采用FaceNet实现人脸识别

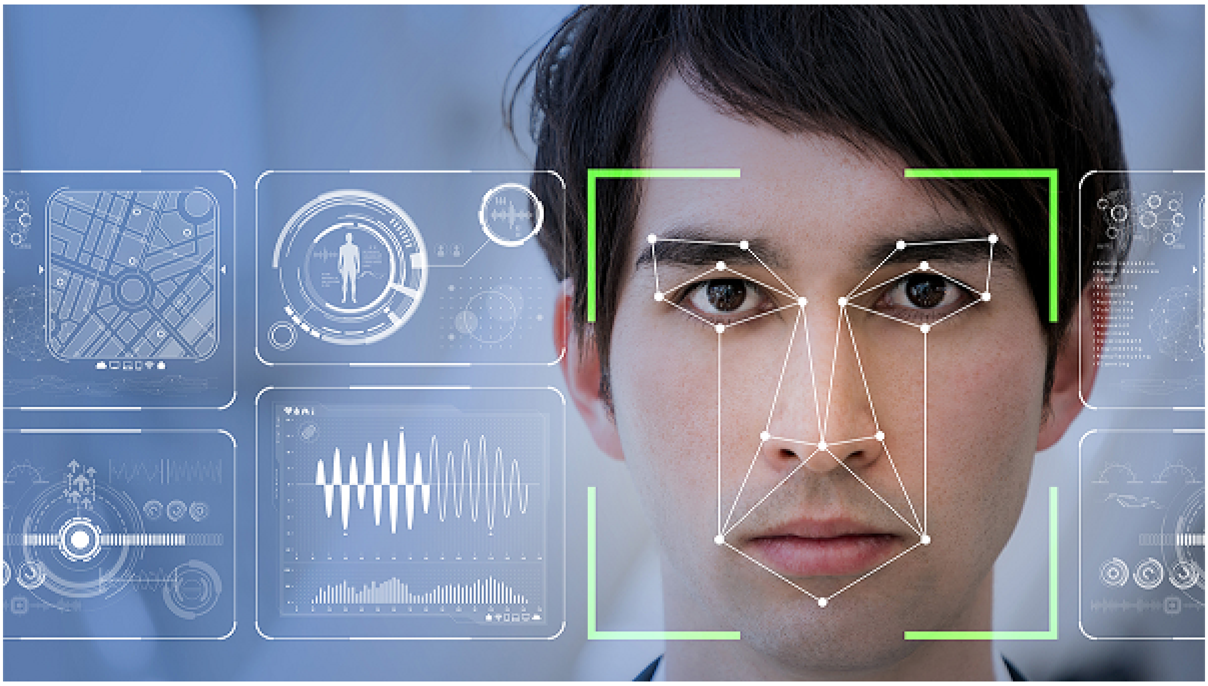
运行

更新于 2020-10-29 09:24:45 6 0

人脸识别指计算机分析人脸特征，自动进行身份鉴别的技术。相比于指纹、虹膜等传统生物识别手段，人脸识别具有无接触、不易盗取等优势，因此在保障公共安全、信息安全、公司和个人财产安全上有强烈的需求。

近些年来随着深度卷积神经网络的发展，人脸识别的准确率得以大幅提升。人脸识别考勤、刷脸支付等相关应用，已开始逐步投入使用，效果显著。

本案例使用FaceNet算法训练LFW数据集，实现人脸识别。



目录

1 数据集简介

LFW(Labled Faces in the Wild)人脸数据集，被广泛应用于评价人脸识别算法的性能，是这一领域重要的数据集。

其中的人脸图像均来源于生活中的自然场景。由于多姿态、光照、表情、年龄、遮挡等因素影响，即使是同一个人的照片差别也较大。对于包含多张人脸的图像，仅选择中心坐标的人脸进行识别，其他区域的人脸则视为背景干扰。

LFW数据集包含5749人的13233张人脸图像，每张图像都标注出了对应的人名，约1680个人有两张以上人脸图像。每张图片的像素大小为250×250，绝大部分为彩色图像，存在少许黑白人脸图像。

