```
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
#data augmemntation (training accuracy is expected to be lower)
transform_train = transforms.Compose(
    [
     transforms.RandomHorizontalFlip(),
     transforms.RandomGrayscale(),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

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

trainset = torchvision.datasets.CIFAR10(root=data_path, train=True, download=False, transform=transform)
trainset_trans = torchvision.datasets.CIFAR10(root=data_path, train=True, download=DOWNLOAD, transform=t
testset = torchvision.datasets.CIFAR10(root=data_path, train=False, download=DOWNLOAD, transform=transfo

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 = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')
print(trainset.data.shape)
print(testset.data.shape)
```

```
Files already downloaded and verified
Files already downloaded and verified
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        #in, out, kernel
        self.conv1 = nn.Conv2d(3,   64,  3) #3 32 32, 64 30 30
```

```
        self.conv2 = nn.Conv2d(64,  128, 3) #64 15 15, 128 15 15
        self.conv3 = nn.Conv2d(128, 256, 3) #128 7 7, 256 4 4
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(256*2*2, 128) #256 2 2, 128
        self.fc2 = nn.Linear(128, 256)     #128 256
        self.fc3 = nn.Linear(256, 10)      #256 10

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 256*2*2)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def forward_h(self, x):
        #print(x.shape)
        a_s = []
        h1 = F.relu(self.conv1(x)) # 3 32 32 64 30 30
        a_s.append(h1)
        p1 = self.pool(h1)  # 64 15 15
        h2 = F.relu(self.conv2(p1))
        a_s.append(h2)
        p2 = self.pool(h2) # 128 6 6
        h3 = F.relu(self.conv3(p2))
        a_s.append(h3)
        p3 = self.pool(h3) # 256 2 2
        _hidden = p3.view(-1, 256*2*2)
        _hidden = F.relu(self.fc1(_hidden))
        _hidden = F.relu(self.fc2(_hidden))
        y = self.fc3(_hidden)
        a_s.append(y)
        #x = F.log_softmax(x, dim=1)
        return a_s
#testmodel = CNN()
#testmodel(torch.Tensor((np.transpose(testset.data[0:10], (0, 3, 1, 2))))).shape


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
```

```
            #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["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


#test model code
#model = CNN()
#model(torch.Tensor((np.transpose(testset.data[0:10], (0, 3, 1, 2))))).shape


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

INIT = False
LOAD = True
SAVE = False
#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 = 'cifar-10-cnn-model2.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 = "cifar-10-cnn-model-small-l2-1e-3.pt"
  rec_name = "rec2-small-no-l2-1e-3.pkl"
  save_path = model_path
else:
  print('Save NOT!! Enabled.')
  save_name = None
  save_path = None
```

```
Loaded model parameters from disk.
Loding tranning records from disk.
Save NOT!! 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"
#####################################################################################################
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=20, model_path=save_path, save_
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.')
```

