ImageClassification

October 26, 2020

[1]:

```
import numpy as np
import matplotlib.pyplot as plt
import time
import random
from sklearn import metrics
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from skimage import io
import os
import cv2
import pandas as pd
```

[2]:

```
train_set = torchvision.datasets.FashionMNIST(root='.', download=True,train=True)
train_image = np.array(train_set.data)
train_label = np.array(train_set.targets)
class_name = train_set.classes
test_set = torchvision.datasets.FashionMNIST(root='.', download=True,␣
 ↪train=False)
test_image = np.array(test_set.data)
test_label = np.array(test_set.targets)
dict_={0: 'T-shirt/top', 1: 'Trousers',2: 'Pullover',3: 'Dress',4: 'Coat',5:␣
 ↪'Sandal',6: 'Shirt',7: 'Sneaker',8: 'Bag',9: 'Ankle boot'}
```

[3]:

```
print("Dimension of the training set: ")
print(train_set.data.shape[0], "datapoints with image size equal to:␣
 ↪","(",train_set.data.shape[1],"*",train_set.data.shape[2],")")
print()
print("Dimension of the test set: ")
print(test_set.data.shape[0],"datapoints with image size equal to:␣
 ↪","(",test_set.data.shape[1],"*",test_set.data.shape[2],")")
```

```
Dimension of the training set:
60000 datapoints with image size equal to:  ( 28 * 28 )
```

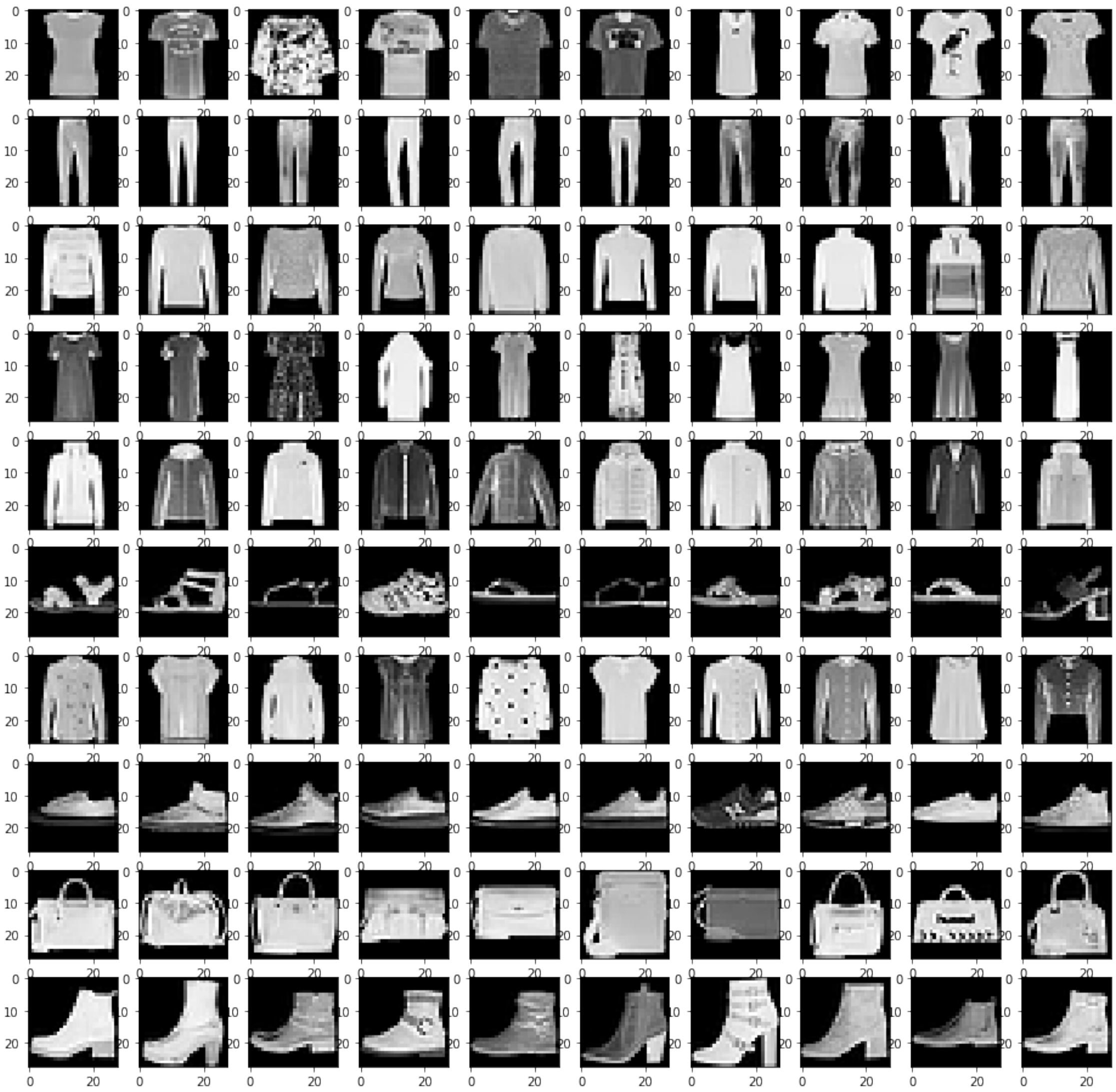
```
Dimension of the test set:
10000 datapoints with image size equal to:  ( 28 * 28 )
```

