CS535 DL Assignment3 Dongkyu Kim

Question 1

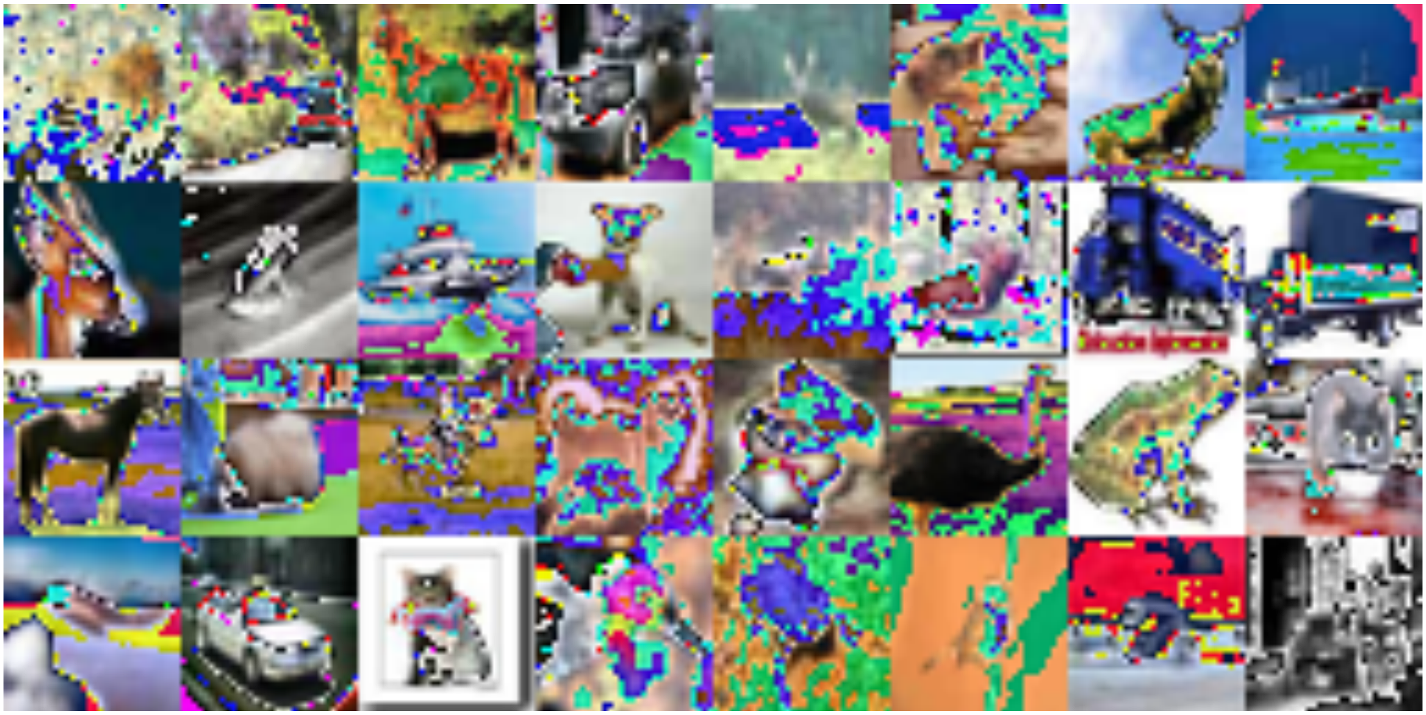
- Add a batch normalization layer after the first fully-connected layer(fc1) (8 points).
- Save the model after training(Checkout our tutorial on how to save your model). Becareful that batch normalization layer performs differently between training and evalation process, make sure you understand how to convert your model between training mode and evaluation mode(you can find hints in my code).
- Observe the difference of final training/testing accuracy with/without batch normalization layer.

