# 一 环境及数据集介绍

## 1.1 代码环境

以<https://github.com/BobLiu20/mtcnn_tf>为base版本.

## 1.2 训练数据集

### 1.2.1 WIDER train数据集.

WIDER face dataset数据集.( [**WIDER FACE: A Face Detection Benchmark**)](http://mmlab.ie.cuhk.edu.hk/projects/WIDERFace/)

特点:

1. Face detection benchmark datasets, 图片均为公共图片
2. 含有32203张图, 有393703个人脸(scale, pose,occlusion(遮盖)等多变性图片).
3. 含有61分类事件. 40%/10%/50%的train,val,test占比.
4. 对于test图片,并不release对应的boundingbox ground truth.用户需要提交预测文件.由WIDER来做预测.



### 1.2.2 landmark 数据集

对于人脸关键点数据集采用的是:[CNN\_FacePoint](http://mmlab.ie.cuhk.edu.hk/archive/CNN_FacePoint.htm)其特点是:

1. 含有5590个LFW图像和7876个网络下载的图像.
2. trainImageList.txt和testImageList.txt分别保存有train,val数据集.

其格式是: 图片名字+矩形框+landmark的5个关键点(five facial points).

# 二 训练MTCNN

2.1 训练命令

|  |
| --- |
| #!/bin/bash  **set** -e  ### All of your tmp data will be saved in ./tmp folder  **echo** "Hello! I will prepare training data and starting to training step by step."  # 1. checking dataset if OK  **if** **[** **!** **-d** "./dataset/WIDER\_train/images" **];** **then**  **echo** "Error: The WIDER\_train/images is not exist. Read dataset/README.md to get useful info."  **exit**  **fi**  **if** **[** **!** **-d** "./dataset/lfw\_5590" **];** **then**  **echo** "Error: The lfw\_5590 is not exist. Read dataset/README.md to get useful info."  **exit**  **fi**  **echo** "Checking dataset pass."  **if** **[** **-d** "./tmp" **];** **then**  **echo** "Warning: The tmp folder is not empty. A good idea is to run ./clearAll.sh to clear it before training."  **fi**  # 2. stage: P-Net  ### generate training data(Face Detection Part) for PNet  **echo** "Preparing P-Net training data: bbox"  **python** prepare\_data**/**gen\_hard\_bbox\_pnet.py  ### generate training data(Face Landmark Detection Part) for PNet  **echo** "Preparing P-Net training data: landmark"  **python** prepare\_data**/**gen\_landmark\_aug.py **--**stage**=**pnet  ### generate tfrecord file for tf training  **echo** "Preparing P-Net tfrecord file"  **python** prepare\_data**/**gen\_tfrecords.py **--**stage**=**pnet  ### start to training P-Net  **echo** "Start to training P-Net"  **python** training**/**train.py **--**stage**=**pnet  # 3. stage: R-Net  ### generate training data(Face Detection Part) for RNet  **echo** "Preparing R-Net training data: bbox"  **python** prepare\_data**/**gen\_hard\_bbox\_rnet\_onet.py **--**stage**=**rnet  ### generate training data(Face Landmark Detection Part) for RNet  **echo** "Preparing R-Net training data: landmark"  **python** prepare\_data**/**gen\_landmark\_aug.py **--**stage**=**rnet  ### generate tfrecord file for tf training  **echo** "Preparing R-Net tfrecord file"  **python** prepare\_data**/**gen\_tfrecords.py **--**stage**=**rnet  ### start to training R-Net  **echo** "Start to training R-Net"  **python** training**/**train.py **--**stage**=**rnet  # 4. stage: O-Net  ### generate training data(Face Detection Part) for ONet  **echo** "Preparing O-Net training data: bbox"  **python** prepare\_data**/**gen\_hard\_bbox\_rnet\_onet.py **--**stage**=**onet  ### generate training data(Face Landmark Detection Part) for ONet  **echo** "Preparing O-Net training data: landmark"  **python** prepare\_data**/**gen\_landmark\_aug.py **--**stage**=**onet  ### generate tfrecord file for tf training  **echo** "Preparing O-Net tfrecord file"  **python** prepare\_data**/**gen\_tfrecords.py **--**stage**=**onet  ### start to training O-Net  **echo** "Start to training O-Net"  **python** training**/**train.py **--**stage**=**onet  # 5. Done  **echo** "Congratulation! All stages had been done. Now you can going to testing and hope you enjoy your result."  **echo** "haha...bye bye" |

## 2.1 生成Pnet训练数据

### 2.1.1 生成Neg,pos等boxes

prepare\_data**/**gen\_hard\_bbox\_pnet.py

|  |
| --- |
| **def** gen\_hard\_bbox\_pnet**(**srcDataSet**,** srcAnnotations**):**  srcDataSet **=** os**.**path**.**join**(**rootPath**,** srcDataSet**)**  srcAnnotations **=** os**.**path**.**join**(**rootPath**,** srcAnnotations**)**  saveFolder **=** os**.**path**.**join**(**rootPath**,** "tmp/data/pnet/"**)**  **print(**">>>>>> Gen hard samples for pnet..."**)**  **## 根据gtbox,原始图片,裁剪生成positive, negative,not care的矩形数据库.因为mtcnn输入是12x12的图,原图太大了. 所以要做一些crop.**  **同时保证正负样本均衡分布.**  typeName **=** **[**"pos"**,** "neg"**,** "part"**]**  saveFiles **=** **{}**  **for** tp **in** typeName**:**  \_saveFolder **=** os**.**path**.**join**(**saveFolder**,** tp**)**  **if** **not** os**.**path**.**isdir**(**\_saveFolder**):**  os**.**makedirs**(**\_saveFolder**)**  saveFiles**[**tp**]** **=** open**(**os**.**path**.**join**(**saveFolder**,** "{}.txt"**.**format**(**tp**)),** 'w'**)**  annotationsFile **=** open**(**srcAnnotations**,** "r"**)**  pIdx **=** 0 # positive  nIdx **=** 0 # negative  dIdx **=** 0 # dont care  idx **=** 0  **for** annotation **in** annotationsFile**:**  annotation **=** annotation**.**strip**().**split**(**' '**)**  # image path  imPath **=** annotation**[**0**]**  # boxed change to float type  bbox **=** map**(**float**,** annotation**[**1**:])**  # gt. each row mean bounding box  boxes **=** np**.**array**(**bbox**,** dtype**=**np**.**float32**).**reshape**(-**1**,** 4**)**  #load image  img **=** cv2**.**imread**(**os**.**path**.**join**(**srcDataSet**,** imPath **+** '.jpg'**))**  idx **+=** 1  height**,** width**,** channel **=** img**.**shape  # 1. NEG: random to crop negative sample image  **## 生成规定格式的负样本(iou<0.3).**  **Region大小是变化的.后来在把它们同意resize到12x12(pnet输入是12x12)的.**    **上图蓝色框是随机的neg框.生成过程是:**   1. **每次选一个size, 在0到虚线间找一个点做为左上角点(这个虚线到末尾的距离是size,同时蓝框是正方形,其变长也是size).** 2. **这个size是可以变化的,如右图所示变大了.** 3. **对于每个size,都只有一个篮框.** 4. **蓝框描述的Neg样本,其与gtbox的iou不能大于0.3.**      1. **~~这些不同尺寸的负样本框,需要resize到12x12(pnet输入shape).~~**   negNum **=** 0  **while** negNum **<** 50**:**  size **=** np**.**random**.**randint**(**12**,** min**(**width**,** height**)** **/** 2**)**  # top\_left  nx **=** np**.**random**.**randint**(**0**,** width **-** size**)**  ny **=** np**.**random**.**randint**(**0**,** height **-** size**)**  # random crop  **## 生成矩形框,为了先计算一下iou,如果iou合适(小于0.3)才会把图像的数据保存下来.**  crop\_box **=** np**.**array**([**nx**,** ny**,** nx **+** size**,** ny **+** size**])**  # cal iou and iou must below 0.3 for neg sample  iou **=** IoU**(**crop\_box**,** boxes**)**  **if** np**.**max**(**iou**)** **>=** 0.3**:**  **continue ## 负样本不满足要求就重新random,此时不消耗negNum**  # crop sample image  cropped\_im **=** img**[**ny **:** ny **+** size**,** nx **:** nx **+** size**,** **:]**  resized\_im **=** cv2**.**resize**(**cropped\_im**,** **(**12**,** 12**),** interpolation**=**cv2**.**INTER\_LINEAR**)**  # now to save it  save\_file **=** os**.**path**.**join**(**saveFolder**,** "neg"**,** "%s.jpg"**%**nIdx**)**  **saveFiles['neg'].write(save\_file + ' 0\n')**  cv2**.**imwrite**(**save\_file**,** resized\_im**)**  nIdx **+=** 1  negNum **+=** 1  **for** box **in** boxes**:**  # box (x\_left, y\_top, x\_right, y\_bottom)  x1**,** y1**,** x2**,** y2 **=** box  #bbox's width and height  w**,** h **=** x2 **-** x1 **+** 1**,** y2 **-** y1 **+** 1  # ignore small faces  # in case the ground truth boxes of small faces are not accurate  **## gtbox中可能有些人脸太小了,mtcnn对这部分支持不好.去掉它们**  **if** max**(**w**,** h**)** **<** 40 **or** x1 **<** 0 **or** y1 **<** 0**:**  **continue**  # 2. NEG: random to crop sample image in bbox inside  **## 此处创造一些和gtbox相交的负样本.**     1. **红色虚线框是负样本左上角可能落在的位置(是四个红色矩阵框)** 2. **它们的特点是蓝色矩形框都能和gt box相交.** 3. **Size是从12到w,h的最小值的一半.** 4. **和gtbox的Iou大于0.3的不需要.** 5. **当x1<size时, 如下图的”蓝色虚线框”描述的是”可能框的左上角落的范围”, 蓝色框是超过图像尺寸的非法部分.这部分被去掉.**   max**(-**size**,** **-**x1**)的作用.**    **for** i **in** range**(**5**):**  size **=** np**.**random**.**randint**(**12**,** min**(**width**,** height**)** **/** 2**)**  # delta\_x and delta\_y are offsets of (x1, y1)  delta\_x **=** np**.**random**.**randint**(**max**(-**size**,** **-**x1**),** w**)**  delta\_y **=** np**.**random**.**randint**(**max**(-**size**,** **-**y1**),** h**)**  nx1 **=** int**(**max**(**0**,** x1 **+** delta\_x**))**  ny1 **=** int**(**max**(**0**,** y1 **+** delta\_y**))**  **if** nx1 **+** size **>** width **or** ny1 **+** size **>** height**:**  **continue**  crop\_box **=** np**.**array**([**nx1**,** ny1**,** nx1 **+** size**,** ny1 **+** size**])**  Iou **=** IoU**(**crop\_box**,** boxes**)**  **if** np**.**max**(**iou**)** **>=** 0.3**:**  **continue**  cropped\_im **=** img**[**ny1**:** ny1 **+** size**,** nx1**:** nx1 **+** size**,** **:]**  resized\_im **=** cv2**.**resize**(**cropped\_im**,** **(**12**,** 12**),** interpolation**=**cv2**.**INTER\_LINEAR**)**  save\_file **=** os**.**path**.**join**(**saveFolder**,** "neg"**,** "%s.jpg"**%**nIdx**)**  saveFiles**[**'neg'**].**write**(**save\_file **+** ' 0\n'**)**  cv2**.**imwrite**(**save\_file**,** resized\_im**)**  nIdx **+=** 1  # 3. POS and PART  **## 生成POS和PART框.**   1. **根据左上角,找到中心点.**   x1 **+** w **/** 2   1. **在中心点左右gtbox宽1/5内随机.**   delta\_x **=** np**.**random**.**randint**(-**w **\*** 0.2**,** w **\*** 0.2**)**   1. **根据这个锚点,找到POS框的左上角**   **-** size **/** 2   1. **然后找到右下角**   **相当于POS矩形端点到中心点的x差,y差都是size/2.**  **进而POS矩形窗口的变长为size.**   1. **并且需要保证IOU在0.65以上(含)**     **for** i **in** range**(**20**):**  # pos and part face size [minsize\*0.8,maxsize\*1.25]  size **=** np**.**random**.**randint**(**int**(**min**(**w**,** h**)** **\*** 0.8**),** np**.**ceil**(**1.25 **\*** max**(**w**,** h**)))**  # delta here is the offset of box center  delta\_x **=** np**.**random**.**randint**(-**w **\*** 0.2**,** w **\*** 0.2**)**  delta\_y **=** np**.**random**.**randint**(-**h **\*** 0.2**,** h **\*** 0.2**)**  #show this way: nx1 = max(x1+w/2-size/2+delta\_x)  nx1 **=** max**(**x1 **+** w **/** 2 **+** delta\_x **-** size **/** 2**,** 0**)**  #show this way: ny1 = max(y1+h/2-size/2+delta\_y)  ny1 **=** max**(**y1 **+** h **/** 2 **+** delta\_y **-** size **/** 2**,** 0**)**  nx2 **=** nx1 **+** size  ny2 **=** ny1 **+** size  **if** nx2 **>** width **or** ny2 **>** height**:**  **continue**  crop\_box **=** np**.**array**([**nx1**,** ny1**,** nx2**,** ny2**])**  #yu gt de offset  offset\_x1 **=** **(**x1 **-** nx1**)** **/** float**(**size**)**  offset\_y1 **=** **(**y1 **-** ny1**)** **/** float**(**size**)**  offset\_x2 **=** **(**x2 **-** nx2**)** **/** float**(**size**)**  offset\_y2 **=** **(**y2 **-** ny2**)** **/** float**(**size**)**  #crop  cropped\_im **=** img**[**int**(**ny1**)** **:** int**(**ny2**),** int**(**nx1**)** **:** int**(**nx2**),** **:]**  #resize  resized\_im **=** cv2**.**resize**(**cropped\_im**,** **(**12**,** 12**),** interpolation**=**cv2**.**INTER\_LINEAR**)**  box\_ **=** box**.**reshape**(**1**,** **-**1**)**  **if** IoU**(**crop\_box**,** box\_**)** **>=** 0.65**:**  save\_file **=** os**.**path**.**join**(**saveFolder**,** "pos"**,** "%s.jpg"**%**pIdx**)**  saveFiles**[**'pos'**].**write**(**save\_file **+** ' **1** %.2f %.2f %.2f %.2f\n'**%(**offset\_x1**,** offset\_y1**,** offset\_x2**,** offset\_y2**))**  cv2**.**imwrite**(**save\_file**,** resized\_im**)**  pIdx **+=** 1  **elif** IoU**(**crop\_box**,** box\_**)** **>=** 0.4**:**  save\_file **=** os**.**path**.**join**(**saveFolder**,** "part"**,** "%s.jpg"**%**dIdx**)**  saveFiles**[**'part'**].**write**(**save\_file **+** ' **-1** %.2f %.2f %.2f %.2f\n'**%(**offset\_x1**,** offset\_y1**,** offset\_x2**,** offset\_y2**))**  cv2**.**imwrite**(**save\_file**,** resized\_im**)**  dIdx **+=** 1  printStr **=** "\r[{}] pos: {} neg: {} part:{}"**.**format**(**idx**,** pIdx**,** nIdx**,** dIdx**)**  sys**.**stdout**.**write**(**printStr**)**  sys**.**stdout**.**flush**()**  **for** f **in** saveFiles**.**values**():**  f**.**close**()**  **print** '\n' |

