (29%)]      Loss: 1.797307
Train Epoch: 4 [17920/60000 (30%)]      Loss: 1.762917
Train Epoch: 4 [18560/60000 (31%)]      Loss: 1.712731
Train Epoch: 4 [19200/60000 (32%)]      Loss: 1.659544
Train Epoch: 4 [19840/60000 (33%)]      Loss: 1.733150
Train Epoch: 4 [20480/60000 (34%)]      Loss: 1.794868
Train Epoch: 4 [21120/60000 (35%)]      Loss: 1.697301
Train Epoch: 4 [21760/60000 (36%)]      Loss: 1.697963
Train Epoch: 4 [22400/60000 (37%)]      Loss: 1.620830
Train Epoch: 4 [23040/60000 (38%)]      Loss: 1.818815
Train Epoch: 4 [23680/60000 (39%)]      Loss: 1.901196
Train Epoch: 4 [24320/60000 (41%)]      Loss: 1.680654
Train Epoch: 4 [24960/60000 (42%)]      Loss: 1.728107
Train Epoch: 4 [25600/60000 (43%)]      Loss: 1.766663
Train Epoch: 4 [26240/60000 (44%)]      Loss: 1.699308
Train Epoch: 4 [26880/60000 (45%)]      Loss: 1.778490
Train Epoch: 4 [27520/60000 (46%)]      Loss: 1.899015
Train Epoch: 4 [28160/60000 (47%)]      Loss: 1.805346
Train Epoch: 4 [28800/60000 (48%)]      Loss: 1.691985
Train Epoch: 4 [29440/60000 (49%)]      Loss: 1.776999
Train Epoch: 4 [30080/60000 (50%)]      Loss: 1.857860
Train Epoch: 4 [30720/60000 (51%)]      Loss: 1.782542
Train Epoch: 4 [31360/60000 (52%)]      Loss: 2.069970
Train Epoch: 4 [32000/60000 (53%)]      Loss: 1.939806
Train Epoch: 4 [32640/60000 (54%)]      Loss: 1.615477
Train Epoch: 4 [33280/60000 (55%)]      Loss: 1.705473
Train Epoch: 4 [33920/60000 (57%)]      Loss: 1.684943
Train Epoch: 4 [34560/60000 (58%)]      Loss: 1.936043
Train Epoch: 4 [35200/60000 (59%)]      Loss: 1.863819
Train Epoch: 4 [35840/60000 (60%)]      Loss: 1.703910
```

```
Train Epoch: 4 [36480/60000 (61%)]      Loss: 1.564308
Train Epoch: 4 [37120/60000 (62%)]      Loss: 1.727909
Train Epoch: 4 [37760/60000 (63%)]      Loss: 1.735572
Train Epoch: 4 [38400/60000 (64%)]      Loss: 1.653822
Train Epoch: 4 [39040/60000 (65%)]      Loss: 1.794901
Train Epoch: 4 [39680/60000 (66%)]      Loss: 1.854849
Train Epoch: 4 [40320/60000 (67%)]      Loss: 1.993804
Train Epoch: 4 [40960/60000 (68%)]      Loss: 1.768725
Train Epoch: 4 [41600/60000 (69%)]      Loss: 1.847163
Train Epoch: 4 [42240/60000 (70%)]      Loss: 1.547858
Train Epoch: 4 [42880/60000 (71%)]      Loss: 1.623012
Train Epoch: 4 [43520/60000 (72%)]      Loss: 1.906509
Train Epoch: 4 [44160/60000 (74%)]      Loss: 1.780949
Train Epoch: 4 [44800/60000 (75%)]      Loss: 1.967527
Train Epoch: 4 [45440/60000 (76%)]      Loss: 1.588181
Train Epoch: 4 [46080/60000 (77%)]      Loss: 1.726179
Train Epoch: 4 [46720/60000 (78%)]      Loss: 1.807101
Train Epoch: 4 [47360/60000 (79%)]      Loss: 1.867216
Train Epoch: 4 [48000/60000 (80%)]      Loss: 1.844177
Train Epoch: 4 [48640/60000 (81%)]      Loss: 1.688753
Train Epoch: 4 [49280/60000 (82%)]      Loss: 1.733385
Train Epoch: 4 [49920/60000 (83%)]      Loss: 1.735752
Train Epoch: 4 [50560/60000 (84%)]      Loss: 1.640139
Train Epoch: 4 [51200/60000 (85%)]      Loss: 1.686671
Train Epoch: 4 [51840/60000 (86%)]      Loss: 1.560983
Train Epoch: 4 [52480/60000 (87%)]      Loss: 1.823435
Train Epoch: 4 [53120/60000 (88%)]      Loss: 1.624515
Train Epoch: 4 [53760/60000 (90%)]      Loss: 1.682707
Train Epoch: 4 [54400/60000 (91%)]      Loss: 1.727903
Train Epoch: 4 [55040/60000 (92%)]      Loss: 1.703596
Train Epoch: 4 [55680/60000 (93%)]      Loss: 1.803922
Train Epoch: 4 [56320/60000 (94%)]      Loss: 1.939805
Train Epoch: 4 [56960/60000 (95%)]      Loss: 1.921829
Train Epoch: 4 [57600/60000 (96%)]      Loss: 1.932332
Train Epoch: 4 [58240/60000 (97%)]      Loss: 1.645238
Train Epoch: 4 [58880/60000 (98%)]      Loss: 1.819905
Train Epoch: 4 [59520/60000 (99%)]      Loss: 1.826699

