人脸识别V2.0

笔记本: 个人笔记
创建时间: 2018/1/24 10:01 **更新时间:** 2018/1/24 15:40
作者: 桔兰梗
URL: file:///C:/Users/yeziy/Documents/My%20Knowledge/temp/941fffa4-b61f-440b-9aff-b6cb0ef8ff61/128/ind...

1.简介

本文以Siamese网络为基础，改进了原有的通过训练分类器来检测人脸的方法，大大提高了识别的准确度。该系统基python3.5/dlib/opencv2/tensorflow1.3-gpu环境，主要实现了人脸识别的功能。本代码地址:链接

Siamese网络是一种相似性度量方法，当类别数多，但每个类别的样本数量少的情况下可用于类别的识别、分类等。Siamese网络也非常适合于进行人脸识别研究，被测者仅需提供最少1张照片，即可正确识别出被测者。这与利用分类器来识别人脸有着很大的不同，利用分类器进行训练的方法需要每一类中有足够多的样本才行，这其实在实际生活中是不现实的。原因如下：无法对每一个人采取足够多的样本，当类别过多，每一类别的样本也足够多时，机器性能也就跟不上了，无法训练出合适的模型；无法对陌生人脸进行划分，主要指不属于分类器中的类别会被划分为分类器中的类别，例如，分类器可以识别3个人的人脸，当第4个人要识别时，他就会被误识别为这3个人中的其中一个。

Siamese 网络的主要思想是通过一个函数将输入映射到目标空间，在目标空间使用简单的距离（欧式距离等）进行对比相似度。在训练阶段去最小化来自相同类别的一对样本的损失函数值，最大化来自不同类别的一堆样本的损失函数值。Siamese网络训练时的输入是一对图片，这一对图片是X_1,X_2，标签是y。当X_1,X_2是同类时，y为0，当x_1,X_2不同类时，y为1 。Siamese网络的损失函数为L=（1-y)L_G(EW(X_1，X_2))+yL_I(EW(X_1，X_2))，其中E_W(X_1,X_2)=||G_W(X_1)-G_W(X_2)||。G_W(X)就是神经网络中的参数，其中L_G是只计算相同类别对图片的损失函数，L_I是只计算不相同类别对图片的损失函数。GitHub上有人使用Tensorflow在MNIST实现Siamese网络，链接。

然而，这还不够，仅仅利用Siamese网络来训练模型，还达不到好的效果。根据Siamese网络的主要思想，有人发明了Triplet 网络。triplet是一个三元组，这个三元组是这样构成的：从训练数据集中随机选一个样本，该样本称为Anchor，然后再随机选取一个和Anchor (记为x_a)属于同一类的样本和不同类的样本,这两个样本对应的称为Positive (记为x_p)和Negative (记为x_n)，由此构成一个（Anchor，Positive，Negative）三元组。有了上面的triplet的概念， triplet loss就好理解了。针对三元组中的每个元素（样本），训练一个参数共享或者不共享的网络，得到三个元素的特征表达，分别记为：f (x_i^a),f (x_i^p),f (x_i^n),。triplet loss的目的就是通过学习，让x_a和x_p特征表达之间的距离尽可能小，而x_a和x_n的特征表达之间的距离尽可能大，并且要让x_a与x_n之间的距离和x_a与x_p之间的距离之间有一个最小的间隔t。公式化表示就是：||f (x_i^a)-f (x_i^p)||+t<||f (x_i^a)-f (x_i^n)||,损失函数就是L={ ||f (x_i^a)-f (x_i^p)||-||f (x_i^a)-f (x_i^n)||+t }+。这里距离用欧式距离度量，+表示{ }内的值大于零的时候，取该值为损失，小于零的时候，损失为零。

推荐几篇论文供详细研究：文章末尾 - 论文推荐。

2.代码思路

2.1 数据集

本人用到过2种数据集，一种是LFW数据集，另一种是CAS-PEAL-R1 数据集。CAS-PEAL-R1 数据集训练了一段时间，效果不太好，就舍弃掉了。现在用LFW数据集，如有条件，选用更大的数据集。

2.2 结构

主要有face_lib文件夹、model文件夹、out文件夹、temp文件夹、train_faces文件夹、get_align_face.py文件、lfw_test.py文件、run.py文件、run_new.py文件、test.py文件。

2.2.1 face_lib文件夹

该文件夹下有align_dlib.py文件、inference.py文件、my_api.py文件、shape_predictor_68_face_landmarks.dat文件。

a. 其中align_dlib.py文件和shape_predictor_68_face_landmarks.dat文件取之于Openface,主要作用是对齐人脸。我主要调用的是align方法。

