# 人脸识别模型测试与优化

# 模型重训练

预训练模型已经达到比较高的精度了，但是因为预训练模型使用的数据集中欧美人脸较多，我们可以使用自己的数据重新训练一个模型，使之更加适合亚洲人脸的识别。

## 准备训练集

训练集使用的是自行采集的数据，共2w+张，年龄集中在20岁左右，其中男性人数约是女性的三倍。

训练集的文件结构如下：



每个文件均已重新命名，每个文件代表一个人物身份。

## 1.2.开始训练模型

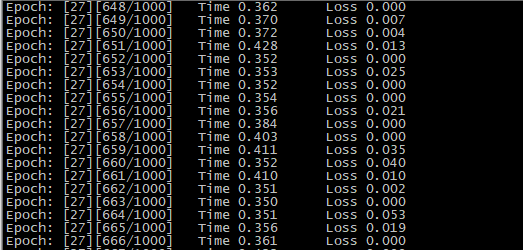
在重新训练模型之前需要设定好以下几个参数：

|  |  |
| --- | --- |
| 参数 | 说明 |
| logs\_base\_dir | 该参数是记录训练过程的文件保存的位置，可以用tensorbord来查看训练过程中损失的变化 |
| models\_base\_dir | 这是模型保存的位置，增量训练过程中，每过一个周期就会对模型进行一次保存 |
| gpu\_memory\_fraction | 该参数是GPU的使用率，为0-1之间的浮点数，如果设置过高可能是会造成显存不足 |
| pretrained\_model | 该参数是预训练模型的ckpt.data文件名前缀的字符串，若该参数为空字符，则重新开始训练一个模型 |
| data\_dir | 该参数是训练数据所在的文件夹 |
| model\_def | 该参数是模型所使用的网络结构名称 |
| batch\_size | 该参数是每批次的图片数量，因为训练时使用单个元素长度为3的FIFO队列，所以该参数必须为3的倍数 |
| image\_size | 该参数是图片随机裁剪后的尺寸，也是输入网络的图片尺寸 |
| random\_crop | 该参数用于判断是否进行随机裁剪。在裁剪过程中，若训练集图片像素大于网络输入尺寸，则将其裁剪成输入尺寸，若小网络输入尺寸，则改参数应指定为False,此时程序会对图片进行填充。 |
| random\_flip | 若指定该参数则对图片进行随机旋转 |

在初始训练的时候，使用inception\_resnet\_v1进行训练。这时，需要设定训练集所在的路径、模型保存的路径、log记录文件所在的路径，保证训练数据符合inception\_resnet\_v1的输入要求（160\*160）。然后运行train.py文件就可以开始训练。

## 1.3.记录损失

训练过程中，每一个批次的迭代，都会把loss函数的值输入到屏幕上。通过查看loss值的变化，可以初步了解到模型的训练情况。



该loss值记录了选取三元组的损失函数值，它并不会像交叉熵那样呈现出持续下降的趋势。当模型读取的批次数据中出现需要优化的三元组时，会产生大于0的loss，否则loss值为0。在模型训练初始，loss值大部分会比较大，随着训练的进行，loss的均值不断减小。