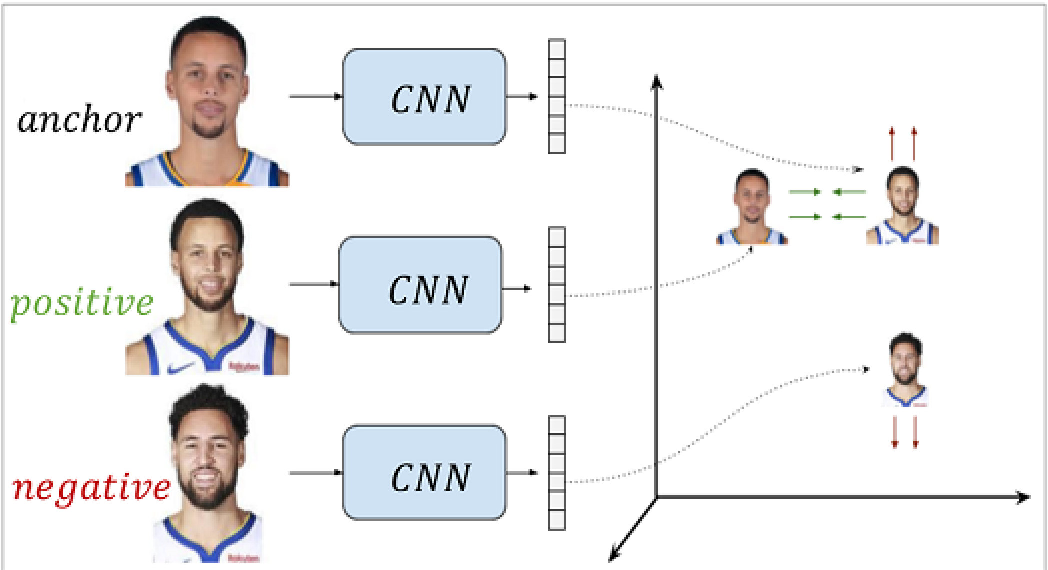
2 模型介绍

Siamese网络的原理是，利用神经网络提取图像特征向量，再根据特征向量判断相似度。

FaceNet将图像映射到欧式空间，且空间距离和图像相似度相关：同一个人的不同人脸图像的距离很小，不同人的图像在空间中距离较大。且FaceNet提出三元组损失（Triplet Loss），需要每个样本包含三张图像：基准图像（Anchor）、正样本（Positive）、负样本

（Negative）。基准图像和正样本是同一人的人脸图像，基准图像和负样本不属于同一人。

通过神经网络，我们希望基准图像和正样本的特征向量距离尽可能小，而基准图像和负样本的特征向量距离尽可能大。



3 数据处理

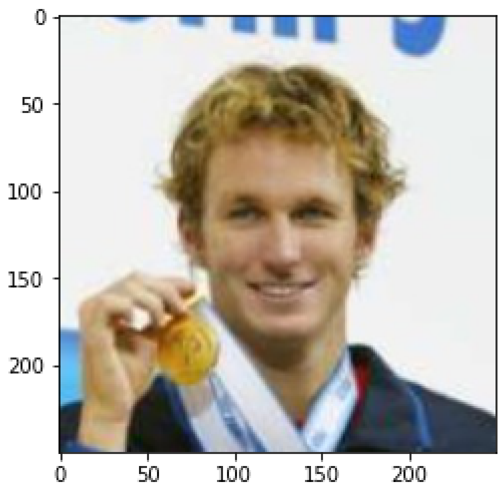
3.1 数据集介绍

首先加载所需要的库，在进行数据处理和模型训练时需要使用。

```
import csv
import random
import os
import torch
import torchvision
import numpy as np
import pandas as pd
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader,Dataset
from skimage import io
from torch.autograd import Variable,Function
from torch.nn.modules.distance import PairwiseDistance
from sklearn.metrics.pairwise import euclidean_distances
import scipy.stats as ss
```

再加载LFW数据集，通过解压 `zip` 文件，可以得到数据集中的图像。 下图展示了数据集中的一张人脸图像。

```
# 加载数据集
import os
!apt install unzip
!unzip lfw.zip

# 展示人脸图像
img = plt.imread("/content/lfw/Aaron_Peirsol/Aaron_Peirsol_0001.jpg")
plt.imshow(img)
plt.show()
```



加载 `lfw.csv` 文件， `id` 列为图像名， `name` 列为图像中人脸对应的人名， `class` 中的数字为人的编号，共有5749人。

```
columns = ['id','name','class']
dataset = pd.read_csv('/content/lfw.csv')
print(dataset)
```

```
                        id                name  class
0              AJ_Cook_0001             AJ_Cook      0
1             AJ_Lamas_0001            AJ_Lamas      1
2        Aaron_Eckhart_0001       Aaron_Eckhart      2
3          Aaron_Guiel_0001         Aaron_Guiel      3
4       Aaron_Patterson_0001     Aaron_Patterson      4
...                     ...                 ...    ...
13228     Zorica_Radovic_0001      Zorica_Radovic   5744
13229     Zulfiqar_Ahmed_0001      Zulfiqar_Ahmed   5745
13230       Zumrati_Juma_0001        Zumrati_Juma   5746
13231     Zurab_Tsereteli_0001     Zurab_Tsereteli   5747
13232  Zydrunas_Ilgauskas_0001  Zydrunas_Ilgauskas   5748

[13233 rows x 3 columns]
```