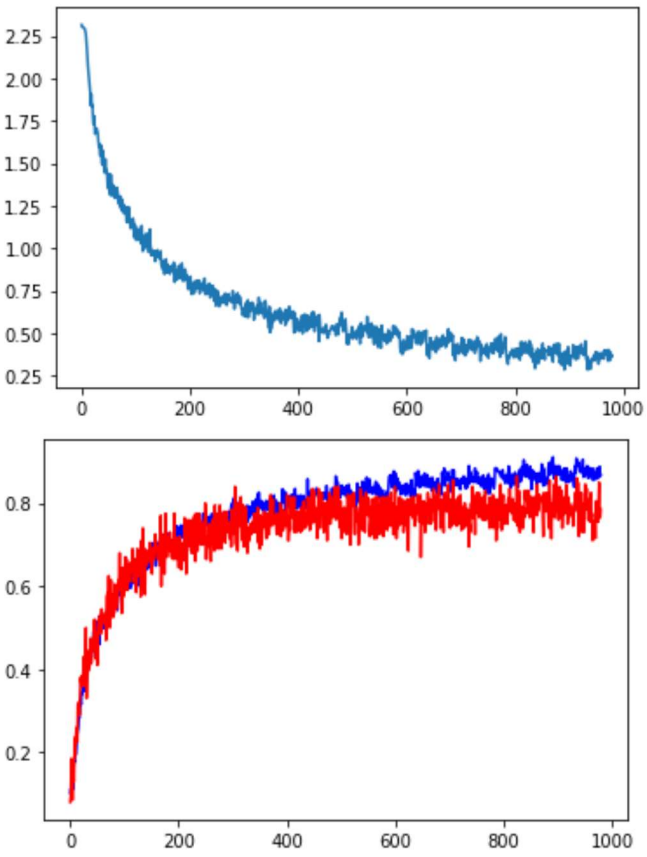
```
epoch 19, interation 9800, batchsize 5 : loss = 0.36941475576910304, train_ac = 0.868, test_ac = 0.
Finished Training.
Saved model parameters to /content/gdrive/My Drive/Colab Notebooks/DeepLearning19/HW2-CIPHAR10/mode
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-9-6c97be6c15b9> in <module>()
      9 if SAVE:
     10     import pickle
---> 11     with open(model_path+rec_name) as fo:
     12         pickle.dump(rec,fo)
     13         print('Saved extended tranning records to disk.')

FileNotFoundError: [Errno 2] No such file or directory: '/content/gdrive/My Drive/Colab Notebooks/D

SEARCH STACK OVERFLOW
```

```
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.')
```

```
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()
```



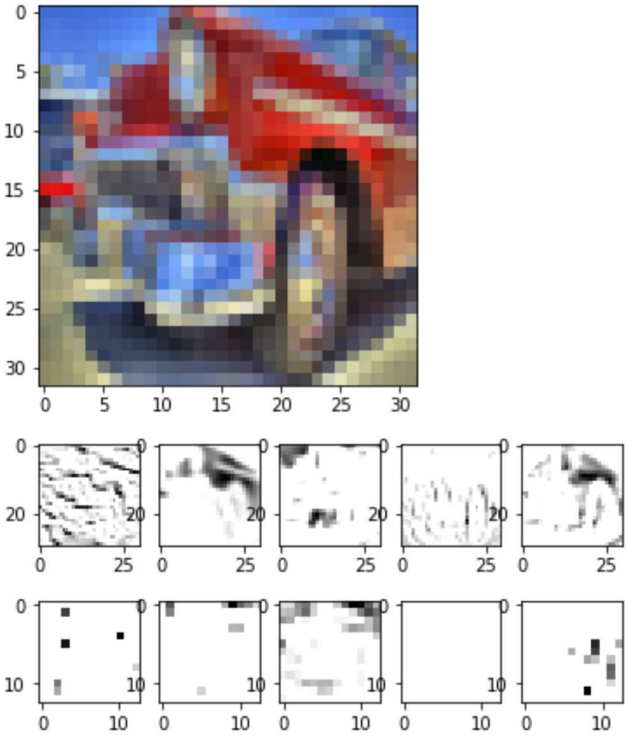

```
calc_accuracy(model, test_all_loader, device)
```

```
0.7808
```