Code:

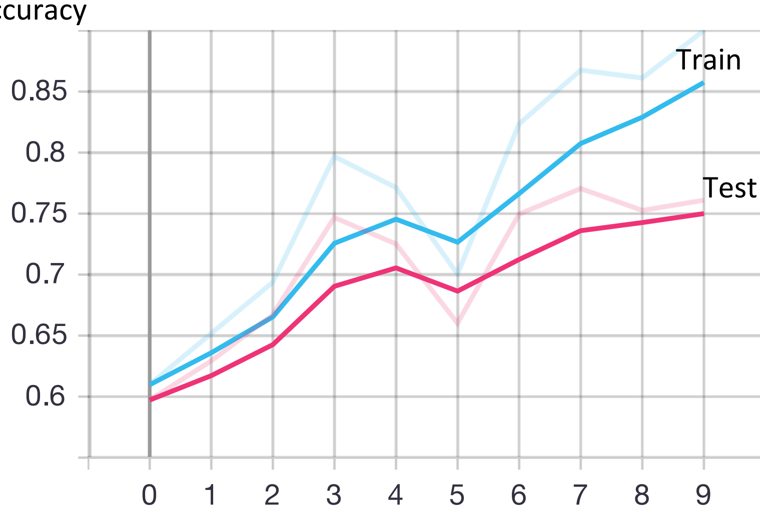
```
## Q1
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv4 = nn.Conv2d(64, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 8 * 8, 512)
        self.fc2 = nn.Linear(512, 10)
        self.bnorm = nn.BatchNorm1d(512)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.pool(x)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = self.bnorm(x)
        x = self.fc2(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

CS535 DL Assignment3 Dongkyu Kim

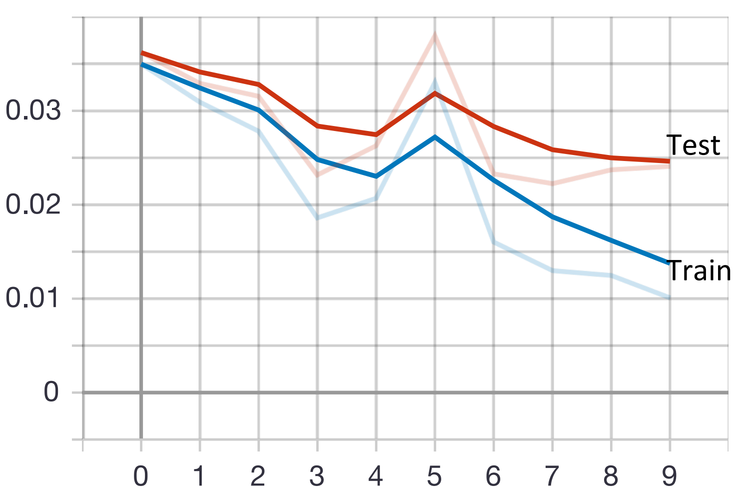
Result:



Accuracy



Loss



CS535 DL Assignment3 Dongkyu Kim
Question 2

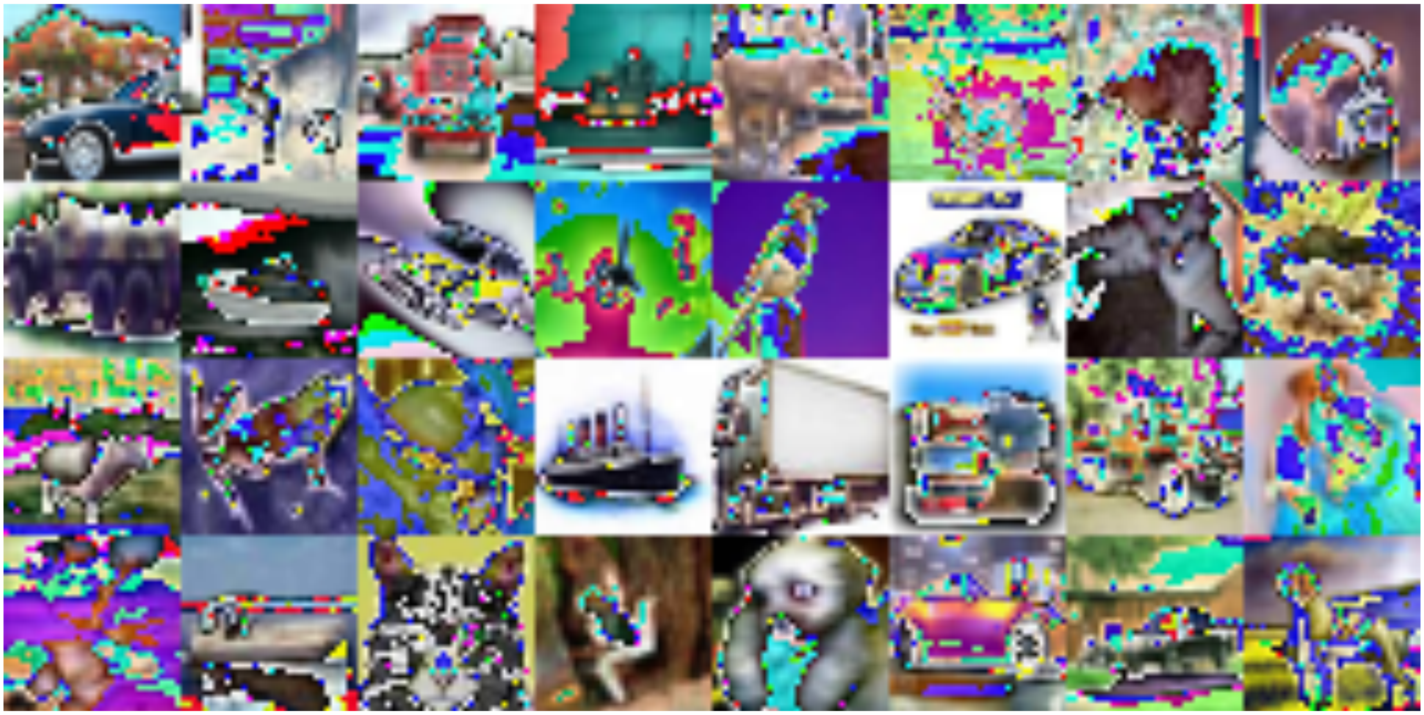
- Modify our model by adding another fully connected layer with 512 nodes at the second-to-last layer (before the fc2 layer) (8 points).