Test set: Avg. loss: 1.5197, Accuracy: 5435/10000 (54%)

Train Epoch: 5 [0/60000 (0%)]   Loss: 1.726171
Train Epoch: 5 [640/60000 (1%)] Loss: 1.793507
Train Epoch: 5 [1280/60000 (2%)]        Loss: 1.710463
Train Epoch: 5 [1920/60000 (3%)]        Loss: 1.660764
Train Epoch: 5 [2560/60000 (4%)]        Loss: 1.683865
Train Epoch: 5 [3200/60000 (5%)]        Loss: 1.697552
Train Epoch: 5 [3840/60000 (6%)]        Loss: 1.761178
Train Epoch: 5 [4480/60000 (7%)]        Loss: 1.622196
```

```
Train Epoch: 5 [5120/60000 (9%)]        Loss: 1.862401
Train Epoch: 5 [5760/60000 (10%)]       Loss: 1.652623
Train Epoch: 5 [6400/60000 (11%)]       Loss: 1.737370
Train Epoch: 5 [7040/60000 (12%)]       Loss: 1.751241
Train Epoch: 5 [7680/60000 (13%)]       Loss: 1.719646
Train Epoch: 5 [8320/60000 (14%)]       Loss: 1.519190
Train Epoch: 5 [8960/60000 (15%)]       Loss: 1.640908
Train Epoch: 5 [9600/60000 (16%)]       Loss: 1.952066
Train Epoch: 5 [10240/60000 (17%)]      Loss: 1.785497
Train Epoch: 5 [10880/60000 (18%)]      Loss: 1.915481
Train Epoch: 5 [11520/60000 (19%)]      Loss: 1.627757
Train Epoch: 5 [12160/60000 (20%)]      Loss: 1.641104
Train Epoch: 5 [12800/60000 (21%)]      Loss: 1.864034
Train Epoch: 5 [13440/60000 (22%)]      Loss: 1.738170
Train Epoch: 5 [14080/60000 (23%)]      Loss: 1.900038
Train Epoch: 5 [14720/60000 (25%)]      Loss: 1.539678
Train Epoch: 5 [15360/60000 (26%)]      Loss: 1.517381
Train Epoch: 5 [16000/60000 (27%)]      Loss: 1.688292
Train Epoch: 5 [16640/60000 (28%)]      Loss: 1.695342
Train Epoch: 5 [17280/60000 (29%)]      Loss: 1.749749
Train Epoch: 5 [17920/60000 (30%)]      Loss: 1.650424
Train Epoch: 5 [18560/60000 (31%)]      Loss: 1.701332
Train Epoch: 5 [19200/60000 (32%)]      Loss: 1.621742
Train Epoch: 5 [19840/60000 (33%)]      Loss: 1.617743
Train Epoch: 5 [20480/60000 (34%)]      Loss: 1.989918
Train Epoch: 5 [21120/60000 (35%)]      Loss: 1.611280
Train Epoch: 5 [21760/60000 (36%)]      Loss: 1.606084
Train Epoch: 5 [22400/60000 (37%)]      Loss: 1.831293
Train Epoch: 5 [23040/60000 (38%)]      Loss: 1.541659
Train Epoch: 5 [23680/60000 (39%)]      Loss: 1.690876
Train Epoch: 5 [24320/60000 (41%)]      Loss: 1.805293
Train Epoch: 5 [24960/60000 (42%)]      Loss: 1.757422
Train Epoch: 5 [25600/60000 (43%)]      Loss: 1.475780
Train Epoch: 5 [26240/60000 (44%)]      Loss: 1.517133
Train Epoch: 5 [26880/60000 (45%)]      Loss: 1.438930
Train Epoch: 5 [27520/60000 (46%)]      Loss: 1.681104
Train Epoch: 5 [28160/60000 (47%)]      Loss: 1.536843
Train Epoch: 5 [28800/60000 (48%)]      Loss: 1.665066
Train Epoch: 5 [29440/60000 (49%)]      Loss: 1.538975