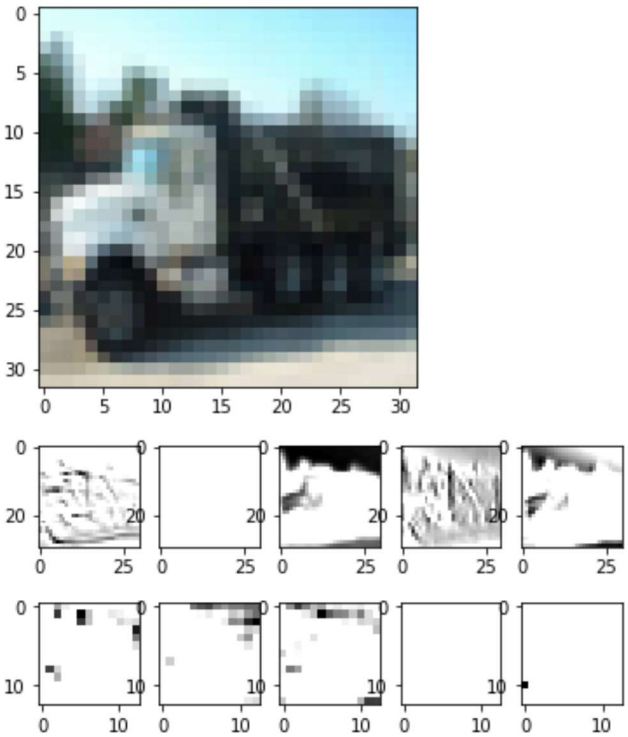
```
total = 0
print('Tainable Valuables:')
for name, param in model.named_parameters():
    if param.requires_grad:
        print(name, '\t', param.numel())
        total += param.numel()
print('\nTotal', '\t', total)
```

```
     Tainable Valuables:
def imshow(img): #for 1 data
    # [-1,1] to [0,1]
    img = img / 2 + 0.5
    if not (type(img).__module__ == np.__name__):
      img = img.numpy()
    # [3,32,32] to [32,32,3]
    plt.imshow(np.transpose(img, (1, 2, 0)))
    plt.show()

def show_hidden(model, device, data, idx, n_show_feature): #data.shape = torch.Size()
    assert(data.shape == torch.Size([1,3,32,32]))
    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))
          imshow(data[0])
          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))
            imshow(data[0])
            show_hidden(model, device, data, 0, 5)
            return

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

```
find mismatch @ 3
label: car
prediction: truck
('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
tensor([[2.8678e-05, 5.3554e-03, 1.3828e-06, 6.2562e-06, 5.1131e-07, 9.0881e-06,
         8.7917e-06, 4.0766e-04, 6.0373e-05, 9.9412e-01]], device='cuda:0',
       grad_fn=<SoftmaxBackward>)
```



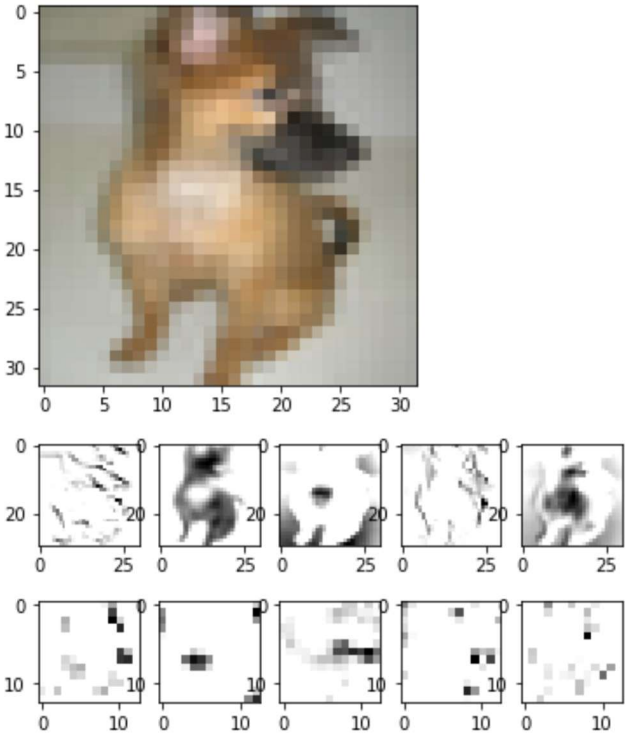
```
find sample of truck @ 5
```



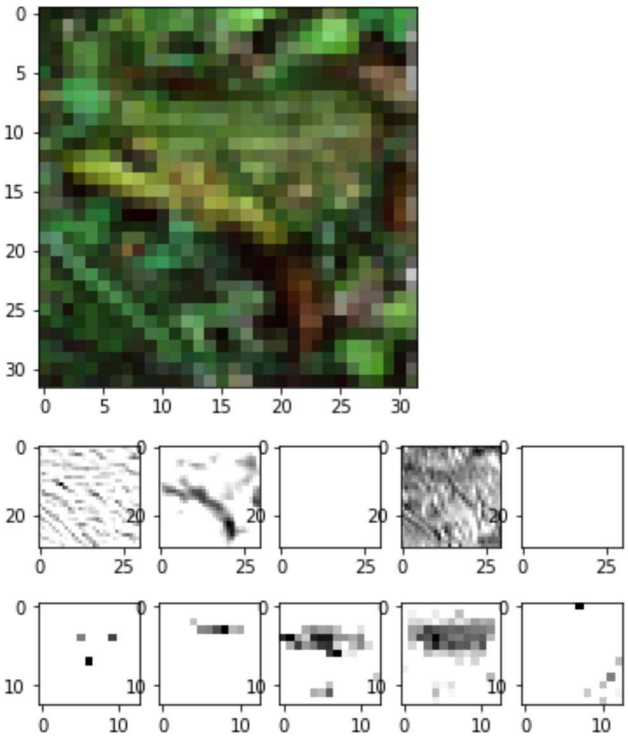
```
def show_by_mismatch():
    test_one_loader = torch.utils.data.DataLoader(testset, shuffle=True)
    for idx, (data,target) in enumerate(test_one_loader):
        output = model(data.to(device))
        prediction = output.max(dim = 1)[1].cpu()
        #print(target)
        if(prediction != target):
          print("find mismatch @ {}".format(idx))
```

