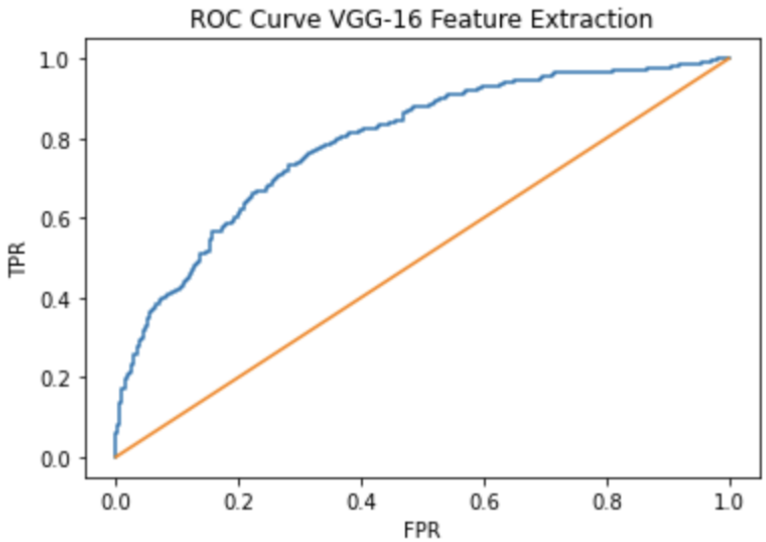
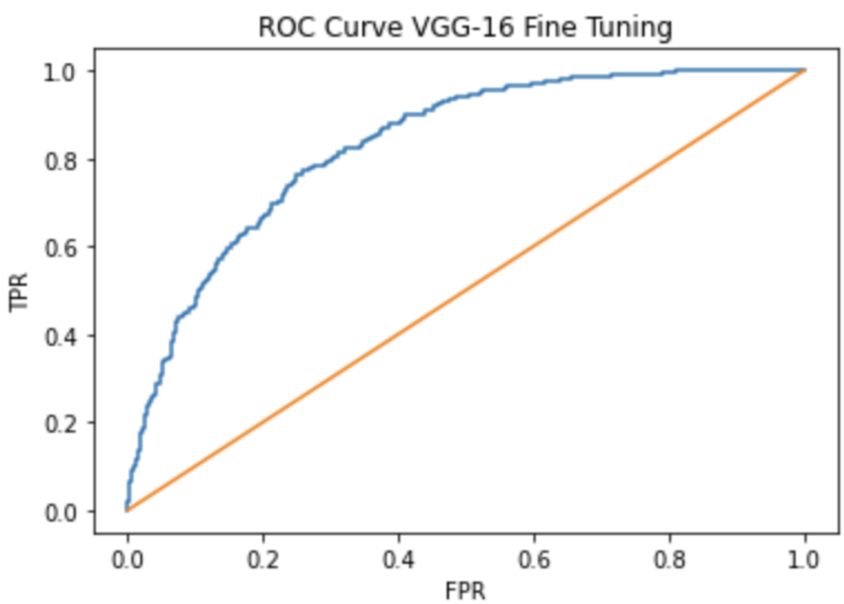


In this section I took a pretrained VGG-16 network and removed the last layer of it. Then ran the lfw pairs through it and used the cosine angle similarity to determine the similarity values for each pair. This is highlighted in the ROC curve above. The AUC for this was: .7914