编写 `get_ids` 函数，将数据集中同一个人的人脸图像分为一类，形式如下。"Bud_Selig_0001"为图像名称，代表姓名为"Bud Selig"的第一张人脸图像，"732"为"Bud Selig"的编号。

```
# 将同一人的人脸图像分成一类
def get_ids(df):
    names = dict()
    for i, ID in enumerate(df['class']):
        if ID not in names:
            names[ID] = []
        names[ID].append(df.iloc[i, 0])
    return names

indices = dataset['class'].unique()
Ids = get_ids(dataset)

# 展示分类结果
print("732:",Ids.get(732))
print("986:",Ids.get(986))
```

```
732: ['Bud_Selig_0001', 'Bud_Selig_0002', 'Bud_Selig_0003', 'Bud_Selig_0004']
986: ['Christopher_Whittle_0001']
```

3.2 划分数据集

为训练模型，将数据集划分为训练集和测试集，比例为训练集：测试集 = 4:1。经过划分后，训练集中有4600个人的人脸图像，测试集中则有1149个人。

```
# 划分训练集和测试集
def train_test_split(dataset, test_split = 0.2):
    # test_split: 测试集占比
    dataset_size = len(dataset)
    indices = list(range(dataset_size))
    shuffle_dataset = True
    split = int(np.floor(test_split * dataset_size))
    random_seed= 42
    if shuffle_dataset :
        np.random.seed(random_seed)
        np.random.shuffle(indices)
    train_indices, test_indices = indices[split:], indices[:split]
    return train_indices, test_indices

# 划分结果
train_indices, test_indices = train_test_split(Ids)
print(len(train_indices))
print(len(test_indices))
```

```
训练集：  4600
测试集：  1149
```

