```
import torchvision
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from sklearn.model_selection import train_test_split

import json
from utils import visualizeWeights

BATCH_SIZE=50
epoch_size = 15
TRAIN_SIZE=10
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
# customize Transform variable is to take input and return a tensor object
# Also by setting normalizer, I scaled pixel values between -1 and 1
#directly taken from
https://medium.com/@aayushbt/fashion-mnist-data-training-using-pytorch-7f6ad71e96f4
cusTransform =
transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5,),(0.5,),)])

# training set
train_data = torchvision.datasets.FashionMNIST('./data', train = True, download = False,
transform = cusTransform)
#Splitting data
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split
.html
train_set, val_set = train_test_split(train_data, test_size=0.1, random_state=42)
# test set
test_data = torchvision.datasets.FashionMNIST('./data', train = False,
transform = cusTransform)

val_generator = torch.utils.data.DataLoader(val_set, batch_size = BATCH_SIZE, shuffle =
False)


# example mlp classifier
class mlp1(torch.nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(mlp1, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Linear(input_size, hidden_size)
        self.fc2 = torch.nn.Linear(hidden_size, num_classes)
        self.relu = torch.nn.ReLU()
    def forward(self, x):
        x = x.view(-1, self.input_size)
        hidden = self.fc1(x)
        relu = self.relu(hidden)
        output = self.fc2(relu)
        return output
# example mlp2 classifier
class mlp2(torch.nn.Module):
    def __init__(self, input_size, hidden_size, hidden_size2, num_classes):
        super(mlp2, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Linear(input_size, hidden_size)
        self.fc2 = torch.nn.Linear(hidden_size, hidden_size2,bias=False)
        self.fc3 = torch.nn.Linear(hidden_size2, num_classes)
        self.relu = torch.nn.ReLU()
    def forward(self, x):
        x = x.view(-1, self.input_size)
        hidden = self.fc1(x)
        relu = self.relu(hidden)
        hidden2 = self.fc2(relu)
        output = self.fc3(hidden2)
        return output
class cnn_3(torch.nn.Module):
    #Layer Definition

#https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-netw
ork-cnn/
    def __init__(self,input_size,num_classes):
        super(cnn_3, self).__init__()
        self.input_size = input_size
        #in_channel = input_size ม 0 ม
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        self.fc1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3,
stride=1, padding='valid')
        self.relu1 = torch.nn.ReLU()
        self.fc2 = torch.nn.Conv2d(in_channels=16, out_channels=8, kernel_size=7,
stride=1, padding='valid')
        self.relu2 = torch.nn.ReLU()
        self.maxpool1 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride=2,padding=0)
        self.fc3 = torch.nn.Conv2d(in_channels=8, out_channels=16, kernel_size=5,
stride=1, padding='valid')
        self.maxpool2 = torch.nn.MaxPool2d(kernel_size=(2,2), stride=2,padding=0)

#https://stackoverflow.com/questions/53580088/calculate-the-output-size-in-convolution-laye
r
        self.fc4 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #It didin't work ??????
        #x = x.view(-1, self.input_size)
        hidden1 = self.fc1(x)
        relu1 = self.relu1(hidden1)
        hidden2 = self.fc2(relu1)
        relu2 = self.relu2(hidden2)
        pool1 = self.maxpool1(relu2)
        hidden3 = self.fc3(pool1)
        pool2 = self.maxpool2(hidden3)
        #Reshaping linear input
        pool2=pool2.view(BATCH_SIZE,144)
        output = self.fc4(pool2)
        return output


class cnn_4(torch.nn.Module):
    #Layer Definition

#https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-netw
ork-cnn/
    def __init__(self,input_size,num_classes):
        super(cnn_4, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3,
stride=1, padding='valid')
        self.relu1 = torch.nn.ReLU()
        self.fc2 = torch.nn.Conv2d(in_channels=16, out_channels=8, kernel_size=5,
stride=1, padding='valid')
        self.relu2 = torch.nn.ReLU()
        self.fc3 = torch.nn.Conv2d(in_channels=8, out_channels=8, kernel_size=3,
stride=1, padding='valid')
        self.relu3 = torch.nn.ReLU()
        self.maxpool1 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2)
        self.fc4 = torch.nn.Conv2d(in_channels=8, out_channels=16, kernel_size=5,
stride=1, padding='valid')
        self.relu4 = torch.nn.ReLU()
        self.maxpool2 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2)
        self.fc5 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #x = x.view(-1, self.input_size)
        #It didin't work ??????