2.2 ROC curve for fine tuning from VGG-16

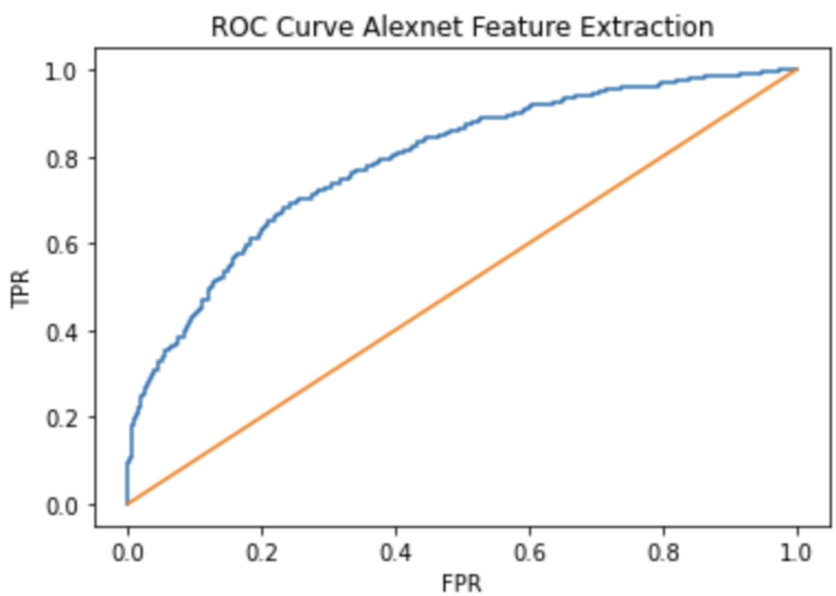


In this section I performed fine tuning of the VGG network, this involved loading the pretrained weights of the model removing the last layer of the model and then training on the dataset for 10 epochs. Training was fed in batches of 60 pairs. The AUC was:.83 , I experienced better performance on the fine-tuned model especially after increasing it up to 10 epochs.

3 Part 2: Models, Alexnet

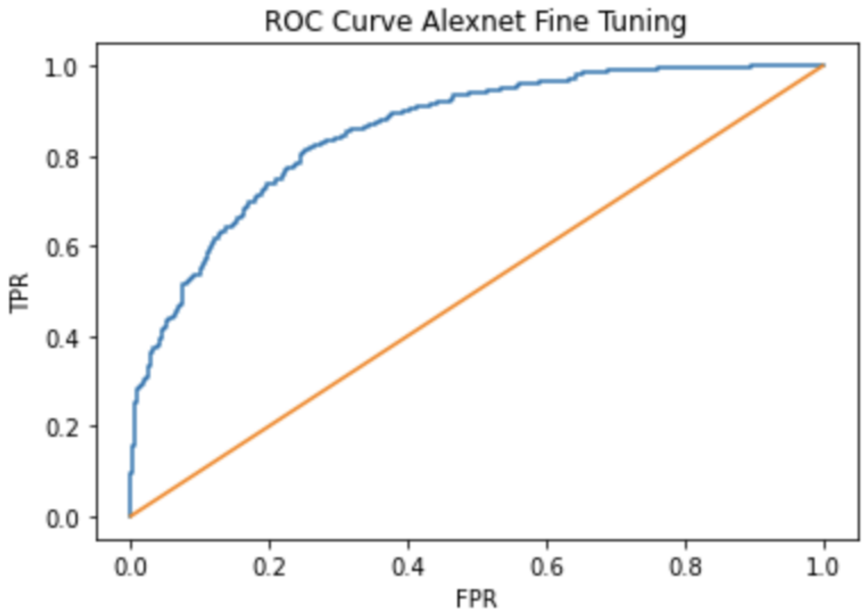
In the Alexnet models I used a binary cross entropy loss function with learning rate of 10^{-4} using the Adam optimizer. I used the cosine similarity measurement to compute the similarity between model outputs for the image pairs.

2.3 Feature Extraction from Alexnet



In this section I took a pretrained Alexnet network and removed the last layer of it. Then ran the lfw pairs through it and used the cosine similarity to determine the similarity values for each pair. This was used in a binary cross entropy loss function as described earlier. This is highlighted in the ROC curve above. The AUC for this was: .79.

2.4 Fine Tuning Alexnet



In this section I performed fine tuning of the Alexnet model, this involved loading the pre-trained weights of the model and then training on the dataset for only 2 epochs. Training was fed in batches of 40 pairs. The AUC was: .85 , I experienced better performance on the fine tuned model and it was interesting to see that smaller epochs gave better results. This probably helped keep the model from over fitting. **This model was my greatest performing one.**

4 Appended Code and References to code

Here is all the code for my work. I wrote everything in python using the framework pytorch on google COLABs Using the GPU setting for faster running. Data was imported using the sklearn libraries and was verified to resemble the lfw dataset.