Train Epoch: 5 [30080/60000 (50%)]      Loss: 1.650898
Train Epoch: 5 [30720/60000 (51%)]      Loss: 1.957263
Train Epoch: 5 [31360/60000 (52%)]      Loss: 1.475897
Train Epoch: 5 [32000/60000 (53%)]      Loss: 1.746179
Train Epoch: 5 [32640/60000 (54%)]      Loss: 1.958112
Train Epoch: 5 [33280/60000 (55%)]      Loss: 1.878716
Train Epoch: 5 [33920/60000 (57%)]      Loss: 1.764753
Train Epoch: 5 [34560/60000 (58%)]      Loss: 1.683838
Train Epoch: 5 [35200/60000 (59%)]      Loss: 1.725957
```

```
Train Epoch: 5 [35840/60000 (60%)]      Loss: 1.628241
Train Epoch: 5 [36480/60000 (61%)]      Loss: 1.722456
Train Epoch: 5 [37120/60000 (62%)]      Loss: 1.736396
Train Epoch: 5 [37760/60000 (63%)]      Loss: 1.565337
Train Epoch: 5 [38400/60000 (64%)]      Loss: 1.734304
Train Epoch: 5 [39040/60000 (65%)]      Loss: 1.612186
Train Epoch: 5 [39680/60000 (66%)]      Loss: 1.676609
Train Epoch: 5 [40320/60000 (67%)]      Loss: 1.609199
Train Epoch: 5 [40960/60000 (68%)]      Loss: 1.568575
Train Epoch: 5 [41600/60000 (69%)]      Loss: 1.617352
Train Epoch: 5 [42240/60000 (70%)]      Loss: 1.503167
Train Epoch: 5 [42880/60000 (71%)]      Loss: 1.509256
Train Epoch: 5 [43520/60000 (72%)]      Loss: 1.542339
Train Epoch: 5 [44160/60000 (74%)]      Loss: 1.746523
Train Epoch: 5 [44800/60000 (75%)]      Loss: 1.570948
Train Epoch: 5 [45440/60000 (76%)]      Loss: 1.522895
Train Epoch: 5 [46080/60000 (77%)]      Loss: 1.611730
Train Epoch: 5 [46720/60000 (78%)]      Loss: 1.827677
Train Epoch: 5 [47360/60000 (79%)]      Loss: 1.818088
Train Epoch: 5 [48000/60000 (80%)]      Loss: 1.643548
Train Epoch: 5 [48640/60000 (81%)]      Loss: 1.852963
Train Epoch: 5 [49280/60000 (82%)]      Loss: 1.455582
Train Epoch: 5 [49920/60000 (83%)]      Loss: 1.632366
Train Epoch: 5 [50560/60000 (84%)]      Loss: 1.583847
Train Epoch: 5 [51200/60000 (85%)]      Loss: 1.568905
Train Epoch: 5 [51840/60000 (86%)]      Loss: 1.634936
Train Epoch: 5 [52480/60000 (87%)]      Loss: 1.792859
Train Epoch: 5 [53120/60000 (88%)]      Loss: 1.581836
Train Epoch: 5 [53760/60000 (90%)]      Loss: 1.590799
Train Epoch: 5 [54400/60000 (91%)]      Loss: 1.546345
Train Epoch: 5 [55040/60000 (92%)]      Loss: 1.595215
Train Epoch: 5 [55680/60000 (93%)]      Loss: 1.765202
Train Epoch: 5 [56320/60000 (94%)]      Loss: 1.734186
Train Epoch: 5 [56960/60000 (95%)]      Loss: 1.616785
Train Epoch: 5 [57600/60000 (96%)]      Loss: 1.606584
Train Epoch: 5 [58240/60000 (97%)]      Loss: 1.687896
Train Epoch: 5 [58880/60000 (98%)]      Loss: 1.768910
Train Epoch: 5 [59520/60000 (99%)]      Loss: 1.530133

