```
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


#mount google drive on colab for saving session variables
from google.colab import drive
import os
drive.mount('/content/gdrive/')
save_path = "/content/gdrive/My Drive/Colab Notebooks/DeepLearning19/HW2-CIPHAR10/models/CNN/"
data_path = "/content/gdrive/My Drive/Colab Notebooks/DeepLearning19/HW2-CIPHAR10/dataset"
```

```
Drive already mounted at /content/gdrive/; to attempt to forcibly remount, call drive.mount("/conte
```

```
#downloaded to drive by colab
DOWNLOAD = True

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (1.0,))])

trainset = torchvision.datasets.MNIST(root=data_path, train=True, download=DOWNLOAD, transform=transform
trainset_trans = torchvision.datasets.MNIST(root=data_path, train=True, download=DOWNLOAD, transform=tra
testset = torchvision.datasets.MNIST(root=data_path, train=False, download=DOWNLOAD, transform=transform

train_loader = torch.utils.data.DataLoader(trainset_trans, batch_size=5, shuffle=True) #batchsize 50 won
train_ac_loader = torch.utils.data.DataLoader(trainset, batch_size=200, shuffle=False)
test_loader = torch.utils.data.DataLoader(testset, batch_size=50, shuffle=False)
test_ac_loader = torch.utils.data.DataLoader(testset, batch_size=200, shuffle=True) #ammortizing error o
test_all_loader = torch.utils.data.DataLoader(testset, batch_size=10000, shuffle=False)

classes = list(range(10))

print(trainset.data.shape)
print(testset.data.shape)
```

```
torch.Size([60000, 28, 28])
torch.Size([10000, 28, 28])
```

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        #in, out, kernel, padding(both sides=>*2)
        self.conv1 = nn.Conv2d(1, 32, 5, padding=2) # 1 28 28 -> 32 28 28 -> 32 14 14
        self.conv2 = nn.Conv2d(32, 64, 5, padding=2)# 32 14 14 -> 64 14 14 -> 64 7 7
        self.pool = nn.MaxPool2d(2, 2, padding=0)
        self.fc1 = nn.Linear(64*7*7, 1024)          #64*7*7 -> 1024
        self.fc2 = nn.Linear(1024, 10)              #1024   -> 10
        self.dpout = nn.Dropout(p=0.5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64*7*7)
        x = F.relu(self.fc1(x))
```

```
        x = self.dpout(x)
        x = self.fc2(x)
        #print(x.shape) is important
        return x

    def forward_h(self, x):
        a_s = []
        h1 = F.relu(self.conv1(x)) # 32 28 28
        a_s.append(h1)
        p1 = self.pool(h1)  # 32 14 14
        h2 = F.relu(self.conv2(p1)) #64 14 14
        a_s.append(h2)
        p2 = self.pool(h2) # 64 7 7
        _hidden = p2.view(-1, 64*7*7)
        _hidden = F.relu(self.fc1(_hidden))
        y = self.fc2(_hidden)
        a_s.append(y)
        return a_s
#testmodel = CNN()
#testmodel(testset.data[0:2].float().reshape((-1, 1, 28, 28))).shape
```

```
torch.Size([2, 10])
```

```
def calc_accuracy(model, loader, device):
    dataiter = iter(loader)
    data, target  = dataiter.next()
    data, target = data.to(device), target.to(device)
    output = model(data)
    max_index = output.max(dim = 1)[1]
    accuracy = (max_index == target).sum().item()/len(target)
    return accuracy

def train(model, device, train_loader, criterion, optimizer, epoch, model_path = None, save_name = None,
    rec = {
        "loss": [],
        "train_ac": [],
        "test_ac": []
    }
    model.train(mode=True)
    batch_size = train_loader.batch_size
    show_frequency = 1000 #show per x data
    for ep_i in range(epoch):
        correct_prediction = 0
        current_loss = 0 #weighted with batch size
        for batch_idx, (data, target) in enumerate(train_loader):
            data, target = data.to(device), target.to(device)
            #forward+backpro
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target) #criterion
            correct_prediction += ((output.max(dim = 1)[1] == target).sum().item())
            current_loss += loss.item()*batch_size
            loss.backward()
            optimizer.step()
            #show statistics
            if batch_idx*batch_size % show_frequency == 0 and batch_idx!=0:
                train_loss = current_loss/show_frequency
                current_loss = 0
                train_accuracy = correct_prediction/show_frequency
                correct_prediction = 0
                test_accuracy = calc_accuracy(model, test_ac_loader, device)
                rec["loss"].append(train_loss)
                rec["train_ac"].append(train_accuracy)
```

