This lab is based on the following https://github.com/adambielski/siamese-triplet/blob/master/Experiments_MNIST.ipynb

We will train on MNIST dataset using pairwise and triplet ranking losses. From the implementation point of view the trickiest thing is constructing the dataset loader. We will use the pre-constructed data loaders that we will download from the file below

```
get https://raw.githubusercontent.com/adambielski/siamese-triplet/master/datasets.py
```

```
--2021-04-17 00:07:29--  https://raw.githubusercontent.com/adambielski/siamese-tripl
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133,
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|
HTTP request sent, awaiting response... 200 OK
Length: 8106 (7.9K) [text/plain]
Saving to: ‘datasets.py’

datasets.py         100%[===================>]   7.92K  --.-KB/s    in 0s

2021-04-17 00:07:29 (110 MB/s) - ‘datasets.py’ saved [8106/8106]
```

```
from torchvision.datasets import MNIST
from torchvision import transforms

mean, std = 0.1307, 0.3081

train_dataset = MNIST('../data/MNIST', train=True, download=True,
                             transform=transforms.Compose([
                                 transforms.ToTensor(),
                                 transforms.Normalize((mean,), (std,))
                             ]))
test_dataset = MNIST('../data/MNIST', train=False, download=True,
                            transform=transforms.Compose([
                                transforms.ToTensor(),
                                transforms.Normalize((mean,), (std,))
                            ]))
n_classes = 10
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ../data/M
                                            9913344/? [04:14<00:00, 38920.10it/s]


Extracting ../data/MNIST/MNIST/raw/train-images-idx3-ubyte.gz to ../data/MNIST/MNIST

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ../data/M
                                            29696/? [00:26<00:00, 1104.01it/s]


Extracting ../data/MNIST/MNIST/raw/train-labels-idx1-ubyte.gz to ../data/MNIST/MNIST

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ../data/MN
                                            1649664/? [00:38<00:00, 42641.00it/s]


Extracting ../data/MNIST/MNIST/raw/t10k-images-idx3-ubyte.gz to ../data/MNIST/MNIST/
```

```
!ls
```

```
sample_data
```

Below is the dataloader you will use to train this, and an illustration of what it returns. Run this just to get an idea of what the dataloader is doing here. Essentially for a batch size of 32 it will return 32 triplets (indexed by j below)

```
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:502: UserWarnin
```

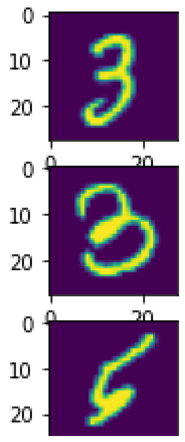
```
from datasets import TripletMNIST
import torch

triplet_train_dataset = TripletMNIST(train_dataset)
triplet_train_loader = torch.utils.data.DataLoader(triplet_train_dataset, batch_size=32,


(im1,im2,im3),_ = next(iter(triplet_train_loader))
#anchor, positive, negative
import matplotlib.pyplot as plt

plt.figure()
j=0
#subplot(r,c) provide the no. of rows and columns
f, axarr = plt.subplots(3,1)
# use the created array to output your multiple images. In this case I have stacked 4 ima
print(im1[j].shape)
print(im1[j].squeeze().shape)
axarr[0].imshow(im1[j].squeeze())
axarr[1].imshow(im2[j].squeeze())
axarr[2].imshow(im3[j].squeeze())
```

```
torch.Size([1, 28, 28])
torch.Size([28, 28])
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:54: UserWarning
  warnings.warn("train_labels has been renamed targets")
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:64: UserWarning
  warnings.warn("train_data has been renamed data")
<matplotlib.image.AxesImage at 0x7fe20e0cb8d0>
<Figure size 432x288 with 0 Axes>
```



Train the model using the triplet loss (https://pytorch.org/docs/stable/generated/torch.nn.TripletMarginLoss.html). At each iteration load the 3 samples using the dataloader to act as anchor, positive, negative. Use 20 epochs for training and the adam optimizer with a suggested learning rate of 1e-3. You may experiment with margin parameters (suggest 0.05 to 0.5).

```
import torch.nn as nn
class EmbeddingNet(nn.Module):
    def __init__(self):
        super(EmbeddingNet, self).__init__()
        self.convnet = nn.Sequential(nn.Conv2d(1, 32, 5), nn.PReLU(),
                                     nn.MaxPool2d(2, stride=2),
                                     nn.Conv2d(32, 64, 5), nn.PReLU(),
                                     nn.MaxPool2d(2, stride=2))

        self.fc = nn.Sequential(nn.Linear(64 * 4 * 4, 256),
                                nn.PReLU(),
                                nn.Linear(256, 256),
                                nn.PReLU(),
                                nn.Linear(256, 4)
                                )

    def forward(self, x):
        output = self.convnet(x)
        output = output.view(output.size()[0], -1)
        output = self.fc(output)
        return output

    def get_embedding(self, x):
        return self.forward(x)
```

```
model=EmbeddingNet()
(im1,im2,im3),_ = next(iter(triplet_train_loader))
print(im1.shape)
output1 = model(im1)
print(output1.shape)
```

```
torch.Size([100, 1, 28, 28])
torch.Size([100, 4])
```

```
#Use this model
#Use these dataloaders
from datasets import TripletMNIST
triplet_loss = nn.TripletMarginLoss(margin=0.2, p=2)
triplet_train_dataset = TripletMNIST(train_dataset)
triplet_train_loader = torch.utils.data.DataLoader(triplet_train_dataset, batch_size=100,
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
loss=[]
counter=[]
iteration_number = 0
for epoch in range(20):
    all_loss=[]
    for (im1,im2,im3),_  in triplet_train_loader:
      optimizer.zero_grad()
      output1 = model(im1)
      output2 = model(im2)
      output3 = model(im3)
      loss = triplet_loss(output1,output2,output3)
      all_loss.append(loss.item())
      loss.backward()
      optimizer.step()
    print("Epoch {} Avg loss {}".format(epoch, sum(all_loss)/len(all_loss)))
```

```
Epoch 0 Avg loss 0.013994431830988106
Epoch 1 Avg loss 0.004463962324440823
Epoch 2 Avg loss 0.003418645114419784
Epoch 3 Avg loss 0.0028855749425239687
Epoch 4 Avg loss 0.002465450398946511
```