### 2.1.2 生成关键点

prepare\_data/gen\_landmark\_aug.py

|  |
| --- |
| **def** gen\_landmark\_data**(**srcTxt**,** net**,** augment**=False):**  '''  srcTxt: each line is: 0=path, 1-4=bbox, 5-14=landmark 5points  net: PNet or RNet or ONet  augment: if enable data augmentation  **## srcTxt是个label文件.**  **格式: path, 4个gtbox数值, 10个landmark坐标(5对,分x,y方向).**  '''  **print(**">>>>>> Start landmark data create...Stage: %s"**%(**net**))**  srcTxt **=** os**.**path**.**join**(**rootPath**,** srcTxt**)**  saveFolder **=** os**.**path**.**join**(**rootPath**,** "tmp/data/%s/"**%(**net**))**  saveImagesFolder **=** os**.**path**.**join**(**saveFolder**,** "landmark"**)**  sizeOfNet **=** **{**"pnet"**:** 12**,** "rnet"**:** 24**,** "onet"**:** 48**}**  **if** net **not** **in** sizeOfNet**:**  **raise** Exception**(**"The net type error!"**)**  **if** **not** os**.**path**.**isdir**(**saveImagesFolder**):**  os**.**makedirs**(**saveImagesFolder**)**  saveF **=** open**(**join**(**saveFolder**,** "landmark.txt"**),** 'w'**)**  imageCnt **=** 0  # image\_path bbox landmark(5\*2)  **for** **(**imgPath**,** bbox**,** landmarkGt**)** **in** getBboxLandmarkFromTxt**(**srcTxt**): ## 取出label配置.**  **F\_imgs = [] ## 是个list, 存imgs**  **F\_landmarks = []**  **## 是list, 存landmarks**  img **=** cv2**.**imread**(**imgPath**)**  **assert(**img **is** **not** **None)**  img\_h**,** img\_w**,** img\_c **=** img**.**shape  gt\_box **=** np**.**array**([**bbox**.**left**,** bbox**.**top**,** bbox**.**right**,** bbox**.**bottom**])**  f\_face **=** img**[**bbox**.**top**:** bbox**.**bottom**+**1**,** bbox**.**left**:** bbox**.**right**+**1**]**  f\_face **=** cv2**.**resize**(**f\_face**,** **(**sizeOfNet**[**net**],** sizeOfNet**[**net**])) ## 把每张图缩小,pnet输入要求12x12的.**  landmark **=** np**.**zeros**((**5**,** 2**)) ## landmark数组.**  #normalize  **for** index**,** one **in** enumerate**(**landmarkGt**):**  **## 因为上面图像缩小了.**  **Landmark也需要同比例缩小到12x12的图上.**  **Landmark在gtbox中比例位置是不变的.**  rv **=** **((**one**[**0**]-**gt\_box**[**0**])/(**gt\_box**[**2**]-**gt\_box**[**0**]),** **(**one**[**1**]-**gt\_box**[**1**])/(**gt\_box**[**3**]-**gt\_box**[**1**]))**  landmark**[**index**]** **=** rv  **F\_imgs.append(f\_face) ##保存face图.**  **F\_landmarks.append(landmark.reshape(10)) ## 保存landmark**  landmark **=** np**.**zeros**((**5**,** 2**))**  **if** augment**:**  x1**,** y1**,** x2**,** y2 **=** gt\_box  #gt's width  gt\_w **=** x2 **-** x1 **+** 1  #gt's height  gt\_h **=** y2 **-** y1 **+** 1  **if** max**(**gt\_w**,** gt\_h**)** **<** 40 **or** x1 **<** 0 **or** y1 **<** 0**:**  **continue**  **## 随机平移:**     1. **在crop层面,缩小0.8到扩大1.25之间.缩放** 2. **平移,以x轴为例,左右偏移1/5w.y轴上是上下偏移1/5y.**   #random shift  **for** i **in** range**(**10**):**  bbox\_size **=** np**.**random**.**randint**(**int**(**min**(**gt\_w**,** gt\_h**)** **\*** 0.8**),** np**.**ceil**(**1.25 **\*** max**(**gt\_w**,** gt\_h**)))**  delta\_x **=** np**.**random**.**randint**(-**gt\_w **\*** 0.2**,** gt\_w **\*** 0.2**)**  delta\_y **=** np**.**random**.**randint**(-**gt\_h **\*** 0.2**,** gt\_h **\*** 0.2**)**  nx1 **=** max**(**x1**+**gt\_w**/**2**-**bbox\_size**/**2**+**delta\_x**,**0**)**  ny1 **=** max**(**y1**+**gt\_h**/**2**-**bbox\_size**/**2**+**delta\_y**,**0**)**    nx2 **=** nx1 **+** bbox\_size  ny2 **=** ny1 **+** bbox\_size  **if** nx2 **>** img\_w **or** ny2 **>** img\_h**:**  **continue**  crop\_box **=** np**.**array**([**nx1**,**ny1**,**nx2**,**ny2**])**  cropped\_im **=** img**[**ny1**:**ny2**+**1**,**nx1**:**nx2**+**1**,:]**  resized\_im **=** cv2**.**resize**(**cropped\_im**,** **(**sizeOfNet**[**net**],** sizeOfNet**[**net**]))**  #cal iou  iou **=** IoU**(**crop\_box**,** np**.**expand\_dims**(**gt\_box**,**0**))**  **if** iou **<=** 0.65**:**  **continue**  F\_imgs**.**append**(**resized\_im**)**  #normalize  **for** index**,** one **in** enumerate**(**landmarkGt**):**  rv **=** **((**one**[**0**]-**nx1**)/**bbox\_size**,** **(**one**[**1**]-**ny1**)/**bbox\_size**)**  landmark**[**index**]** **=** rv  F\_landmarks**.**append**(**landmark**.**reshape**(**10**))**  landmark **=** np**.**zeros**((**5**,** 2**))**  landmark\_ **=** F\_landmarks**[-**1**].**reshape**(-**1**,**2**)**  bbox **=** BBox**([**nx1**,**ny1**,**nx2**,**ny2**])**  **## 在”shift”基础上做mirror.**   1. **有概率(50%)的去做mirror.** 2. **利用flip()实现镜像.位于tools\landmark\_utils.py.实现的是x轴方向的镜像.**   **相应的要把landmark也给镜像了(左眼和右眼互换,左右嘴角互换.)**  #mirror  **if** random**.**choice**([**0**,**1**])** **>** 0**:**  face\_flipped**,** landmark\_flipped **=** flip**(**resized\_im**,** landmark\_**)**  face\_flipped **=** cv2**.**resize**(**face\_flipped**,** **(**sizeOfNet**[**net**],** sizeOfNet**[**net**]))**  #c\*h\*w  F\_imgs**.**append**(**face\_flipped**)**  F\_landmarks**.**append**(**landmark\_flipped**.**reshape**(**10**))**  #rotate  **if** random**.**choice**([**0**,**1**])** **>** 0**:**  **## 也是概率的去做旋转.**   1. **先逆时针旋转5度.** 2. **然后偏移,flip** 3. **再转回来(顺时针5度).** 4. **旋转使用cv2. getRotationMatrix2D函数.** 5. **它需要待旋转的centerX,CenterY.** 6. **旋转角度** 7. **缩放比例** 8. **输出一个rot\_mat矩阵.矩阵如下:**   **对应位置相乘,然后加上最后一列.**    face\_rotated\_by\_alpha**,** landmark\_rotated **=** rotate**(**img**,** bbox**,** \  bbox**.**reprojectLandmark**(**landmark\_**),** 5**)**#逆时针旋转  #landmark\_offset  landmark\_rotated **=** bbox**.**projectLandmark**(**landmark\_rotated**)**  face\_rotated\_by\_alpha **=** cv2**.**resize**(**face\_rotated\_by\_alpha**,** **(**sizeOfNet**[**net**],** sizeOfNet**[**net**]))**  F\_imgs**.**append**(**face\_rotated\_by\_alpha**)**  F\_landmarks**.**append**(**landmark\_rotated**.**reshape**(**10**))**    #flip  face\_flipped**,** landmark\_flipped **=** flip**(**face\_rotated\_by\_alpha**,** landmark\_rotated**)**  face\_flipped **=** cv2**.**resize**(**face\_flipped**,** **(**sizeOfNet**[**net**],** sizeOfNet**[**net**]))**  F\_imgs**.**append**(**face\_flipped**)**  F\_landmarks**.**append**(**landmark\_flipped**.**reshape**(**10**))**    #inverse clockwise rotation  **if** random**.**choice**([**0**,**1**])** **>** 0**:**  face\_rotated\_by\_alpha**,** landmark\_rotated **=** rotate**(**img**,** bbox**,** \  bbox**.**reprojectLandmark**(**landmark\_**),** **-**5**)**#顺时针旋转  landmark\_rotated **=** bbox**.**projectLandmark**(**landmark\_rotated**)**  face\_rotated\_by\_alpha **=** cv2**.**resize**(**face\_rotated\_by\_alpha**,** **(**sizeOfNet**[**net**],** sizeOfNet**[**net**]))**  F\_imgs**.**append**(**face\_rotated\_by\_alpha**)**  F\_landmarks**.**append**(**landmark\_rotated**.**reshape**(**10**))**    face\_flipped**,** landmark\_flipped **=** flip**(**face\_rotated\_by\_alpha**,** landmark\_rotated**)**  face\_flipped **=** cv2**.**resize**(**face\_flipped**,** **(**sizeOfNet**[**net**],** sizeOfNet**[**net**]))**  F\_imgs**.**append**(**face\_flipped**)**  F\_landmarks**.**append**(**landmark\_flipped**.**reshape**(**10**))**  F\_imgs**,** F\_landmarks **=** np**.**asarray**(**F\_imgs**),** np**.**asarray**(**F\_landmarks**)**  **for** i **in** range**(**len**(**F\_imgs**)): ## 保存**  path **=** os**.**path**.**join**(**saveImagesFolder**,** "%d.jpg"**%(**imageCnt**))**  cv2**.**imwrite**(**path**,** F\_imgs**[**i**])**  landmarks **=** map**(**str**,** list**(**F\_landmarks**[**i**]))**  saveF**.**write**(**path **+** " -2 " **+** " "**.**join**(**landmarks**)+**"\n"**)**  imageCnt **+=** 1  printStr **=** "\rCount: {}"**.**format**(**imageCnt**)**  sys**.**stdout**.**write**(**printStr**)**  sys**.**stdout**.**flush**()**  saveF**.**close**()**  **print** "\nLandmark create done!" |

### 2.1.3 生成Pnet用的tfrecord

首先,要拿到datasets

|  |
| --- |
| **def** \_\_get\_dataset**(**net**,** iterType**):**  dataset **=** **[]**  **for** line **in** **\_\_iter\_all\_data(**net**,** iterType**):** |

|  |
| --- |
| **def** \_\_iter\_all\_data**(**net**,** iterType**):**  saveFolder **=** os**.**path**.**join**(**rootPath**,** "tmp/data/%s/"**%(**net**))**  **if** net **not** **in** **[**'pnet'**,** 'rnet'**,** 'onet'**]:**  **raise** Exception**(**"The net type error!"**)**  **if** **not** os**.**path**.**isfile**(**os**.**path**.**join**(**saveFolder**,** 'pos.txt'**)):**  **raise** Exception**(**"Please gen pos.txt in first!"**)**  **if** **not** os**.**path**.**isfile**(**os**.**path**.**join**(**saveFolder**,** 'landmark.txt'**)):**  **raise** Exception**(**"Please gen landmark.txt in first!"**)**  **if** iterType **==** 'all'**:**  **with** open**(**os**.**path**.**join**(**saveFolder**,** 'pos.txt'**),** 'r'**)** **as** f**:**  pos **=** f**.**readlines**()**  **with** open**(**os**.**path**.**join**(**saveFolder**,** 'neg.txt'**),** 'r'**)** **as** f**:**  neg **=** f**.**readlines**()**  **with** open**(**os**.**path**.**join**(**saveFolder**,** 'part.txt'**),** 'r'**)** **as** f**:**  part **=** f**.**readlines**()**  # keep sample ratio [neg, pos, part] = [3, 1, 1]  base\_num **=** min**([**len**(**neg**),** len**(**pos**),** len**(**part**)])**  **if** len**(**neg**)** **>** base\_num **\*** 3**:**  neg\_keep **=** np**.**random**.**choice**(**len**(**neg**),** size**=**base\_num **\*** 3**,** replace**=False)**  **else:**  neg\_keep **=** np**.**random**.**choice**(**len**(**neg**),** size**=**len**(**neg**),** replace**=False)**  pos\_keep **=** np**.**random**.**choice**(**len**(**pos**),** size**=**base\_num**,** replace**=False)**  part\_keep **=** np**.**random**.**choice**(**len**(**part**),** size**=**base\_num**,** replace**=False)**  **for** i **in** pos\_keep**:**  **yield** pos**[**i**] ## 按照Pos,Neg,Part顺序记录到tfrecord中.**  **for** i **in** neg\_keep**:**  **yield** neg**[**i**]**  **for** i **in** part\_keep**:**  **yield** part**[**i**]**  **for** item **in** open**(**os**.**path**.**join**(**saveFolder**,** 'landmark.txt'**),** 'r'**):**  **yield** item  **elif** iterType **in** **[**'pos'**,** 'neg'**,** 'part'**,** 'landmark'**]:**  **for** line **in** open**(**os**.**path**.**join**(**saveFolder**,** '%s.txt'**%(**iterType**))):**  **yield** line  **else:**  **raise** Exception**(**"Unsupport iter type."**)** |

何为yield操作:

如下实例:

|  |
| --- |
| >>> **def** g(n): ... **for** i **in** range(n): ... **yield** i \*\*2 ... >>> **for** i **in** g(5): **## g(5)在编译时候是常量了. 需要展开, g(5),在yield参与下,返回出0,1,4,9,16的值.** ... **print** i,":", ... 0 : 1 : 4 : 9 : 16 : |

## 2.1A understanding Positive data



##### 回归框的目标是什么, 以及回归框和gtboxes关系.

区别于Region的物体检测(frcnn,frcn等), mtcnn是没有anchors(或region)的概念的. 它是怎么做到回归框呢?

