```
import torch
import numpy as np
import collections
from torchvision import datasets
import torchvision.transforms as transforms
from torch.utils.data.sampler import SubsetRandomSampler

device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
print(device)
```

```
cuda
```

```
# specify the image classes
classes = ['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck

num_workers = 0
batch_size = 20
valid_size = 0.2

# convert data to a normalized torch.FloatTensor
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

# choose the training and test datasets
train_data = datasets.CIFAR10('data', train=True,
                              download=True, transform=transform)
test_data = datasets.CIFAR10('data', train=False,
                             download=True, transform=transform)

# obtain training indices that will be used for validation
num_train = len(train_data)
indices = list(range(num_train))
np.random.shuffle(indices)
split = int(np.floor(valid_size * num_train))
train_idx, valid_idx = indices[split:], indices[:split]

# define samplers for obtaining training and validation batches
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)
```

```
all_acc_dict = collections.OrderedDict()
```

```
train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
    sampler=train_sampler, num_workers=num_workers)
valid_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
    sampler=valid_sampler, num_workers=num_workers)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=batch_size,
    num_workers=num_workers)
```

2a.

Build a ResNet based Convolutional Neural Network, like what we built in lectures (with skip connections), to classify the images across all 10 classes in CIFAR 10. For this problem, let's use 10 blocks for ResNet and call it ResNet-10. Use the similar dimensions and channels as we need in lectures. Train your network for 300 epochs. Report your training time, training loss, and evaluation accuracy after 300 epochs. Analyze your results in your report and compare them against problem 1.b on training time, achieved accuracy, and model size. Make sure to submit your code by providing the GitHub URL of your course repository for this course.

```
from tqdm import tqdm
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

```
def validate(model, train_loader, val_loader):
    accdict = {}
    for name, loader in [("train", train_loader), ("val", valid_loader)]:
        correct = 0
        total = 0

        with torch.no_grad():
            for imgs, labels in tqdm(loader):
                imgs = imgs.to(device=device)
                labels = labels.to(device=device)
                outputs = model(imgs)
                _, predicted = torch.max(outputs, dim=1) # <1>
                total += labels.shape[0]
                correct += int((predicted == labels).sum())

        print("Accuracy {}: {:.2f}".format(name , correct / total))
        accdict[name] = correct / total
    return accdict

```

```
def training_loop(n_epochs, optimizer, model, loss_fn, train_loader):
    for epoch in range(1, n_epochs + 1):
        loss_train = 0.0
```

```
        for imgs, labels in train_loader:
            imgs = imgs.to(device=device)
            labels = labels.to(device=device)
            outputs = model(imgs)
            loss = loss_fn(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            loss_train += loss.item()

        if epoch == 1 or epoch % 25 == 0:
            print('\nEpoch {}, \nTraining loss {}'.format(epoch,
                loss_train / len(train_loader)))
            validate(model, train_loader, valid_loader)
```

Model architecture:

ResNet-10

> NetResDeep
>
> > ResBLock

```
class ResBlock(nn.Module):
    def __init__(self, n_chans):
        super(ResBlock, self).__init__()
        self.conv = nn.Conv2d(n_chans, n_chans, kernel_size=3,
                              padding=1, bias=False)  # <1>
        self.batch_norm = nn.BatchNorm2d(num_features=n_chans)
        torch.nn.init.kaiming_normal_(self.conv.weight,
                                      nonlinearity='relu')  # <2>
        torch.nn.init.constant_(self.batch_norm.weight, 0.5)
        torch.nn.init.zeros_(self.batch_norm.bias)

    def forward(self, x):
        out = self.conv(x)
        out = self.batch_norm(out)
        out = torch.relu(out)
        return out + x
```

```
class NetResDeep(nn.Module):
    def __init__(self, n_chans1=32, n_blocks=10):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.resblocks = nn.Sequential(
            *(n_blocks * [ResBlock(n_chans=n_chans1)]))
```