- Apply the model weights you saved at step 1 to initialize to the new model(only up to fc2 layer since after that all layers are newly created) before training.
- Train and save the model (Hint: check the end of the assignment description to see how to partially restore weights from a pretrained weights file).

Code:

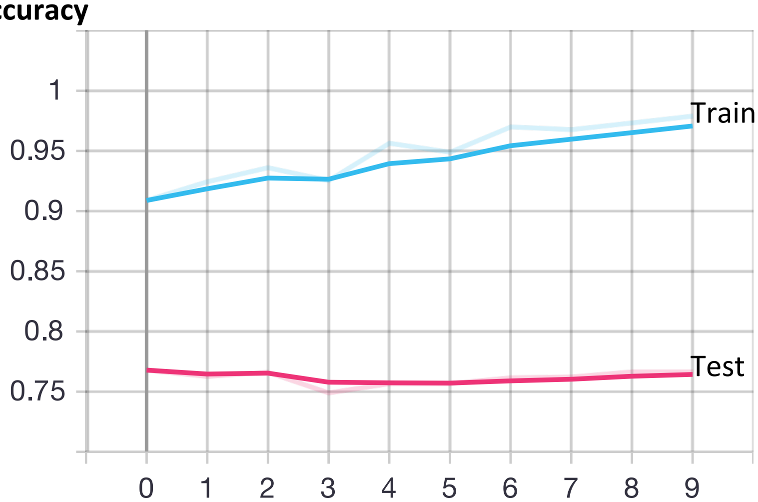
```
## Q2-1
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv4 = nn.Conv2d(64, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 8 * 8, 512)
        self.fc_q2 = nn.Linear(512, 512)
        self.fc2 = nn.Linear(512, 10)
        self.bnorm = nn.BatchNorm1d(512)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.pool(x)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = self.bnorm(x)
        x = F.relu(self.fc_q2(x))
        x = self.fc2(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features