# 模型测试

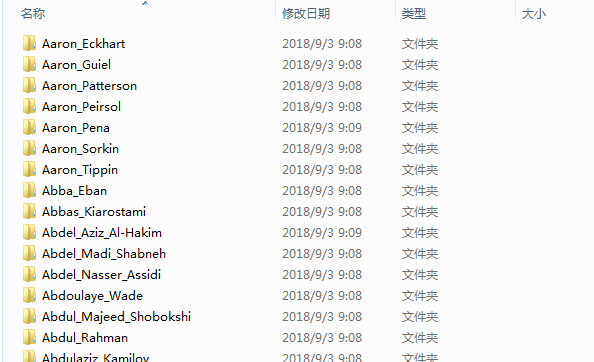
## 准备测试集

模型测试需要选择合适的测试集。在采集的数据集中，我们留下了30个人物的样本作为验证。此外还准备了lfw数据集作为辅助验证，防止模型对采集的数据集过度拟合。

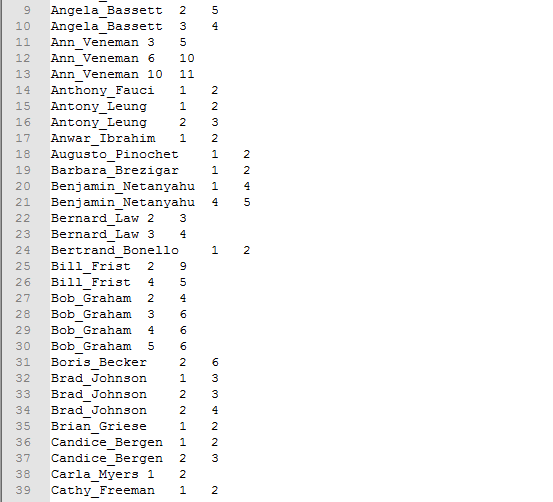
## 准备pairs文件

测试模型的时候需要两类样本：正样本对和正负样本对，即相同的人脸和不同的人脸，模型应该尽可能地把这两类样本分开。为了便捷地获得多个样本对，我们从文件中去读取这样的样本名称，然后根据名称到文件中获得样本。

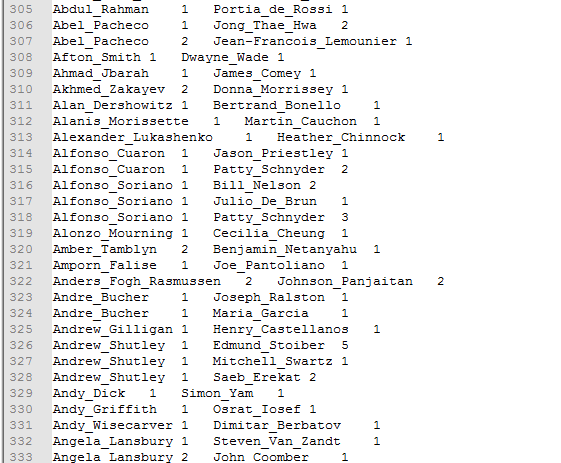
其中，测试集目录结构如下：



正样本对如下：



正样本-负样本对如下：



人物名称对应文件夹的名称，数字表示该人物的第n张照片。使用这种方法，读取300个正样本对和300个正负样本对，并重复10次做10折交叉验证。

## 生成pairs文件

LFW数据集中的pairs文件是随数据集附带的，当我们想要在自己的数据集上做验证时，就要自行制作pairs文件。制作pairs文件有很多种算法，总的来说只要满足将n个正样本对+n个负样本对的名称重复交叉验证的k-fold次数就可以了。

在LFW数据集提供的pairs文件中，n取值为300，k-fold取值为10。

## 2.4.模型验证指标

这一步使用k-fold交叉验证来读取pairs文件，程序会预先将pairs文件的内容读进内存，然后从重复k批次的正样本对和正负样本对中随机读取其中的一批次，用读取出的样本进行验证。

在第一次验证中，程序会从0-4每隔0.01为步长确定threshold值，对每个threshold值计算tp、tn、fp、fn四个指标，构成ROC曲线的要素。选取准确率最高的threshold作为分隔不同人脸的依据，输出该值下的准确率。

一次验证结束后，程序会读取pairs文件中余下的批次做同样的验证，并取k次验证的平均值作为最终验证结果。

该模块的主要衡量指标如下所示：

|  |  |  |
| --- | --- | --- |
| 指标 | 类型 | 说明 |
| nrof\_folds | int | 交叉验证的fold数 |
| tp | int | 测试样本中，判断为同一人，实际为同一人的样本数 |
| fp | int | 测试样本中，判断为同一人，实际为不同人的样本数 |
| tn | int | 测试样本中，判断为不同人，实际为不同人的样本数 |
| fn | int | 测试样本中，判断为不同人，实际为同一人的样本数 |