```
                rec["train_ac"].append(train_accuracy)
                rec["test_ac"].append(test_accuracy)
                print('\repoch {}, interation {}, batchsize {} : loss = {}, train_ac = {}, test_ac = {}'
    print('\nFinished Training.')
    model.eval()
    #saving
    if model_path is not None:
        torch.save(model.state_dict(), model_path+save_name)
        print('Saved model parameters to {}.'.format(model_path))

    return rec


#################################chooese to init or load#################################################
model_path = "/content/gdrive/My Drive/Colab Notebooks/DeepLearning19/HW2-CIPHAR10/models/CNN/"

INIT = True
LOAD = False
SAVE = True
#init
#########################################################################################################
if INIT:
    rec = {
        "loss": [],
        "train_ac": [],
        "test_ac": []
    }
    model = CNN()
    print("Initialized New Network")
#load
#########################################################################################################
if LOAD:
    load_path = model_path
    load_model_name = 'ministl2-cnn-model.pt'
    model = CNN()
    model.load_state_dict(torch.load(load_path+load_model_name))
    print('Loaded model parameters from disk.')
    import pickle
    with open(load_path+"rec.pkl", "rb") as fo:
        rec = pickle.load(fo)
    print('Loding tranning records from disk.')

if SAVE:
  print('Save Enabled.')
  save_name = 'minist-cnn-model.pt'
  rec_name = "rec-ministl2-1e-3.pkl"
  save_path = model_path
else:
  print('Save NOT!! Enabled.')
  save_name = None
  save_path = None
```

```
Initialized New Network
Save Enabled.
```

```
#################################use gpu if can  ###################################################
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)
print("model device : ", device)
```

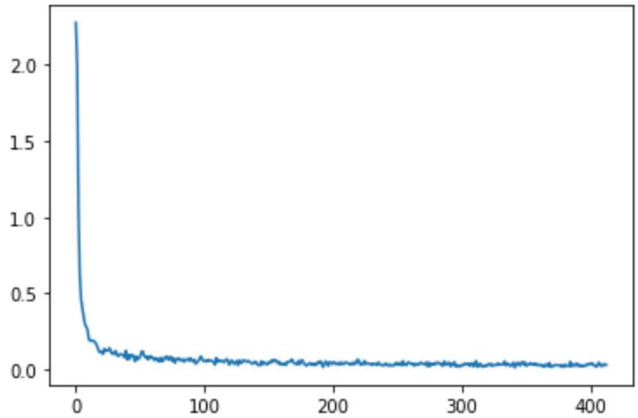
```
model device :  cuda:0
```

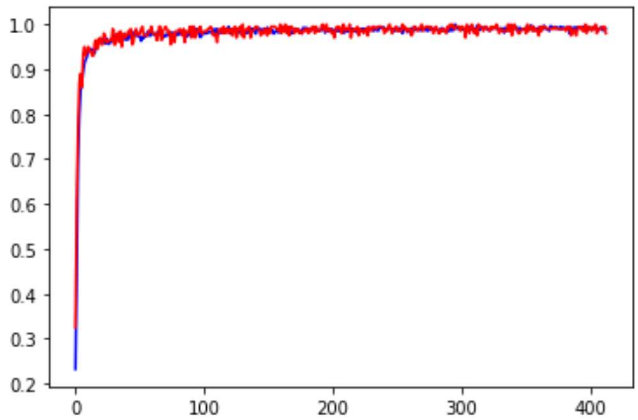
```
##train +save to disk"
#tf version iteration * batch = 80000*50 => 7epoch
```

```
#tf version iteration = batch = 60000/50 => 1epoch
########################################################################################################
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9, weight_decay=0.001) #L2 0.01 underfit

new_rec = {"loss": [],"train_ac": [],"test_ac": []}
new_rec = train(model, device, train_loader, criterion, optimizer, epoch=5, model_path=save_path, save_n
rec["loss"].extend(new_rec["loss"])
rec["train_ac"].extend(new_rec["train_ac"])
rec["test_ac"].extend(new_rec["test_ac"])
if SAVE:
    import pickle
    with open(model_path+rec_name, "wb") as fo: #wb
        pickle.dump(rec,fo)
        print('Saved extended tranning records to disk.')
```

```
epoch 4, interation 11800, batchsize 5 : loss = 0.031851229297450344, train_ac = 0.99, test_ac = 0.
Finished Training.
Saved model parameters to /content/gdrive/My Drive/Colab Notebooks/DeepLearning19/HW2-CIPHAR10/mode
Saved extended tranning records to disk.
```