3.3 基准图像和正负样本

训练模型的时候，我们需要同时输入基准图像，及其对应的正负样本，并最小化三元组损失。因此我们需要编写函数，在训练集中标注出基准图像、正负样本。

```
# 数据格式转化
class ToTensor(object):
    def _call_(self, sample):
        image, label = sample['image'], sample['label']
        image = image.transpose((2, 0, 1))
        return {'image': torch.from_numpy(image),
                'label': torch.from_numpy(label)}

# 划分基准图像、正负样本
class triplet_dataset(torch.utils.data.DataLoader):
    # 读取变量值
    def __init__(self, root_dir= "/content/lfw/", transform = None, train=0
):
        # train 0 : 三元组损失; train 1 : 划分训练集图像; train 2 : 划分测试集
图像
        self.csv_dir = "/content/lfw_allnames.csv"
        self.train = train
        self.root_dir = root_dir
        self.transform = transform
        self.train_dataset,self.test_dataset = self.get_data_set()
        temp = self.train_dataset.groupby(['name']).image_path.apply(lambda x
: list(x.values)).reset_index()
        print(temp.head())
        self.training_triplets = self.make_triplets(temp['name'], temp['image_
path'], temp.index)

    def get_data_set(self,):
        # 整理数据格式
        train_set = pd.read_csv(self.csv_dir)
        train_set = train_set.loc[train_set.index.repeat(train_set['images'])]
        train_set['image_path'] = 1 + train_set.groupby('name').cumcount()
        train_set['image_path'] = train_set.image_path.apply(lambda x: '{0:0>
4}'.format(x))
        train_set['image_path'] = train_set.name + "/" + train_set.name + "_"
+ train_set.image_path + ".jpg"
        train_set['label'] = train_set.index
        temp = train_set.where(train_set['images'] >= 2).dropna()

        # 随机选择数据
        random_class = random.sample(range(temp.index[0], temp.index[-1]), 10)
        test_set = pd.DataFrame(columns= train_set.columns)
        for i in random_class:
            index = temp.index[i]
            x = temp[temp.label == index]
            x = x.dropna()
            test_set = test_set.append(x, ignore_index = True)
```

```
        # 划分训练集和测试集
        train_set = pd.merge(train_set,test_set, indicator=True, how='outer').
query('_merge=="left_only"').drop('_merge', axis=1)
        train_set = train_set.drop(['images'], axis= 1)
        test_set = test_set.drop(['images'], axis= 1)
        test_set = test_set.reset_index().drop(['index'], axis= 1)
        train_set.drop(['label'],axis= 1)
        return train_set,test_set

    # 选择基准图像、正负样本
    def make_triplets(self, names, Ids, classes):
        print(names[5],Ids[5],classes[5])

        triplet_size = len(classes)
        triplets = []
        for i in range(triplet_size):
            # 选择正样本
            pos_index = np.random.choice(classes)
            while len(Ids[pos_index]) < 2:
                pos_index = np.random.choice(classes)
            # 选择基准图像
            anchor = np.random.randint(0, len(Ids[pos_index]))
            postive = np.random.randint(0, len(Ids[pos_index]))
            while anchor == postive:
                postive = np.random.randint(0, len(Ids[pos_index]))

            # 选择负样本
            neg_index = np.random.choice(classes)
            while pos_index == neg_index:
                neg_index = np.random.choice(classes)
            negative = np.random.randint(0, len(Ids[neg_index]))

            postive_class = names[pos_index]
            negative_class = names[neg_index]

            triplets.append([Ids[pos_index][anchor], Ids[pos_index][postive],
                                  Ids[neg_index][negative], postive_class, negati
ve_class, pos_index, neg_index])
        return triplets

    # 划分数据集
    def __getitem__(self, idx):
        if torch.is_tensor(idx):
          idx = idx.tolist()

        # 划分基准图像和正负样本
        if(self.train == 0):
            # 人脸身份
            anc_id, pos_id, neg_id, positive_class, negative_class, pos_index,
neg_index = self.training_triplets[idx]
            anc_img   = os.path.join(self.root_dir, anc_id)
            pos_img   = os.path.join(self.root_dir, pos_id)
            neg_img   = os.path.join(self.root_dir, neg_id)
```

```
            # 图像标注为基准图片/正负样本
            anc_img   = io.imread(anc_img)
            pos_img   = io.imread(pos_img)
            neg_img   = io.imread(neg_img)

            pos_class = torch.from_numpy(np.array([pos_index]).astype('long'))
            neg_class = torch.from_numpy(np.array([neg_index]).astype('long'))

            sample = {'anchor': anc_img, 'postive': pos_img, 'negative': neg_i
mg,
                      'postive_label': positive_class, 'negative_label': negat
ive_class}

            if self.transform:
                sample['anchor'] = self.transform(sample['anchor'])
                sample['postive'] = self.transform(sample['postive'])
                sample['negative'] = self.transform(sample['negative'])

    # 划分训练集
    elif(self.train == 1):
        anc_img   = os.path.join(self.root_dir, self.train_dataset.iloc[idx][
'image_path'])
        img   = io.imread(anc_img)
        klass = self.train_dataset.iloc[idx]['name']
        # 图像及人脸身份
        sample = {'img': img, 'class': klass}
        if self.transform:
            sample['img'] = self.transform(sample['img'])

    # 划分测试集
    else:
        anc_img   = os.path.join(self.root_dir, self.test_dataset.iloc[idx]['i
mage_path'])
        img   = io.imread(anc_img)
        klass = self.test_dataset.iloc[idx]['name']
        # 图像及人脸身份
        sample = {'img': img, 'class': klass}
        if self.transform:
            sample['img'] = self.transform(sample['img'])

    return sample

    def __len__(self):
        if(self.train == 0):
            return len(self.training_triplets)
        elif(self.train == 1):
            return len(self.train_dataset)
        else:
            return len(self.test_dataset)
```

```
def get_data_loaders(batchSize=8, numWorker=4, train=0):
    # 转换数据格式
    data_transforms = transforms.Compose([
              transforms.ToPILImage(),
              transforms.RandomHorizontalFlip(),
```

```
            transforms.Resize([220,220]),
            transforms.ToTensor(),
            transforms.Normalize(mean = [0.5, 0.5, 0.5], std = [0.5, 0.5, 0.
5])])
    # 划分基准图像、正负样本
    dataset = triplet_dataset(transform = data_transforms, train = train)
    dataloaders = torch.utils.data.DataLoader(dataset, batch_size = batchSize,
shuffle = False, num_workers = numWorker)
    return dataset,dataloaders

# 划分后的训练集
dataset,train_loader = get_data_loaders(batchSize=64, train=0)
```

```
              name                                image_path
0          AJ_Cook                    [AJ_Cook/AJ_Cook_0001.jpg]
1         AJ_Lamas                  [AJ_Lamas/AJ_Lamas_0001.jpg]
2    Aaron_Eckhart          [Aaron_Eckhart/Aaron_Eckhart_0001.jpg]
3      Aaron_Guiel              [Aaron_Guiel/Aaron_Guiel_0001.jpg]
4  Aaron_Patterson  [Aaron_Patterson/Aaron_Patterson_0001.jpg]
Aaron_Peirsol ['Aaron_Peirsol/Aaron_Peirsol_0001.jpg', 'Aaron_Peirsol/Aaron_Pe
irsol_0002.jpg', 'Aaron_Peirsol/Aaron_Peirsol_0003.jpg', 'Aaron_Peirsol/Aaron_
Peirsol_0004.jpg'] 5
```