代码示例如下:

```
detector = align_dlib.AlignDlib(PREDICTOR_PATH) # PREDICTOR_PATH代表shape_predictor_68_face_landmarks.dat的目录
img = cv2.imread(path_name)                     # 读取图片
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)   # 转为RGB图片
face_align_rgb = detector.align(size, img_rgb)   # img_rgb必须为RGB图片，size我设为96
```

b. inference.py文件只有一个Siamese类，主要进行神经网络的配置。

network方法中：X为输入图片，keep_f为dropout的比率，采用的是VGG16的经典网络结构。输出的f3是一个128维的向量。

loss_with_spring方法：计算loss值，阈值margin=5.0。

look_like方法：计算2张照片的欧氏距离。

c. my_api.py文件包含我自己写的一些类与方法。

GetAlignedFace类：这个主要是把一张正常图片转化为大小为96*96对齐后的人脸图片。

Traversal类: 包括2个方法，get_triplet_data方法和generate_train_data方法。

get_triplet_data方法，读取一个文件夹 返回标签数(文件夹数)、图片数组和图片id，返回的都是列表（list）类型。generate_train_data方法，以遍历方式生成3元组训练数据。假设有4个类别，第一个类别有1张图片，第二个类别有2张图片，第三个类别有3张图片，第四个类别有4张图片。那么标签数是4，图片数组face_array, 第一个类别的 第1张图片可用face_array[0][0]表示，图片id用id_array表示，id_array=[[0], [0, 1], [0, 1, 2],[0, 1, 2, 3]]。id_array[0]中的0表示第一类别，其中id_array[0]=[0]表示第一类的第一张图片。这2个列表一个是图片，一个是图片id，face_array[0][0]在内存中占用4KB（图片大小4kb）的空间，而id_array[0][0]几乎忽略不计。以遍历方式生成的3元组训练数据，只生成图片id的排列。取第一类的第一张图片为Anchor，即x_a =id_array[0][0],我

用0_0表示，由于第一类只有一张图片，故x_p=0_0,x_n就从另外3类中取一张图片。遍历生成的3元组是{ 0_0,0_0,1_0；0_0,0_0,1_1；0_0,0_0,2_0；0_0,0_0,2_1；0_0,0_0,2_2；0_0,0_0,3_0；0_0,0_0,3_1； 0_0,0_0,3_2;0_0,0_0,3_3；1_0,1_1,0_0；1_0,1_1,2_0；1_0,1_1,2_1；1_0,1_1,2_2；1_0,1_1,3_0；1_0,1_1,3_1；1_0,1_1,3_2；1_0,1_1,3_3；2_0,2_1,0_0；2_0,2_1,1_0；2_0,2_1,1_1；2_0,2_1,3_0；2_0,2_1,3_1；2_0,2_1,3_2；2_0,2_1,3_3；2_0,2_2,0_0；2_0,2_2,1_0；2_0,2_2,1_1； 2_0,2_2,3_0；2_0,2_2,3_1； 2_0,2_2,3_2； 2_0,2_2,3_3； 2_1,2_2,0_0；2_1,2_2,1_0;2_1,2_2,1_1； 2_1,2_2,3_0； 2_1,2_2,3_1； 2_1,2_2,3_2； 2_1,2_2,3_3;3_0,3_1,0_0； 3_0,3_1,1_0； 3_0,3_1,1_1； 3_0,3_1,2_0； 3_0,3_1,2_1;3_0,3_1,2_2； 3_0,3_2,0_0； 3_0,3_2,1_0； 3_0,3_2,1_1； 3_0,3_2,2_0;3_0,3_2,2_1；3_0,3_2,2_2... }，等等，后面就不列出来了。

Random类：只有一个方法：generate_train_data方法。

这个方法是随机生成3元组，以上面的例子为例，随机生成的3元组是{ 0_0,0_0,1_0；1_0,1_1,2_1；2_0,2_2,3_0；3_0,3_3,2_1；}。思想是第一类中取2张图片，当作x_a,x_p，在随机从另外3类中取一张图片当做x_n，该方法一次只能生成4个3元组。

LfwTest类：这个主要利用Lfw数据集测试模型的性能的。negative_pairs.txt包含3000对人脸，这些人脸对不属于同一类；postive_pairs.txt也包含3000对人脸，这些人脸对属于同一类。针对这共6000对人脸来计算出他们的欧式距离。

LfwPlot类：这个主要根据阈值和准确率来画出表格。针对negative_pairs，欧式距离大于阈值的则认为判断正确；针对postive_pairs，欧式距离小于于阈值的则认为判断正确；分别计算他们的准确率，并画出表格。