首先, 它的回归目标是Postive样本, 这些Postive样本和gtboxes是有区别的. Mtcnn没有直接选择gt boxes做为回归目标. 应该是考虑到正样本太少了(毕竟一幅图中人脸的个数,面接都太小了).

然后, Mtcnn选择依据gtboxes,生成一组更多的pos样本.

1. 如上图, 对每一个gtbox都随机生成和他相邻的一组的矩形框(iou>0.7). 作为pos的样本(也就是网络回归目的, 计算它和gtbox的offset loss).
2. 样本是围绕gtboxes的.矩形框最开始是任意尺寸的. 等到满足iou>0.7的框找到后, 再把这些任意尺寸的pos框resize到12\*12尺寸.
3. 生成的pos样本中还会记录下pos样本和其对应的gtboxes的offset值.这些值也作为回归目标(loss目标).
4. 网络训练的目的也并非让offset值越小越好, 而是达到一个效果是,让预测框(区别于最终候选框)更加接近真实人脸框的iou值(>0.7), 越能让iou>0.7越好.
5. 所以当网络train好后做预测时, 其推断输出不是说一下子就是gt boxes的位置(矩形框). 而是输出一组围绕着”真值(真实人脸)”的一组框以及该框对应的offset值.
6. 通过把输出结果的矩形框尺寸(x1,y1,x2,y2)和回归出来的offset组合(对应位置相加,例 x1+offset\_x1)得到一个”校验后”(calibration)的候选框.这种候选框可以输出给下一层网络.也可以作为最终推断结果.

## 2.2 训练Pnet

training/train.pyt

Pnet的训练参数以及Loss组成占比.

|  |
| --- |
| **if** net **==** 'pnet'**:**  image\_size **=** 12  ratio\_cls\_loss**,** ratio\_bbox\_loss**,** ratio\_landmark\_loss **=** 1.0**,** 0.5**,** 0.5 |

Mtcnn训练的配置

|  |
| --- |
| config**.**BATCH\_SIZE **=** 384 ## 每次处理384张图  config**.**CLS\_OHEM **=** **True**  config**.**CLS\_OHEM\_RATIO **=** 0.7 ## Iou阈值(nms)  config**.**BBOX\_OHEM **=** **False**  config**.**BBOX\_OHEM\_RATIO **=** 0.7  config**.**EPS **=** 1e-14  config**.**LR\_EPOCH **=** **[**6**,** 14**,** 20**]** |

|  |
| --- |
| #define placeholder  **##　输入的图和label.**  input\_image **=** tf**.**placeholder**(**tf**.**float32**,** shape**=[**config**.**BATCH\_SIZE**,** image\_size**,** image\_size**,** 3**],** name**=**'input\_image'**)**  label **=** tf**.**placeholder**(**tf**.**float32**,** shape**=[**config**.**BATCH\_SIZE**],** name**=**'label'**)**  bbox\_target **=** tf**.**placeholder**(**tf**.**float32**,** shape**=[**config**.**BATCH\_SIZE**,** 4**],** name**=**'bbox\_target'**)**  landmark\_target **=** tf**.**placeholder**(**tf**.**float32**,**shape**=[**config**.**BATCH\_SIZE**,**10**],**name**=**'landmark\_target'**)**  #class,regression  **## 这个netFactory指的是PNET,RNET,ONET的model.**  cls\_loss\_op**,**bbox\_loss\_op**,**landmark\_loss\_op**,**L2\_loss\_op**,**accuracy\_op **=** [**netFactory**](#_何为netFactory之P_Net.)**(**input\_image**,** label**,** bbox\_target**,**landmark\_target**,**training**=True)**  #train,update learning rate(3 loss)  ## 整体的Loss计算:    train\_op**,** lr\_op **=** [**train\_model**](#_Train_mode对lr的修改)**(**baseLr**,** ratio\_cls\_loss**\***cls\_loss\_op **+** ratio\_bbox\_loss**\***bbox\_loss\_op **+** ratio\_landmark\_loss**\***landmark\_loss\_op **+** L2\_loss\_op**,** total\_num**)**  # init  init **=** tf**.**global\_variables\_initializer**()**  gpu\_options **=** tf**.**GPUOptions**(**allow\_growth**=True)**  sess **=** tf**.**Session**(**config**=**tf**.**ConfigProto**(**gpu\_options**=**gpu\_options**))**  #save model  saver **=** tf**.**train**.**Saver**(**max\_to\_keep**=**0**)**  sess**.**run**(**init**)**  #visualize some variables  tf**.**summary**.**scalar**(**"cls\_loss"**,**cls\_loss\_op**)**#cls\_loss  tf**.**summary**.**scalar**(**"bbox\_loss"**,**bbox\_loss\_op**)**#bbox\_loss  tf**.**summary**.**scalar**(**"landmark\_loss"**,**landmark\_loss\_op**)**#landmark\_loss  tf**.**summary**.**scalar**(**"cls\_accuracy"**,**accuracy\_op**)**#cls\_acc  summary\_op **=** tf**.**summary**.**merge\_all**()**  logs\_dir **=** os**.**path**.**join**(**rootPath**,** "tmp"**,** "logs"**,** net**)**  **if** os**.**path**.**exists**(**logs\_dir**)** **==** **False:**  os**.**makedirs**(**logs\_dir**)**  writer **=** tf**.**summary**.**FileWriter**(**logs\_dir**,** sess**.**graph**)**  #begin  coord **=** tf**.**train**.**Coordinator**()**  #begin enqueue thread  threads **=** tf**.**train**.**start\_queue\_runners**(**sess**=**sess**,** coord**=**coord**)**  i **=** 0  #total steps  MAX\_STEP **=** int**(**total\_num **/** config**.**BATCH\_SIZE **+** 1**)** **\*** endEpoch  **print** "\n\nTotal step: "**,** MAX\_STEP  epoch **=** 0  sess**.**graph**.**finalize**()**  **try:**  **for** step **in** range**(**MAX\_STEP**):**  i **=** i **+** 1  **if** coord**.**should\_stop**():**  **break**  image\_batch\_array**,** label\_batch\_array**,** bbox\_batch\_array**,**landmark\_batch\_array **=** sess**.**run**([**image\_batch**,** label\_batch**,** bbox\_batch**,**landmark\_batch**])**  #random flip  image\_batch\_array**,**landmark\_batch\_array **=** random\_flip\_images**(**image\_batch\_array**,**label\_batch\_array**,**landmark\_batch\_array**)**  '''  print image\_batch\_array.shape  print label\_batch\_array.shape  print bbox\_batch\_array.shape  print landmark\_batch\_array.shape  print label\_batch\_array[0]  print bbox\_batch\_array[0]  print landmark\_batch\_array[0]  '''  \_**,**\_**,**summary **=** sess**.**run**([train\_op, lr\_op** **,**summary\_op**],** feed\_dict**={**input\_image**:** image\_batch\_array**,** label**:** label\_batch\_array**,** bbox\_target**:** bbox\_batch\_array**,**landmark\_target**:**landmark\_batch\_array**})**    **if** **(**step**+**1**)** **%** display **==** 0**:**  #acc = accuracy(cls\_pred, labels\_batch)  cls\_loss**,** bbox\_loss**,**landmark\_loss**,**L2\_loss**,**lr**,**acc **=** sess**.**run**([**cls\_loss\_op**,** bbox\_loss\_op**,**landmark\_loss\_op**,**L2\_loss\_op**,**lr\_op**,**accuracy\_op**],**  feed\_dict**={**input\_image**:** image\_batch\_array**,** label**:** label\_batch\_array**,** bbox\_target**:** bbox\_batch\_array**,** landmark\_target**:** landmark\_batch\_array**})**  **print(**"%s [%s] Step: %d, accuracy: %3f, cls loss: %4f, bbox loss: %4f, landmark loss: %4f,L2 loss: %4f,lr:%f " **%** **(**  datetime**.**now**(),** net**,** step**+**1**,** acc**,** cls\_loss**,** bbox\_loss**,** landmark\_loss**,** L2\_loss**,** lr**))**  #save every two epochs  **if** i **\*** config**.**BATCH\_SIZE **>** total\_num**\***2**:**  epoch **=** epoch **+** 1  i **=** 0  saver**.**save**(**sess**,** modelPrefix**,** global\_step**=**epoch**\***2**)**  writer**.**add\_summary**(**summary**,**global\_step**=**step**)**  **except** tf**.**errors**.**OutOfRangeError**:**  **print(**"Done!"**)**  **finally:**  coord**.**request\_stop**()**  writer**.**close**()**  coord**.**join**(**threads**)**  sess**.**close**()** |

### 2.2.1 何为netFactory之P\_Net.

在training/mtcnn\_model.py中

|  |
| --- |
| #construct Pnet  #label:batch  **def** P\_Net**(**inputs**,** label**=None,** bbox\_target**=None,** landmark\_target**=None,** training**=True):**  #define common param  **with** slim**.**arg\_scope**([**slim**.**conv2d**],**  activation\_fn**=**prelu**,## prelu激活函数**  weights\_initializer**=**slim**.**xavier\_initializer**(),**  biases\_initializer**=**tf**.**zeros\_initializer**(),**  weights\_regularizer**=**slim**.**l2\_regularizer**(**0.0005**),**  padding**=**'valid'**):**  net **=** slim**.**conv2d**(**inputs**,** 10**,** 3**,** stride**=**1**,**scope**=**'conv1'**)**  net **=** slim**.**max\_pool2d**(**net**,** kernel\_size**=[**2**,**2**],** stride**=**2**,** scope**=**'pool1'**)**  net **=** slim**.**conv2d**(**net**,** num\_outputs**=**16**,** kernel\_size**=[**3**,**3**],** stride**=**1**,** scope**=**'conv2'**)**  net **=** slim**.**conv2d**(**net**,** num\_outputs**=**32**,** kernel\_size**=[**3**,**3**],** stride**=**1**,** scope**=**'conv3'**)**  **# 三个任务输出.**   1. **2分类, 是否人脸** 2. **Boundingbox回归, 4个尺寸.** 3. **Landmark回归, 5对10个尺寸.**     #batch\*H\*W\*2  conv4\_1 **=** slim**.**conv2d**(**net**,** num\_outputs**=**2**,** kernel\_size**=[**1**,**1**],** stride**=**1**,** scope**=**'conv4\_1'**,** activation\_fn**=**tf**.**nn**.**softmax**)**  #batch\*H\*W\*4  bbox\_pred **=** slim**.**conv2d**(**net**,** num\_outputs**=**4**,** kernel\_size**=[**1**,**1**],** stride**=**1**,** scope**=**'conv4\_2'**,** activation\_fn**=None)**  #batch\*H\*W\*10  landmark\_pred **=** slim**.**conv2d**(**net**,** num\_outputs**=**10**,** kernel\_size**=[**1**,**1**],** stride**=**1**,** scope**=**'conv4\_3'**,** activation\_fn**=None)**  **if** training**:**  #batch\*2  cls\_prob **=** tf**.**squeeze**(**conv4\_1**,** **[**1**,**2**],** name**=**'cls\_prob'**)**  cls\_loss **=** [**cls\_ohem**](#_何为cls_ohem.)**(**cls\_prob**,** label**)**  #batch  bbox\_pred **=** tf**.**squeeze**(**bbox\_pred**,** **[**1**,** 2**],** name**=**'bbox\_pred'**) ## 已经把[b 1 1 4]中间两个去掉了.**  bbox\_loss **=** [bbox\_ohem](#_何为bbox_ohem)**(**bbox\_pred**,** bbox\_target**,** label**)**  #batch\*10  landmark\_pred **=** tf**.**squeeze**(**landmark\_pred**,** **[**1**,** 2**],** name**=**"landmark\_pred"**)**  landmark\_loss **=** [landmark\_ohem](#_何为landmark_ohem)**(**landmark\_pred**,** landmark\_target**,** label**)**  accuracy **=** cal\_accuracy**(**cls\_prob**,** label**)##　计算精度.**  L2\_loss **=** tf**.**add\_n**(**slim**.**losses**.**get\_regularization\_losses**()) ## 添加L2正则化**  **return** cls\_loss**,** bbox\_loss**,** landmark\_loss**,** L2\_loss**,** accuracy  **else:** # testing  #when test, batch\_size = 1  cls\_pro\_test **=** tf**.**squeeze**(**conv4\_1**,** axis**=**0**)**  bbox\_pred\_test **=** tf**.**squeeze**(**bbox\_pred**,** axis**=**0**)**  landmark\_pred\_test **=** tf**.**squeeze**(**landmark\_pred**,** axis**=**0**)**  **return** cls\_pro\_test**,** bbox\_pred\_test**,** landmark\_pred\_test |

##### 何为cls\_ohem.

为了产生cls的Loss.

1. 输入有Pnet的预测, 以及label的ground truth.

|  |
| --- |
| **def** cls\_ohem**(**cls\_prob**,** label**):**  zeros **=** tf**.**zeros\_like**(**label**)**  #label=-1 --> label=0net\_factory  **## label值负数重置为0值.**  **考虑到会有Neg的样本,**    **如上图tf.where(A). A的tensor分布如上图(左图,输入).**   1. **输入是T,F的tensor.** 2. **输入元素中一个有5个true. 所以tf.where输出是5行(每行代表这个T的位置)** 3. **输入的阶(区别于矩阵的阶,这里是输入tensor的维度)是3. 所以tf.where的输出有3列.** 4. **[0 0 0]表示第dim0的#0成员, dim1的#0成员,dim2的#1成员是第一个True.**   **另一种tf.where用法:**  **tf.where(input, a,b)**   1. **input是参考, 值为True和False. Input,a,b的格式都一样的.** 2. **tf.where的输出以a为基本, 然后input中true的位置在a的同位置上保留a的值. 然后input的false位置在a上的同位置用b上的同位置的值替换之.**   **本处tf.where是吧label中小于0的部分用zero替换.**  label\_filter\_invalid **=** tf**.**where**(**tf**.**less**(**label**,**0**),** zeros**,** label**)**  num\_cls\_prob **=** **tf.size(**cls\_prob**)**  **## tf.size会把cls\_prob这个tensor整理成0-d的格式.**  **然后再计算有多少个元素.**  cls\_prob\_reshape **=** tf**.**reshape**(**cls\_prob**,[**num\_cls\_prob**,-**1**])**  label\_int **=** tf**.**cast**(**label\_filter\_invalid**,**tf**.**int32**)**  num\_row **=** tf**.**to\_int32**(**cls\_prob**.**get\_shape**()[**0**])**  row **=** tf**.**range**(**num\_row**)\***2  **## 这部分是找到label描述的对应的概率.** [相关数据见下面实验](#_Cls_ohem实验)   1. **比如label描述当前图片应该是neg的. 计算loss的概率就取neg的概率(因为一个prob结果会出2个概率)** 2. **如果当前描述的是pos的,就取出pos的概率做loss.**   **原始cls\_prob是[batch 2]的.**    **为了方便取出对应的pos的概率,将其reshape成一列.**    **具体如下的计算就是找到计算Loss需要的概率:**   1. **一个label对应prob\_reshape预测的2行.**   **row = tf.range(num\_row)\*2**   1. **下面的label\_int描述了pos的偏移.** 2. **下面row和label\_int之和就是需要进入Loss计算的概率了.**   **蓝线表示需要那个概率.**    **indices\_ = row + label\_int**  label\_prob **=** tf**.**squeeze**(**tf**.**gather**(**cls\_prob\_reshape**,** indices\_**))**  loss **=** **-**tf**.**log**(**label\_prob**+**1e-10**)## epsilon避免梯度消失(爆炸).**  zeros **=** tf**.**zeros\_like**(**label\_prob**,** dtype**=**tf**.**float32**)**  ones **=** tf**.**ones\_like**(**label\_prob**,**dtype**=**tf**.**float32**)**  valid\_inds **=** tf**.**where**(**label **<** zeros**,**zeros**,**ones**)## 将有阈值的部分整理成1, 构成一个01的tensor为valid\_inds**  num\_valid **=** tf**.**reduce\_sum**(**valid\_inds**) ## 预测中Pos个数.TP+FP**  keep\_num **=** tf**.**cast**(**num\_valid**\***num\_keep\_radio**,**dtype**=**tf**.**int32**)**  #set 0 to invalid sample  loss **=** loss **\*** valid\_inds  loss**,**\_ **=** tf**.**nn**.**top\_k**(**loss**,** k**=**keep\_num**)**  **return** tf**.**reduce\_mean**(**loss**)** |