```
        self.fc1 = nn.Linear(8 * 8 * n_chans1, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = F.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = self.resblocks(out)
        out = F.max_pool2d(out, 2)
        out = out.view(-1, 8 * 8 * self.n_chans1)
        out = torch.relu(self.fc1(out))
        out = self.fc2(out)
        return out
```

Model 1: training & testing

```
model = NetResDeep(n_chans1=32, n_blocks=10).to(device=device)
optimizer = optim.SGD(model.parameters(), lr=3e-3)
criterion = nn.CrossEntropyLoss() # loss function
```

```
import datetime
```

```
%%time
training_loop(
    n_epochs = 300,
    optimizer = optimizer,
    model = model,
    loss_fn = criterion,
    train_loader = train_loader,
)
```

```
Epoch 125,
Training loss 0.059915671777538135
100%|██████████| 2000/2000 [00:11<00:00, 171.16it/s]
Accuracy train: 0.98
100%|██████████| 500/500 [00:02<00:00, 176.05it/s]
Accuracy val: 0.65

Epoch 150,
Training loss 0.05428413086901128
100%|██████████| 2000/2000 [00:11<00:00, 171.11it/s]
Accuracy train: 0.98
100%|██████████| 500/500 [00:02<00:00, 170.62it/s]
Accuracy val: 0.64

Epoch 175,
Training loss 0.03712588283987088
100%|██████████| 2000/2000 [00:11<00:00, 169.07it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:02<00:00, 170.85it/s]
Accuracy val: 0.64
```

```
Epoch 200,
Training loss 0.036101794352949586
100%|██████████| 2000/2000 [00:11<00:00, 172.52it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:03<00:00, 165.78it/s]
Accuracy val: 0.65

Epoch 225,
Training loss 0.039580269681479874
100%|██████████| 2000/2000 [00:11<00:00, 171.35it/s]
Accuracy train: 0.98
100%|██████████| 500/500 [00:03<00:00, 165.71it/s]
Accuracy val: 0.64

Epoch 250,
Training loss 0.030393678966140213
100%|██████████| 2000/2000 [00:11<00:00, 168.46it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:03<00:00, 165.26it/s]
Accuracy val: 0.64


Epoch 275,
Training loss 0.02017068582485471
100%|██████████| 2000/2000 [00:11<00:00, 168.04it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:02<00:00, 169.43it/s]
Accuracy val: 0.64

Epoch 300,
Training loss 0.03243725258223584
100%|██████████| 2000/2000 [00:11<00:00, 168.84it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:03<00:00, 165.15it/s]Accuracy val: 0.65
CPU times: user 1h 22min 23s, sys: 29.5 s, total: 1h 22min 53s
Wall time: 1h 22min 38s
```

```
torch.save(model.state_dict(), 'model_cifar.pt')
model.load_state_dict(torch.load('model_cifar.pt'))
```

```
<All keys matched successfully>
```

```
# track test loss
test_loss = 0.0
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))

model.eval()
# iterate over test data
for data, target in test_loader:
    # move tensors to GPU if CUDA is available
    data, target = data.to(device), target.to(device)
    # forward pass: compute predicted outputs by passing inputs to the model
```

```
    output = model(data)
    # calculate the batch loss
    loss = criterion(output, target)
    # update test loss
    test_loss += loss.item()*data.size(0)
    # convert output probabilities to predicted class
    _, pred = torch.max(output, 1)
    # compare predictions to true label
    correct_tensor = pred.eq(target.data.view_as(pred))
    correct = np.squeeze(correct_tensor.cpu().numpy())
    # calculate test accuracy for each object class
    for i in range(batch_size):
        label = target.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1

# average test loss
test_loss = test_loss/len(test_loader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))

for i in range(10):
    if class_total[i] > 0:
        print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (
            classes[i], 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))
    else:
        print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (
    100. * np.sum(class_correct) / np.sum(class_total),
    np.sum(class_correct), np.sum(class_total)))
```