Test set: Avg. loss: 1.3522, Accuracy: 6033/10000 (60%)

Train Epoch: 6 [0/60000 (0%)]   Loss: 1.606163
Train Epoch: 6 [640/60000 (1%)] Loss: 1.655597
Train Epoch: 6 [1280/60000 (2%)]        Loss: 1.637015
Train Epoch: 6 [1920/60000 (3%)]        Loss: 1.735553
Train Epoch: 6 [2560/60000 (4%)]        Loss: 1.758767
Train Epoch: 6 [3200/60000 (5%)]        Loss: 1.790750
Train Epoch: 6 [3840/60000 (6%)]        Loss: 1.429236
```

```
Train Epoch: 6 [4480/60000 (7%)]        Loss: 1.754624
Train Epoch: 6 [5120/60000 (9%)]        Loss: 1.541237
Train Epoch: 6 [5760/60000 (10%)]       Loss: 1.446383
Train Epoch: 6 [6400/60000 (11%)]       Loss: 1.676746
Train Epoch: 6 [7040/60000 (12%)]       Loss: 1.872014
Train Epoch: 6 [7680/60000 (13%)]       Loss: 1.868678
Train Epoch: 6 [8320/60000 (14%)]       Loss: 1.601902
Train Epoch: 6 [8960/60000 (15%)]       Loss: 1.496411
Train Epoch: 6 [9600/60000 (16%)]       Loss: 1.943234
Train Epoch: 6 [10240/60000 (17%)]      Loss: 1.751435
Train Epoch: 6 [10880/60000 (18%)]      Loss: 1.594096
Train Epoch: 6 [11520/60000 (19%)]      Loss: 1.760990
Train Epoch: 6 [12160/60000 (20%)]      Loss: 1.736537
Train Epoch: 6 [12800/60000 (21%)]      Loss: 1.624633
Train Epoch: 6 [13440/60000 (22%)]      Loss: 1.627194
Train Epoch: 6 [14080/60000 (23%)]      Loss: 1.730525
Train Epoch: 6 [14720/60000 (25%)]      Loss: 1.326702
Train Epoch: 6 [15360/60000 (26%)]      Loss: 1.746469
Train Epoch: 6 [16000/60000 (27%)]      Loss: 1.582238
Train Epoch: 6 [16640/60000 (28%)]      Loss: 1.668810
Train Epoch: 6 [17280/60000 (29%)]      Loss: 1.366605
Train Epoch: 6 [17920/60000 (30%)]      Loss: 1.587680
Train Epoch: 6 [18560/60000 (31%)]      Loss: 1.723737
Train Epoch: 6 [19200/60000 (32%)]      Loss: 1.602955
Train Epoch: 6 [19840/60000 (33%)]      Loss: 1.558135
Train Epoch: 6 [20480/60000 (34%)]      Loss: 1.533307
Train Epoch: 6 [21120/60000 (35%)]      Loss: 1.788374
Train Epoch: 6 [21760/60000 (36%)]      Loss: 1.684695
Train Epoch: 6 [22400/60000 (37%)]      Loss: 1.492282
Train Epoch: 6 [23040/60000 (38%)]      Loss: 1.727158
Train Epoch: 6 [23680/60000 (39%)]      Loss: 1.846760
Train Epoch: 6 [24320/60000 (41%)]      Loss: 1.444536
Train Epoch: 6 [24960/60000 (42%)]      Loss: 1.648739
Train Epoch: 6 [25600/60000 (43%)]      Loss: 1.632702
Train Epoch: 6 [26240/60000 (44%)]      Loss: 1.576805
Train Epoch: 6 [26880/60000 (45%)]      Loss: 1.449404
Train Epoch: 6 [27520/60000 (46%)]      Loss: 1.391297
Train Epoch: 6 [28160/60000 (47%)]      Loss: 1.626914
Train Epoch: 6 [28800/60000 (48%)]      Loss: 1.416876
Train Epoch: 6 [29440/60000 (49%)]      Loss: 1.689113
Train Epoch: 6 [30080/60000 (50%)]      Loss: 1.536542
Train Epoch: 6 [30720/60000 (51%)]      Loss: 1.573219
Train Epoch: 6 [31360/60000 (52%)]      Loss: 1.619847
Train Epoch: 6 [32000/60000 (53%)]      Loss: 1.666977
Train Epoch: 6 [32640/60000 (54%)]      Loss: 1.524066
Train Epoch: 6 [33280/60000 (55%)]      Loss: 1.801936
Train Epoch: 6 [33920/60000 (57%)]      Loss: 1.569296
Train Epoch: 6 [34560/60000 (58%)]      Loss: 1.565236
```

