# 一 facenet复述

## 1.1 facenet在lfw上评估标准的算法

A) Roc评估标准

**B) val的评估标准.**

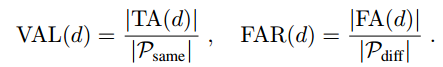
1. 在LFW上和youtube.
2. 按照”对”来做准确性评估(计数等)
3. 每队含有两个人脸图
4. 把两个人脸图分别输入到网络推断中(test), 等到两个人脸向量(embedding)
5. 计算两个embedding之间的距离(平方的L2距离)
6. 怎么计算是true accepts?



1. d 是超参数, 阈值.描述多”近”的距离算是”同一人脸”
2. Psame是gt. 是说在gt(真实标注对)上, 满足不大于d距离的样本对的个数.
3. 只在same标注对上去算被网络认为是同一个人的个数(“对”的个数)
4. Xi和xj是对应样本对的两个样本(两个人脸图)
5. D(xi,xj)是计算它们的一个L2距离.
6. 什么是false accepts?(定义false accepts)



1. 只在标注是false的对上做评估, 如果两个人脸图距离不大于d则认为是同一张人脸(显然这个是错误的推断结果)
2. 相应的VAL(d)和FAR(d)的含义



1. VAL是 正判正/正样本对的概率
2. FAR是 正判负/负样本的概率

|  |
| --- |
| thresholds,  阈值组(400个, 0~3.99, 间隔0.01)  embeddings1,embedding2,  两个人脸对. 是batchsize大小的.  actual\_issame, 含有true,false, 表示gt的这一对人脸是否同一人脸. 是dataset大小的.  nrof\_folds, k折.  KFold提供k折的方法  **def calculate\_val(thresholds, embeddings1, embeddings2, actual\_issame, far\_target, nrof\_folds=10):** |

1 def calculate\_val(thresholds, embeddings1, embeddings2, actual\_issame, far\_target, nrof\_folds=10):

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

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

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

5 nrof\_thresholds = len(thresholds)

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

7

8 val = np.zeros(nrof\_folds)

9 far = np.zeros(nrof\_folds)

10

11 diff = np.subtract(embeddings1, embeddings2)

12 dist = np.sum(np.square(diff),1)

13 indices = np.arange(nrof\_pairs)

14

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

16

17 # Find the threshold that gives FAR = far\_target

18 far\_train = np.zeros(nrof\_thresholds)

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

20 \_, far\_train[threshold\_idx] = calculate\_val\_far(threshold, dist[train\_set], actual\_issame[train\_set])

21 if np.max(far\_train)>=far\_target:

22 f = interpolate.interp1d(far\_train, thresholds, kind='slinear')

23 threshold = f(far\_target)

24 else:

25 threshold = 0.0

26

27 val[fold\_idx], far[fold\_idx] = calculate\_val\_far(threshold, dist[test\_set], actual\_issame[test\_set])

28

29 val\_mean = np.mean(val)

30 far\_mean = np.mean(far)

31 val\_std = np.std(val)

32 return val\_mean, val\_std, far\_mean

33

34

35 def calculate\_val\_far(threshold, dist, actual\_issame):

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

37 true\_accept = np.sum(np.logical\_and(predict\_issame, actual\_issame))

38 false\_accept = np.sum(np.logical\_and(predict\_issame, np.logical\_not(actual\_issame)))

39 n\_same = np.sum(actual\_issame)

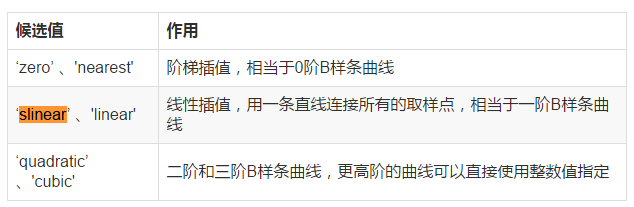
40 n\_diff = np.sum(np.logical\_not(actual\_issame))

41 val = float(true\_accept) / float(n\_same)

42 far = float(false\_accept) / float(n\_diff)

43 return val, fa

1. 何为interpolate.interp1d



## 1.2 facenet align利用mtcnn产生候选框的后处理算法

输入:

1. Out0是人脸框, out1是人脸概率.

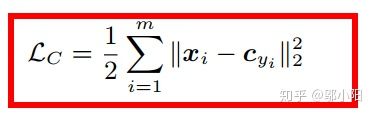
|  |
| --- |
| out **=** pnet**(**img\_y**)** **// 这个img\_y是传入的单张图的, 所以不太考虑batch.**  **out0 = np.transpose(out[0], (0,2,1,3))**  **out1 = np.transpose(out[1], (0,2,1,3))## h,w换个位置.**  boxes**,** \_ **=** generateBoundingBox**(**out1**[**0**,:,:,**1**].**copy**(),** out0**[**0**,:,:,:].**copy**(),** scale**,** threshold**[**0**])** |

|  |
| --- |
| **def** generateBoundingBox**(**imap**,** reg**,** scale**,** t**):**  # use heatmap to generate bounding boxes  stride**=**2  cellsize**=**12  imap **=** np**.**transpose**(**imap**) ## 只有是人脸的情况下才有意义(人脸概率大于阈值).**  dx1 **=** np**.**transpose**(**reg**[:,:,**0**])**  dy1 **=** np**.**transpose**(**reg**[:,:,**1**])**  dx2 **=** np**.**transpose**(**reg**[:,:,**2**])**  dy2 **=** np**.**transpose**(**reg**[:,:,**3**])**  y**,** x **=** np**.**where**(**imap **>=** t**)## 取出括号事件为True的”坐标”**  **if** y**.**shape**[**0**]==**1**:**  dx1 **=** np**.**flipud**(**dx1**)**  dy1 **=** np**.**flipud**(**dy1**)**  dx2 **=** np**.**flipud**(**dx2**)**  dy2 **=** np**.**flipud**(**dy2**)**  score **=** imap**[(**y**,**x**)]**  reg **=** np**.**transpose**(**np**.**vstack**([** dx1**[(**y**,**x**)],** dy1**[(**y**,**x**)],** dx2**[(**y**,**x**)],** dy2**[(**y**,**x**)]** **]))**  **if** reg**.**size**==**0**:**  reg **=** np**.**empty**((**0**,**3**))**  bb **=** np**.**transpose**(**np**.**vstack**([**y**,**x**]))**  **## 从12x12的金字塔层图缩放(扩张)到原图中的人脸框尺寸.**  q1 **=** np**.**fix**((**stride**\***bb**+**1**)/**scale**)**  q2 **=** np**.**fix**((**stride**\***bb**+**cellsize**-**1**+**1**)/**scale**)**  boundingbox **=** np**.**hstack**([**q1**,** q2**,** np**.**expand\_dims**(**score**,**1**),** reg**])**  **return** boundingbox**,** reg |

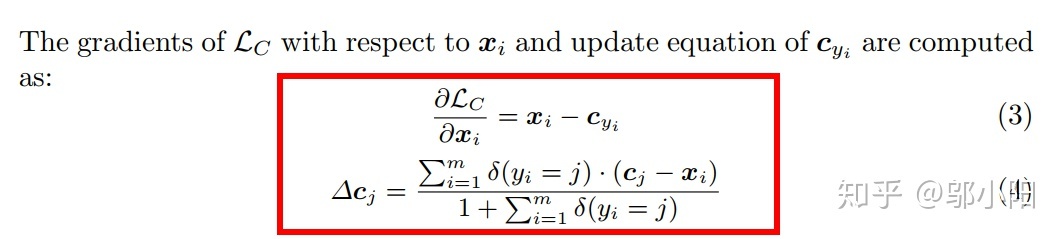
|  |
| --- |
| reg **=** np**.**transpose**(**np**.**vstack**([** dx1**[(**y**,**x**)],** dy1**[(**y**,**x**)],** dx2**[(**y**,**x**)],** dy2**[(**y**,**x**)]** **]))** |

## 1.3 centerloss算法

1. 为每一个类提供(初始化)一个类中心.
2. 对于每次batch, 最小化batch内每个样本对其对应类中心的距离.
3. 公式:
   1. Cyi是某次batch中,第i个样本对应类别的中心.(和特征Xi的维度一样).
   2. Xi是第i个样本对应的特征.
   3. 这个距离就是2范数.