```
Test Loss: 7.892042

Test Accuracy of airplane:  6% (64/1000)
Test Accuracy of automobile:  0% ( 4/1000)
Test Accuracy of  bird: 20% (206/1000)
Test Accuracy of   cat: 33% (330/1000)
Test Accuracy of  deer: 62% (623/1000)
Test Accuracy of   dog: 11% (111/1000)
Test Accuracy of  frog: 76% (766/1000)
Test Accuracy of horse:  3% (39/1000)
Test Accuracy of  ship: 50% (506/1000)
Test Accuracy of truck:  2% (20/1000)

Test Accuracy (Overall): 26% (2669/10000)
```

2b.

Develop three additional trainings and evaluations for your ResNet-10 to assess the impacts of regularization to your ResNet-10.

- Weight Decay with lambda of 0.001
- Dropout with p=0.3
- Batch Normalization

Weight Decay with lambada of 0.001

```
model1 = NetResDeep(n_chans1=32, n_blocks=10).to(device=device)
optimizer = optim.SGD(model1.parameters(), lr=3e-3,  weight_decay=1e-4)
loss_fn = nn.CrossEntropyLoss()
```

```
%%time
training_loop(
    n_epochs = 300,
    optimizer = optimizer,
    model = model1,
    loss_fn = loss_fn,
    train_loader = train_loader,
)
```

```
Epoch 125,
Training loss 0.05250040679562426
100%|██████████| 2000/2000 [00:11<00:00, 172.78it/s]
Accuracy train: 0.98
100%|██████████| 500/500 [00:02<00:00, 172.33it/s]
Accuracy val: 0.65

Epoch 150,
Training loss 0.060921908087027986
100%|██████████| 2000/2000 [00:11<00:00, 171.47it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:02<00:00, 169.08it/s]
Accuracy val: 0.65

Epoch 175,
Training loss 0.047182706075353964
100%|██████████| 2000/2000 [00:11<00:00, 167.95it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:02<00:00, 169.65it/s]
Accuracy val: 0.66

Epoch 200,
Training loss 0.04347056663020521
100%|██████████| 2000/2000 [00:11<00:00, 170.99it/s]
Accuracy train: 0.98
100%|██████████| 500/500 [00:02<00:00, 172.72it/s]
Accuracy val: 0.65

Epoch 225,
Training loss 0.03333504017757514
100%|██████████| 2000/2000 [00:11<00:00, 171.78it/s]
```

```
100%|██████████| 2000/2000 [00:11<00:00, 171.78it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:02<00:00, 172.38it/s]
Accuracy val: 0.65

Epoch 250,
Training loss 0.024144101363983282
100%|██████████| 2000/2000 [00:11<00:00, 170.47it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:02<00:00, 168.58it/s]
Accuracy val: 0.65

Epoch 275,
Training loss 0.014662986142694876
100%|██████████| 2000/2000 [00:11<00:00, 169.58it/s]
Accuracy train: 1.00
100%|██████████| 500/500 [00:02<00:00, 170.10it/s]
Accuracy val: 0.65

Epoch 300,
Training loss 0.047725999124603104
100%|██████████| 2000/2000 [00:11<00:00, 173.03it/s]
Accuracy train: 0.99
100%|██████████| 500/500 [00:03<00:00, 165.54it/s]Accuracy val: 0.65
CPU times: user 1h 22min 19s, sys: 27.8 s, total: 1h 22min 47s
Wall time: 1h 22min 33s
```

```
# save model
torch.save(model1.state_dict(), 'model1_cifar.pt')
model1.load_state_dict(torch.load('model1_cifar.pt'))

# track test loss
test_loss = 0.0
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))

model1.eval()
# iterate over test data
for data, target in test_loader:
    data, target = data.to(device), target.to(device)
    output = model1(data)
    loss = criterion(output, target)
    test_loss += loss.item()*data.size(0)
    _, pred = torch.max(output, 1)
    correct_tensor = pred.eq(target.data.view_as(pred))
    correct = np.squeeze(correct_tensor.cpu().numpy())
    # calculate test accuracy for each object class
    for i in range(batch_size):
        label = target.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1

```