Embedd the entire training set and perform classification on the 500 test set images using a nearest neighbor approach. Report the overall accuracy. Finally show the 5 nearest neighbor for 2 randomly selected samples.

```
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=500, shuffle=False)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=500, shuffle=True)
embs=[]
targets=[]
for img,label in train_loader:
    img_emb=model(img)
    for idx in range(len(img_emb)):
      embs.append(img_emb[idx].detach().numpy())
      targets.append(label[idx])
```

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=5)
neigh.fit(embs,targets)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
 test_imgs,test_labels =next(iter(test_loader))
 test_embs=model(test_imgs)
 correct=0
 for idx in range(len(test_labels)):
    nn_class=neigh.predict([test_embs[idx].detach().numpy()])
    org_class=test_labels[idx].item()
    # print(type(nn_class[0]))
    # print(type(org_class))
    if nn_class[0] == org_class:
      correct=correct+1

    # print(nn_class,test_labels[idx])
print("test classification accuracy=",correct/len(test_labels))
```

```
test classification accuracy= 0.988
```

```
from scipy.spatial import distance
import numpy as np
for idx in np.random.randint(500, size=3):
  p1=test_embs[idx].detach().numpy()
  for idx2 in range(len(embs)):
    dis=distance.euclidean(p1,embs[idx2])
    if dis<0.035:
        print("test point=",p1," label=",test_labels[idx].item(),
        "nearest train point=",embs[idx2]," label=",targets[idx2]," dist=",dis)
  print("\n")
```

```
   1.0564456   0.23062131 -0.43402073]  label= tensor(3)  dist= 0.030030006542801857
   1.0564634   0.23337558 -0.45503718]  label= tensor(3)  dist= 0.022409332916140556
   1.0798577   0.25700772 -0.46699774]  label= tensor(3)  dist= 0.034628208726644516
   1.0602124   0.23511738 -0.46315226]  label= tensor(3)  dist= 0.03448755666613579
   1.0476727   0.22428265 -0.4391755 ]  label= tensor(3)  dist= 0.033146247267723083
   1.0736427   0.24992588 -0.47012144]  label= tensor(3)  dist= 0.027466386556625366
   1.0710776   0.25660524 -0.44151852]  label= tensor(3)  dist= 0.016465920954942703
   1.0514135   0.22496894 -0.43810537]  label= tensor(3)  dist= 0.027172092348337173
   1.0855343   0.25285625 -0.46801072]  label= tensor(3)  dist= 0.03407539799809456
   1.0752157   0.2393105  -0.45126918]  label= tensor(3)  dist= 0.019106265157461166
   1.086365    0.23438361 -0.44538394]  label= tensor(3)  dist= 0.03100321814417839
   1.06264     0.24362132 -0.46604943]  label= tensor(3)  dist= 0.02782380022108555
   1.0521002   0.25046077 -0.47549698]  label= tensor(3)  dist= 0.03012542426586151
```

```
    1.0753038   0.22904468 -0.4361196 ]  label= tensor(3)  dist= 0.02784881182014942


5   1.018239    0.18084818 -0.54018766]  label= tensor(3)  dist= 0.029130926355719566
4   0.9898162   0.20619187 -0.5472096 ]  label= tensor(3)  dist= 0.028144920244812965
7   0.99916637  0.18517703 -0.5377188 ]  label= tensor(3)  dist= 0.03250732272863388
4   1.0194106   0.18908694 -0.56992733]  label= tensor(3)  dist= 0.032263461500406265
1  1.0041096  0.1994327 -0.5642025]  label= tensor(3)  dist= 0.019045783206820488


).5260905  -0.5422998   0.61258334]  label= tensor(4)  dist= 0.0340864397585392
).56821626 -0.5345587   0.60764295]  label= tensor(4)  dist= 0.019442226737737656
).53995407 -0.5339299   0.61120486]  label= tensor(4)  dist= 0.028582962229847908
).5702605  -0.57084125  0.61595714]  label= tensor(4)  dist= 0.03414198383688927
).5578974  -0.54031986  0.6176654 ]  label= tensor(4)  dist= 0.017355065792798996
).5692823  -0.5570332   0.600352  ]  label= tensor(4)  dist= 0.026520201936364174
).56297743 -0.5461674   0.59872174]  label= tensor(4)  dist= 0.025871671736240387
).5681932  -0.5504158   0.59254813]  label= tensor(4)  dist= 0.019851570948958397
).5501481  -0.5294229   0.6025038 ]  label= tensor(4)  dist= 0.02090076170861721
).58138126 -0.54639     0.60655576]  label= tensor(4)  dist= 0.03146378695964813
).5415361  -0.54515177  0.5866125 ]  label= tensor(4)  dist= 0.022887440398335457
).5519217  -0.56786096  0.58848655]  label= tensor(4)  dist= 0.030409300699830055
).5724522  -0.55783576  0.61658394]  label= tensor(4)  dist= 0.027936290949583054
).5561033  -0.51930714  0.59356886]  label= tensor(4)  dist= 0.031413834542036057
```

✓ 1s completed at 11:13 PM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.