```
            print("label: {}".format(classes[target]))
            print("prediction: {}".format(classes[prediction]))
            print(classes)
            print(F.softmax(output, dim=1))
            imshow(data[0])
            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))
                imshow(data[0])
                show_hidden(model, device, data, 0, 5)
                return

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

```
find mismatch @ 3
label: dog
prediction: frog
('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
tensor([[4.2304e-04, 9.5143e-04, 5.2991e-02, 9.1608e-02, 6.4651e-02, 1.4262e-01,
         6.4220e-01, 4.0775e-03, 2.2732e-04, 2.5478e-04]], device='cuda:0',
       grad_fn=<SoftmaxBackward>)
```



```
find sample of frog @ 22
```

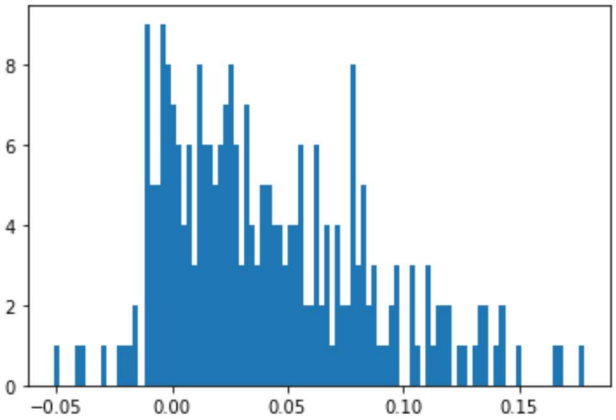


```
CNN()
```

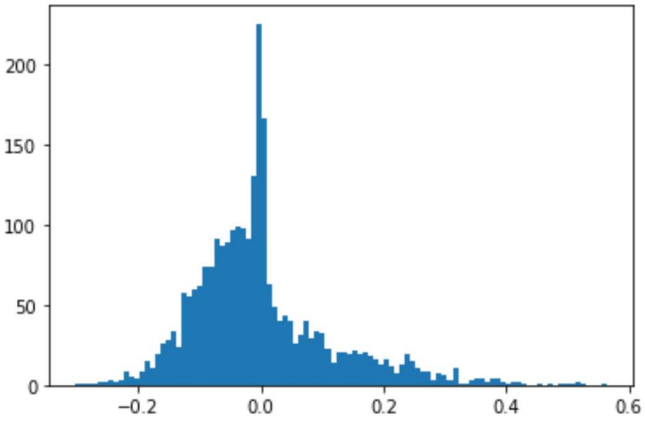
```
CNN(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1024, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=10, bias=True)
)
```

```
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(wi_flatten,100)
        plt.show()

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

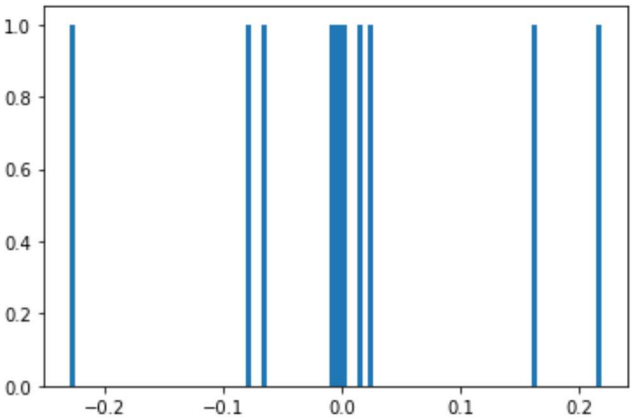