        hidden1 = self.fc1(x)
        relu1 = self.relu1(hidden1)
        hidden2 = self.fc2(relu1)
        relu2 = self.relu2(hidden2)
        hidden3 = self.fc3(relu2)
        relu3 = self.relu3(hidden3)
        pool1 = self.maxpool1(relu3)
        hidden4 = self.fc4(pool1)
        relu4 = self.relu4(hidden4)
        pool2 = self.maxpool2(relu4)
        pool2=pool2.view(50,144)
        #Reshaping linear input
        output = self.fc5(pool2)
        return output
class cnn_5(torch.nn.Module):
    #Layer Definition
```

```
#https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-network-cnn/
    def __init__(self,input_size,num_classes):
        super(cnn_5, self).__init__()
        self.input_size = input_size
        self.fc1 = torch.nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding='valid')
        self.relu1 = torch.nn.ReLU()
        self.fc2 = torch.nn.Conv2d(in_channels=16, out_channels=8, kernel_size=3, stride=1, padding='valid')
        self.relu2 = torch.nn.ReLU()
        self.fc3 = torch.nn.Conv2d(in_channels=8, out_channels=8, kernel_size=3, stride=1, padding='valid')
        self.relu3 = torch.nn.ReLU()
        self.fc4 = torch.nn.Conv2d(in_channels=8, out_channels=8, kernel_size=3, stride=1, padding='valid')
        self.relu4 = torch.nn.ReLU()
        self.maxpool4 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2)
        self.fc5 = torch.nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3, stride=1, padding='valid')
        self.relu5 = torch.nn.ReLU()
        self.fc6 = torch.nn.Conv2d(in_channels=16, out_channels=16, kernel_size=3, stride=1, padding='valid')
        self.relu6 = torch.nn.ReLU()
        self.maxpool6 = torch.nn.MaxPool2d(kernel_size=(2, 2), stride= 2)
        self.fc7 = torch.nn.Linear(in_features=144, out_features=num_classes)
    def forward(self, x):
        #x = x.view(-1, self.input_size)
        #It didin't work ??????
        hidden1 = self.fc1(x)
        relu1 = self.relu1(hidden1)
        hidden2 = self.fc2(relu1)
        relu2 = self.relu2(hidden2)
        hidden3 = self.fc3(relu2)
        relu3 = self.relu3(hidden3)
        hidden4 = self.fc4(relu3)
        relu4 = self.relu4(hidden4)
        pool4 = self.maxpool4(relu4)
        #pool4=pool4.view()
        hidden5 = self.fc5(pool4)
        relu5 = self.relu5(hidden5)
        hidden6 = self.fc6(relu5)
        relu6 = self.relu6(hidden6)
        pool6 = self.maxpool6(relu6)
        #Reshaping linear input
        pool6=pool6.view(50,144)
        output = self.fc7(pool6)
        return output
#Dictionary for json
dictonary ={
    'name': None,
    'loss_curve':None,
    'train_acc_curve':None,
    'val_acc_curve':None,
    'test_acc':None,
    'weights':None,
    }
#Model Types

models=['mlp_1','mlp_2','cnn_3','cnn_4','cnn_5']
for modelselected in models:
    print(f"Training is started for model {modelselected}")
    train_losses_total=[]
    train_accus_total=[]
    valid_accus_total=[]
    weight_best=None
    maxPerformanceTask=0
    for stepX in range(TRAIN_SIZE):
        print(f"Step {stepX+1} is started")
        if modelselected == 'mlp_1':
            model_mlp = mlp1(784,64,10).to(device)
        elif modelselected == 'mlp_2':
            model_mlp= mlp2(784,16,64,10).to(device)
        elif modelselected == 'cnn_3':
```

```
            model_mlp = cnn_3(784,10).to(device)
        elif modelselected == 'cnn_4':
            model_mlp = cnn_4(784,10).to(device)
        elif modelselected == 'cnn_5':
            model_mlp = cnn_5(784,10).to(device)
        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model_mlp.parameters(), lr = 0.01)

        #Recorded values for each try
        train_losses=[]
        train_accus=[]
        valid_accus=[]
        test_accus=[]
        for epoch in range(epoch_size):
            print(f"Epoch is {epoch+1}/{epoch_size}")
            total=0
            correct=0
            train_generator = torch.utils.data.DataLoader(train_set, batch_size =
BATCH_SIZE, shuffle = True)
            total_step = len(train_generator)
            #https://stackoverflow.com/questions/62833157/cnn-model-using-pytorch
            #Train DATA

#https://androidkt.com/calculate-total-loss-and-accuracy-at-every-epoch-and-plot-using-matp
lotlib-in-pytorch/
            for i, (images, labels) in enumerate(train_generator):
                model_mlp.train()
                # Move tensors to the configured device