`dataset` 中包含了基准图像、及其正负样本的像素值，并标注了图像对应的人名，共分为5739组，划分为90批次（batch），每批次包含64组图像。

```
print("train_loader:",len(train_loader))
print("dataset:",len(dataset))
```

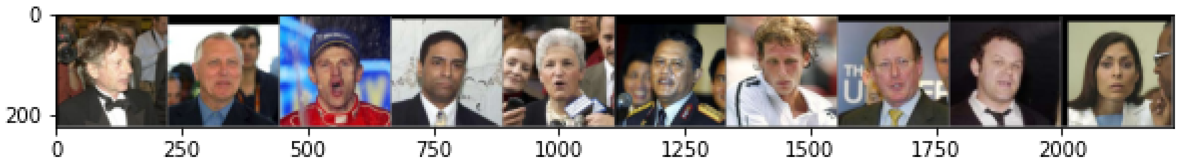
```
train_loader: 90
dataset: 5740
```

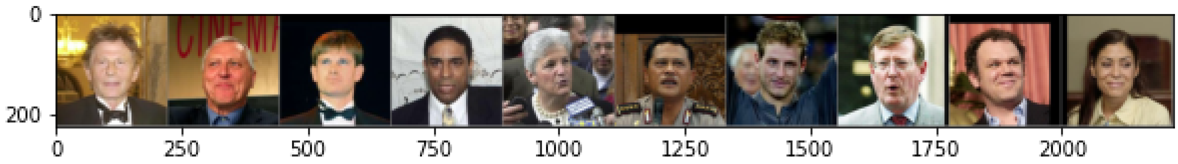
下图展示了10组基准图像、正负样本，及其基准图像对应的人名。可以看到第一、二行为基准图像和正样本，属于同一个人的人脸图像，第三行图像为负样本，是另一个人的人脸图像。

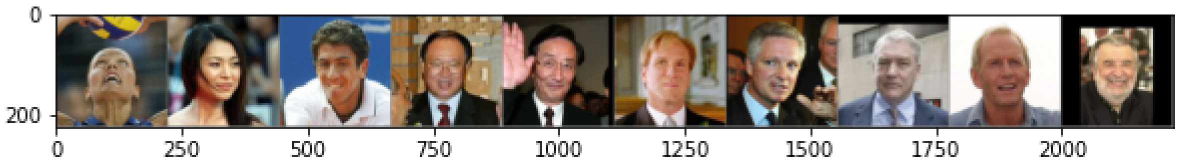
```
# 展示图像
def showImg(image):
    image = image/2 + 0.5
    plt.figure(figsize=(10,10))
    plt.imshow(np.transpose(image, (1,2,0)))

    for i, batch in enumerate(train_loader):
        # 分别展示基准图像、正负样本
        if i == 1:
            print(batch['postive_label'][0:10])
            showImg(torchvision.utils.make_grid(batch['anchor'][0:10], nrow=10
))
```

```
        showImg(torchvision.utils.make_grid(batch['postive'][0:10], nrow=10))
        showImg(torchvision.utils.make_grid(batch['negative'][0:10], nrow=10))
        break
```

['Roman_Polanski', 'Peter_Greenaway', 'Marcus_Gronholm', 'Oscar_Elias_Biscet', 'Lynn_Abraham', 'Dai_Bachtiar', 'David_Nalbandian', 'David_Trimble', 'John_Reilly', 'Laura_Hernandez']







4 构建模型

获得了标注好的训练集后，我们开始构建FaceNet网络。

4.1 三元组损失函数

因为FaceNet网络采用三元组损失函数，首先编写这一损失函数 `triplet_loss` 。损失函数的计算公式为：$\sum_i^N \left[\left\| f\left(x_i^a\right) - f\left(x_i^p\right) \right\|_2^2 - \left\| f\left(x_i^a\right) - f\left(x_i^n\right) \right\|_2^2 + margin \right]_+$

- $\left\| f\left(x_i^a\right) - f\left(x_i^p\right) \right\|_2^2$: 基准图像和正样本特征向量的欧式距离
- $\left\| f\left(x_i^a\right) - f\left(x_i^n\right) \right\|_2^2$: 基准图像和负样本特征向量的欧式距离
- $margin$: 基准图像与正负样本的特征向量距离，存在最小间隔

```
def triplet_loss(anchor, positive, negative, margin= 0.2):
    dist_pos = (anchor - positive).pow(2).sum(1) # 欧式距离
    dist_neg = (anchor - negative).pow(2).sum(1)
    loss = F.relu(dist_pos - dist_neg + margin) # relu函数计算损失值
    loss = loss.mean() # 三元组损失值
    return loss
```