HW4_Final

April 29, 2021

[1]:
```
#HW 4 Vedant Jain



import torch
import torch.nn as nn
import matplotlib.pyplot as plt
from torch.utils.data import Dataset,DataLoader
import torchvision.transforms as transforms
from torchvision.transforms import ToPILImage
import torch.nn.functional as F
import numpy as np
import os
import random
import sys
from PIL import Image

import torchvision.models as models

from sklearn.metrics import roc_curve, auc
```

[2]:
```
torch.cuda.is_available()
```

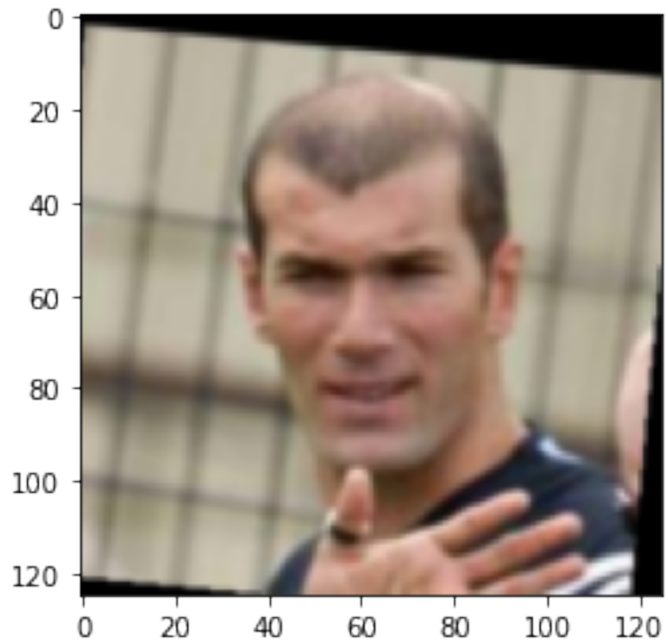
[2]:
```
True
```

[3]:
```
#%% load in the images

from sklearn.datasets import fetch_lfw_pairs
lfw_train = fetch_lfw_pairs(subset='train',color=True, slice_=(slice(0, 250),␣
↪slice(0, 250)))
lfw_test = fetch_lfw_pairs(subset='test',color=True, slice_=(slice(0, 250),␣
↪slice(0, 250)))
```

[4]:
```
first = lfw_train.pairs[1099,0,:,:,:]
second = lfw_train.pairs[1099,1,:,:,:]
lfw_train.pairs.shape
```

[4]:
```
(2200, 2, 125, 125, 3)
```

[5]:
```
plt.imshow(second.astype('uint8'))
plt.show()
```



[6]:
```
#%% preprocess image
preprocess = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

#input_tensor = preprocess(singleImage.astype(np.uint8))
```

[7]:
```
#%% get labels and individual images
trainLabels = lfw_train.target
testLabels = lfw_test.target
```

[8]:
```
#%% throw into dataloader:

class NumbersDataset(Dataset):
    def __init__(self,trainSel):
        self.trainingData = trainSel
        if trainSel == 1:
            self.numEntries = len(trainLabels)
```

```python
        else:
            self.numEntries = len(testLabels)

    def __len__(self):
        return self.numEntries

    def __getitem__(self, idx):
        if self.trainingData == 1:
            singleImage = lfw_train.pairs[idx,0,:,:,:]
            image1 = preprocess(singleImage.astype(np.uint8))
            secondImage = lfw_train.pairs[idx,1,:,:,:]
            image2 = preprocess(secondImage.astype(np.uint8))
            label = trainLabels[idx]
            return image1, image2, label
        else:
            singleImage = lfw_test.pairs[idx,0,:,:,:]
            image1 = preprocess(singleImage.astype(np.uint8))
            secondImage = lfw_test.pairs[idx,1,:,:,:]
            image2 = preprocess(secondImage.astype(np.uint8))
            label = testLabels[idx]
            return image1, image2, label
```