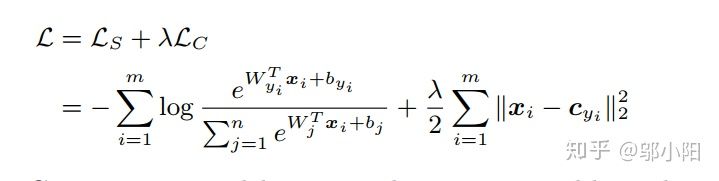


1. 距离LC求导如下:



* 1. 某个类别j的中心Cj只依赖于该类的特征计算得来.

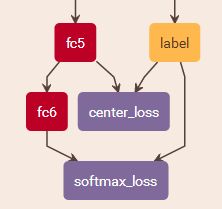
1. 训练总体loss是softmax Loss和center Loss之和.



1. [具体center loss的Cyi如何计算](#_实验结果)(可以结合后面实验来理解)

**center就像一个参数一样，先随机初始化，然后再每个迭代后在当前类别中更新一次。**

1. 理论上讲(从顾名思义角度), Cyi就是某种类别的中心, 在每次计算loss(在做bp前计算loss), 都要根据当前图片类别(当前样本所属类别)的所有图片计算一个Cyi(类别中心).这样太耗时.
2. 也可以,随机初始化Cyi, 当bp的时候,也计算Loss对Cyi的偏导数.同时更新Cyi的值.(这是一种近似).
3. center loss的原理主要是在softmax loss的基础上，通过对训练集的每个类别在特征空间分别维护一个类中心，在训练过程，增加样本经过网络映射后在特征空间与类中心的距离约束，从而兼顾了类内聚合与类间分离。  
   同样是作为训练阶段的辅助loss，**center loss相对于contrastive和triplet loss的优点显然省去了复杂并且含糊的样本对构造过程，只需要在特征输出层中引入即可**，看下图

  
另一个角度上说，center loss采取的是在训练过程中用空间换取时间的策略，对于不惜计算资源的深度学习任务而言也算是提供了一种新的思路.

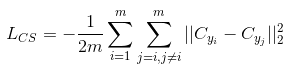
输入:

|  |
| --- |
| Features: [B, 128]  Label: [B]  Alfa: scaler  Nrof\_classes: scaler 10000+ |

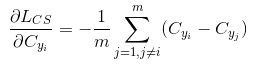
|  |
| --- |
| **def** center\_loss**(**features**,** label**,** alfa**,** nrof\_classes**):**  """Center loss based on the paper "A Discriminative Feature Learning Approach for Deep Face Recognition"  (http://ydwen.github.io/papers/WenECCV16.pdf)  """  nrof\_features **=** features**.**get\_shape**()[**1**]**  centers **=** tf**.**get\_variable**(**'centers'**,** **[**nrof\_classes**,** nrof\_features**],** dtype**=**tf**.**float32**,**  initializer**=**tf**.**constant\_initializer**(**0**),** trainable**=False)**  label **=** tf**.**reshape**(**label**,** **[-**1**])**  centers\_batch **=** tf**.**gather**(**centers**,** label**)**  diff **=** **(**1 **-** alfa**)** **\*** **(**centers\_batch **-** features**)**  centers **=** tf**.**scatter\_sub**(**centers**,** label**,** diff**)## label对应位置的centers得到更新**  loss **=** tf**.**reduce\_mean**(**tf**.**square**(**features **-** centers\_batch**))**  **return** loss**,** centers |

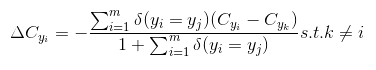
## 1.4 mutual loss 算法

Mutal Loss的损失函数如下,描述了不同类yi和yj之间的距离.但这个函数变小则是说明类间距离小. 我们需要整理一个损失函数:

[](file:///E:\cvdocs\a\cvdocs\faceDetectAndRecæ ¸å¿ä»£ç å¤è¿°.docx#_msocom_1)

 对应的更新量是:





         1. https://img2018.cnblogs.com/blog/1674599/201905/1674599-20190508115020132-1504238049.png当yi等于yj时, 该表达式值为1, 否则值为0.