4.2 Siamese网络

4.2 Siamese网络

Siamese网络输出图像的特征向量，网络由卷积层和全连接层组成。

```
class FaceNetSiameseNetwork(nn.Module):
    def __init__(self):
        super(FaceNetSiameseNetwork, self).__init__()
        # 卷积层
        self.convnn = nn.Sequential(
            nn.Conv2d(3, 64, 7, stride=2, padding=3),
            nn.MaxPool2d(3, 2,padding=1),
            nn.BatchNorm2d(64),
            nn.Conv2d(64, 64, 1, stride=1, padding=0),
            nn.Conv2d(64, 192, 3, stride=1, padding=1),
            nn.BatchNorm2d(192),
            nn.MaxPool2d(3, 2,padding=1),
            nn.Conv2d(192, 192, 1, stride=1, padding=0),
            nn.Conv2d(192, 384, 3, stride=1, padding=1),
            nn.MaxPool2d(3, 2,padding=1),
            nn.Conv2d(384, 384, 1, stride=1, padding=0),
            nn.Conv2d(384, 256, 3, stride=1, padding=1),
            nn.Conv2d(256, 256, 1, stride=1, padding=0),
            nn.Conv2d(256, 256, 3, stride=1, padding=1),
            nn.Conv2d(256, 256, 1, stride=1, padding=0),
            nn.Conv2d(256, 256, 3, stride=1, padding=1),
            nn.MaxPool2d(3, 2,padding=1),
            )

        # 全连接层
        self.fcnn1 = nn.Sequential(
            nn.Linear(7*7*256, 128),
            nn.Linear(128, 128),
            nn.Linear(128, 128),
            )

    def forward_on_one_datapt(self, datapt):
        output = self.convnn(datapt)
        output = output.view(output.size()[0], -1)
        output = self.fcnn1(output)
        output = output.view(output.size()[0], -1)
        output = F.normalize(output)
        return output

    def forward(self, a_img):
        output1 = self.forward_on_one_datapt(a_img)

        return output1
```

我们选择适用于大数据集和高维空间的Adam算法优化参数。

```
# 选择GPU/CPU计算
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net = FaceNetSiameseNetwork()
net = net.float()
net = net.to(device)

# 选择优化算法
optimizer = optim.Adam(net.parameters(), lr = 0.00005 )
train_loader.train = 0
optimizer
```

```
Adam (
Parameter Group 0
    amsgrad: False
    betas: (0.9, 0.999)
    eps: 1e-08
    lr: 5e-05
    weight_decay: 0
)
```

5 模型训练

构造了FaceNet模型后，我们可以将训练集代入模型，进行训练。训练过程为：首先通过Siamese网络，计算基准图像、正样本、负样本的特征向量；再计算三元组损失函数；根据损失函数优化Siamese网络参数；当损失值小于设定的阈值后，结束训练。

```
# 训练模型
counter = []
loss_history = []
iteration_number= 0

for epoch in range(15):
     for i, batch in enumerate(train_loader):
        # 计算基准图像、正负样本的特征向量
        anc_img,pos_img,neg_img,label1,label2= batch['anchor'],batch['postive'
],batch['negative'],batch['postive_label'],batch['negative_label']
        anc_img = anc_img.to(device)
        pos_img = pos_img.to(device)
        neg_img = neg_img.to(device)
        out1 = net(anc_img)
        out2 = net(pos_img)
        out3 = net(neg_img)
        optimizer.zero_grad()

        # 计算三元组损失函数
        loss_triplet = triplet_loss(out1,out2,out3)
        loss_triplet.backward()

         # 优化参数
         optimizer.step()
        if(iteration_number % 10 == 0):
            loss_history.append([iteration_number,loss_triplet])
```