```
Train Epoch: 6 [35200/60000 (59%)]      Loss: 1.738132
Train Epoch: 6 [35840/60000 (60%)]      Loss: 1.558911
Train Epoch: 6 [36480/60000 (61%)]      Loss: 1.441554
Train Epoch: 6 [37120/60000 (62%)]      Loss: 1.880547
Train Epoch: 6 [37760/60000 (63%)]      Loss: 1.650763
Train Epoch: 6 [38400/60000 (64%)]      Loss: 1.428139
Train Epoch: 6 [39040/60000 (65%)]      Loss: 1.541226
Train Epoch: 6 [39680/60000 (66%)]      Loss: 1.571661
Train Epoch: 6 [40320/60000 (67%)]      Loss: 1.567901
Train Epoch: 6 [40960/60000 (68%)]      Loss: 1.404655
Train Epoch: 6 [41600/60000 (69%)]      Loss: 1.479782
Train Epoch: 6 [42240/60000 (70%)]      Loss: 1.477000
Train Epoch: 6 [42880/60000 (71%)]      Loss: 1.836013
Train Epoch: 6 [43520/60000 (72%)]      Loss: 1.695755
Train Epoch: 6 [44160/60000 (74%)]      Loss: 1.736305
Train Epoch: 6 [44800/60000 (75%)]      Loss: 1.709165
Train Epoch: 6 [45440/60000 (76%)]      Loss: 1.689896
Train Epoch: 6 [46080/60000 (77%)]      Loss: 1.667838
Train Epoch: 6 [46720/60000 (78%)]      Loss: 1.531822
Train Epoch: 6 [47360/60000 (79%)]      Loss: 1.471065
Train Epoch: 6 [48000/60000 (80%)]      Loss: 1.632969
Train Epoch: 6 [48640/60000 (81%)]      Loss: 1.575340
Train Epoch: 6 [49280/60000 (82%)]      Loss: 1.741369
Train Epoch: 6 [49920/60000 (83%)]      Loss: 1.515056
Train Epoch: 6 [50560/60000 (84%)]      Loss: 1.350115
Train Epoch: 6 [51200/60000 (85%)]      Loss: 1.473876
Train Epoch: 6 [51840/60000 (86%)]      Loss: 1.527722
Train Epoch: 6 [52480/60000 (87%)]      Loss: 1.532131
Train Epoch: 6 [53120/60000 (88%)]      Loss: 1.405500
Train Epoch: 6 [53760/60000 (90%)]      Loss: 1.446242
Train Epoch: 6 [54400/60000 (91%)]      Loss: 1.549795
Train Epoch: 6 [55040/60000 (92%)]      Loss: 1.565743
Train Epoch: 6 [55680/60000 (93%)]      Loss: 1.504326
Train Epoch: 6 [56320/60000 (94%)]      Loss: 1.729680
Train Epoch: 6 [56960/60000 (95%)]      Loss: 1.480865
Train Epoch: 6 [57600/60000 (96%)]      Loss: 1.485847
Train Epoch: 6 [58240/60000 (97%)]      Loss: 1.525082
Train Epoch: 6 [58880/60000 (98%)]      Loss: 1.590319
Train Epoch: 6 [59520/60000 (99%)]      Loss: 1.661578