##### Cls\_ohem实验



|  |
| --- |
| **def** cls\_ohem**(**cls\_prob**,** label**):**  zeros\_tsr **=** tf**.**zeros**([**2**,** 3**])** ##为了调用tf.Print做的dummy.  zeros **=** tf**.**zeros\_like**(**label**)**  #label=-1 --> label=0net\_factory  label\_filter\_invalid **=** tf**.**where**(**tf**.**less**(**label**,**0**),** zeros**,** label**)**  num\_cls\_prob **=** tf**.**size**(**cls\_prob**)**  cls\_prob\_reshape **=** tf**.**reshape**(**cls\_prob**,[**num\_cls\_prob**,-**1**])**  label\_int **=** tf**.**cast**(**label\_filter\_invalid**,**tf**.**int32**)**  num\_row **=** tf**.**to\_int32**(**cls\_prob**.**get\_shape**()[**0**])**  row **=** tf**.**range**(**num\_row**)\***2  indices\_ **=** row **+** label\_int  **tfprint.cls\_ohem = tf.Print(zeros\_tsr,**  **["cls\_ohem",tf.shape(row),row,tf.shape(label\_int),label\_int,tf.shape(indices\_),indices\_,tf.shape(cls\_prob),tf.shape(label),tf.shape(cls\_prob\_reshape),cls\_prob\_reshape],summarize=8)**  label\_prob **=** tf**.**squeeze**(**tf**.**gather**(**cls\_prob\_reshape**,** indices\_**))**  loss **=** **-**tf**.**log**(**label\_prob**+**1e-10**)**  zeros **=** tf**.**zeros\_like**(**label\_prob**,** dtype**=**tf**.**float32**)**  ones **=** tf**.**ones\_like**(**label\_prob**,**dtype**=**tf**.**float32**)**  valid\_inds **=** tf**.**where**(**label **<** zeros**,**zeros**,**ones**)**  num\_valid **=** tf**.**reduce\_sum**(**valid\_inds**)**  keep\_num **=** tf**.**cast**(**num\_valid**\***num\_keep\_radio**,**dtype**=**tf**.**int32**)**  #set 0 to invalid sample  loss **=** loss **\*** valid\_inds  loss**,**\_ **=** tf**.**nn**.**top\_k**(**loss**,** k**=**keep\_num**)**  **return** tf**.**reduce\_mean**(**loss**)** |

实验结果:

|  |
| --- |
| [cls\_ohem][384][0 2 4 6 8 10 12 14...][384][0 1 0 0 0 1 1 1...][384][0 3 4 6 8 11 13 15...][384 2][384][768 1][[0.492263049][0.507736921][0.513266802][0.486733228][0.528568387][0.471431583][0.526716888][0.473283082]...]  [cls\_ohem][384][0 2 4 6 8 10 12 14...][384][0 1 0 0 0 1 0 1...][384][0 3 4 6 8 11 12 15...][384 2][384][768 1][[0.526763141][0.473236829][0.520561635][0.479438335][0.472027242][0.527972698][0.460573941][0.539426]...]  [cls\_ohem][384][0 2 4 6 8 10 12 14...][384][0 0 0 0 0 0 1 0...][384][0 2 4 6 8 10 13 14...][384 2][384][768 1][[0.540893495][0.459106475][0.520356238][0.479643732][0.462826669][0.537173331][0.463837773][0.536162198]...]  [cls\_ohem][384][0 2 4 6 8 10 12 14...][384][0 0 0 0 0 0 1 0...][384][0 2 4 6 8 10 13 14...][384 2][384][768 1][[0.511262059][0.488737941][0.501042485][0.498957515][0.493958622][0.506041408][0.577940941][0.422059059]...]  [cls\_ohem][384][0 2 4 6 8 10 12 14...][384][0 0 0 0 1 0 0 0...][384][0 2 4 6 9 10 12 14...][384 2][384][768 1][[0.537541568][0.462458402][0.527794123][0.472205877][0.624005318][0.375994742][0.506848097][0.493151873]...]  依次应为:  **tf.Print(zeros\_tsr,["cls\_ohem",tf.shape(row),row,tf.shape(label\_int),label\_int,tf.shape(indices\_),indices\_],summarize=8)**  **row shape: 384 # batch就是384**  **row: [0 2 4 6 8 ..2b]**  **label\_int shape: 384**  **label\_int 值: [0 1 0 0 1..] # 将小于0的part,修正为0.**  **Indices\_ shape: 384**  **Indices\_ 值: [0 3 4 6 …] # 真是row和label\_int的对应位置和.**  **cls\_prob\_reshape 的shape是: 768 aka 2\*batch**  **cls\_prob shape是 384** |

##### 何为bbox\_ohem



|  |
| --- |
| #label=1 or label=-1 then do regression  **def** bbox\_ohem**(**bbox\_pred**,**bbox\_target**,**label**):**  zeros\_index **=** tf**.**zeros\_like**(**label**,** dtype**=**tf**.**float32**)**  ones\_index **=** tf**.**ones\_like**(**label**,**dtype**=**tf**.**float32**)**  valid\_inds **=** tf**.**where**(**tf**.**equal**(**tf**.**abs**(**label**),** 1**),**ones\_index**,**zeros\_index**)**  **## 把不相交的预测框的位置筛掉,只留下pos和part的框的位置.**  #(batch,)  **##bbox\_pred是[b 4]格式的. 在每个batch上都和”4”个端点计算欧式距离.**  **然后,在dim1(也就是”4”这个维度)上求和.(reduce\_sum)**  **Square\_error就剩下batch个error值.**  square\_error **=** tf**.**square**(**bbox\_pred**-**bbox\_target**)**  square\_error **=** tf**.**reduce\_sum**(**square\_error**,**axis**=**1**)**  #keep\_num scalar  num\_valid **=** tf**.**reduce\_sum**(**valid\_inds**)**  keep\_num **=** tf**.**cast**(**num\_valid**\***num\_keep\_radio**,**dtype**=**tf**.**int32**)**  **## 取topk的做Loss,其他的不做loss.**   1. **这些做loss的就是part和pos的. 只不过这些正样本的回归有偏差.** 2. **取出其中最k大的偏差做Loss.**   #keep valid index square\_error  square\_error **=** square\_error**\***valid\_inds  \_**,** k\_index **=** tf**.**nn**.**top\_k**(**square\_error**,** k**=**keep\_num**)**  square\_error **=** tf**.**gather**(**square\_error**,** k\_index**)**  **return** tf**.**reduce\_mean**(**square\_error**)** |

##### 何为landmark\_ohem



类似bbox\_ohem,同为回归问题.同样采用欧式距离.

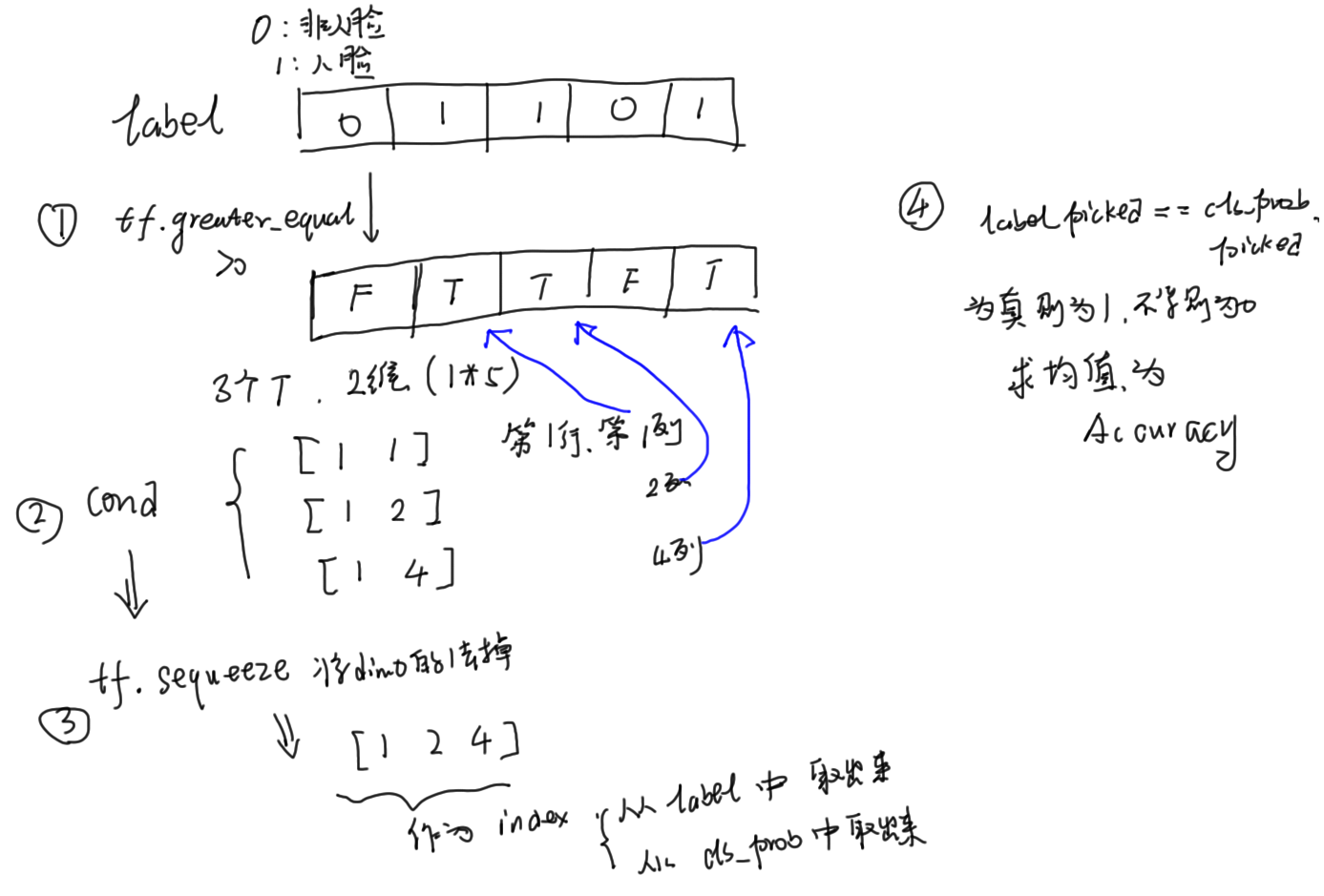
|  |
| --- |
| **def** landmark\_ohem**(**landmark\_pred**,**landmark\_target**,**label**):**  #keep label =-2 then do landmark detection  ones **=** tf**.**ones\_like**(**label**,**dtype**=**tf**.**float32**)**  zeros **=** tf**.**zeros\_like**(**label**,**dtype**=**tf**.**float32**)**  valid\_inds **=** tf**.**where**(**tf**.**equal**(**label**,-**2**),**ones**,**zeros**)**  square\_error **=** tf**.**square**(**landmark\_pred**-**landmark\_target**)**  square\_error **=** tf**.**reduce\_sum**(**square\_error**,**axis**=**1**)**  num\_valid **=** tf**.**reduce\_sum**(**valid\_inds**)**  keep\_num **=** tf**.**cast**(**num\_valid**\***num\_keep\_radio**,**dtype**=**tf**.**int32**)**  square\_error **=** square\_error**\***valid\_inds  \_**,** k\_index **=** tf**.**nn**.**top\_k**(**square\_error**,** k**=**keep\_num**)**  square\_error **=** tf**.**gather**(**square\_error**,** k\_index**)**  **return** tf**.**reduce\_mean**(**square\_error**)** |

### 2.2.2 Train\_mode对lr的修改

|  |
| --- |
| **def** train\_model**(**baseLr**,** loss**,** data\_num**):**  """  train model  :param baseLr: base learning rate  :param loss: loss  :param data\_num:  :return:  train\_op, lr\_op  """  lr\_factor **=** 0.1  global\_step **=** tf**.**Variable**(**0**,** trainable**=False)**  #LR\_EPOCH [8,14]  #boundaried [num\_batch,num\_batch]  boundaries **=** **[**int**(**epoch **\*** data\_num **/** config**.**BATCH\_SIZE**)** **for** epoch **in** config**.**LR\_EPOCH**]**  #lr\_values[0.01,0.001,0.0001,0.00001]  lr\_values **=** **[**baseLr **\*** **(**lr\_factor **\*\*** x**)** **for** x **in** range**(**0**,** len**(**config**.**LR\_EPOCH**)** **+** 1**)]**  **## 生成分段global steps对应的learning rate.**  **达到下面例子的效果,不同的steps区间就使用不同learning rate.**   1. **Lr变化和steps是有关的,分段函数.** 2. **随着steps增大,lr是减小的.(避免更好的收敛).**     #control learning rate  lr\_op **=** **tf.train.piecewise\_constant(**global\_step**,** boundaries**,** lr\_values**)**  optimizer **=** tf**.**train**.**MomentumOptimizer**(**lr\_op**,** 0.9**)**  train\_op **=** optimizer**.**minimize**(**loss**,** global\_step**)**  **return** train\_op**,** lr\_op |

### 2.2.3 understanding accuracy

|  |
| --- |
| **def** cal\_accuracy**(**cls\_prob**,**label**):**  pred **=** tf**.**argmax**(**cls\_prob**,**axis**=**1**)**  label\_int **=** tf**.**cast**(**label**,**tf**.**int64**)**  cond **=** tf**.**where**(**tf**.**greater\_equal**(**label\_int**,**0**))**  picked **=** tf**.**squeeze**(**cond**)**  label\_picked **=** tf**.**gather**(**label\_int**,**picked**)**  pred\_picked **=** tf**.**gather**(**pred**,**picked**)**  accuracy\_op **=** tf**.**reduce\_mean**(**tf**.**cast**(**tf**.**equal**(**label\_picked**,**pred\_picked**),**tf**.**float32**))**  **return** accuracy\_op |