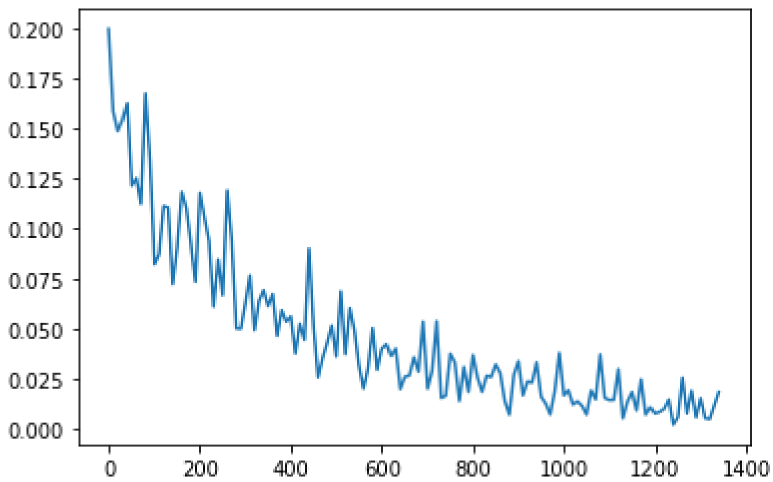
```
        loss_history.append([iteration_number, loss_triplet])
        iteration_number +=1

    # 损失值小于阈值时结束训练
    if loss_triplet <= 0.001:
        break
    print ("Cost after epoch %i: %f" % (epoch + 1, loss_triplet))
```

```
Cost after epoch 1: 0.085702
Cost after epoch 2: 0.062274
Cost after epoch 3: 0.062042
Cost after epoch 4: 0.033101
Cost after epoch 5: 0.034050
Cost after epoch 6: 0.011423
Cost after epoch 7: 0.016538
Cost after epoch 8: 0.012612
Cost after epoch 9: 0.011083
Cost after epoch 10: 0.007054
Cost after epoch 11: 0.010787
Cost after epoch 12: 0.010134
Cost after epoch 13: 0.007737
Cost after epoch 14: 0.007672
Cost after epoch 15: 0.005255
```

绘制损失函数的折线图，可以看到随着训练次数的增加，损失函数的值逐渐降低。

```
X = np.array([i for i,j in loss_history])
Y = np.array([j.cpu().detach().numpy() for i,j in loss_history])
plt.plot(X,Y)
```

```
[<matplotlib.lines.Line2D at 0x7f1ea8dc22b0>]
```



保存训练后得到的模型参数，在进行人脸识别时调用。

```
torch.save(net.state_dict(), "/content/faceNet.pth")
net.state_dict()
```

6 人脸识别

现在我们使用测试集，进行人脸识别，并观察识别效果。首先调用上述编写的 `get_data_loaders` 函数，整理测试集数据格式。

```
test_set,test_loaders  = get_data_loaders(train=2)
```

通过训练好的FaceNet网络，计算出每张人脸图像的特征向量。

```
images = []
labels = []
i = 0
for i_batch, sample_batched in enumerate(test_loaders):
    img, lbl = sample_batched['img'], sample_batched['class']
    # 图像的特征向量
    trained_img = net(img.to(device))
    images.append(trained_img.cpu().detach().numpy())
    labels.append(lbl)
```

将图像及其特征向量一一对应，放入表格中展示。

```
# 对应的人名，及其特征向量
imgs_flatten = []
labels_flatten = []
for i in range(len(images)):
    for j in range(len(labels[i])): # batch size
        imgs_flatten.append(images[i][j])
        labels_flatten.append(labels[i][j])

images = imgs_flatten
labels = labels_flatten

# 用dataframe格式展示
test_set = pd.DataFrame(list(zip(labels,images)), columns= ["Name","model"])
test_set
```

	Name	model
0	George_W_Bush	[0.009774414, -0.018760437, 0.04569927, 0.1805...
1	George_W_Bush	[0.06294935, -0.022815503, 0.06530338, 0.14501...
2	George_W_Bush	[-0.0068228394, 0.00064193405, 0.11699534, 0.1...
3	George_W_Bush	[0.027908, 0.08238165, 0.056143515, 0.14268696...
4	George_W_Bush	[-0.05615644, 0.07643677, 0.14157344, 0.062561...
...	...	...

789	Donald_Rumsfeld	[0.00532799, 0.0014077309, 0.11482674, 0.08033...
790	Antony_Leung	[-0.112804405, 0.08369416, 0.045227364, 0.0565...
791	Antony_Leung	[-0.08913507, 0.07487048, 0.19725062, -0.00458...
792	Antony_Leung	[-0.11247574, 0.0440643, 0.13153228, 0.0136022...
793	Antony_Leung	[-0.06941409, 0.09075172, 0.1906489, -0.015462...

794 rows × 2 columns

将同一个人的图像及特征向量整理到一行中。

```
test_set = test_set.groupby(['Name']).model.apply(lambda x: list(x.values)).r
eset_index()
test_set
```

	Name	model
0	Antony_Leung	[[-0.112804405, 0.08369416, 0.045227364, 0.056...
1	Carlos_Manuel_Pruneda	[[-0.1477274, 0.06181724, 0.08542253, 0.125788...
2	Donald_Rumsfeld	[[0.09454071, -0.020301603, 0.102025814, 0.073...
3	George_W_Bush	[[0.009774414, -0.018760437, 0.04569927, 0.180...
4	Gloria_Macapagal_Arroyo	[[0.0014552873, 0.07929268, -0.039004464, 0.08...
5	Harrison_Ford	[[-0.055699687, 0.1466127, 0.1053287, 0.187506...
6	Hilmi_Ozkok	[[-0.08276133, 0.008318439, 0.07294396, 0.0653...
7	John_Ashcroft	[[-0.0040352833, 0.032061446, 0.17385659, 0.05...
8	Jose_Maria_Aznar	[[-0.17265135, 0.00075782003, 0.054814138, 0.1...
9	Judy_Genshaft	[[0.08437047, 0.0999114, -0.09351107, 0.168229...