```
# average test loss
test_loss = test_loss/len(test_loader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))

for i in range(10):
    if class_total[i] > 0:
        print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (
            classes[i], 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))
    else:
        print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (
    100. * np.sum(class_correct) / np.sum(class_total),
    np.sum(class_correct), np.sum(class_total)))
```

```
Test Loss: 7.160267

Test Accuracy of airplane: 13% (134/1000)
Test Accuracy of automobile:  2% (21/1000)
Test Accuracy of  bird:  7% (70/1000)
Test Accuracy of   cat:  4% (47/1000)
Test Accuracy of  deer: 90% (908/1000)
Test Accuracy of   dog:  5% (50/1000)
Test Accuracy of  frog: 66% (668/1000)
Test Accuracy of horse:  5% (53/1000)
Test Accuracy of  ship: 25% (258/1000)
Test Accuracy of truck: 23% (238/1000)

Test Accuracy (Overall): 24% (2447/10000)
```

Dropout with p = 0.3

```
class NetResDeepDropout(nn.Module):
    def __init__(self, n_chans1=32, n_blocks=10):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.resblocks = nn.Sequential(
            *(n_blocks * [ResBlock(n_chans=n_chans1)]))
        self.fc1 = nn.Linear(8 * 8 * n_chans1, 32)
        self.fc2 = nn.Linear(32, 10)
        self.dropout2d = nn.Dropout2d(p=0.3)
        self.dropout = nn.Dropout(p=0.3)

    def forward(self, x):
        out = F.max_pool2d(torch.relu(self.conv1(x)), 2)
        out = self.dropout2d(out)
        out = self.resblocks(out)
        out = F.max_pool2d(out, 2)
        out = self.dropout2d(out)
```

```
        out = out.view(-1, 8 * 8 * self.n_chans1)
        out = torch.relu(self.fc1(out))
        out = self.dropout(out)
        out = self.fc2(out)
        return out
```

```
model2 = NetResDeepDropout(n_chans1=32, n_blocks=10).to(device=device)
optimizer2 = optim.SGD(model2.parameters(), lr=9e-3)
criterion2 = nn.CrossEntropyLoss()
```

```
%%time
training_loop(
    n_epochs = 300,
    optimizer = optimizer2,
    model = model2,
    loss_fn = criterion2,
    train_loader = train_loader,
)
```

```
Epoch 125,
Training loss 1.144781928882003
100%|██████████| 2000/2000 [00:14<00:00, 136.15it/s]
Accuracy train: 0.61
100%|██████████| 500/500 [00:03<00:00, 135.45it/s]
Accuracy val: 0.56

Epoch 150,
Training loss 1.1063170250952243
100%|██████████| 2000/2000 [00:15<00:00, 125.93it/s]
Accuracy train: 0.61
100%|██████████| 500/500 [00:04<00:00, 122.61it/s]
Accuracy val: 0.56

Epoch 175,
Training loss 1.0939056973308325
100%|██████████| 2000/2000 [00:15<00:00, 132.08it/s]
Accuracy train: 0.62
100%|██████████| 500/500 [00:03<00:00, 127.64it/s]
Accuracy val: 0.57

Epoch 200,
Training loss 1.0832975043654443
100%|██████████| 2000/2000 [00:15<00:00, 131.06it/s]
Accuracy train: 0.62
100%|██████████| 500/500 [00:03<00:00, 131.80it/s]
Accuracy val: 0.58

Epoch 225,
Training loss 1.08542363370955
100%|██████████| 2000/2000 [00:15<00:00, 132.32it/s]
Accuracy train: 0.61
100%|██████████| 500/500 [00:03<00:00, 131.82it/s]
Accuracy val: 0.57
```

