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


BATCH_SIZE=50
epoch_size =20
#Faster
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/@aaysbt/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 = True,
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




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 mı 0 mı
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

#Model Types
train_losses_total=[]
valid_accus_total=[]
lrs=[0.1,0.01,0.001] #Learning Rates
```

```
for LR in lrs:
    print(f"Training is started for Learining Rate {LR}") # Printing LR state
    model_mlp = cnn_3(784,10).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model_mlp.parameters(), lr = LR, momentum=0.00)#Setting LR
    #Recorded values for each try
    train_losses=[]
    valid_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
                running_loss = loss.item()
                val_generator = torch.utils.data.DataLoader(val_set, batch_size =
BATCH_SIZE, shuffle = False)
                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
                train_losses.append(running_loss)
                valid_accus.append(val_accu)
    train_losses_total.append(train_losses)
    valid_accus_total.append(valid_accus)
    #Dictionary for json
dictonary ={
     'name': 'cnn3',
     'loss_curve_1':train_losses_total[0],
     'loss_curve_01':train_losses_total[1],
     'loss_curve_001':train_losses_total[2],
     'val_acc_curve_1':valid_accus_total[0],
     'val_acc_curve_01':valid_accus_total[1],
     'val_acc_curve_001':valid_accus_total[2],
```

```
    }
#Recording Results
with open("Q4cnn3_1.json","w") as outfile:
    json.dump(dictonary,outfile)
```