         2. Cyk是除去Cyi(两个不同类)的**任意一个**类.

         3. 计算某次类和其他任意一个类的距离.

 输入:

Features: [B, 128]

Label: [B]

Alfa: scaler

Nrof\_classes: scaler 10000+

def mutual\_loss(features, label, alfa, nrof\_classes):

"""Mutual loss

"""

if args\_helper.facenet\_open\_debug ==True:

zeros\_tsr = tf.zeros([2, 3]) ##涓轰簡璋冪敤tf.Print鍋氱殑dummy.

nrof\_features = features.get\_shape()[1]

with tf.variable\_scope('center\_loss', reuse=True):

centers = tf.get\_variable('centers', [nrof\_classes, nrof\_features])

if args\_helper.facenet\_open\_debug ==True:

centers\_1 = centers

label = tf.reshape(label, [-1])

centers\_batch = tf.gather(centers, label)

centers\_batch\_ots = tf.gather(centers, label+1)

diff = -1\*(1 - alfa) \* (centers\_batch - centers\_batch\_ots)

centers = tf.scatter\_sub(centers, label, diff)

if args\_helper.facenet\_open\_debug ==True:

centers\_2 = centers

loss = tf.reduce\_mean(tf.square(features - centers\_batch))

if args\_helper.facenet\_open\_debug ==True:

tfprint.center\_loss = tf.Print(zeros\_tsr,["features,label,nrof\_classes, centers\_1, centers\_batch, diff, centers\_2, loss, ",tf.shape(features),tf.shape(label),nrof\_classes,tf.shape(centers\_1),tf.shape(centers\_batch),tf.shape(diff),tf.shape(centers\_2),loss],summarize=4)

return loss, centers

## 1.5 arcface loss

输入:

embedding: [B,F]

labels: [B]

num\_cls: scaler, 10000+

margin: scaler, 4

[复制代码](javascript:void(0);)

'''

embeddings.shape = [None, 10575]

embeddings.transpose.shape = [10575, None]

'''

def angular\_softmax\_loss(embeddings,labels,num\_cls, margin=4):

l = 0.

print("embeddings.shape = ",embeddings.get\_shape().as\_list())

#embeddings=tf.transpose(embeddings) # 128, 90 [featuremap num, batchsize], batchsize 榛樿鍊兼槸90

#embeddings=tf.transpose(embeddings) # 128, 90 [featuremap num, batchsize], batchsize 默认值是90

# 范数(norm)，是具有“长度”概念的函数。

# 1. tf.norm的axis为1, 是把每一行作为向量, 计算每一行的向量范数. 然后把这些向量范数值(标量)组合在一起.

# 2. 把输入的矩阵认为是n个向量.

# 3. 特别地,当axis为None时, 会把整个矩阵认为是一个向量来算.

# 4. 具体到这里, embedding本来是[B 128]的, 然后输入tf.norm的axis=1.

# a) 应该是把每个batch的128维度数值作为一个向量(作为一行)(这样才能解释通,才有物理意义.), 计算这个向量的范数.

# b) 对比于把90个batch作为向量, 计算的是90个batch下,128各自维度的"长度"..., 这个更有意义!!!

'''embeddings\_norm = tf.norm(embeddings, axis=1)'''

zeros\_tsr = tf.zeros([2, 3])

print("embeddings.transpose.shape = ",embeddings.get\_shape().as\_list())

inputs\_shape = embeddings.get\_shape().as\_list()

''' # debug shape鍜宎s\_list. 灏芥棭閲囩敤print鍑烘潵as\_list涔嬪悗鐨勫€?

weight = tf.Variable(initial\_value=tf.random\_normal((num\_cls,inputs\_shape[0])) \* tf.sqrt(2 / inputs\_shape[0]),dtype=tf.float32,name='weights') # shaep =classes, features,

tfprint.asoftmax\_loss = tf.Print(zeros\_tsr,["label shape",tf.shape(labels),tf.shape(weight)],summarize=4)

return zeros\_tsr

'''

with tf.variable\_scope("softmax"):

'''<1> embedding及 eb/||eb||, embedding\_norm'''