```
Epoch 250,
Training loss 1.0706671068519353
100%|██████████| 2000/2000 [00:15<00:00, 131.81it/s]
Accuracy train: 0.62
100%|██████████| 500/500 [00:03<00:00, 132.49it/s]
Accuracy val: 0.57

Epoch 275,
Training loss 1.061357958495617
100%|██████████| 2000/2000 [00:15<00:00, 132.76it/s]
Accuracy train: 0.63
100%|██████████| 500/500 [00:03<00:00, 130.11it/s]
Accuracy val: 0.58

Epoch 300,
Training loss 1.0545503511428833
100%|██████████| 2000/2000 [00:15<00:00, 130.91it/s]
Accuracy train: 0.63
100%|██████████| 500/500 [00:03<00:00, 133.17it/s]Accuracy val: 0.57
CPU times: user 1h 51min 2s, sys: 30.7 s, total: 1h 51min 33s
Wall time: 1h 51min 17s
```

```python
# save model
torch.save(model2.state_dict(), 'model2_cifar.pt')
model2.load_state_dict(torch.load('model2_cifar.pt'))

# track test loss
test_loss = 0.0
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))

model2.eval()
# iterate over test data
for data, target in test_loader:
    data, target = data.to(device), target.to(device)
    output = model2(data)
    loss = criterion(output, target)
    test_loss += loss.item()*data.size(0)
    _, pred = torch.max(output, 1)
    correct_tensor = pred.eq(target.data.view_as(pred))
    correct = np.squeeze(correct_tensor.cpu().numpy())
    # calculate test accuracy for each object class
    for i in range(batch_size):
        label = target.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1

# average test loss
test_loss = test_loss/len(test_loader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))

```

```
for i in range(10):
    if class_total[i] > 0:
        print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (
            classes[i], 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))
    else:
        print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (
    100. * np.sum(class_correct) / np.sum(class_total),
    np.sum(class_correct), np.sum(class_total)))
```

```
Test Loss: 1.921139

Test Accuracy of airplane: 22% (226/1000)
Test Accuracy of automobile: 36% (368/1000)
Test Accuracy of  bird:  1% (13/1000)
Test Accuracy of   cat: 17% (174/1000)
Test Accuracy of  deer:  0% ( 7/1000)
Test Accuracy of   dog:  4% (48/1000)
Test Accuracy of  frog:  4% (49/1000)
Test Accuracy of horse: 79% (794/1000)
Test Accuracy of  ship:  6% (61/1000)
Test Accuracy of truck: 88% (882/1000)

Test Accuracy (Overall): 26% (2622/10000)
```

Batch Normalization

```
class NetResDeepBatchNorm(nn.Module):
    def __init__(self, n_chans1=32, n_blocks=10):
        super().__init__()
        self.n_chans1 = n_chans1
        self.conv1 = nn.Conv2d(3, n_chans1, kernel_size=3, padding=1)
        self.conv1_norm = nn.BatchNorm2d(n_chans1)
        self.resblocks = nn.Sequential(
            *(n_blocks * [ResBlock(n_chans=n_chans1)]))
        self.fc1 = nn.Linear(8 * 8 * n_chans1, 32)
        self.fc1_norm = nn.BatchNorm1d(32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        out = F.max_pool2d(torch.relu(self.conv1_norm(self.conv1(x))), 2)
        out = self.resblocks(out)
        out = F.max_pool2d(out, 2)
        out = out.view(-1, 8 * 8 * self.n_chans1)
        out = torch.relu(self.fc1_norm(self.fc1(out)))
        out = self.fc2(out)
        return out
```

```
model3 = NetResDeepBatchNorm(n_chans1=32, n_blocks=10).to(device=device)
optimizer3 = optim.SGD(model3.parameters(), lr=9e-3)
criterion3 = nn.CrossEntropyLoss()
```

```
%%time
training_loop(
    n_epochs = 300,
    optimizer = optimizer3,
    model = model3,
    loss_fn = criterion3,
    train_loader = train_loader,
)
```