# 三 使用MTCNN做推断(测试)

## 3.1 数据预处理

|  |
| --- |
| **def** test**(**stage**,** testFolder**):**  **print(**"Start testing in %s"**%(**testFolder**))**  detectors **=** **[None,** **None,** **None]**  **if** stage **in** **[**'pnet'**,** 'rnet'**,** 'onet'**]:**  modelPath **=** os**.**path**.**join**(**rootPath**,** 'tmp/model/pnet/'**)**  a **=** **[**b**[**5**:-**6**]** **for** b **in** os**.**listdir**(**modelPath**)** **if** b**.**startswith**(**'pnet-'**)** **and** b**.**endswith**(**'.index'**)]**  maxEpoch **=** max**(**map**(**int**,** a**))** # auto match a max epoch model  modelPath **=** os**.**path**.**join**(**modelPath**,** "pnet-%d"**%(**maxEpoch**))**  **print(**"Use PNet model: %s"**%(**modelPath**))**  **detectors[0] = FcnDetector(P\_Net,modelPath)**  **if** stage **in** **[**'rnet'**,** 'onet'**]:**  modelPath **=** os**.**path**.**join**(**rootPath**,** 'tmp/model/rnet/'**)**  a **=** **[**b**[**5**:-**6**]** **for** b **in** os**.**listdir**(**modelPath**)** **if** b**.**startswith**(**'rnet-'**)** **and** b**.**endswith**(**'.index'**)]**  maxEpoch **=** max**(**map**(**int**,** a**))**  modelPath **=** os**.**path**.**join**(**modelPath**,** "rnet-%d"**%(**maxEpoch**))**  **print(**"Use RNet model: %s"**%(**modelPath**))**  **detectors[1] = Detector(R\_Net, 24, 1, modelPath)**  **if** stage **in** **[**'onet'**]:**  modelPath **=** os**.**path**.**join**(**rootPath**,** 'tmp/model/onet/'**)**  a **=** **[**b**[**5**:-**6**]** **for** b **in** os**.**listdir**(**modelPath**)** **if** b**.**startswith**(**'onet-'**)** **and** b**.**endswith**(**'.index'**)]**  maxEpoch **=** max**(**map**(**int**,** a**))**  modelPath **=** os**.**path**.**join**(**modelPath**,** "onet-%d"**%(**maxEpoch**))**  **print(**"Use ONet model: %s"**%(**modelPath**))**  **detectors[2] = Detector(O\_Net, 48, 1, modelPath)**  mtcnnDetector **=** **MtcnnDetector(**detectors**=**detectors**,** min\_face\_size **=** 24**,** threshold**=[**0.9**,** 0.6**,** 0.7**])**  testImages **=** **[]**  **for** name **in** os**.**listdir**(**testFolder**):**  testImages**.**append**(**os**.**path**.**join**(**testFolder**,** name**))**  testDatas **=** TestLoader**(**testImages**)**  # Now to detect  ## 推断  allBoxes**,** allLandmarks **=** **mtcnnDetector.detect\_face(**testDatas**)**  **print(**"\n"**)**  # Save it  **for** idx**,** imagePath **in** enumerate**(**testImages**):**  image **=** cv2**.**imread**(**imagePath**)**  **for** bbox **in** allBoxes**[**idx**]:**  **## 在原图上画矩形**  cv2**.**putText**(**image**,**str**(**np**.**round**(**bbox**[**4**],**2**)),(**int**(**bbox**[**0**]),**int**(**bbox**[**1**])),**cv2**.**FONT\_HERSHEY\_TRIPLEX**,**1**,**color**=(**255**,**0**,**255**))**  cv2**.**rectangle**(**image**,** **(**int**(**bbox**[**0**]),**int**(**bbox**[**1**])),(**int**(**bbox**[**2**]),**int**(**bbox**[**3**])),(**0**,**0**,**255**))**  allLandmark **=** allLandmarks**[**idx**]**  **if** allLandmark **is** **not** **None:** # pnet and rnet will be ignore landmark  **for** landmark **in** allLandmark**:**  **for** i **in** range**(**len**(**landmark**)/**2**):**  cv2**.**circle**(**image**,** **(**int**(**landmark**[**2**\***i**]),**int**(**int**(**landmark**[**2**\***i**+**1**]))),** 3**,** **(**0**,**0**,**255**))**  savePath **=** os**.**path**.**join**(**rootPath**,** 'testing'**,** 'results\_%s'**%(**stage**))**  **if** **not** os**.**path**.**isdir**(**savePath**):**  os**.**makedirs**(**savePath**)**  cv2**.**imwrite**(**os**.**path**.**join**(**savePath**,** "result\_%d.jpg" **%(**idx**)),** image**)**  **print(**"Save image to %s"**%(**savePath**))** |

#### 构造Pnet,Rnet,Onet结构.

|  |
| --- |
| **class** **Detector(**object**):**  #net\_factory:rnet or onet  #datasize:24 or 48  **def** \_\_init\_\_**(**self**,** net\_factory**,** data\_size**,** batch\_size**,** model\_path**):**  graph **=** tf**.**Graph**()**  **with** graph**.**as\_default**():**  self**.**image\_op **=** tf**.**placeholder**(**tf**.**float32**,** shape**=[**batch\_size**,** data\_size**,** data\_size**,** 3**],** name**=**'input\_image'**)**  #figure out landmark  **## net\_factory代表Pnet,Rnet,Onet结构.**  **self.cls\_prob, self.bbox\_pred, self.landmark\_pred** **=** net\_factory**(**self**.**image\_op**,** training**=False)** |

#### 检测入口

|  |
| --- |
| **def** detect\_face**(**self**,** test\_data**):**  all\_boxes **=** **[]** #save each image's bboxes **## List,存boundingBox**  landmarks **=** **[] ## 是list,存landmark**  batch\_idx **=** 0  **for** databatch **in** test\_data**: ## 每次取一张图,是numpy格式的.**  # print info  printStr **=** "\rDone images: {}"**.**format**(**batch\_idx**)**  sys**.**stdout**.**write**(**printStr**)**  sys**.**stdout**.**flush**()**  batch\_idx **+=** 1  im **=** databatch  # pnet  **if** self**.**pnet\_detector**:**  **##　利用Pnet推断.**   1. **输入numpy img数组** 2. **输出含有boundingbox的 boxes\_c.**   #ignore landmark  boxes**,** boxes\_c**,** landmark **=** self**.**detect\_pnet**(**im**)**  **if** boxes\_c **is** **None:**  all\_boxes**.**append**(**np**.**array**([]))**  landmarks**.**append**(**np**.**array**([]))**  **continue**  # rnet  **if** self**.**rnet\_detector**:**  #ignore landmark  boxes**,** boxes\_c**,** landmark **=** self**.**detect\_rnet**(**im**,** boxes\_c**)**  **if** boxes\_c **is** **None:**  all\_boxes**.**append**(**np**.**array**([]))**  landmarks**.**append**(**np**.**array**([]))**  **continue**  # onet  **if** self**.**onet\_detector**:**  boxes**,** boxes\_c**,** landmark **=** self**.**detect\_onet**(**im**,** boxes\_c**)**  **if** boxes\_c **is** **None:**  all\_boxes**.**append**(**np**.**array**([]))**  landmarks**.**append**(**np**.**array**([]))**  **continue**  all\_boxes**.**append**(**boxes\_c**)**  landmarks**.**append**(**landmark**)**  **return** all\_boxes**,**landmarks |

##### 3.1.1 Pnet推断数据预处理

|  |
| --- |
| **def** detect\_pnet**(**self**,** im**):**  """Get face candidates through pnet  Parameters:  ----------  im: numpy array  input image array  Returns:  -------  ## 返回两个   1. 校验前的raw检测框. 2. 校验后的检测框.   boxes: numpy array  detected boxes before calibration  boxes\_c: numpy array  boxes after calibration  """  h**,** w**,** c **=** im**.**shape  net\_size **=** 12  **## 预处理**   1. **min\_face\_size是24.** 2. **初始scale是12/24=0.5.** 3. **按照这个比例对图像做缩小.**   **比如640x480的图,先缩小0.5到**  **320x240, 第二次迭代时候,320仍然大于net\_size的12.**  **仍然需要缩小0.5到160x120.直到不大于net\_size.**  **这样实现一个图像金字塔.**    current\_scale **=** float**(**net\_size**)** **/** self**.**min\_face\_size # find initial scale  im\_resized **=** self**.**processed\_image**(**im**,** current\_scale**)**  current\_height**,** current\_width**,** \_ **=** im\_resized**.**shape  # for fcn  all\_boxes **=** list**()**  **while** min**(**current\_height**,** current\_width**)** **>** net\_size**:**  #return the result predicted by pnet  #cls\_cls\_map : H\*w\*2  #reg: H\*w\*4  **## 这里也能看到h,w并不是1的,所以输入的尺寸可以是任意的**  cls\_cls\_map**,** reg **=** self**.pnet\_detector.predict(**im\_resized**)**  #boxes: num\*9(x1,y1,x2,y2,score,x1\_offset,y1\_offset,x2\_offset,y2\_offset)  boxes **=** **self.generate\_bbox(**cls\_cls\_map**[:,** **:,**1**],** reg**,** current\_scale**,** self**.**thresh**[**0**])**  current\_scale **\*=** self**.**scale\_factor **##scale\_factor值为0.79**  im\_resized **=** self**.**processed\_image**(**im**,** current\_scale**)**  current\_height**,** current\_width**,** \_ **=** im\_resized**.**shape  **if** boxes**.**size **==** 0**:**  **continue**  keep **=** **py\_nms(**boxes**[:,** **:**5**],** 0.5**,** 'Union'**)**  boxes **=** boxes**[**keep**]**  **all\_boxes.append(**boxes**)**  **if** len**(**all\_boxes**)** **==** 0**:**  **return** **None,** **None,** **None**  all\_boxes **=** np**.**vstack**(**all\_boxes**)**  # merge the detection from first stage  keep **=** py\_nms**(**all\_boxes**[:,** 0**:**5**],** 0.7**,** 'Union'**)**  all\_boxes **=** all\_boxes**[**keep**]**  boxes **=** all\_boxes**[:,** **:**5**]**  bbw **=** all\_boxes**[:,** 2**]** **-** all\_boxes**[:,** 0**]** **+** 1  bbh **=** all\_boxes**[:,** 3**]** **-** all\_boxes**[:,** 1**]** **+** 1  # refine the boxes  boxes\_c **=** np**.**vstack**([**all\_boxes**[:,** 0**]** **+** all\_boxes**[:,** 5**]** **\*** bbw**,**  all\_boxes**[:,** 1**]** **+** all\_boxes**[:,** 6**]** **\*** bbh**,**  all\_boxes**[:,** 2**]** **+** all\_boxes**[:,** 7**]** **\*** bbw**,**  all\_boxes**[:,** 3**]** **+** all\_boxes**[:,** 8**]** **\*** bbh**,**  all\_boxes**[:,** 4**]])**  boxes\_c **=** boxes\_c**.**T  **return** boxes**,** boxes\_c**,** **None** |

网络推断

基本网络依赖

|  |
| --- |
| detectors**[**0**]** **=** FcnDetector**(**P\_Net**,**modelPath**)**  detectors**[**1**]** **=** Detector**(**R\_Net**,** 24**,** 1**,** modelPath**)**  detectors**[**2**]** **=** Detector**(**O\_Net**,** 48**,** 1**,** modelPath**)** |

对于pnet,使用的是FcnDetector.

|  |
| --- |
| # 下面的net\_factory是Pnet  self**.**cls\_prob**,** self**.**bbox\_pred**,** \_ **=** net\_factory**(**image\_reshape**,** training**=False)**    **## pnet不需要推断landmark.**  **def** predict**(**self**,** databatch**):**  height**,** width**,** \_ **=** databatch**.**shape  # print(height, width)  cls\_prob**,** bbox\_pred **=** **self.sess.run([self.cls\_prob,** **self.bbox\_pred],**  feed\_dict**={**self**.**image\_op**:** databatch**,** self**.**width\_op**:** width**,**  self**.**height\_op**:** height**})** |

##### 3.1.2 Pnet的推断

|  |
| --- |
| **def** detect\_pnet**(**self**,** im**):**  **## 拿到Pnet的候选框.**   1. 输入是img. 2. 输出是候选框(boxes, boxes\_c) 3. 方法是 4. 对输入图做图像金字塔. 最小face是24(min\_face\_size). 5. Pnet是fcn结构, 对输入图的hw尺寸无限制.   """Get face candidates through pnet  Parameters:  ----------  im: numpy array  input image array  Returns:  -------  boxes: numpy array  detected boxes before calibration  boxes\_c: numpy array  boxes after calibration  """  h**,** w**,** c **=** im**.**shape  net\_size **=** 12    current\_scale **=** float**(**net\_size**)** **/** self**.**min\_face\_size # find initial scale  im\_resized **=** **self.processed\_image(**im**,** current\_scale**)**  current\_height**,** current\_width**,** \_ **=** im\_resized**.**shape  # for fcn  all\_boxes **=** list**()**  **while** min**(**current\_height**,** current\_width**)** **>** net\_size**:**  #return the result predicted by pnet  #cls\_cls\_map : H\*w\*2  #reg: H\*w\*4  **## Pnet网路的推断结构**   1. **输入金字塔的一层(小图).** 2. **输出该层金子塔上可能的人脸回归框及置信度(分类).** 3. **输出是raw的格式,还不能给Rnet使用.需要整理一下.** 4. **整理后的格式为:**   *(x1,y1,x2,y2,score,x1\_offset,y1\_offset,x2\_offset,y2\_offset)*   1. *是在Pnet输入原图中boundingboxes的尺寸.* 2. *Score为概率* 3. *Offset为Pnet预测的boundingboxes与gtboxes的offset.(缩略图,金字塔层图).*   cls\_cls\_map**,** reg **=** **self.pnet\_detector.predict(**im\_resized**)**  #boxes: num\*9(x1,y1,x2,y2,score,x1\_offset,y1\_offset,x2\_offset,y2\_offset)  boxes **=** **self.generate\_bbox(**cls\_cls\_map**[:,** **:,**1**],** reg**,** current\_scale**,** self**.**thresh**[**0**])**  **## 从原图中~~找到~~(生成)下一层的金字塔**  current\_scale **\*=** self**.**scale\_factor  im\_resized **=** self**.**processed\_image**(**im**,** current\_scale**)**  current\_height**,** current\_width**,** \_ **=** im\_resized**.**shape  **if** boxes**.**size **==** 0**:**  **continue**  keep **=** **py\_nms(**boxes**[:,** **:**5**],** 0.5**,** 'Union'**)**  boxes **=** boxes**[**keep**]**  **all\_boxes.**append**(**boxes**) ## 保存Pnet所有金字塔层boundingboxes**  **if** len**(**all\_boxes**)** **==** 0**:**  **return** **None,** **None,** **None**  all\_boxes **=** np**.**vstack**(**all\_boxes**)**  # merge the detection from first stage  **## 第二次执行Nms, 对所有金字塔层输出的boundingboxes做nms.**  keep **=** py\_nms**(**all\_boxes**[:,** 0**:**5**],** 0.7**,** 'Union'**)**  all\_boxes **=** all\_boxes**[**keep**]**  boxes **=** all\_boxes**[:,** **:**5**]**  bbw **=** all\_boxes**[:,** 2**]** **-** all\_boxes**[:,** 0**]** **+** 1  bbh **=** all\_boxes**[:,** 3**]** **-** all\_boxes**[:,** 1**]** **+** 1  # refine the boxes  boxes\_c **=** np**.**vstack**([**all\_boxes**[:,** 0**]** **+** all\_boxes**[:,** 5**]** **\*** bbw**,**  all\_boxes**[:,** 1**]** **+** all\_boxes**[:,** 6**]** **\*** bbh**,**  all\_boxes**[:,** 2**]** **+** all\_boxes**[:,** 7**]** **\*** bbw**,**  all\_boxes**[:,** 3**]** **+** all\_boxes**[:,** 8**]** **\*** bbh**,**  all\_boxes**[:,** 4**]])**  boxes\_c **=** boxes\_c**.**T  **return** boxes**,** boxes\_c**,** **None** |