Test set: Avg. loss: 1.2691, Accuracy: 6348/10000 (63%)

Train Epoch: 7 [0/60000 (0%)]   Loss: 1.470528
Train Epoch: 7 [640/60000 (1%)] Loss: 1.269132
Train Epoch: 7 [1280/60000 (2%)]        Loss: 1.319869
Train Epoch: 7 [1920/60000 (3%)]        Loss: 1.537299
Train Epoch: 7 [2560/60000 (4%)]        Loss: 1.461959
Train Epoch: 7 [3200/60000 (5%)]        Loss: 1.630617
```

```
Train Epoch: 7 [3840/60000 (6%)]        Loss: 1.475696
Train Epoch: 7 [4480/60000 (7%)]        Loss: 1.572340
Train Epoch: 7 [5120/60000 (9%)]        Loss: 1.469757
Train Epoch: 7 [5760/60000 (10%)]       Loss: 1.346283
Train Epoch: 7 [6400/60000 (11%)]       Loss: 1.513430
Train Epoch: 7 [7040/60000 (12%)]       Loss: 1.569073
Train Epoch: 7 [7680/60000 (13%)]       Loss: 1.614485
Train Epoch: 7 [8320/60000 (14%)]       Loss: 1.367619
Train Epoch: 7 [8960/60000 (15%)]       Loss: 1.706758
Train Epoch: 7 [9600/60000 (16%)]       Loss: 1.377922
Train Epoch: 7 [10240/60000 (17%)]      Loss: 1.668646
Train Epoch: 7 [10880/60000 (18%)]      Loss: 1.546496
Train Epoch: 7 [11520/60000 (19%)]      Loss: 1.450217
Train Epoch: 7 [12160/60000 (20%)]      Loss: 1.432647
Train Epoch: 7 [12800/60000 (21%)]      Loss: 1.617009
Train Epoch: 7 [13440/60000 (22%)]      Loss: 1.521451
Train Epoch: 7 [14080/60000 (23%)]      Loss: 1.372225
Train Epoch: 7 [14720/60000 (25%)]      Loss: 1.536443
Train Epoch: 7 [15360/60000 (26%)]      Loss: 1.456600
Train Epoch: 7 [16000/60000 (27%)]      Loss: 1.393358
Train Epoch: 7 [16640/60000 (28%)]      Loss: 1.451180
Train Epoch: 7 [17280/60000 (29%)]      Loss: 1.632435
Train Epoch: 7 [17920/60000 (30%)]      Loss: 1.444274
Train Epoch: 7 [18560/60000 (31%)]      Loss: 1.598967
Train Epoch: 7 [19200/60000 (32%)]      Loss: 1.347860
Train Epoch: 7 [19840/60000 (33%)]      Loss: 1.285892
Train Epoch: 7 [20480/60000 (34%)]      Loss: 1.568453
Train Epoch: 7 [21120/60000 (35%)]      Loss: 1.598737
Train Epoch: 7 [21760/60000 (36%)]      Loss: 1.620939
Train Epoch: 7 [22400/60000 (37%)]      Loss: 1.454626
Train Epoch: 7 [23040/60000 (38%)]      Loss: 1.432974
Train Epoch: 7 [23680/60000 (39%)]      Loss: 1.518465
Train Epoch: 7 [24320/60000 (41%)]      Loss: 1.404043
Train Epoch: 7 [24960/60000 (42%)]      Loss: 1.503339
Train Epoch: 7 [25600/60000 (43%)]      Loss: 1.714111
Train Epoch: 7 [26240/60000 (44%)]      Loss: 1.517891
Train Epoch: 7 [26880/60000 (45%)]      Loss: 1.516451
Train Epoch: 7 [27520/60000 (46%)]      Loss: 1.424855
Train Epoch: 7 [28160/60000 (47%)]      Loss: 1.778215
Train Epoch: 7 [28800/60000 (48%)]      Loss: 1.781990
Train Epoch: 7 [29440/60000 (49%)]      Loss: 1.563949
Train Epoch: 7 [30080/60000 (50%)]      Loss: 1.427419
Train Epoch: 7 [30720/60000 (51%)]      Loss: 1.536928
Train Epoch: 7 [31360/60000 (52%)]      Loss: 1.573296
Train Epoch: 7 [32000/60000 (53%)]      Loss: 1.530559
Train Epoch: 7 [32640/60000 (54%)]      Loss: 1.572730
Train Epoch: 7 [33280/60000 (55%)]      Loss: 1.635172
Train Epoch: 7 [33920/60000 (57%)]      Loss: 1.775551
```