[9]:
```python
#%% load the dataset
train_set = NumbersDataset(1)
test_set = NumbersDataset(0)
train_loader = DataLoader(train_set, batch_size=50, shuffle=True, num_workers=0)
test_loader = DataLoader(test_set, batch_size=1, shuffle=False, num_workers=0)
```

[10]:
```python
torch.cuda.is_available()
```

[10]: True

trueLabel = []; predLabel = [];

for i,data in enumerate(test_loader, 0): model.train(mode=False) image1s,image2s,labels=data image1s = image1s.to(device) image2s = image2s.to(device) labels = labels.to(device) trueLabel.append(labels.cpu().numpy()) with torch.no_grad(): output = model(image1s) output2 = model(image2s) #getLabels.append(output) cos = nn.CosineSimilarity(dim=1, eps=1e-6) cos_sim = cos(output, output2) predLabel.append(cos_sim.cpu().numpy()) #print(i)

[11]:
```python
#%%
def train(train_loader, model, loss_fn, optimizer):
    for i, data in enumerate(train_loader):
        model.train(mode=True)
        image1s,image2s,labels=data
        labels = labels.to(torch.float32)
```

```python
        image1s = image1s.to(device)
        image2s = image2s.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()

        # Compute prediction error
        output = model(image1s)
        output2 = model(image2s)
    #getLabels.append(output)
        cos = nn.CosineSimilarity(dim=1, eps=1e-8)
        cos_sim = cos(output, output2)
        predVal = cos_sim

        # print(predVal)
        # print('True value: ')
        # print(labels)

        loss = loss_fn(predVal, labels)
        #print(loss)

        # Backpropagation
        loss.backward()
        optimizer.step()
    print('one Epoch done')


def test(dataloader, model):
    trueLabel = [];
    predLabel = [];
    model.train(mode=False)
    model.eval()
    with torch.no_grad():
        for i, data in enumerate(dataloader):
            image1s,image2s,labels=data
            #labels = labels.to(torch.float32)
            image1s = image1s.to(device)
            image2s = image2s.to(device)
            #labels = labels.to(device)
            output = model(image1s)
            output2 = model(image2s)
            #trueLabel.append(labels.cpu().numpy())
            cos = nn.CosineSimilarity(dim=1, eps=1e-6)
            cos_sim = cos(output, output2)
            predLabel.append(cos_sim.cpu().numpy())
            trueLabel.append(labels)
    return trueLabel, predLabel
```

[12]:
```
#%% reset Dataset
train_set = NumbersDataset(1)
test_set = NumbersDataset(0)
train_loader = DataLoader(train_set, batch_size=60, shuffle=True, num_workers=0)
test_loader = DataLoader(test_set, batch_size=1, shuffle=False, num_workers=0)
```

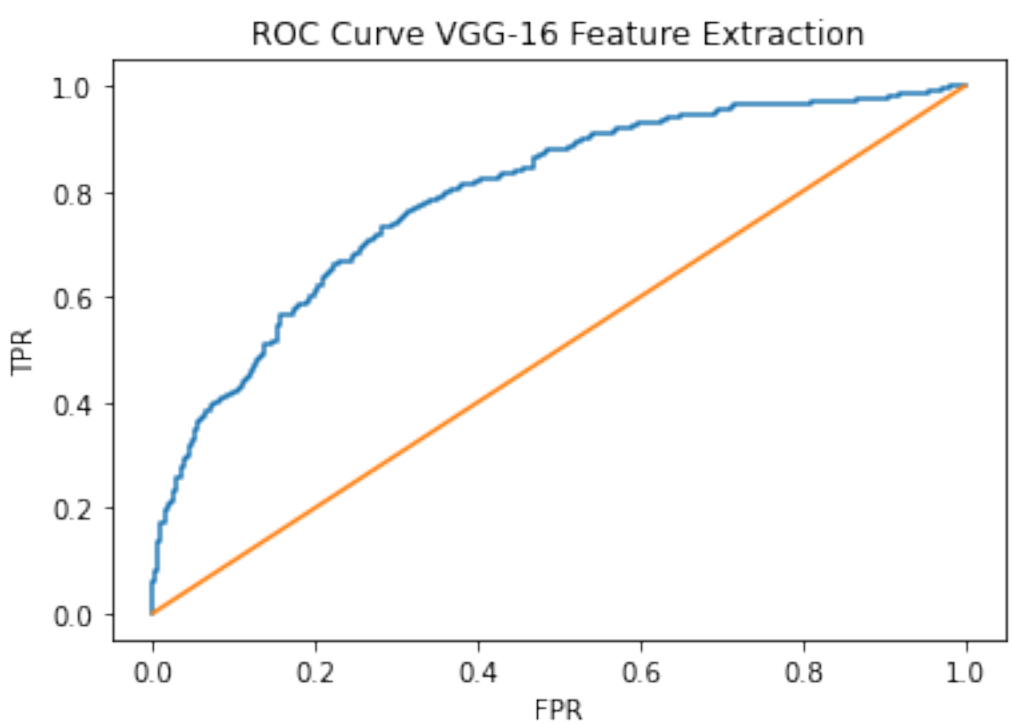
[13]:
```
#%% define model to use here its VGG
#model = torch.hub.load('pytorch/vision:v0.6.0', 'alexnet', pretrained=True)
model = models.vgg16(pretrained=True)
model.eval()
new_classifier = nn.Sequential(*list(model.classifier.children())[:-1])
model.classifier = new_classifier
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)
```