[4]:

```
def get_images_for_class(class_label):
#create final tensor
    images=torch.Tensor(0,28,28)
    images=images.type(torch.uint8)
    for x in label:
        label_indexes = random.choice(np.where(np.array(train_label) == x)).
 ↪tolist()
        sub_images = train_set.data[random.sample(label_indexes,k=10)]
    #get_random images matching the labels
        images = torch.cat((images,sub_images),0)
    return images
def show_images(images,number,x,y):
    fig = plt.figure(figsize = (x, y))
    for i in range(1, number+1):
        fig.add_subplot(10, 10, i)
        plt.imshow(images[i-1],cmap='gray')
```

[5]:

```
label = dict_.keys()
images = get_images_for_class(label)
show_images(images,100,16,16)
```



[7]:

```
print("Number of t-shirt:",np.sum(train_label==0))
print("Number of trousers:",np.sum(train_label==1))
print("Number of pullover:",np.sum(train_label==2))
print("Number of dresses:",np.sum(train_label==3))
print("Number of coats:",np.sum(train_label==4))
print("Number of sandals:",np.sum(train_label==5))
print("Number of shirts:",np.sum(train_label==6))
print("Number of sneakers:",np.sum(train_label==7))
print("Number of bags:",np.sum(train_label==8))
print("Number of ankle boots:",np.sum(train_label==9))
```

```
Number of t-shirt: 6000
Number of trousers: 6000
Number of pullover: 6000
Number of dresses: 6000
Number of coats: 6000
Number of sandals: 6000
Number of shirts: 6000
Number of sneakers: 6000
Number of bags: 6000
Number of ankle boots: 6000
```

[9]:

```
class Network(nn.Module):
    def __init__(self):
        super(Network,self).__init__()
        self.layer_1 = nn.Sequential(nn.
        Conv2d(1,16,kernel_size=3,padding=1,stride=1),nn.ReLU(),nn.MaxPool2d(2))
        self.layer_2 = nn.Sequential(nn.
        Conv2d(16,32,kernel_size=3,padding=1,stride=1),nn.ReLU(),nn.MaxPool2d(2))
        self.drop_out = nn.Dropout()
        self.fc_1 = nn.Linear(7*7*32,500) #full connection
        self.fc_2 = nn.Linear(500,10)
    def forward(self,y):
        output = self.layer_1(y)
        output = self.layer_2(output)
        output = output.view(output.size(0), -1)
        output = self.drop_out(output)
        output = self.fc_1(output)
        output = self.fc_2(output)
        return output
```

[27]:

```
batch_size = 500
epochs = 25
learning_rate = [0.01,0.1,0.01,0.001]
loss_function = nn.CrossEntropyLoss()
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

#optimser defined in train as it changes with learning rates
```