###### 何为generate\_bbox

|  |
| --- |
| **def** generate\_bbox**(**self**,** cls\_map**,** reg**,** scale**,** threshold**):**  """  generate bbox from feature cls\_map  Parameters:  ----------  cls\_map: numpy array , **n x m**  detect score for each position  reg: numpy array , n x m x 4  bbox  scale: float number  scale of this detection  threshold: float number  detect threshold  Returns:  -------  bbox array  """  cellsize **=** 12  t\_index **=** np**.**where**(**cls\_map **>** threshold**)## 阈值0.9**  # find nothing  **if** t\_index**[**0**].**size **==** 0**:**  **return** np**.**array**([])**  **#offset**  **## clsmap是h\*w\*1的. 找到每个”1”中阈值大于0.9的.对应的id.**   1. **比如第i个dim0,第j个dim1那个”1”的值大于0.9.**   **这里的t\_index[0]描述的是”n”的位置(高h).**  **t\_index[1]描述是”m”位置(宽w).**   1. **根据上面的I,j找到对应的boundingbox的坐标.**      1. **由于是图像金字塔进入到Pnet中. Pnet的输出boundingbox需要”放大”到原图尺寸中.** 2. **Stride和scale是缩放时候的参数.scale是系数, stride是跳行数.**     dx1**,** dy1**,** dx2**,** dy2 **=** **[**reg**[**t\_index**[**0**],** t\_index**[**1**],** i**]** **for** i **in** range**(**4**)]**  reg **=** np**.**array**([**dx1**,** dy1**,** dx2**,** dy2**])**  score **=** cls\_map**[**t\_index**[**0**],** t\_index**[**1**]]**  boundingbox **=** np**.**vstack**([**np**.**round**((**self**.**stride **\*** t\_index**[**1**])** **/** scale**),**  np**.**round**((**self**.**stride **\*** t\_index**[**0**])** **/** scale**),**  np**.**round**((**self**.**stride **\*** t\_index**[**1**]** **+** cellsize**)** **/** scale**),**  np**.**round**((**self**.**stride **\*** t\_index**[**0**]** **+** cellsize**)** **/** scale**),**  score**,**  reg**])**  **return** boundingbox**.**T |

###### 何为py\_nms

|  |
| --- |
| **def** py\_nms**(**dets**,** thresh**,** mode**=**"Union"**):**  **## 把预测框中阈值大于thresh的框去掉(保留score最大的那个框,其他的去掉).**   1. **输入是boxes候选框(含有reg在原图中的位置及大小, cls的人脸概率值).** 2. **输出候选框中带保存的indexes.**   """  greedily select boxes with high confidence  keep boxes overlap <= thresh  rule out overlap > thresh  :param dets: [[x1, y1, x2, y2 score]]  :param thresh: retain overlap <= thresh  :return: indexes to keep  """  x1 **=** dets**[:,** 0**]**  y1 **=** dets**[:,** 1**]**  x2 **=** dets**[:,** 2**]**  y2 **=** dets**[:,** 3**]**  scores **=** dets**[:,** 4**]**  areas **=** **(**x2 **-** x1 **+** 1**)** **\*** **(**y2 **-** y1 **+** 1**)**  order **=** scores**.**argsort**()[::-**1**] ## 按人脸概率(得分)降序排序.**  keep **=** **[]**  **while** order**.**size **>** 0**:**  i **=** order**[**0**] ## 取最大值(最大得分)对应的index.**  keep**.**append**(**i**) ## 需要保存的(因为是最大的).**  **## 向量化处理,计算Iou.有两种方式.**   1. **计算交并比.** 2. **计算相交部分和最小面积的比.**   xx1 **=** np**.**maximum**(**x1**[**i**],** x1**[**order**[**1**:]])**  yy1 **=** np**.**maximum**(**y1**[**i**],** y1**[**order**[**1**:]])**  xx2 **=** np**.**minimum**(**x2**[**i**],** x2**[**order**[**1**:]])**  yy2 **=** np**.**minimum**(**y2**[**i**],** y2**[**order**[**1**:]])**  w **=** np**.**maximum**(**0.0**,** xx2 **-** xx1 **+** 1**)**  h **=** np**.**maximum**(**0.0**,** yy2 **-** yy1 **+** 1**)**  inter **=** w **\*** h  **if** mode **==** "Union"**:**  ovr **=** inter **/** **(**areas**[**i**]** **+** areas**[**order**[**1**:]]** **-** inter**)**  **elif** mode **==** "Minimum"**:**  ovr **=** inter **/** np**.**minimum**(**areas**[**i**],** areas**[**order**[**1**:]])**  #keep  inds **=** np**.**where**(**ovr **<=** thresh**)[**0**]**  order **=** order**[**inds **+** 1**]**  **return** keep |

###### Understanding mtcnn raw reg(bbox)

|  |
| --- |
| 格式:  num\*9(x1,y1,x2,y2,score,x1\_offset,y1\_offset,x2\_offset,y2\_offset) |

1. 9个成员.
2. X1,y1,x2,y2是12\*12的小图中,第(x1,y1)像素是人脸框的左上角, 第(x2,y2)像素是人脸框的右下角.
3. Score是该人脸框的得分.
4. X1\_offset,…是距离gtbox的距离(线性距离).
5. 因为mtcnn的回归目标是Positive的样本.
6. 这些pos样本的生成是gtboxes相邻iou>0.7的矩形框.
7. 所以回归出来(推断出来)的结果也是与”真值”相邻的矩形框.
8. 所以需要对推断的值做offset的修正(校验, calibration).

## 3.2 pnet数据后处理

## 3.3 图像金字塔

# 四 一些问题

## 4.1 关于算法后期优化若干问题

|  |
| --- |
| 问: 楼主，你好，当我重新按照你给的方式训练完模型（数据集采用WIDER\_train），发现误检率还是挺高的，同时，人脸检测过程耗时严重（图像：1262\*750）约3.6s，请问楼主有相同的困扰吗？或者可以指明一下优化思路，感谢！ |

窗体顶端

|  |
| --- |
| 问: [**@homedawn**](https://github.com/homedawn) 你好，后面你解决了耗时和误检率的问题了不 |

|  |
| --- |
| **耗时没多大研究，误检可能之前的训练方法不对，后面重新训练几次有改善一些，主要可能还是训练集的问题** |
| 问: [**@homedawn**](https://github.com/homedawn) 您好，可以分享一下训练经验吗？我fddb上误检300时，true positive最多也就90%，而原版有92%。尝试了很多种训练方法都没有用。 |

|  |
| --- |
| 试一试在widerface上训练一下。关于耗时问题发现pNet最耗时几乎占了整个识别的90%时间，在研究pNet加入多进程来提升效率，碰到了一些问题还没解决，有人一起研究一下么？ ps:我发现github上的@不懂用，哪位大神可否告知一下哈 |
| 问: [**@homedawn**](https://github.com/homedawn) 我一直是在wider上进行训练的，但是一直达不到作者论文里fddb的曲线。 |
| **训练的时候是Pnet训练完再训练Rnet，Rnet里传入Pnet训练完的模型** |
| 问: [**@homedawn**](https://github.com/homedawn) 你是否在wider face的validation set上验证过？我在easy/medium/hard set上跑出来的精度大约只有30%, 30%, 20%。 |
| [**@htjacky**](https://github.com/htjacky)**没有验证，目前在看减少耗时的问题** |
| 问: [**@homedawn**](https://github.com/homedawn) 请问减少耗时的方法你找到了么？用作者的模型，GPU K40M上检测，每次都需要1s以上才能出结果，而且经常出来两个人脸 |
| **可以参考这个的解决方法**[**https://github.com/AITTSMD/MTCNN-Tensorflow/issues/220，但是目前测试结果看，主要耗时在三个网络层哪里，这个不好弄！！**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/220%EF%BC%8C%E4%BD%86%E6%98%AF%E7%9B%AE%E5%89%8D%E6%B5%8B%E8%AF%95%E7%BB%93%E6%9E%9C%E7%9C%8B%EF%BC%8C%E4%B8%BB%E8%A6%81%E8%80%97%E6%97%B6%E5%9C%A8%E4%B8%89%E4%B8%AA%E7%BD%91%E7%BB%9C%E5%B1%82%E5%93%AA%E9%87%8C%EF%BC%8C%E8%BF%99%E4%B8%AA%E4%B8%8D%E5%A5%BD%E5%BC%84%EF%BC%81%EF%BC%81) |

## 4.2 模型对图片通道有要求？

问:用PIL.Image读取的时候检测不到人脸，或者检测到很少的人脸；

使用cv2读取图片的时候就很正常了，难道模型对RGB或者BGR有要求？

**@dear-john 对，注意通道顺序**

# 五 faceNet训练

## 5.1 mtcnn align和多人脸间compare

Facenet

|  |
| --- |
| https://github.com/davidsandberg/facenet |

**下载数据集**

|  |
| --- |
| wget http://vis-www.cs.umass.edu/lfw/lfw.tgz |

**环境位置**

|  |
| --- |
| cd src  export PYTHONPATH=$PYTHONPATH:`pwd`  cd .. |

可以写成一个envsetup.sh脚本

|  |
| --- |
| #!/bin/bash  cd src  export PYTHONPATH=$PYTHONPATH:`pwd`  cd .. |

chmod a+x envsetup.sh

直接执行(./envsetup.sh)并不能使export生效.

需要使用source命令

**source envsetup.sh**

**将lfw数据集做align处理**

|  |
| --- |
| python src/align/align\_dataset\_mtcnn.py ./lfw/raw ./lfw/lfw\_mtcnnpy\_160 --image\_size 160 --margin 32 --random\_order |

**在lfw上测试下facenet pretrained model**

|  |
| --- |
| mkdir pretrained  cp /data/chapter\_6\_data/\* pretrained/  python src/validate\_on\_lfw.py ./lfw/lfw\_mtcnnpy\_160 ./pretrained/ |

实验结果:

|  |
| --- |
| Runnning forward pass on LFW images  Accuracy: 0.992+-0.003  Validation rate: 0.97467+-0.01477 @ [FAR](#_何为FAR?)=0.00133  Area Under Curve (AUC): 1.000  Equal Error Rate (EER): 0.007 |

**检查如下3张照片的相似度(距离矩阵).**



|  |
| --- |
| python src/compare.py ./pretrained/ ./test\_imgs/1.jpg ./test\_imgs/2.jpg ./test\_imgs/3.jpg |

实验结果:

|  |
| --- |
| Distance matrix  0 1 2  0 0.0000 0.7270 1.1283  1 0.7270 0.0000 1.0913  2 1.1283 1.0913 0.0000 |

### 何为lfw?

网址在: <http://vis-www.cs.umass.edu/lfw/>

*(Labeled Faces in the Wild Home)*

其目的是, 为了研究非限制的人脸识别(unconstrained face recognition).

1. 数据包含13000张来自web的人脸.
2. 含有4套(sets)lfw.
3. 一套origin的.
4. 3套aligned
5. 其一为funneled imgs
6. 其二为lwf-a, 采用非公开的aligned方法.
7. 其三为deep funneled imgs.
8. 可以作为train和test使用. 采用10折交叉
9. 也可以作为benchmark来使用.采用10折交叉.
10. 其分析结果最好是做出ROC曲线以及Accuracy值.

### 何为FAR?

**FAR**

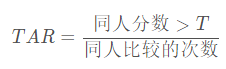
接下来看人脸识别中的 常用的指标TAR和FAR。TAR（True Accept Rate）表示正确接受的比例，**FAR（False Accept Rate）**表示错误接受的比例。**所谓的接受就是在进行人脸验证的过程中，两张图像被认为是同一个人**。在网上到的了FAR（False Accept Rate）的计算方式如下：



对于人脸识别不太了解的小伙伴可能不太了解上面公式的意思，下面我就来解释一下。做人脸验证的时候我们会给出两张图像让算法判断两张图片是不是同一个人的。一般是先将两张图片表示成两个高维的特征向量，然后计算两个特征向量的相似度或者距离。在这里定义FAR时使用的是相似度，公式中分数就是指的相似度。在比较的过程中我们希望同一个人的图像相似度比较高，**不同人的相似度**比较低。我们会给定一个相似度阈值T，比如0.6, 如果两张图像的相似度大于T我们就认为两张图片是一个人的，如果小于T我们就认为两证图像 是不同人的。但是无论将T设置成什么样值都会有一定得错误率，就是FAR，因为我们提取的图像的特征向量总是不够好，并不总能 满足：同一个人的图像相似度比较高，不同人的相似度比较低。偶尔也会出现不同人的图像的相似度大于给定的阈值T，这样我们就会犯错误接受的错误。**FAR就是我们比较不同人的图像时，把其中的图像对当成同一个人图像的比例。**我们希望**FAR越小越好**。

**TAR**

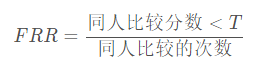
TAR**（True Accept Rate）**表示正确接受的比例。所谓的就是在进行人脸验证的过程中，两张图像被认为是同一个人。我在网上并没有查到TAR的定义，在这里我参照FAR的定义猜测一下，不一定正确，如有错误还望批评指正。



参考上面对 FAR的解释，我们知道对相同人的图片对进行比较也会出现相似度小于阈值T的情况，这是我们就会犯错误，这个错误就是FRR( False Reject Rate),就是把相同的人的图像当做不同人的了，这个下面我们再讲。可以理解 TAR就是对相同人的图片对进行比较，我们计算出的相似度大于阈值的图像对所占的比例。**我们希望TAR越大越好**。

**FRR**

上面讲解TAR的时候，提到了**FRR（False Reject Rate）**就是错误拒绝率。**就是把相同的人的图像当做不同人**的了。有了上面的基础，FRR就很好理解了：



根据上述公式的定义，我们知道

FRR + TAR = 1 , TAR=1−FRR

FAR和FRR的定义可以概括如下：



## 5.2 facenet train

需要CASIA-WebFace数据集.需要申请.可在百度网盘找到.