## Q2-2
def partially_restore_weights(filepath):
    pretrained_dict = torch.load(filepath)
    model_dict = net.state_dict()
    pretrained_dict = {key: val for key, val in pretrained_dict.items() if key in model_dict}
    model_dict.update(pretrained_dict)
    net.load_state_dict(model_dict)

    return net
```

CS535 DL Assignment3 Dongkyu Kim

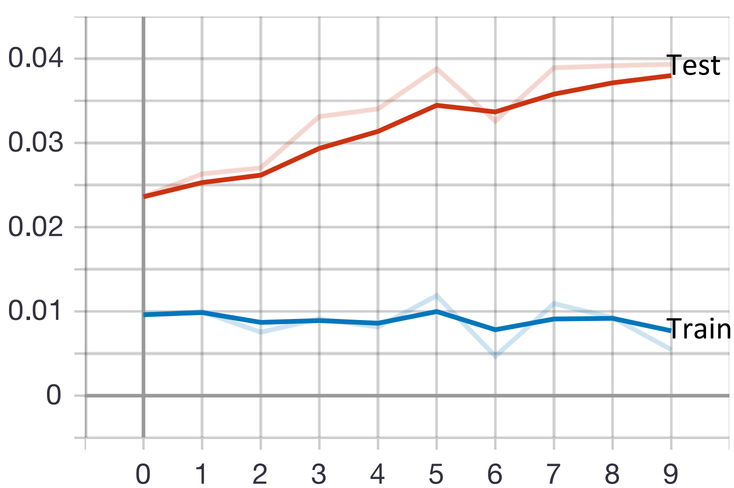
Result:



Accuracy



Loss



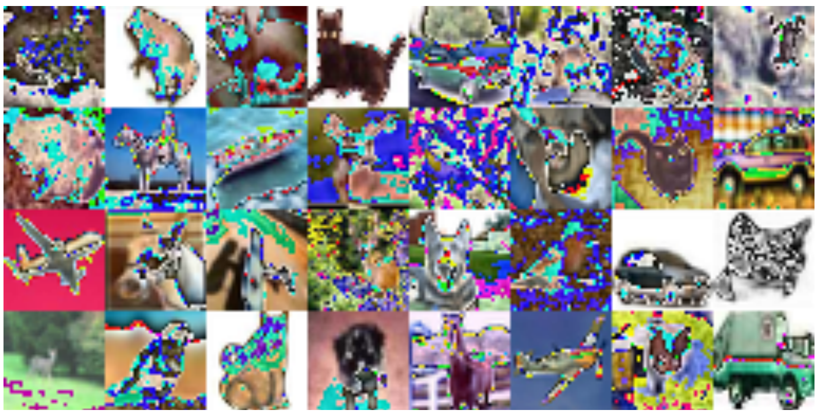
CS535 DL Assignment3 Dongkyu Kim
Question 3

- Try to use an adaptive schedule to tune the learning rate, you can choose from RMSprop, Adagrad and Adam (Hint: you don't need to implement any of these, look at Pytorch documentation please) (8 points).

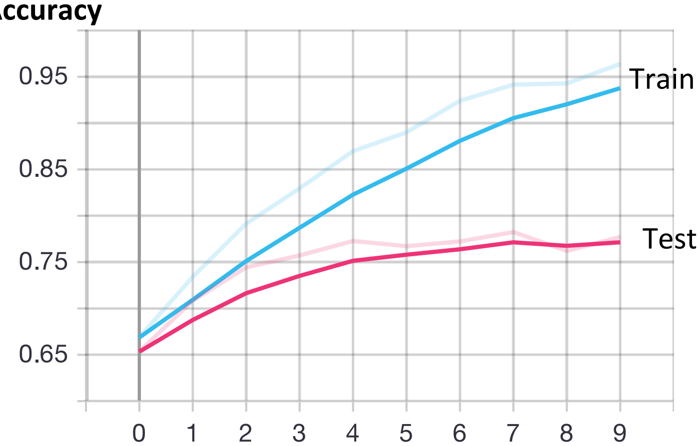
Code:

```
criterion = nn.CrossEntropyLoss()
# optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
optimizer = optim.Adam(net.parameters(), lr=0.001, amsgrad=True) # Q3 & Q4
```

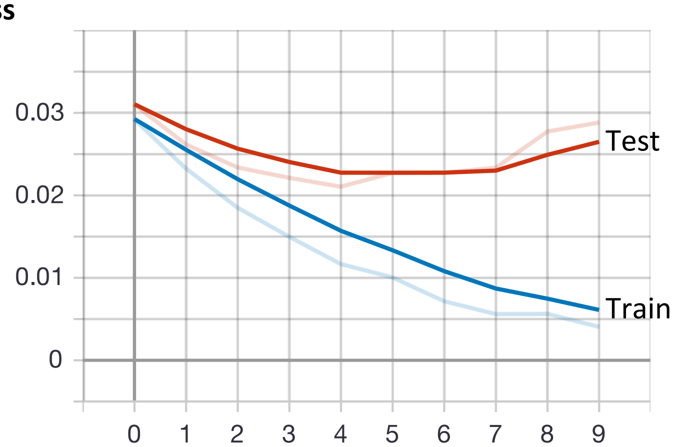
Result:



Accuracy



Loss



CS535 DL Assignment3 Dongkyu Kim

Question 4

- Try to tune your network in another way (e.g. add/remove a layer, change the activation function, add/remove regularizer, change the number of hidden units, more batch normalization layers) not described in the previous four. You can start from random initialization or previous results as you wish (8 points).

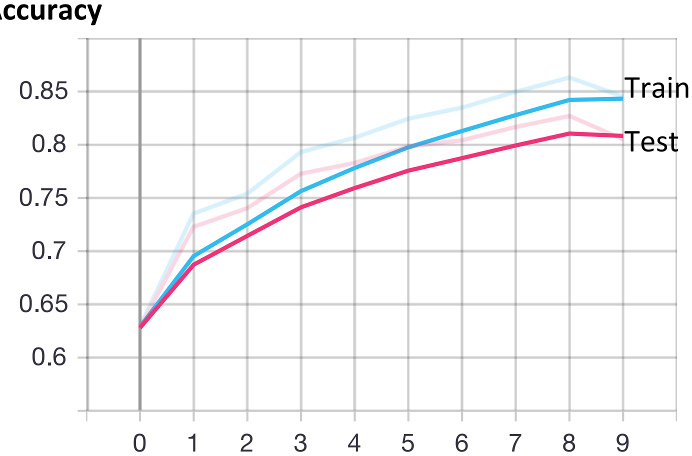
Code:

```
## Q4
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv4 = nn.Conv2d(64, 64, 3, padding=1)
        self.conv5 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv6 = nn.Conv2d(128, 128, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(128 * 4 * 4, 512)
        self.fc_q2 = nn.Linear(512, 512)
        self.fc2 = nn.Linear(512, 10)
        self.bnorm1d1 = nn.BatchNorm1d(512)
        self.bnorm1d2 = nn.BatchNorm1d(512)
        self.bnorm2d1 = nn.BatchNorm2d(32)
        self.bnorm2d2 = nn.BatchNorm2d(64)
        self.bnorm2d3 = nn.BatchNorm2d(128)
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        # Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift,https://arxiv.org/abs/1801.05134
        x = F.relu(self.conv1(x))
        x = self.bnorm2d1(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = self.dropout(x)
        x = F.relu(self.conv3(x))
        x = self.bnorm2d2(x)
        x = F.relu(self.conv4(x))
        x = self.pool(x)
        x = self.dropout(x)
        x = F.relu(self.conv5(x))
        x = self.bnorm2d3(x)
        x = F.relu(self.conv6(x))
        x = self.pool(x)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = self.bnorm1d1(x)
        x = F.relu(self.fc_q2(x))
        x = self.bnorm1d2(x)
        x = self.fc2(x)

        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
```

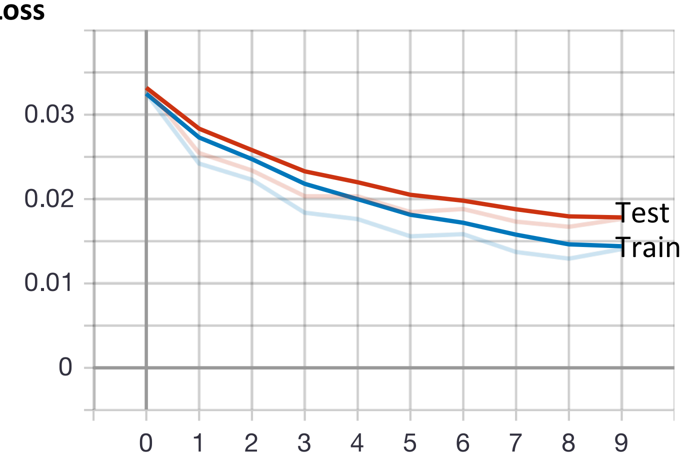
```
    num_features = 1
    for s in size:
        num_features *= s
    return num_features

# data augmentation for Q4
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(), # randomly flip and rotate
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

criterion = nn.CrossEntropyLoss()
# optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
optimizer = optim.Adam(net.parameters(), lr=0.001, amsgrad=True) # Q3 & Q4
```

Result:

Accuracy



Loss



CS535 DL Assignment3 Dongkyu Kim
Source Code

```
from __future__ import print_function
from __future__ import division
import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
# from tensorboardX import SummaryWriter  # for pytorch below 1.14
from torch.utils.tensorboard import SummaryWriter # for pytorch above or equal 1.14

# check if CUDA is available
train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
  print('CUDA is not available.  Training on CPU ...')
else:
  print('CUDA is available!  Training on GPU ...')