```
Train Epoch: 7 [34560/60000 (58%)]      Loss: 1.373432
Train Epoch: 7 [35200/60000 (59%)]      Loss: 1.544963
Train Epoch: 7 [35840/60000 (60%)]      Loss: 1.483714
Train Epoch: 7 [36480/60000 (61%)]      Loss: 1.639524
Train Epoch: 7 [37120/60000 (62%)]      Loss: 1.390156
Train Epoch: 7 [37760/60000 (63%)]      Loss: 1.506042
Train Epoch: 7 [38400/60000 (64%)]      Loss: 1.638519
Train Epoch: 7 [39040/60000 (65%)]      Loss: 1.658675
Train Epoch: 7 [39680/60000 (66%)]      Loss: 1.337207
Train Epoch: 7 [40320/60000 (67%)]      Loss: 1.354470
Train Epoch: 7 [40960/60000 (68%)]      Loss: 1.805172
Train Epoch: 7 [41600/60000 (69%)]      Loss: 1.547708
Train Epoch: 7 [42240/60000 (70%)]      Loss: 1.472515
Train Epoch: 7 [42880/60000 (71%)]      Loss: 1.469830
Train Epoch: 7 [43520/60000 (72%)]      Loss: 1.420005
Train Epoch: 7 [44160/60000 (74%)]      Loss: 1.536278
Train Epoch: 7 [44800/60000 (75%)]      Loss: 1.610308
Train Epoch: 7 [45440/60000 (76%)]      Loss: 1.925429
Train Epoch: 7 [46080/60000 (77%)]      Loss: 1.380297
Train Epoch: 7 [46720/60000 (78%)]      Loss: 1.534966
Train Epoch: 7 [47360/60000 (79%)]      Loss: 1.577414
Train Epoch: 7 [48000/60000 (80%)]      Loss: 1.614012
Train Epoch: 7 [48640/60000 (81%)]      Loss: 1.556711
Train Epoch: 7 [49280/60000 (82%)]      Loss: 1.694106
Train Epoch: 7 [49920/60000 (83%)]      Loss: 1.427527
Train Epoch: 7 [50560/60000 (84%)]      Loss: 1.294647
Train Epoch: 7 [51200/60000 (85%)]      Loss: 1.579789
Train Epoch: 7 [51840/60000 (86%)]      Loss: 1.582177
Train Epoch: 7 [52480/60000 (87%)]      Loss: 1.656181
Train Epoch: 7 [53120/60000 (88%)]      Loss: 1.586301
Train Epoch: 7 [53760/60000 (90%)]      Loss: 1.359378
Train Epoch: 7 [54400/60000 (91%)]      Loss: 1.621071
Train Epoch: 7 [55040/60000 (92%)]      Loss: 1.559799
Train Epoch: 7 [55680/60000 (93%)]      Loss: 1.423362
Train Epoch: 7 [56320/60000 (94%)]      Loss: 1.691877
Train Epoch: 7 [56960/60000 (95%)]      Loss: 1.495156
Train Epoch: 7 [57600/60000 (96%)]      Loss: 1.565975
Train Epoch: 7 [58240/60000 (97%)]      Loss: 1.334616
Train Epoch: 7 [58880/60000 (98%)]      Loss: 1.615387
Train Epoch: 7 [59520/60000 (99%)]      Loss: 1.705570