```
Epoch 125,
Training loss 0.13427692255144938
100%|██████████| 2000/2000 [00:11<00:00, 168.07it/s]
Accuracy train: 0.96
100%|██████████| 500/500 [00:03<00:00, 160.87it/s]
Accuracy val: 0.64

Epoch 150,
Training loss 0.1219282886844012
100%|██████████| 2000/2000 [00:12<00:00, 164.09it/s]
Accuracy train: 0.96
100%|██████████| 500/500 [00:03<00:00, 165.22it/s]
Accuracy val: 0.64

Epoch 175,
Training loss 0.11227119853644399
100%|██████████| 2000/2000 [00:12<00:00, 165.55it/s]
Accuracy train: 0.97
100%|██████████| 500/500 [00:03<00:00, 164.52it/s]
Accuracy val: 0.64

Epoch 200,
Training loss 0.103699491652078
100%|██████████| 2000/2000 [00:12<00:00, 164.89it/s]
Accuracy train: 0.97
100%|██████████| 500/500 [00:02<00:00, 167.59it/s]
Accuracy val: 0.65

Epoch 225,
Training loss 0.09236694530025125
100%|██████████| 2000/2000 [00:11<00:00, 167.72it/s]
Accuracy train: 0.97
100%|██████████| 500/500 [00:02<00:00, 169.13it/s]
Accuracy val: 0.65

Epoch 250,
Training loss 0.09012942448526155
100%|██████████| 2000/2000 [00:12<00:00, 161.64it/s]
Accuracy train: 0.97
100%|██████████| 500/500 [00:03<00:00, 160.06it/s]
Accuracy val: 0.64
```

```
Epoch 275,
Training loss 0.0830427543593396
100%|██████████| 2000/2000 [00:12<00:00, 162.71it/s]
Accuracy train: 0.97
100%|██████████| 500/500 [00:03<00:00, 163.85it/s]
Accuracy val: 0.64

Epoch 300,
Training loss 0.08433806079177884
100%|██████████| 2000/2000 [00:12<00:00, 164.83it/s]
Accuracy train: 0.97
100%|██████████| 500/500 [00:03<00:00, 162.87it/s]Accuracy val: 0.64
CPU times: user 1h 24min 51s, sys: 33.4 s, total: 1h 25min 25s
Wall time: 1h 25min 15s
```

```python
# save model
torch.save(model3.state_dict(), 'model3_cifar.pt')
model3.load_state_dict(torch.load('model3_cifar.pt'))

# track test loss
test_loss = 0.0
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))

model3.eval()
# iterate over test data
for data, target in test_loader:
    data, target = data.to(device), target.to(device)
    output = model3(data)
    loss = criterion3(output, target)
    test_loss += loss.item()*data.size(0)
    _, pred = torch.max(output, 1)
    correct_tensor = pred.eq(target.data.view_as(pred))
    correct = np.squeeze(correct_tensor.cpu().numpy())
    # calculate test accuracy for each object class
    for i in range(batch_size):
        label = target.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1

# average test loss
test_loss = test_loss/len(test_loader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))

for i in range(10):
    if class_total[i] > 0:
        print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (
            classes[i], 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))
    else:
```

```
        print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (
    100. * np.sum(class_correct) / np.sum(class_total),
    np.sum(class_correct), np.sum(class_total)))
```

```
Test Loss: 4.055020

Test Accuracy of airplane:  0% ( 0/1000)
Test Accuracy of automobile: 34% (348/1000)
Test Accuracy of  bird:  0% ( 0/1000)
Test Accuracy of   cat:  0% ( 0/1000)
Test Accuracy of  deer:  0% ( 0/1000)
Test Accuracy of   dog:  0% ( 1/1000)
Test Accuracy of  frog:  0% ( 1/1000)
Test Accuracy of horse:  0% ( 0/1000)
Test Accuracy of  ship:  0% ( 0/1000)
Test Accuracy of truck: 65% (656/1000)

Test Accuracy (Overall): 10% (1006/10000)
```