对tp、fp、tn、fn进行进一步计算，得到以下指标：

|  |  |  |
| --- | --- | --- |
| 指标 | 计算方式 | 说明 |
| TPR | tp/(tp+fn) | 查全率（召回率），实际为正类中预测为正类的比率 |
| FPR（FAR） | fp/(fp+tn) | 假正类率，实际为负类中预测为正类的比率 |
| PPV | tp/(tp+fp) | 假负类率，预测为正类中实际为正类的比率 |
| ACC | (tp+tn)/(tp+fp+tn+fn) | 准确率，即模型判断正确的比率 |
| VAl | 当FPR在低于某个比率时的TPR | 当预测错误比率低于某个数值时，样本的召回率 |

相关代码如下：

def calculate\_roc(thresholds, embeddings1, embeddings2, actual\_issame, nrof\_folds=10, distance\_metric=0, subtract\_mean=False):

assert(embeddings1.shape[0] == embeddings2.shape[0])

assert(embeddings1.shape[1] == embeddings2.shape[1])

#lfw中点对的个数

nrof\_pairs = min(len(actual\_issame), embeddings1.shape[0])

#这里thresholds是一个长度为400的数组，从0开始，间隔0.01

nrof\_thresholds = len(thresholds)

k\_fold = KFold(n\_splits=nrof\_folds, shuffle=False)

#tprs是一个10x400的数组

tprs = np.zeros((nrof\_folds,nrof\_thresholds))

fprs = np.zeros((nrof\_folds,nrof\_thresholds))

accuracy = np.zeros((nrof\_folds))

indices = np.arange(nrof\_pairs)

#将数据分成10份，一份是测试，9份是训练

for fold\_idx, (train\_set, test\_set) in enumerate(k\_fold.split(indices)):

if subtract\_mean:

mean = np.mean(np.concatenate([embeddings1[train\_set], embeddings2[train\_set]]), axis=0)

else:

mean = 0.0

dist = distance(embeddings1-mean, embeddings2-mean, distance\_metric)

acc\_train = np.zeros((nrof\_thresholds))

#在0到4，每间隔0.01的数字作为阈值，然后再选择其中值最大的作为阈值

for threshold\_idx, threshold in enumerate(thresholds):

#在当前阈值下，求训练数据的准确度

\_, \_, acc\_train[threshold\_idx] = calculate\_accuracy(threshold, dist[train\_set], actual\_issame[train\_set])

#获取0-4之间阈值下的最佳值

best\_threshold\_index = np.argmax(acc\_train)

for threshold\_idx, threshold in enumerate(thresholds):

tprs[fold\_idx,threshold\_idx], fprs[fold\_idx,threshold\_idx], \_ = calculate\_accuracy(threshold, dist[test\_set], actual\_issame[test\_set])

\_, \_, accuracy[fold\_idx] = calculate\_accuracy(thresholds[best\_threshold\_index], dist[test\_set], actual\_issame[test\_set])

#由于还有交叉验证，因此还需要求10次tpr和fpr均值

tpr = np.mean(tprs,0)

fpr = np.mean(fprs,0)

return tpr, fpr, accuracy

def calculate\_accuracy(threshold, dist, actual\_issame):

predict\_issame = np.less(dist, threshold)

tp = np.sum(np.logical\_and(predict\_issame, actual\_issame))

fp = np.sum(np.logical\_and(predict\_issame, np.logical\_not(actual\_issame)))

tn = np.sum(np.logical\_and(np.logical\_not(predict\_issame), np.logical\_not(actual\_issame)))

fn = np.sum(np.logical\_and(np.logical\_not(predict\_issame), actual\_issame))

tpr = 0 if (tp+fn==0) else float(tp) / float(tp+fn)

fpr = 0 if (fp+tn==0) else float(fp) / float(fp+tn)

acc = float(tp+tn)/dist.size

return tpr, fpr, acc

# 模型测试和优化

在确定模型训练和测试的策略之后，开始对训练模型的性能进行测试。该过程会在第一次训练出来的模型的基础上进行优化，并比较每一次优化之后的结果。

## 3.1.初次训练

该次模型训练的初始条件如下:

|  |  |
| --- | --- |
| 训练集人物数量 | 670人 |
| 训练集男女比例 | 约3：1 |
| 学习率 | 固定0.1 |
| 优化器 | Adagrad |
| 网络 | Inception\_resnet\_v1 |
| 测试集 | 测试集1：数据集中留下的28个人，700张图片，五官清晰可辨。  测试集2：LFW数据集 |
| 测试指标 | 准确率：模型将正负样本识别错误的概率  验证率：模型在准确率达到99%的时的召回率 |