[13]:
```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
```

```
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
  )
)
```

[14]:

```
trueLabel, predLabel = test(test_loader, model)
fpr, tpr, thresholds = roc_curve(trueLabel, predLabel)
plt.plot(fpr, tpr)
plt.plot(np.linspace(0,1,100),np.linspace(0,1,100))
plt.title('ROC Curve VGG-16 Feature Extraction')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



[15]:
```
aucVal = auc(fpr, tpr)
aucVal
```

[15]: 0.791432

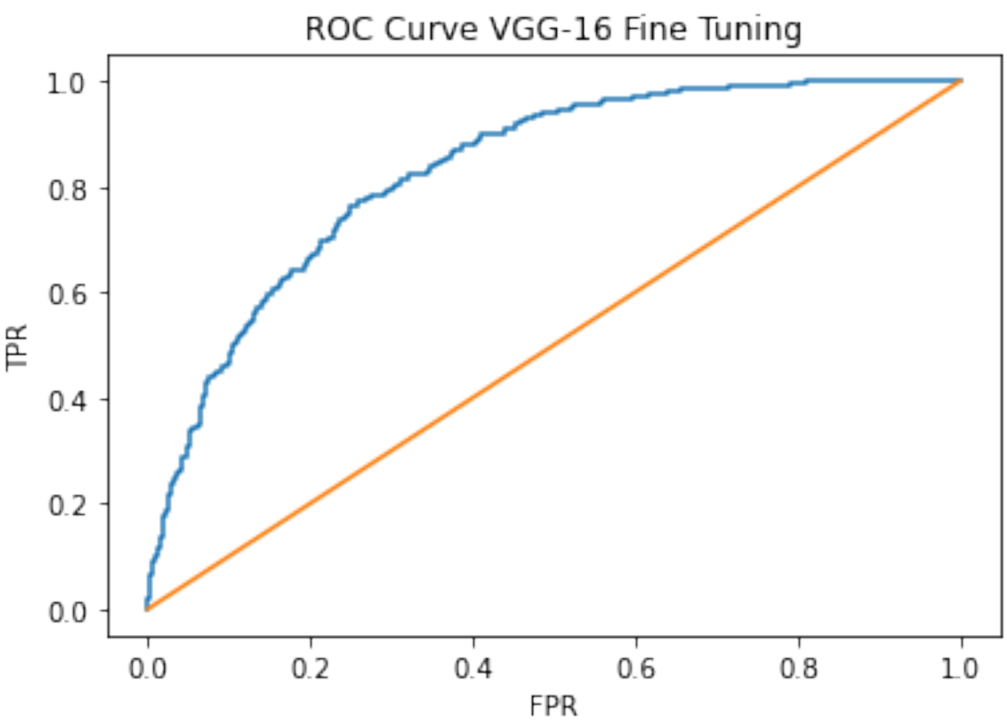
[16]:
```
loss_fn = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-5) #loss earlier was 1e-4

epochs = 10

#%%

for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_loader, model, loss_fn, optimizer)
print("Done!")
```