## Q4
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
    self.conv2 = nn.Conv2d(32, 32, 3, padding=1)
    self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
    self.conv4 = nn.Conv2d(64, 64, 3, padding=1)
    self.conv5 = nn.Conv2d(64, 128, 3, padding=1)
    self.conv6 = nn.Conv2d(128, 128, 3, padding=1)
    self.pool = nn.MaxPool2d(2, 2)
    self.fc1 = nn.Linear(128 * 4 * 4, 512)
    self.fc_q2 = nn.Linear(512, 512)
    self.fc2 = nn.Linear(512, 10)
    self.bnorm1d1 = nn.BatchNorm1d(512)
    self.bnorm1d2 = nn.BatchNorm1d(512)
    self.bnorm2d1 = nn.BatchNorm2d(32)
    self.bnorm2d2 = nn.BatchNorm2d(64)
    self.bnorm2d3 = nn.BatchNorm2d(128)
    self.dropout = nn.Dropout(0.25)

  def forward(self, x):
    # Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift,
https://arxiv.org/abs/1801.05134
    x = F.relu(self.conv1(x))
    x = self.bnorm2d1(x)
```

CS535 DL Assignment3 Dongkyu Kim

```
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = self.dropout(x)
        x = F.relu(self.conv3(x))
        x = self.bnorm2d2(x)
        x = F.relu(self.conv4(x))
        x = self.pool(x)
        x = self.dropout(x)
        x = F.relu(self.conv5(x))
        x = self.bnorm2d3(x)
        x = F.relu(self.conv6(x))
        x = self.pool(x)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = self.bnorm1d1(x)
        x = F.relu(self.fc_q2(x))
        x = self.bnorm1d2(x)
        x = self.fc2(x)

        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features

def eval_net(dataloader):
    correct = 0
    total = 0
    total_loss = 0
    net.eval() # Why would I do this?
    criterion = nn.CrossEntropyLoss(reduction='mean')
    for data in dataloader:
        images, labels = data
        images, labels = Variable(images).cuda(), Variable(labels).cuda()
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels.data).sum()
        loss = criterion(outputs, labels)
        total_loss += loss.item()
    net.train() # Why would I do this?
    return total_loss / total, correct.float() / total

if __name__ == "__main__":
    BATCH_SIZE = 32 #mini_batch size
```

CS535 DL Assignment3 Dongkyu Kim

```
MAX_EPOCH = 10 #maximum epoch to train

# transform = transforms.Compose(
#     [transforms.ToTensor(),
#      transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]) #torchvision.transforms.Normalize(mean, std)

# data augmentation for Q4
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(), # randomly flip and rotate
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=BATCH_SIZE,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
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')

print('Building model...')
net = Net().cuda()
# net = partially_restore_weights('mytraining1.pth') #Q2
net.train() # Why would I do this?

images, labels = next(iter(trainloader))
grid = torchvision.utils.make_grid([images.cuda()])

writer = SummaryWriter('part4')
writer.add_images('images', grid)
writer.add_graph(net, [images.cuda()])

criterion = nn.CrossEntropyLoss()
# optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
optimizer = optim.Adam(net.parameters(), lr=0.001, amsgrad=True) # Q3 & Q4

print('Start training...')
for epoch in range(MAX_EPOCH):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
```

```
            # get the inputs
            inputs, labels = data

            # wrap them in Variable
            inputs, labels = Variable(inputs).cuda(), Variable(labels).cuda()

            # 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 % 500 == 499:    # print every 2000 mini-batches
                print('    Step: %5d avg_batch_loss: %.5f' %
                      (i + 1, running_loss / 500))
                running_loss = 0.0
        print('    Finish training this EPOCH, start evaluating...')
        train_loss, train_acc = eval_net(trainloader)
        test_loss, test_acc = eval_net(testloader)
        print('EPOCH: %d train_loss: %.5f train_acc: %.5f test_loss: %.5f test_acc %.5f' %
              (epoch+1, train_loss, train_acc, test_loss, test_acc))

        #writer.add_scalar('train_loss', train_loss,epoch)
        #writer.add_scalar('train_acc', train_acc,epoch)
        #writer.add_scalar('test_loss', test_loss,epoch)
        #writer.add_scalar('test_acc', test_acc,epoch)

        writer.add_scalars('loss', {'train_loss': train_loss,
                                    'test_loss': test_loss},epoch)

        writer.add_scalars('acc', {'train_acc': train_acc,
                                   'test_acc': test_acc},epoch)


    # writer.close()
    print('Finished Training')
    print('Saving model...')
    torch.save(net.state_dict(), 'mytraining4.pth')
```