'''<2> weights 及 w/||w||, weights\_norm'''

## 1. 做norm时,消除的都是Feature的个数.

## a) embedding [B F], 按照axis=1消除F.

## b) weights [F C], 按照axis=0消除F.

## 2. 区别tf.nn.normalize和tf.norm

## a) tf.nn.norm 是求解矩阵的范数. 当axis有值时, axis=1,则把每行当成向量, 计算向量的范数.

## b) tf.nn.normalize 是两部分的组合.

## i) 先做tf.nn.norm,求出范数.

## ii) 再利用上面的范数对原来的矩阵做归一化.

## iii) 效果是 T/||T||

embeddings\_div\_embedding\_norm = tf.nn.l2\_normalize(embeddings, axis=1)

embeddings\_norm = tf.norm(embeddings,axis=1)

weights = tf.get\_variable(name='embedding\_weights',

#shape=[inputs\_shape[0], inputs\_shape[0]], # 10575 10575 澶ぇ浜? GPU鍐呭瓨涓嶅浜?

shape=[inputs\_shape[1],num\_cls],

#classes,inputs\_shape[1]

# 1. 做成 feature dim, classnum=10575

# 2. feature dim是和basebone相关的.

# 3. 所以此处是 [128, num\_cls] ,aka [128, 10575]

initializer=tf.contrib.layers.xavier\_initializer())

w\_origin = weights

weights = tf.nn.l2\_normalize(weights, axis=0)

weights\_norm = tf.norm(weights,axis=0)

weights\_div\_weights\_norm = weights

'''<3> weights和label相关的'''

weight\_unit\_batch = tf.gather(weights\_div\_weights\_norm,labels) # shaep =batch,features\_num,

'''<4> 计算cos\_theta'''

cos\_theta = tf.reduce\_sum(tf.multiply(embeddings\_div\_embedding\_norm,weight\_unit\_batch),axis=1) # shaep =batch,

cos\_theta\_power = tf.square(cos\_theta)

cos\_theta\_biq = tf.pow(cos\_theta, 4)

sign0 = tf.sign(cos\_theta)

sign3 = tf.multiply(tf.sign(2\*cos\_theta\_power-1), sign0)

sign4 = 2\*sign0 + sign3 -3

result=sign3\*(8\*cos\_theta\_biq-8\*cos\_theta\_power+1) + sign4

'''<5> cos\_4theta并不能天然解决问题,需要再算成prediction的logits(margin\_logits)'''

logits\_inputs = tf.reduce\_sum(tf.multiply(embeddings,weight\_unit\_batch),axis=1) # shaep =batch,

logits = tf.matmul(embeddings,weights\_div\_weights\_norm) #shape = [batch,classes] x \* w\_unit ## 鍘熷logits

##angu\_theta = tf.acos(cos\_theta)

##cos\_4Theta = tf.cos(4\*angu\_theta) ## result 的算法和实际要求的cos(4θ)的值不同,这是何意?

margin\_logits = tf.multiply(result, embeddings\_norm)

f = 1.0/(1.0+l) # l:lambda

ff = 1.0 - f

index\_range = tf.range(start=0,limit= tf.shape(embeddings,out\_type=tf.int64)[0],delta=1,dtype=tf.int64)

index\_labels = tf.stack([index\_range, labels], axis = 1)

index\_logits = tf.scatter\_nd(index\_labels,tf.subtract(margin\_logits,logits\_inputs), tf.shape(logits,out\_type=tf.int64))# 鍏冲績鐨勬墠鏈夊€?鍏朵粬鐨勪负0鍊?

logits\_final = f \* logits + ff \* index\_logits

loss = tf.reduce\_mean(tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(labels=labels,logits=logits\_final))

pred\_prob = tf.nn.softmax(logits=logits\_final)

return loss

[复制代码](javascript:void(0);)

# 二 物体检测

## 2.1 SPPNet

### 2.1.1 SPP Pooling算法

## 2.2 Fast R-CNN

### 2.2.1 多任务损失函数

### 2.2.2 RoI Pooling算法

注意处理重叠区域的算法

### 2.2.3 RoI pooling的前项推断

### 2.2.4 RoI pooling的Bp算法