2.2.2 model文件夹

该文件夹又包括random、traversal文件夹，分别存放以随机方法训练的模型和以遍历方法训练的模型。

2.2.3 out文件夹

主要是各种CSV文件的输出目录。

2.2.4 temp文件夹

不详细介绍。

2.2.5 train_faces文件夹

存放训练样本集，里面的人脸都经过裁剪与对齐。

2.2.6 get_align_face.py文件

以多进程的方式裁剪和对齐人脸图片。

2.2.7 lfw_test.py文件

Flag = 0,则以多进程的方式同时处理negative_pairs 、postive_pairs，并把计算得到的欧式距离存放在./temp/lfw/result文件夹中。Flag为1或者为2则分别计算其中一 个。Flag = 3,则通过得到的欧式距离画出表格。

2.2.8 run.py文件

以遍历方式训练人脸数据。该训练方式消耗资源特别巨大，样本数每多一点，3元组并成几何数增长，区区几百张图片，即可产生几千万种3元组，以我电脑上的2G显存的 GTX-960M显卡来训练，训练一个批次，则至少需要一个星期。

2.2.9 run_new.py文件

以随机方式训练人脸数据。因为在该方式下，一个批次产生的3元组数与类目数相同，所以训练一个批次所需时间非常少，但是损失值不容易收敛。硬件条件不行的推荐采用这种方式训练。

3. 程序运行流程：

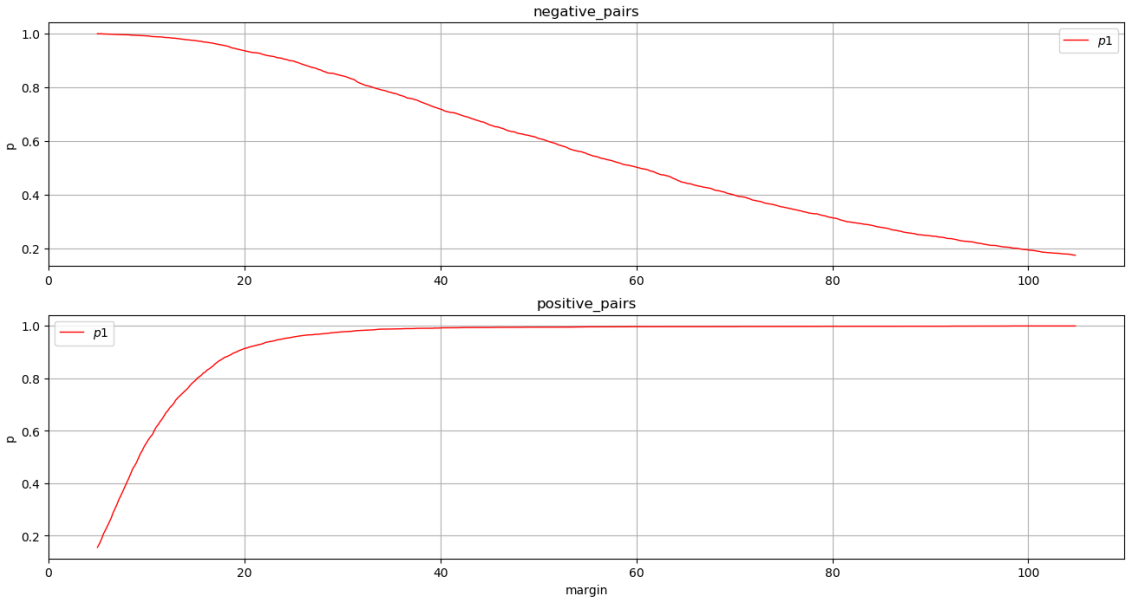
3.1 首先运行get_align_face.py文件。生成对齐过的人脸训练数据。

3.2 运行run_new.py文件，会出现 “我们发现模型，是否需要预训练 [yes/no]?” 第一次训练输入no，按回车，即开始训练。这样做的目的是可以间断训练，每次训练完一段时间退出，可以接着训练。

3.3 训练完成后，首先改lfw_test.py中的Flag=0,然后运行，运行完成后改Falg=3,再运行，即可看到准确率的表格。

4. 人脸识别准确率：

时间及硬件条件有限，在以随机方式训练人脸数据2天以后，其模型性能如下表格所示：



由此可以看出，当阈值为20时，negative_pairs 和postive_pairs对应的准确率都能达到90%以上。相信经过更长时间的训练，其准确率会更高。

5.总结

a. 多看大牛的论文，还可以对其网络结构进行改进，进一步提高准确率。

b. 可以把模型放在具体现实生活中进行识别人脸，看其表现如何（目前正在开发有关人脸识别的小程序）。

6. 论文推荐

1. **Hamming Distance Metric Learning**
2. **Targeting Ultimate Accuracy: Face Recognition via Deep Embedding**
3. **FaceNet: A Unified Embedding for Face Recognition and Clustering**
4. **Learning Deep Face Representation**
5. **DeepFace: Closing the Gap to Human-Level Performance in Face Verification**
6. **Learning a Similarity Metric Discriminatively, with Application to Face Verification**
7. **Siamese Neural Networks for One-shot Image Recognition**