Test set: Avg. loss: 1.1891, Accuracy: 6776/10000 (68%)
```

15 2.6 Storing the Model

Important Note: In order for the autograder not to overwrite your model with empty (untrained) model, please make sure you wrap your code within the following conditional statement:

```
if perform_computation:
   # Save your trained model here
```

[23]:
```
message = 'Here you should store the model at "./mnist_net.pth" .'
print(message)

if perform_computation:
    # your code here
```

```
  File "<ipython-input-23-d72e5d5b12bc>", line 6

    ^
SyntaxError: unexpected EOF while parsing
```

16 2.7 Evaluating the Trained Model

[ ]:
```
message = 'Here you can visualize a bunch of examples and print the prediction␣
 ↪of the trained classifier (this is optional).'
print(message)

# your code here
raise NotImplementedError
```

[ ]:
```
message = 'Here you can evaluate the overall accuracy of the trained classifier␣
 ↪(this is optional).'
print(message)

# your code here
raise NotImplementedError
```

[ ]:
```
message = 'Here you can evaluate the per-class accuracy of the trained␣
 ↪classifier (this is optional).'
print(message)

# your code here
raise NotImplementedError
```

16.1 2.8 Autograding and Final Tests

[24]:
```
assert 'Net' in globals().keys(), 'The Net class was not defined earlier. ' + \
                                  'Make sure you read and follow the␣
 ↪instructions provided as Important Notes' + \
                                  '(especially, the "Model Class Naming" part).'

mnist_net_path = './mnist_net.pth'

assert os.path.exists(mnist_net_path), 'You have not stored the trained model␣
 ↪properly. ' + \
                                       'Make sure you read and follow the␣
 ↪instructions provided as Important Notes.'

assert os.path.getsize(mnist_net_path) < 1000000, 'The size of your trained␣
 ↪model exceeds 1 MB.'

if 'net' in globals():
    del net
net = Net()
net.load_state_dict(torch.load(mnist_net_path))
net = net.eval()

# Disclaimer: Most of the following code was adopted from Pytorch's␣
 ↪Documentation and Examples
# https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

transformation_list = [transforms.RandomAffine(degrees=60, translate=(0.2, 0.
 ↪2), scale=(0.5, 2.),
                                               shear=None, resample=0,␣
 ↪fillcolor=0),
                       transforms.ToTensor(),
                       transforms.Normalize((0.5,), (0.5,))]

test_pre_tranformation = transforms.Compose(transformation_list)

mnist_root = '/home/jovyan/work/course-lib/data_mnist'
testset = torchvision.datasets.MNIST(root=mnist_root, train=False,
                                     download=False,␣
 ↪transform=test_pre_tranformation)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=1)


class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
```

```
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
print('----------')
print(f'Overall Testing Accuracy: {100. * sum(class_correct) /␣
 ↪sum(class_total)} %%')
```

```
Accuracy of     0 : 70 %
Accuracy of     1 : 98 %
Accuracy of     2 : 42 %
Accuracy of     3 : 53 %
Accuracy of     4 : 63 %
Accuracy of     5 : 50 %
Accuracy of     6 : 77 %
Accuracy of     7 : 82 %
Accuracy of     8 : 35 %
Accuracy of     9 : 50 %
----------
Overall Testing Accuracy: 63.03 %%
```

[ ]:
```
# "Digit Recognition Test: Checking the accuracy on the MNIST Images"
```

[ ]:
```

```