```
def plot_curve(x):
    plt.figure()
    plt.plot(x)
    plt.show()
plot_curve(rec['loss'])
#plot_curve(rec['train_ac'])
#plot_curve(rec['test_ac'])
plt.figure()
plt.plot(rec['train_ac'], color='blue')
plt.plot(rec['test_ac'], color='red')
plt.show()
```






```
calc_accuracy(model, test_all_loader, device)
```

```
0.0026
```

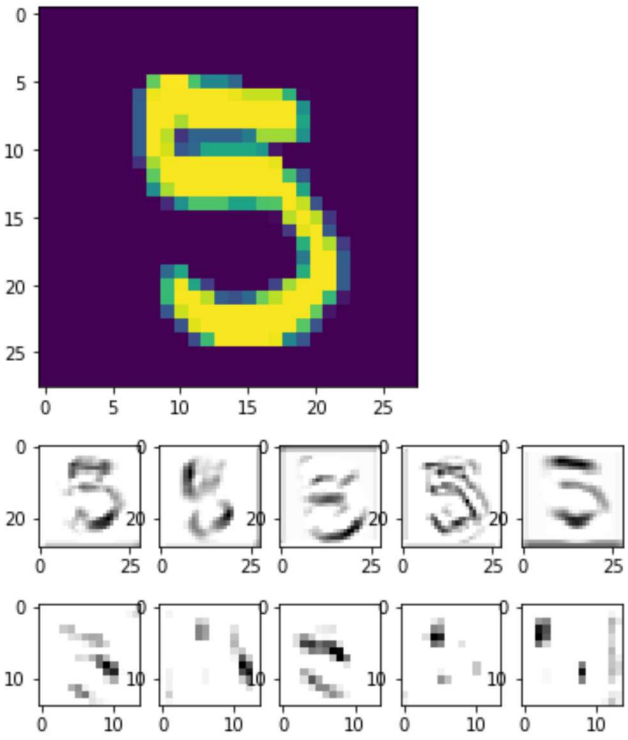
```
total = 0
print('Tainable Valuables:')
for name, param in model.named_parameters():
    if param.requires_grad:
        print(name, '\t', param.numel())
        total += param.numel()
print('\nTotal', '\t', total)
```

```
Tainable Valuables:
conv1.weight     800
conv1.bias       32
conv2.weight     51200
conv2.bias       64
fc1.weight       3211264
fc1.bias         1024
fc2.weight       10240
fc2.bias         10

Total    3274634
```

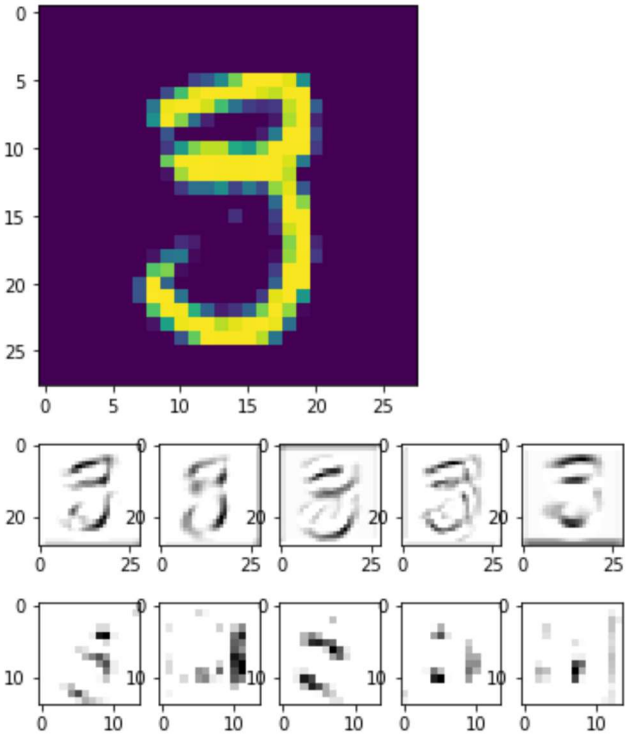
```
def show_hidden(model, device, data, idx, n_show_feature): #data.shape = torch.Size()
    #assert(data.shape == torch.Size([28,28]))
    data = data.to(device)
    a_s = model.forward_h(data)
    h1 = a_s[0].detach().cpu().numpy()[idx]
    h2 = a_s[1].detach().cpu().numpy()[idx]

    for i in range(1, 1+n_show_feature):
        #f, axs = plt.subplots(1,n_show_feature,figsize=(1,1))
        plt.subplot(1,n_show_feature,i)
        plt.imshow(h1[i],cmap=plt.cm.binary)
    plt.show()
    for i in range(1, 1+n_show_feature):
        plt.subplot(1,n_show_feature ,i)
        plt.imshow(h2[i],cmap=plt.cm.binary)
    plt.show()
    return