我们将测试集划分为两部分，一部分设定为已知身份的人脸数据集，作为数据库，变量名为 `Xtrain` ，对应的人名为 `Ytrain` 。一部分为需要进行识别的人脸图像 `Xtest` ，其对应的真实身份为 `Ytest` 。

```
# 将测试集划分为已知数据库Xtrain，和待识别人脸图像Xtest
def train_test_split(test_set, split_size=0.8):
    Xtrain = []; Xtest = []; Ytrain = []; Ytest = []

    for i in range(test_set.shape[0]):
        from_model = test_set.iloc[i]['model']
        label = test_set.iloc[i]['Name']
        from_model_len = len(from_model)
        # 划分数据
        for j in range(from_model_len):
            if j <= int (from_model_len * split_size):
                Xtrain.append(from_model[j])
```

```
                Ytrain.append(label)
            else:
                Xtest.append(from_model[j])
                Ytest.append(label)

    return np.array(Xtrain),np.array(Xtest),np.array(Ytrain),np.array(Ytest)

Xtrain, Xtest, Ytrain, Ytest = train_test_split(test_set)
print(Xtrain.shape, Ytrain.shape,Xtest.shape,Ytest.shape)
```

```
(641, 128) (641,) (153, 128) (153,)
```

通过KNN算法进行人脸识别：特征向量距离近的人脸图像归类为同一人。

```
# KNN函数
class KNN:
    def __init__(self, k, scalefeatures=False):
        self.K=k

    def train(self, X, Y):
        self.X_train=X
        self.Y_train=Y

    def predict(self, X):
        num_test = X.shape[0]
        y_pred = np.zeros(self.K, dtype = self.Y_train.dtype)
        pclass=[]
        # 计算欧氏距离
        compute_distance = euclidean_distances(X, self.X_train)
        # 找到最近距离，预测人脸身份
        for x in range(num_test):
            dist = np.sort(compute_distance[x])
            for y in range(self.K):
                index = np.where(dist[y] == compute_distance[x])
                y_pred[y] = self.Y_train[index][0]
            pclass.append(ss.mode(y_pred)[0][0])
        return np.array(pclass)
```

输入153张待预测人脸图像 `Xtest` ，判断该人脸图像与已知人脸数据集 `Xtrain` 中哪一人匹配。并输出预测结果。

```
clf = KNN(k=3)
clf.train(Xtrain, Ytrain)
y_pred = clf.predict(Xtest)
len(y_pred)
```

```
153
```

7 人脸识别效果

对比预测的人脸身份 `y_pred` 和实际身份 `Ytest` ，可以看到人脸识别模型达到了82.35%的准确率。

```
total= Ytest.shape[0]
correct = 0
for i in range(len(Ytest)):
    if y_pred[i] == Ytest[i]:
        correct += 1
print("Accuracy is : ", (correct/total) * 100)
```

```
Accuracy is :  82.35294117647058
```

通过表格可以直观展示出身份验证准确率。 `result` 中包含每一张人脸图像对应的真实人名和预测人名。

```
result = pd.DataFrame({
    '预测人名': y_pred,
    '真实人名': Ytest
})
```

将表格按照真实人名分组，可以观察识别效果。如"George_W_Bush"的人脸图像共有105张，其中有93张分类准确，即模型预测该图像是"George_W_Bush"的人脸。

```
result_group = result.groupby(['真实人名'])
result_group.describe()
```

	预测人名			
	count	unique	top	freq
真实人名				
Donald_Rumsfeld	24	3	Donald_Rumsfeld	16
George_W_Bush	105	3	George_W_Bush	93
Gloria_Macapagal_Arroyo	8	1	Gloria_Macapagal_Arroyo	8
Harrison_Ford	2	2	George_W_Bush	1
John_Ashcroft	10	4	John_Ashcroft	6
Jose_Maria_Aznar	4	2	Jose_Maria_Aznar	3

8 总结

本案例中，我们使用FaceNet模型进行人脸识别。首先将训练集分为基准图像、正样本、负样本，并将其传入Siamese网络计算特征向量；通过计算三元组损失，我们不断优化Siamese网络，最终使得基准图像和正样本的特征向量距离相近，和负样本的特征向量距离较大；最后传入待识别的人脸图像，根据KNN算法，找出数据库中与其特征向量距离相近的人脸，判断该人脸图像对应的人名。

发表您的讨论

petitio

n/)