```
Epoch 1
-------------------------------
one Epoch done
Epoch 2
-------------------------------
one Epoch done
Epoch 3
-------------------------------
one Epoch done
Epoch 4
-------------------------------
one Epoch done
Epoch 5
-------------------------------
one Epoch done
Epoch 6
-------------------------------
one Epoch done
Epoch 7
-------------------------------
one Epoch done
Epoch 8
-------------------------------
one Epoch done
Epoch 9
-------------------------------
one Epoch done
Epoch 10
```

```
-------------------------------
one Epoch done
Done!
```

[17]:
```
trueLabel, predLabel = test(test_loader, model)
```

[18]:
```
#roc curve graph
fpr, tpr, thresholds = roc_curve(trueLabel, predLabel)
plt.plot(fpr, tpr)
plt.plot(np.linspace(0,1,100),np.linspace(0,1,100))
plt.title('ROC Curve VGG-16 Fine Tuning')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



[19]:
```
aucVal = auc(fpr, tpr)
aucVal
```

[19]: 0.83124

[20]:
```
#%% reset Dataset
train_set = NumbersDataset(1)
test_set = NumbersDataset(0)
train_loader = DataLoader(train_set, batch_size=40, shuffle=True, num_workers=0)
```

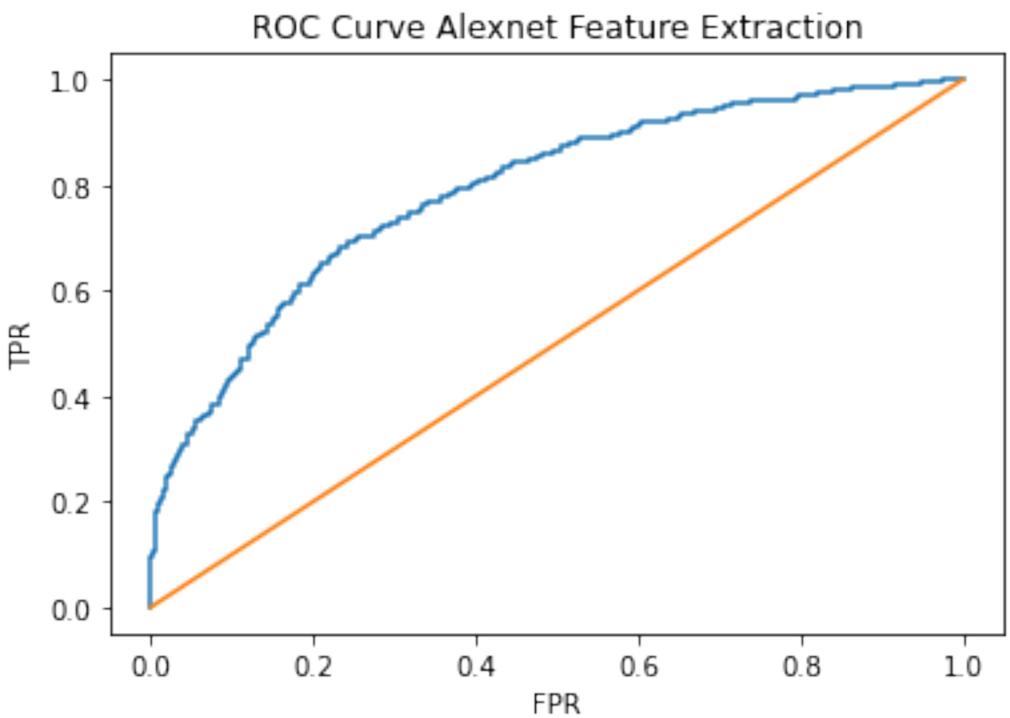
```
test_loader = DataLoader(test_set, batch_size=1, shuffle=False, num_workers=0)
```