                images = images.to(device)
                labels = labels.to(device)
                # Forward pass
                outputs = model_mlp(images)
                loss = criterion(outputs, labels.to(device))

                # Backward and optimize
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()


                if (i+1) % 10 == 0:
                    model_mlp.eval()
                    #Train calculation
                    #Directly taken from
https://discuss.pytorch.org/t/how-does-one-get-the-predicted-classification-label-from-a-py
torch-model/91649/3
                    _, tra_pred=outputs.max(1)
                    tra_size = labels.size(0)
                    tra_corr= tra_pred.eq(labels).sum().item()
                    tra_acc=tra_corr/tra_size
                    train_acc=tra_acc*100
                    running_loss = loss.item()
                    val_total=0
                    val_correct=0
                    #Valid
                    for j, (val_images, val_labels) in enumerate(val_generator):
                        # Accuracy Calculation
                        #
https://androidkt.com/calculate-total-loss-and-accuracy-at-every-epoch-and-plot-using-matpl
otlib-in-pytorch/
                        val_images = val_images.to(device)
                        val_labels = val_labels.to(device)
                        # Forward pass
                        val_outputs = model_mlp(val_images)
                        _, val_predicted = val_outputs.max(1)
                        val_total += val_labels.size(0)
                        val_correct += val_predicted.eq(val_labels).sum().item()
                    val_accu=(val_correct/ val_total)*100
                    #print ('Step {} Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Train
Accuracy {:.4f}, Val Accuracy {:.4f} '
                    #       .format(stepX,epoch+1, epoch_size, i+1, total_step,
running_loss,train_acc,val_accu))
                    train_losses.append(running_loss)
                    train_accus.append(train_acc)
```

```
                    valid_accus.append(val_accu)
            train_losses_total.append(train_losses)
            train_accus_total.append(train_accus)
            valid_accus_total.append(valid_accus)
            #print(train_losses_total)

            #Test
            with torch.no_grad():
                test_generator = torch.utils.data.DataLoader(test_data, batch_size =
BATCH_SIZE, shuffle = False)
                model_mlp.eval()
                correct = 0
                total = 0
                for images, labels in test_generator:
                    images = images.to(device)
                    labels = labels.to(device)
                    outputs = model_mlp(images)
                    _, predicted = outputs.max(1)
                    total += labels.size(0)
                    correct += predicted.eq(labels).sum().item()
                test_accu =(correct/total)*100
                test_accus.append(test_accu)
                if(test_accu > maxPerformanceTask):
                    maxPerformanceTask=test_accu
                    model_mlp.to('cpu')
                    weight_best=model_mlp.fc1.weight.data.numpy()
                    model_mlp.to(device)

            print('For Step {} It is finished'.format(stepX+1))
        #https://www.geeksforgeeks.org/python-column-wise-sum-of-nested-list/
        average_train_losses = [sum(sub_list) / len(sub_list) for sub_list in
zip(*train_losses_total)]
        average_train_accu = [sum(sub_list) / len(sub_list) for sub_list in
zip(*train_accus_total)]
        average_valid_accu = [sum(sub_list) / len(sub_list) for sub_list in
zip(*valid_accus_total)]


        #Dictionary for json
        dictonary ={
              'name': modelselected,
              'loss_curve':average_train_losses,
              'train_acc_curve':average_train_accu,
             'val_acc_curve':average_valid_accu,
             'test_acc':maxPerformanceTask,

'weights':weight_best.tolist(),#https://stackoverflow.com/questions/26646362/numpy-array-is
-not-json-serializable
             }
        #JSON Writing a file (geeksforgeeks.com)
        with open("q2_"+modelselected+".json","w") as outfile:
            json.dump(dictonary,outfile)

        visualizeWeights(weight_best, save_dir='resultQ2',
filename='input_weights_'+modelselected)
```