在该条件下训练若干个周期，抽取训练后的模型在采集的数据集上进行测试，测试结果如下：

|  |  |  |
| --- | --- | --- |
| 测试模型 | 准确率 | 验证率 |
| 自训练100周期 | 0.8595 | 0.489 |
| 自训练125周期 | 0.8698 | 0.539 |
| 自训练150周期 | 0.8648 | 0.483 |
| 自训练200周期 | 0.8648 | 0.445 |
| 预训练模型 | 0.8750 | 0.645 |

在LFW下测试的结果如下：

|  |  |  |
| --- | --- | --- |
| 测试模型 | 准确率 | 验证率 |
| 自训练100周期 | 0.7248 | 0.177 |
| 自训练125周期 | 0.7238 | 0.189 |
| 自训练150周期 | 0.7168 | 0.159 |
| 自训练200周期 | 0.7160 | 0.165 |
| 预训练模型 | 0.9735 | 0.953 |

结果分析：

|  |  |
| --- | --- |
| 评估角度 | 评估结果 |
| 准确率 | 用采集的数据训练出的模型在同样的数据集上准确率相差较小，但是在LFW数据集上预训练模型明显优于自行训练的模型。 |
| 验证率 | 验证率体现了正负样本是否完全分开。测试发现，自行训练的模型验证率不是很高，即使在采集的数据下，自训练模型的验证率也低于预训练模型。 |
| 过拟合 | 模型产生了轻微的过拟合现象，在训练125周期时，准确率和验证率都表现得很好，之后开始略微下降。 |

## 3.2.改变网络结构

初次训练的模型有以下几个问题：

|  |  |
| --- | --- |
| Ⅰ | 泛化能力不佳，可能与训练集数量过少有关 |
| Ⅱ | 完全区分正负样本的能力不强 |
| Ⅲ | 出现了轻微的过拟合现象 |

优化时对训练条件的更改如下：

|  |  |
| --- | --- |
| 使用网络 | Inception\_resnet\_v1 —> inception\_resnet\_v2 |
| 输入维度 | 160\*160 —> 224\*224 |
| 其他 | 剔除偏侧角度过大，且看不到五官的图片 |

超参数更改如下：

|  |  |  |
| --- | --- | --- |
| 参数 | 更改方式 | 说明 |
| batch\_size | 45 —> 90 | 增加每批次样本数量，利于兼顾全局样本 |
| learning\_rate\_decay\_factor | 1.0 —>0.95 | 引入学习率指数衰减机制，该参数为指数衰减的系数 |
| Learning\_rate\_decay\_epochs | 100 —>60 | 该参数是学习率衰减的指数，实时的学习率计算方式为：初始学习率\*学习率衰减系数^(当前周期数/总周期数)。学习率会随着训练的进行而衰减并且衰减速率不断加快。引入该参数有助于模型在后期收敛到底部。 |
| keep\_prob | 1.0 —> 0.8 | 减少全连接层的激活比例，为模型泛化留下更多的冗余空间。 |

在采集的数据上测试结果如下：

|  |  |  |
| --- | --- | --- |
| 测试模型 | 准确率 | 验证率 |
| 自训练75周期 | 0.8853 | 0.365 |
| 自训练115周期 | 0.8805 | 0.431 |
| 自训练135周期 | 0.8650 | 0.472 |
| 自训练160周期 | 0.8690 | 0.468 |
| 自训练200周期 | 0.8702 | 0.463 |
| 预训练模型 | 0.8750 | 0.645 |

在LFW数据集上测试结果如下：

|  |  |  |
| --- | --- | --- |
| 测试模型 | 准确率 | 验证率 |
| 自训练75周期 | 0.6193 | 0.082 |
| 自训练115周期 | 0.6298 | 0.085 |
| 自训练135周期 | 0.6380 | 0.093 |
| 自训练160周期 | 0.6460 | 0.096 |
| 自训练200周期 | 0.6428 | 0.090 |
| 预训练模型 | 0.9735 | 0.975 |