将”层次不齐的”人脸图利用mtcnn,整理出人脸框.

|  |
| --- |
| python src/align/align\_dataset\_mtcnn.py CASIA-WebFace/raw CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182 --image\_size 182 --margin 44 |

### 何为CASIA-WebFace数据集

李子青组的 CASIA-WebFace(50万，1万个人).

## 5.3 facenet性能指标

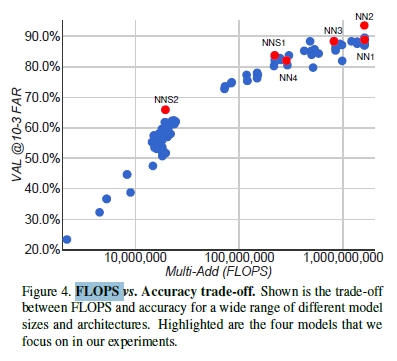
### 何为FLOPS

对于一个卷积层，假设其大小为  （其中c为输入通道数, n为输出通道数），输出的feature map尺寸为  ，则该卷积层的

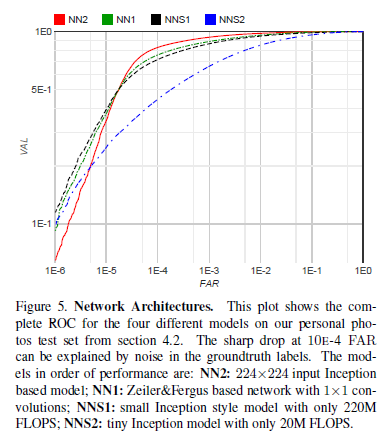


即#FLOPS= H' \* W' \* #paras

1. 随着网络复杂度增加,VAL(10-3的FAR)(10-3的错误识别率的精确率)会增大.

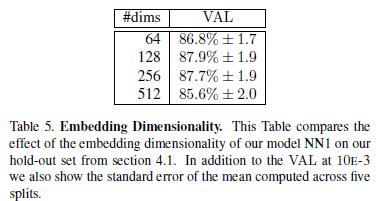


### VAL(TAR)和FAR结果



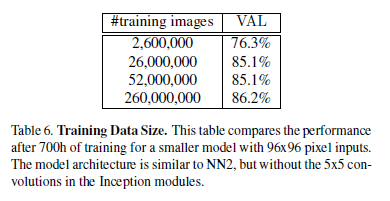
1. VAL越高越好(准确率, 类似TAR)
2. FAR越低越好(错误识别率).
3. 固定一个看另一个才有意义. 比如我们可能会给出一个期望的FAR值(比如1e-3),看此时的val值那个最大.
4. 选择1e-3的FAR,而不是更小,原因是从图上看出在1e-4后Val率会有一个”断崖”下降,这应该不能接受的.

### Embedding维度对val影响



1. 维度增长和val增长并非严格正相关
2. 在128维效果最好.

数据集大小对精度影响



1. 正相关
2. 数据集增大后期,精度提高不明显.
3. 应该是基于同一个FAR上的VAL.

# 六 faceNet之人脸预对齐

采用mtcnn去做的人脸检测以及人脸对齐(利用mtcnn推断的landmark来做旋转,把目标人脸摆正).

在src/align/align\_dataset\_mtcnn.py中.

## 6.1 align 操作

|  |
| --- |
| **def** main**(**args**):**  **## align作用:**   1. **根据mtcnn输出landmark,把人脸摆正.** 2. **Mtcnn可以对图像先做一个人脸框检测, 输出一个候选的人脸框.这些候选人脸框里去做facenet,缩小范围(满足facenet输入的要求).**   sleep**(**random**.**random**())**  output\_dir **=** os**.**path**.**expanduser**(**args**.**output\_dir**)**  **if** **not** os**.**path**.**exists**(**output\_dir**):**  os**.**makedirs**(**output\_dir**)**  # Store some git revision info in a text file in the log directory  src\_path**,** \_ **=** os**.**path**.**split**(**os**.**path**.**realpath**(**\_\_file\_\_**))**  facenet**.**store\_revision\_info**(**src\_path**,** output\_dir**,** ' '**.**join**(**sys**.**argv**))**  **## 打开lfw数据集(获得数据集描述).**  dataset **=** facenet**.**get\_dataset**(**args**.**input\_dir**)**  **print(**'Creating networks and loading parameters'**)**  **with** tf**.**Graph**().**as\_default**():**  gpu\_options **=** tf**.**GPUOptions**(**per\_process\_gpu\_memory\_fraction**=args.gpu\_memory\_fraction) ## 使用gpu的内存限制(例,值0.1, 则只会使用当前gpu显存的10%).**  sess **=** tf**.**Session**(**config**=**tf**.**ConfigProto**(**gpu\_options**=**gpu\_options**,** log\_device\_placement**=False))**  **## 根据pretrained模型得到mtcnn网路输出.**   1. **Mtcnn的参数. 最小人脸,minsize=20** 2. **Pnet,rnet,onet的nms阈值** 3. **图像金字塔的缩小系数.factor.**   **with** sess**.**as\_default**():**  pnet**,** rnet**,** onet **=** **align.detect\_face.create\_mtcnn(**sess**,** **None)**  minsize **=** 20 # minimum size of face  threshold **=** **[**0.6**,** 0.7**,** 0.7**]** # three steps's threshold  factor **=** 0.709 # scale factor  # Add a random key to the filename to allow alignment using multiple processes  random\_key **=** np**.**random**.**randint**(**0**,** high**=**99999**)**  bounding\_boxes\_filename **=** os**.**path**.**join**(**output\_dir**,** 'bounding\_boxes\_%05d.txt' **%** random\_key**)**  **with** open**(**bounding\_boxes\_filename**,** "w"**)** **as** text\_file**:**  **## 使用mtcnn前数据处理以及mtcnn输出的后处理**   1. **给每一个输入图片,建立一个输出图片(aligned)的文件位置.** 2. **输入图片必须是2维或3维的.2维图片要转换(to\_rgb).** 3. **Random.shuffle可以打乱数据集的顺序(图片的顺序).** 4. **涉及到mtcnn输出多张人脸时,根据一个判据来得到最”好”(最大置信)的人脸.**   nrof\_images\_total **=** 0  nrof\_successfully\_aligned **=** 0  **if** args**.**random\_order**:**  random**.**shuffle**(**dataset**)**  **for** cls **in** dataset**:**  output\_class\_dir **=** os**.**path**.**join**(**output\_dir**,** cls**.**name**)**  **if** **not** os**.**path**.**exists**(**output\_class\_dir**):**  os**.**makedirs**(**output\_class\_dir**)**  **if** args**.**random\_order**:**  random**.**shuffle**(**cls**.**image\_paths**)**  **for** image\_path **in** cls**.**image\_paths**:**  nrof\_images\_total **+=** 1  filename **=** os**.**path**.**splitext**(**os**.**path**.**split**(**image\_path**)[**1**])[**0**]**  output\_filename **=** os**.**path**.**join**(**output\_class\_dir**,** filename **+** '.png'**)**  **print(**image\_path**)**  **if** **not** os**.**path**.**exists**(**output\_filename**):**  **try:**  img **=** misc**.**imread**(**image\_path**)**  **except** **(**IOError**,** ValueError**,** IndexError**)** **as** e**:**  errorMessage **=** '{}: {}'**.**format**(**image\_path**,** e**)**  **print(**errorMessage**)**  **else:**  **if** img**.**ndim **<** 2**:**  **print(**'Unable to align "%s"' **%** image\_path**)**  text\_file**.**write**(**'%s\n' **%** **(**output\_filename**))**  **continue**  **if** img**.**ndim **==** 2**:**  img **=** facenet**.to\_rgb(**img**)**  img **=** img**[:,** **:,** 0**:**3**]**  bounding\_boxes**,** \_ **=** [**align.detect\_face.detect\_face**](#_6.3_何为detect_face)**(**img**,** minsize**,** pnet**,** rnet**,** onet**,** threshold**,** factor**)**  nrof\_faces **=** bounding\_boxes**.**shape**[**0**]**    **## 当mtcnn输出多张人脸时, 涉及一个评判标准(类似nms的筛选机制).**   1. **人脸框越大越有信心** 2. **人脸框距离图片中心越小越有信心.** 3. **输出判据最大的那个人脸.** 4. ***np.argmax(bounding\_box\_size - offset\_dist\_squared \* 2.0)***   **if** nrof\_faces **>** 0**:**  det **=** bounding\_boxes**[:,** 0**:**4**]**  img\_size **=** np**.**asarray**(**img**.**shape**)[**0**:**2**]**  **if** nrof\_faces **>** 1**:**  bounding\_box\_size **=** **(**det**[:,** 2**]** **-** det**[:,** 0**])** **\*** **(**det**[:,** 3**]** **-** det**[:,** 1**])**  img\_center **=** img\_size **/** 2  offsets **=** np**.**vstack**([(**det**[:,** 0**]** **+** det**[:,** 2**])** **/** 2 **-** img\_center**[**1**],** **(**det**[:,** 1**]** **+** det**[:,** 3**])** **/** 2 **-** img\_center**[**0**]])**  offset\_dist\_squared **=** np**.**sum**(**np**.**power**(**offsets**,** 2.0**),** 0**)**  index **=** [np**.**argmax](#_理解_align最大置信判据)**(**bounding\_box\_size **-** offset\_dist\_squared **\*** 2.0**)** # some extra weight on the centering  det **=** det**[**index**,** **:]**  det **=** np**.**squeeze**(**det**)**  bb **=** np**.**zeros**(**4**,** dtype**=**np**.**int32**)**  #print(det[0], det[1], det[2], det[3])  bb**[**0**]** **=** np**.**maximum**(**det**[**0**]** **-** args**.**margin **/** 2**,** 0**)**  bb**[**1**]** **=** np**.**maximum**(**det**[**1**]** **-** args**.**margin **/** 2**,** 0**)**  bb**[**2**]** **=** np**.**minimum**(**det**[**2**]** **+** args**.**margin **/** 2**,** img\_size**[**1**])**  bb**[**3**]** **=** np**.**minimum**(**det**[**3**]** **+** args**.**margin **/** 2**,** img\_size**[**0**])**  #print(bb)  cropped **=** img**[**bb**[**1**]:**bb**[**3**],** bb**[**0**]:**bb**[**2**],** **:]**  scaled **=** misc**.**imresize**(**cropped**,** **(**args**.**image\_size**,** args**.**image\_size**),** interp**=**'bilinear'**)**  nrof\_successfully\_aligned **+=** 1  misc**.**imsave**(**output\_filename**,** scaled**)**  text\_file**.**write**(**'%s %d %d %d %d\n' **%** **(**output\_filename**,** bb**[**0**],** bb**[**1**],** bb**[**2**],** bb**[**3**]))**  **else:**  **print(**'Unable to align "%s"' **%** image\_path**)**  text\_file**.**write**(**'%s\n' **%** **(**output\_filename**))**  **print(**'Total number of images: %d' **%** nrof\_images\_total**)**  **print(**'Number of successfully aligned images: %d' **%** nrof\_successfully\_aligned**)** |

## 6.2 understanding align\_datset\_mtcnn.py

1. 区别于传统的”align”含义. 没有依据landmark去做人脸的”摆正”操作.

2. 对于一副原图, 利用matcnn的检测,把人脸框crop出来.

3. facenet需要的图尺寸是160x160的,上一步crop出的人脸框是参差不齐的.需要resize到期望尺寸(160x160). 这个crop, resize的过程,在这里称为”align”

## 6.3 何为detect\_face

### 6.3.1 网络骨架搭建(Pnet)

#### understanding python装饰器(decorator)

|  |  |
| --- | --- |
| **def** layer**(**op**):**  **## 起到动态装饰器作用.**   1. **下面由layer修饰(会被包裹)的函数, 都要被layer的layer\_decorated函数包裹(形成包含)** 2. **类似如下的修饰作用(对函数的包裹).**  |  | | --- | | *@def1*  *@def2***(**arg1**,**arg2**)**  **def** test**(**aug**):**  **pass**  其展开如下：  def1(def2(arg1,arg2)(test(aug))) |   '''Decorator for composable network layers.'''  **def** layer\_decorated**(**self**,** **\***args**,** **\*\***kwargs**):**  # Automatically set a name if not provided.  name **=** kwargs**.**setdefault**(**'name'**,** self**.**get\_unique\_name**(**op**.**\_\_name\_\_**))**  # Figure out the layer inputs.  **if** len**(**self**.**terminals**)** **==** 0**:**  **raise** RuntimeError**(**'No input variables found for layer %s.' **%** name**)**  **elif** len**(**self**.**terminals**)** **==** 1**:**  layer\_input **=** self**.**terminals**[**0**] ## 拿到该层的输入**  **else:**  layer\_input **=** list**(**self**.**terminals**)**  # Perform the operation and get the output.  **## op是”被包裹”的函数(一般是tf.nn.add等)的操作.**  layer\_output **=** **op(**self**,** layer\_input**,** **\***args**,** **\*\***kwargs**)**  # Add to layer LUT.  **self.layers[name] = layer\_output**  # This output is now the input for the next layer.  self**.**feed**(**layer\_output**)**  # Return self for chained calls.  **return** self  **return** layer\_decorated |

#### 被”修饰器”(包裹)实例Pnet

如下的conv层,是被layer装饰器包裹的.