def show_by_mismatch():
    test_one_loader = torch.utils.data.DataLoader(testset, shuffle=True)
    for idx, (data,target) in enumerate(test_one_loader):
        output = model(data.to(device))
        prediction = output.max(dim = 1)[1].cpu()
        #print(target)
        if(prediction != target):
          print("find mismatch @ {}".format(idx))
          print("label: {}".format(classes[target]))
          print("prediction: {}".format(classes[prediction]))
          print(classes)
          print(F.softmax(output, dim=1))
          plt.figure()
          plt.imshow(data[0][0],cmap=plt.cm.binary)
          plt.show()
          show_hidden(model, device, data, 0, 5)
          return prediction, target

def show_by_class(cls):
    test_one_loader = torch.utils.data.DataLoader(testset, shuffle=True)
    for idx, (data,target) in enumerate(test_one_loader):
        if cls == target:
            print("find sample of {} @ {}".format(classes[cls], idx))
            plt.figure()
```

```
            plt.figure()
            plt.imshow(data[0][0],cmap=plt.cm.binary)
            plt.show()
            show_hidden(model, device, data, 0, 5)
            return


pred, tar = show_by_mismatch()
show_by_class(pred)
```

```
find mismatch @ 31
label: 5
prediction: 3
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
tensor([[1.2061e-06, 5.9835e-06, 4.4336e-08, 6.1830e-01, 6.5437e-07, 3.7389e-01,
         1.1896e-06, 1.0466e-05, 1.2395e-04, 7.6585e-03]], device='cuda:0',
       grad_fn=<SoftmaxBackward>)
```

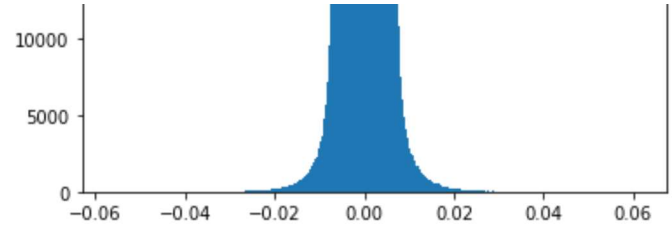


```
find sample of 3 @ 5
```

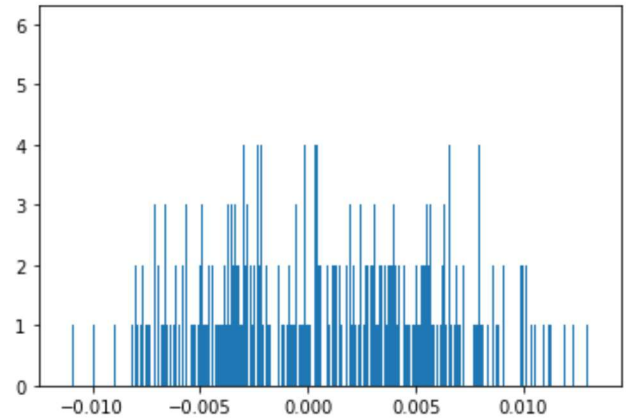


```
W_all = np.array([])
for name, param in model.named_parameters():
    if param.requires_grad:
        print(name, '\t', param.numel())
        wi_flatten = param.cpu().detach().numpy().flatten()
        W_all = np.r_[W_all, wi_flatten]
        #print(param.cpu().detach().numpy())
        plt.figure()
```

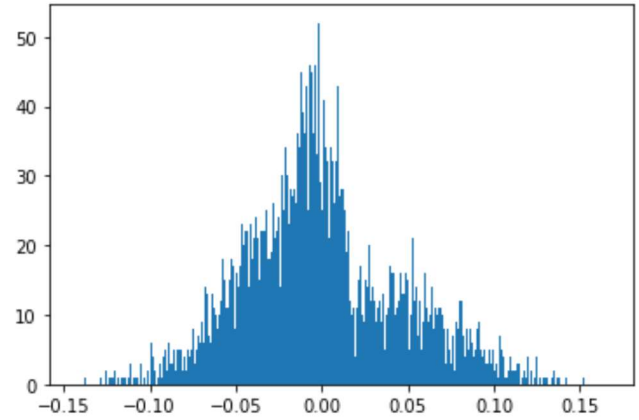
```
            plt.hist(w1_flatten,1000)
            plt.show()

    plt.figure()
    plt.hist(W_all,1000)
    plt.show()
```




```
fc1.bias          1024
```




```
fc2.weight        10240
```




```
fc2.bias          10
```