[28]:

```
model = Network().to(device)
transform = transforms.ToTensor()
loss_list = []
acc_list = []
def train(learning_rate,epochs):
    train_images = transform(np.transpose(train_image)).to(device)
    train_images = train_images.unsqueeze(1)
    train_labels = torch.Tensor(train_label).long().to(device)
    for i in learning_rate:
        optimiser = torch.optim.SGD(model.parameters(),lr=i)
        for e in range(epochs):
            s = list(range(train_set.data.shape[0]))
            random.shuffle(s) #shuffle indexes
            while(len(s)!=0):
                sample = s[-batch_size:]
                sub_images = train_images[sample]
                sub_labels = train_labels[sample]
                optimiser.zero_grad()
                output = model(sub_images)
                loss = loss_function(output,sub_labels)
                loss.backward()
                optimiser.step()
                loss_list.append(loss.item())
                total = sub_labels.size(0)
                _, predicted = torch.max(output.data, 1)
                correct = (predicted==sub_labels).sum().item()
                acc_list.append(correct/total)
                del s[-batch_size:]

start = time.time()
train(learning_rate,epochs)
end = time.time()-start
m, s = divmod(end, 60)
h, m = divmod(m, 60)
```

[29]:

```
print("Time to complete the training:",int(h),"hours",int(m),"minutes␣
 ↪and",int(s),"seconds")
```

```
Time to complete the training: 0 hours 3 minutes and 41 seconds
```

[30]:

```
model.eval()
accuracy = 0
ground_truth = []
predicted_label = []
start = time.time()
with torch.no_grad():
    test_images = transform(np.transpose(test_image)).to(device)
    test_images = test_images.unsqueeze(1)
    test_labels = torch.Tensor(test_label).long().to(device)
    output = model(test_images)
    total = test_labels.size(0)
    _, predicted = torch.max(output.data, 1)
    predicted_label.append(predicted.tolist())
    ground_truth.append(test_labels.tolist())
    correct = (predicted==test_labels).sum().item()
    accuracy = correct/total
print("Time to complete the test:",round(time.time()-start,2),"seconds")
```

```
Time to complete the test: 0.2 seconds
```

```
[31]: print("Test accuracy: ",round(accuracy*100,2),"%",sep = '')
```

```
Test accuracy: 89.4%
```

```
[ ]: ground_truth_flat = [val for sublist in ground_truth for val in sublist]
     predicted_flat = [val for sublist in predicted_label for val in sublist]
     confusion_matrix = metrics.confusion_matrix(ground_truth_flat,predicted_flat)
     print(confusion_matrix)
```

```
[ ]: images = torch.Tensor(0,28,28)
     for filename in os.listdir("pic"):
         img = io.imread(filename, as_gray=True)
         resized = cv2.resize(img, (28,28), interpolation = cv2.INTER_AREA)
         tensor = torch.from_numpy(resized).float()
         tensor = tensor.unsqueeze(0)
         images = torch.cat((images,tensor),0)
     show_images(images,3,40,40)
```

```
[ ]: images = images.unsqueeze(1)
     output = model(images)
     _,predicted = torch.max(output.data,1)
     sm = torch.nn.Softmax(dim=1) #get probabilities for predictions
     probabilities = sm(output)
     print("Exptected: Bag, T-Shirt/Top, Trousers")
     print("Predicted: ",dict_[predicted[0].item()],', ',dict_[predicted[1].
      ↪item()],', ',dict_[predicted[2].item()],sep = '')
```

```
[ ]: probabilities = probabilities.tolist()
     df = pd.DataFrame(np.c_[probabilities[0],probabilities[1],probabilities[2]],␣
      ↪index = dict_.values(),columns = ['Bag','Shirt','Trousers'])
     df.plot.bar()
     plt.xlabel('Classes', fontweight = 'bold', color = 'black', fontsize =␣
      ↪'17',horizontalalignment = 'center')
     plt.ylabel('Predicted probabilities', fontweight = 'bold', color = 'black',␣
      ↪fontsize = '12',horizontalalignment = 'center')
     fig = plt.gcf()
     fig.set_size_inches(10, 5)
     plt.show()
```