[27]:
```
#%% define model to use
#model = torch.hub.load('pytorch/vision:v0.6.0', 'alexnet', pretrained=True)
#model = models.vgg16(pretrained=True)
model2 = models.alexnet(pretrained=True)
model2.eval()
new_classifier = nn.Sequential(*list(model2.classifier.children())[:-1])
model2.classifier = new_classifier
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model2.to(device)
```

[27]:
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
  )
)
```

[22]:
```
trueLabel, predLabel = test(test_loader, model2)
#roc curve graph
fpr, tpr, thresholds = roc_curve(trueLabel, predLabel)
plt.plot(fpr, tpr)
plt.plot(np.linspace(0,1,100),np.linspace(0,1,100))
```

```
plt.title('ROC Curve Alexnet Feature Extraction')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



[23]:
```
aucVal = auc(fpr, tpr)
aucVal
```

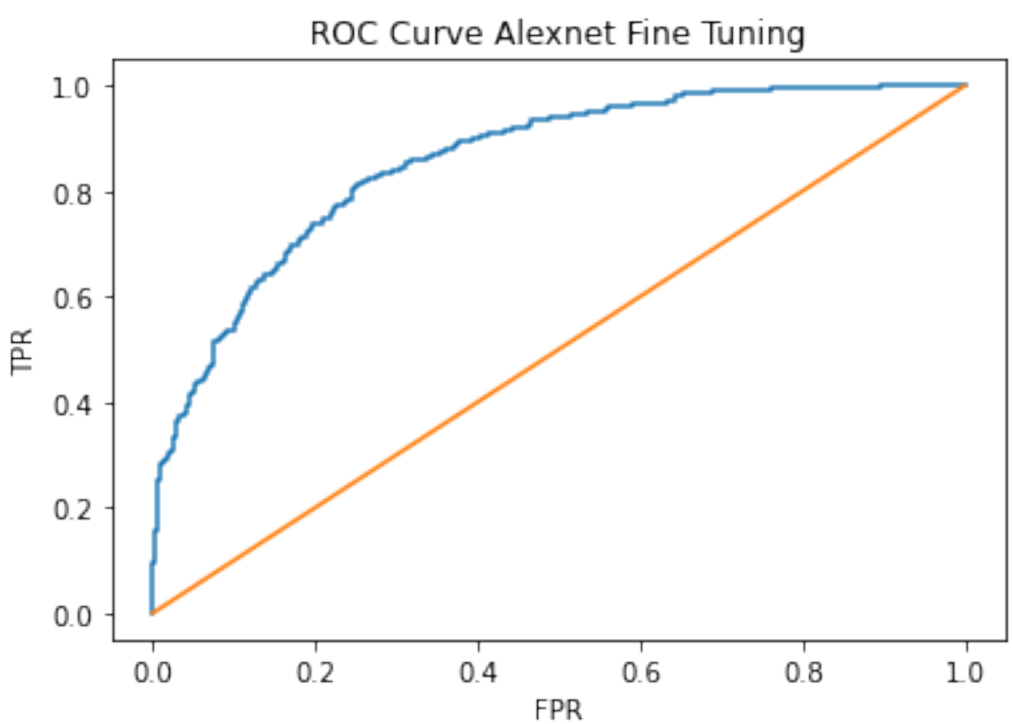
[23]: 0.7899160000000001

[28]:
```
loss_fn = nn.BCELoss()
optimizer = torch.optim.Adam(model2.parameters(), lr=1e-4) #loss earlier was␣
↪1e-4

epochs = 2

#%%

for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_loader, model2, loss_fn, optimizer)
print("Done!")
```

Epoch 1

```
-------------------------------
one Epoch done
Epoch 2
-------------------------------
one Epoch done
Done!
```

[29]:
```
trueLabel, predLabel = test(test_loader, model2)
#roc curve graph
fpr, tpr, thresholds = roc_curve(trueLabel, predLabel)
plt.plot(fpr, tpr)
plt.plot(np.linspace(0,1,100),np.linspace(0,1,100))
plt.title('ROC Curve Alexnet Fine Tuning')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
```



[26]:
```
aucVal = auc(fpr, tpr)
aucVal
```

[26]: 0.8529159999999999

[26]:
```
```

[26]:
```

```

[26]:
```

```