结果分析：由于模型结构复杂而训练数据较少，inception\_resnet\_v2在该训练数据下的泛化能力不强。在120周期左右时，V2在采集数据集上的准确率要略微胜过V1，但是在LFW数据集上的表现要明显低于V1。

## 3.3.增量训练

### 3.3.1.保存模型

当觉得训练不够充分或者训练过程因为意外事件而被打断时，可以在已保存模型的基础上进行增量训练。增量训练之前需要对训练到一定周期的模型进行保存。

保存模型的代码如下：

def save\_variables\_and\_metagraph(sess, saver, summary\_writer, model\_dir, model\_name, step):

print('Saving variables')

start\_time = time.time()

checkpoint\_path = os.path.join(model\_dir, 'model-%s.ckpt' % model\_name)

saver.save(sess, checkpoint\_path, global\_step=step, write\_meta\_graph=False)

save\_time\_variables = time.time() - start\_time

print('Variables saved in %.2f seconds' % save\_time\_variables)

metagraph\_filename = os.path.join(model\_dir, 'model-%s.meta' % model\_name)

save\_time\_metagraph = 0

if not os.path.exists(metagraph\_filename):

print('Saving metagraph')

start\_time = time.time()

saver.export\_meta\_graph(metagraph\_filename)

save\_time\_metagraph = time.time() - start\_time

print('Metagraph saved in %.2f seconds' % save\_time\_metagraph)

保存完毕后，会在相应的文件夹里出现四个文件：



这四个文件用于恢复训练好的模型参数。增量训练使用的网络结构需要和已保存模型的结构保持一致。前一步骤使用inception\_resnet\_v1训练并保存了模型，增量训练时也要使用该网络。

### 3.3.2.恢复模型

在前一步保存的四个文件中，ckpt文件用于保存模型参数，meta文件用于保存网络结构，checkpoint用于告知tensorflow内部函数这是最新的检查点。增量训练主要用到这三个文件。首先需要将网络的结构加载到计算图中，然后把保存好的模型参数注入到网络结构中。

载入网络的代码如下：

import importlib

network = importlib.import\_module(args.model\_def)

prelogits, \_ = network.inference(image\_batch, args.keep\_probability,

phase\_train=phase\_train\_placeholder, bottleneck\_layer\_size=args.embedding\_size,

weight\_decay=args.weight\_decay)

载入模型参数的代码如下：

with sess.as\_default():

if args.pretrained\_model:

print('Restoring pretrained model: %s' % args.pretrained\_model)

saver.restore(sess, args.pretrained\_model)

### 3.3.3.调整参数

增量训练开始前可以调整超参数，改变训练效果。

由于inception\_resnet\_v1比较适合小规模数据，因此最终决定选用该网络做增量训练。在3.1中初次训练的模型中选择训练了100个周期的模型，并在其基础上作如下调整：

|  |  |  |
| --- | --- | --- |
| 参数 | 更改方式 | 说明 |
| batch\_size | 45 —> 90 | 增加每批次样本数量，利于兼顾到全局样本 |
| 优化器 | Adagrad —> Adam | Adagrad随着训练的进行梯度的分母会越来越大，使梯度逐渐接近于0，训练可能提前结束。使用Adam训练时采用自适应学习率，取消了预设的学习率数值。 |
| Alpha | 0.2 —> 0.24 | Alpha为正负样本的边界距离，适当增大以加强区分正负样本的能力。 |
| keep\_prob | 1.0 —> 0.8 | 为了缓解因训练时间过长导致的过拟合问题，将全连接层的激活比例下调。 |

在该条件下训练了200个周期以后，测试结果如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 使用模型 | 数据集 | 准确率 | 验证率 |
| 自训练模型 | LFW | 0.7185 | 0.161 |
| 采集数据 | 0.8810 | 0.560 |
| 预训练模型 | LFW | 0.9735 | 0.867 |
| 采集数据 | 0.8750 | 0.645 |

该模型在采集的数据集上的准确率超过了预训练模型，并且在LFW数据集上也有较好的表现。在测试集10折交叉验证上，除了第二折的最佳阈值为1.060以外，其余九折的最佳阈值均为1.160。