```
fc2.bias          256
```

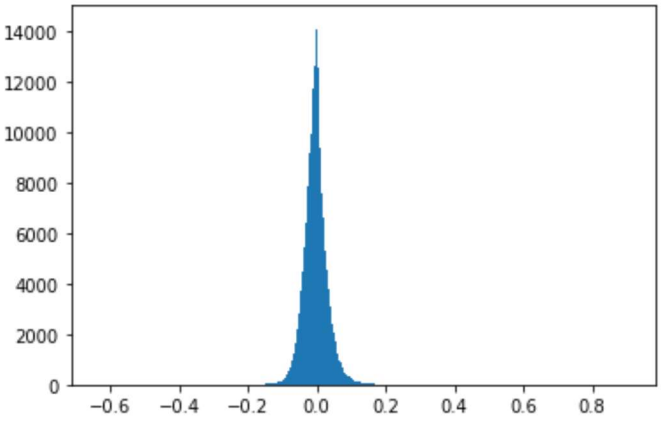



```
fc3.weight        2560
```



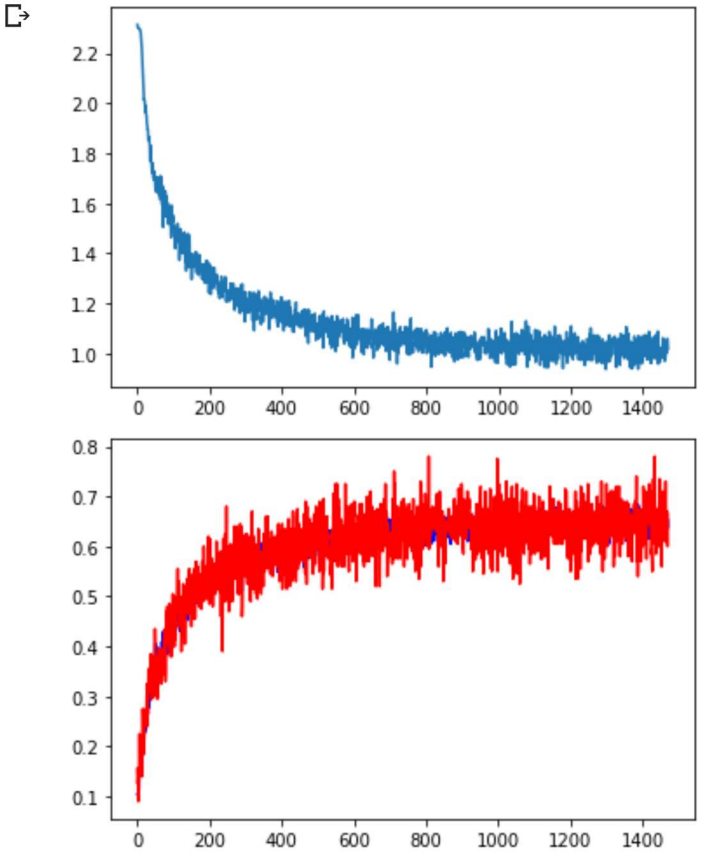

```
fc3.bias          10
```





```
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()
```




```
def imshow(img): #for 1 data
    # [-1,1] to [0,1]
    img = img / 2 + 0.5
    if not (type(img).__module__ == np.__name__):
      img = img.numpy()
    # [3,32,32] to [32,32,3]
    plt.imshow(np.transpose(img, (1, 2, 0)))
    plt.show()

def show_hidden(model, device, data, idx, n_show_feature): #data.shape = torch.Size()
    assert(data.shape == torch.Size([1,3,32,32]))
    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))
              imshow(data[0])
              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))
                imshow(data[0])
                show_hidden(model, device, data, 0, 5)
                return


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