1. 首先要执行layer的layer\_decorated函数.
2. 在layer\_decorated函数执行***”被包裹的”***op操作(op等价于此处的conv函数).
3. 在执行op前,要把op操作需要的input数据准备好(从self.terminals[0]中读取,以传参方式给op操作使用.)
4. 在执行op后,要把op操作的输出output及时更新到self.terminals中(是个list).

|  |
| --- |
| *@layer*  **def** conv**(**self**,**  inp**,**  k\_h**,**  k\_w**,**  c\_o**,**  s\_h**,**  s\_w**,**  name**,**  relu**=True,**  padding**=**'SAME'**,**  group**=**1**,**  biased**=True):**  # Verify that the padding is acceptable  self**.**validate\_padding**(**padding**)**  # Get the number of channels in the input  c\_i **=** int**(**inp**.**get\_shape**()[-**1**])**  # Verify that the grouping parameter is valid  **assert** c\_i **%** group **==** 0  **assert** c\_o **%** group **==** 0  # Convolution for a given input and kernel  convolve **=** **lambda** i**,** k**:** tf**.**nn**.**conv2d**(**i**,** k**,** **[**1**,** s\_h**,** s\_w**,** 1**],** padding**=**padding**)**  **with** tf**.**variable\_scope**(**name**)** **as** scope**:**  kernel **=** self**.**make\_var**(**'weights'**,** shape**=[**k\_h**,** k\_w**,** c\_i **//** group**,** c\_o**])**  # This is the common-case. Convolve the input without any further complications.  output **=** convolve**(**inp**,** kernel**)**  # Add the biases  **if** biased**:**  biases **=** self**.**make\_var**(**'biases'**,** **[**c\_o**])**  output **=** tf**.**nn**.**bias\_add**(**output**,** biases**)**  **if** relu**:**  # ReLU non-linearity  output **=** tf**.**nn**.**relu**(**output**,** name**=**scope**.**name**)**  **return** output |

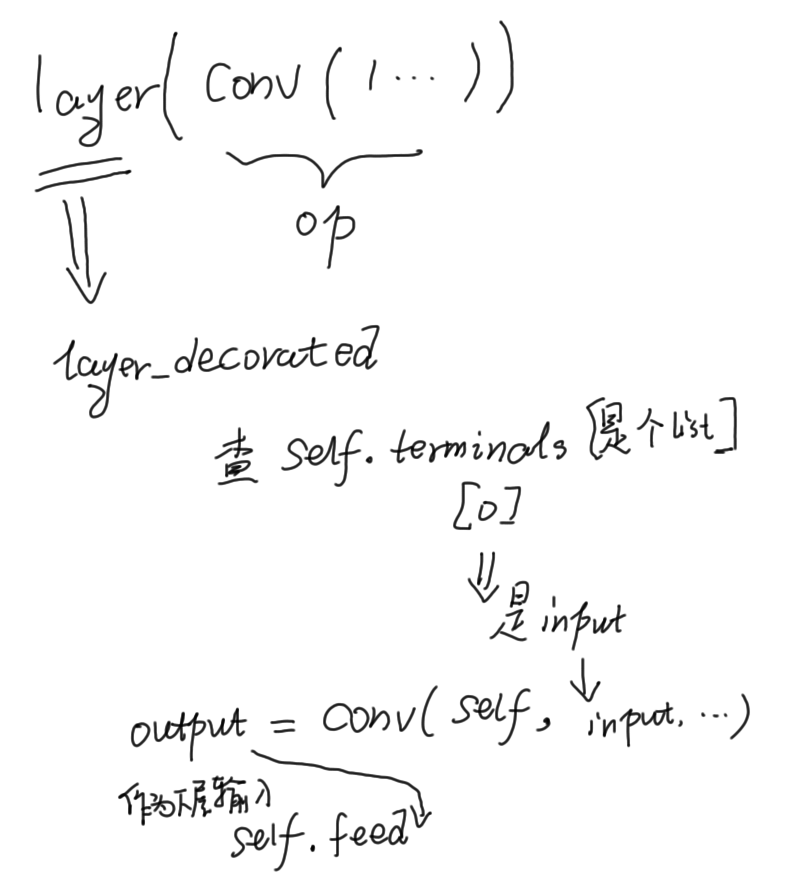
##### Pnet输入

|  |
| --- |
| **def** feed**(**self**,** **\***args**):**  **## 喂给网络输入.**   1. **网络的所有输出都会存在self.terminals中** 2. **Self.terminals是个list** 3. **Self.terminals[0]是整个网络的input.** 4. **Self.terminals[i]是第i层单元(conv或者relu)的输出,也是第i+1层单元的输入.** 5. **从layers中取出对应tensor, 放到self.terminals中.** 6. **layers是”字典”(?)**   '''Set the input(s) for the next operation by replacing the terminal nodes.  The arguments can be either layer names or the actual layers.  '''  **assert** len**(**args**)** **!=** 0  self**.**terminals **=** **[]**  **for** fed\_layer **in** args**:**  **if** isinstance**(**fed\_layer**,** string\_types**):**  **try:**  fed\_layer **=** **self.layers[**fed\_layer**]**  **except** KeyError**:**  **raise** KeyError**(**'Unknown layer name fed: %s' **%** fed\_layer**)**  **self.terminals.append(**fed\_layer**)**  **return** self |

##### Pnet输出

|  |
| --- |
| **def** layer\_decorated**(**self**,** **\***args**,** **\*\***kwargs**):**  # Automatically set a name if not provided.  name **=** kwargs**.**setdefault**(**'name'**,** self**.**get\_unique\_name**(**op**.**\_\_name\_\_**))**  # Figure out the layer inputs.  **if** len**(**self**.**terminals**)** **==** 0**:**  **raise** RuntimeError**(**'No input variables found for layer %s.' **%** name**)**  **elif** len**(**self**.**terminals**)** **==** 1**:**  layer\_input **=** self**.**terminals**[**0**] ## 拿到该层的输入**  **else:**  layer\_input **=** list**(**self**.**terminals**)**  # Perform the operation and get the output.  **## op是”被包裹”的函数(一般是tf.nn.add等)的操作.**  layer\_output **=** **op(**self**,** layer\_input**,** **\***args**,** **\*\***kwargs**)**  # Add to layer LUT.  **self.layers[name] = layer\_output ## 更新输出为下一层的输入.**  # This output is now the input for the next layer.  self**.**feed**(**layer\_output**)**  # Return self for chained calls.  **return** self |

##### 动态修饰器下的Pnet网络简图



### 6.3.2 网络”血液”的填充

#### 从caffemode中复原网络

何为pnet?

|  |
| --- |
| **def** create\_mtcnn**(**sess**,** model\_path**):**  **if** **not** model\_path**:**  model\_path**,**\_ **=** os**.**path**.**split**(**os**.**path**.**realpath**(**\_\_file\_\_**))**  **with** tf**.**variable\_scope**(**'pnet'**):**  data **=** tf**.**placeholder**(**tf**.**float32**,** **(None,None,None,**3**),** 'input'**)**  pnet **=** PNet**({**'data'**:**data**})## 先创建pnet的网络骨架**  pnet**.**load**(**os**.**path**.**join**(**model\_path**,** 'det1.npy'**),** sess**) ##　再加载已经train好的网络权重参数. (注入血液).**  **with** tf**.**variable\_scope**(**'rnet'**):**  data **=** tf**.**placeholder**(**tf**.**float32**,** **(None,**24**,**24**,**3**),** 'input'**)**  rnet **=** RNet**({**'data'**:**data**})**  rnet**.**load**(**os**.**path**.**join**(**model\_path**,** 'det2.npy'**),** sess**)**  **with** tf**.**variable\_scope**(**'onet'**):**  data **=** tf**.**placeholder**(**tf**.**float32**,** **(None,**48**,**48**,**3**),** 'input'**)**  onet **=** ONet**({**'data'**:**data**})**  onet**.**load**(**os**.**path**.**join**(**model\_path**,** 'det3.npy'**),** sess**)**    **pnet\_fun = lambda img : sess.run(('pnet/conv4-2/BiasAdd:0', 'pnet/prob1:0'), feed\_dict={'pnet/input:0':img})**  rnet\_fun **=** **lambda** img **:** sess**.**run**((**'rnet/conv5-2/conv5-2:0'**,** 'rnet/prob1:0'**),** feed\_dict**={**'rnet/input:0'**:**img**})**  onet\_fun **=** **lambda** img **:** sess**.**run**((**'onet/conv6-2/conv6-2:0'**,** 'onet/conv6-3/conv6-3:0'**,** 'onet/prob1:0'**),** feed\_dict**={**'onet/input:0'**:**img**})**  **## 特别要注意, 这里返回的是pnet\_fun(是pnet网络的输出层组),而不是我们代码前面的pnet的网络骨架.**  **return** pnet\_fun**,** rnet\_fun**,** onet\_fun |

### 6.3.2 理解detect\_face

前面分析了detect\_face的核心pnet网路的创建及参数恢复.

下面要解释pnet网络需要的输入的准备(金字塔), 及输出的后处理(多脸的最大判据).

#### Mtcnn输入的预处理(金字塔)

从mtcnn论文可知, 推断时候需要从多尺度去检测人脸.

1. 多尺度的方法就是输入原图的图像金字塔.

|  |
| --- |
| **def** detect\_face**(**img**,** minsize**,** pnet**,** rnet**,** onet**,** threshold**,** factor**):**  # im: input image  # minsize: minimum of faces' size  # pnet, rnet, onet: caffemodel  # threshold: threshold=[th1 th2 th3], th1-3 are three steps's threshold  # fastresize: resize img from last scale (using in high-resolution images) if fastresize==true  ## **分成几个步骤**   1. **图像金字塔的生成(预处理) 以VGA的图为例.** 2. **Minsize是最小人脸,20.** 3. **基础系数m为12/20** 4. **多层金字塔参数表现为:**      1. **利用scales生成新的h’,w’的尺寸(此时仅仅是尺寸生成了)** 2. **利用h’,w’的尺寸在原图img中resample(此时生成数据)** 3. **然后归一化.** 4. **最后对h’,w’的img’做一个转置** 5. **从RGB 转置成GRB.** 6. **RGB可以认为是img’\_x** 7. **GRB就是img’\_y**   factor\_count**=**0  total\_boxes**=**np**.**empty**((**0**,**9**))**  points**=[]**  h**=**img**.**shape**[**0**]**  w**=**img**.**shape**[**1**]**  minl**=**np**.**amin**([**h**,** w**])**  m**=**12.0**/**minsize  minl**=**minl**\***m  # creat scale pyramid  scales**=[]**  **while** minl**>=**12**:**  scales **+=** **[**m**\***np**.**power**(**factor**,** factor\_count**)]**  minl **=** minl**\***factor  factor\_count **+=** 1  # first stage  **for** j **in** range**(**len**(**scales**)):**  scale**=**scales**[**j**]**  hs**=**int**(**np**.**ceil**(**h**\***scale**))**  ws**=**int**(**np**.**ceil**(**w**\***scale**))**  im\_data **=** imresample**(**img**,** **(**hs**,** ws**))**  im\_data **=** **(**im\_data**-**127.5**)\***0.0078125  img\_x **=** np**.**expand\_dims**(**im\_data**,** 0**)**  img\_y **=** np**.**transpose**(**img\_x**,** **(**0**,**2**,**1**,**3**))**  **[out = pnet(img\_y)](#_6.3.3_understanding_of)**  out0 **=** np**.**transpose**(**out**[**0**],** **(**0**,**2**,**1**,**3**))**  out1 **=** np**.**transpose**(**out**[**1**],** **(**0**,**2**,**1**,**3**))**    boxes**,** \_ **=** **generateBoundingBox(**out1**[**0**,:,:,**1**].**copy**(),** out0**[**0**,:,:,:].**copy**(),** scale**,** threshold**[**0**])**    # inter-scale nms  pick **=** **nms(**boxes**.**copy**(),** 0.5**,** 'Union'**)**  **if** boxes**.**size**>**0 **and** pick**.**size**>**0**:**  boxes **=** boxes**[**pick**,:]**  total\_boxes **=** np**.**append**(**total\_boxes**,** boxes**,** axis**=**0**)**  **## pnet推断的后处理**   1. **需要nms剔除重叠框**   numbox **=** total\_boxes**.**shape**[**0**]**  **if** numbox**>**0**:**  pick **=** **nms(**total\_boxes**.**copy**(),** 0.7**,** 'Union'**)**  total\_boxes **=** total\_boxes**[**pick**,:]**  regw **=** total\_boxes**[:,**2**]-**total\_boxes**[:,**0**] ## 是w**  regh **=** total\_boxes**[:,**3**]-**total\_boxes**[:,**1**] ## y2-y1是H**  qq1 **=** total\_boxes**[:,**0**]+total\_boxes[:,5]\***regw  qq2 **=** total\_boxes**[:,**1**]+**total\_boxes**[:,**6**]\***regh  qq3 **=** total\_boxes**[:,**2**]+**total\_boxes**[:,**7**]\***regw  qq4 **=** total\_boxes**[:,**3**]+**total\_boxes**[:,**8**]\***regh  total\_boxes **=** np**.**transpose**(**np**.**vstack**([**qq1**,** qq2**,** qq3**,** qq4**,** total\_boxes**[:,**4**]]))**  total\_boxes **=** rerec**(**total\_boxes**.**copy**())**  total\_boxes**[:,**0**:**4**]** **=** np**.**fix**(**total\_boxes**[:,**0**:**4**]).**astype**(**np**.**int32**)**  dy**,** edy**,** dx**,** edx**,** y**,** ey**,** x**,** ex**,** tmpw**,** tmph **=** pad**(**total\_boxes**.**copy**(),** w**,** h**)**  numbox **=** total\_boxes**.**shape**[**0**]**  **if** numbox**>**0**:**  # second stage  tempimg **=** np**.**zeros**((**24**,**24**,**3**,**numbox**))**  **for** k **in** range**(**0**,**numbox**):**  tmp **=** np**.**zeros**((**int**(**tmph**[**k**]),**int**(**tmpw**[**k**]),**3**))**  tmp**[**dy**[**k**]-**1**:**edy**[**k**],**dx**[**k**]-**1**:**edx**[**k**],:]** **=** img**[**y**[**k**]-**1**:**ey**[**k**],**x**[**k**]-**1**:**ex**[**k**],:]**  **if** tmp**.**shape**[**0**]>**0 **and** tmp**.**shape**[**1**]>**0 **or** tmp**.**shape**[**0**]==**0 **and** tmp**.**shape**[**1**]==**0**:**  tempimg**[:,:,:,**k**]** **=** imresample**(**tmp**,** **(**24**,** 24**))**  **else:**  **return** np**.**empty**()**  tempimg **=** **(**tempimg**-**127.5**)\***0.0078125  tempimg1 **=** np**.**transpose**(**tempimg**,** **(**3**,**1**,**0**,**2**))**  out **=** rnet**(**tempimg1**)**  out0 **=** np**.**transpose**(**out**[**0**])**  out1 **=** np**.**transpose**(**out**[**1**])**  score **=** out1**[**1**,:]**  ipass **=** np**.**where**(**score**>**threshold**[**1**])**  total\_boxes **=** np**.**hstack**([**total\_boxes**[**ipass**[**0**],**0**:**4**].**copy**(),** np**.**expand\_dims**(**score**[**ipass**].**copy**(),**1**)])**  mv **=** out0**[:,**ipass**[**0**]]**  **if** total\_boxes**.**shape**[**0**]>**0**:**  pick **=** nms**(**total\_boxes**,** 0.7**,** 'Union'**)**  total\_boxes **=** total\_boxes**[**pick**,:]**  total\_boxes **=** bbreg**(**total\_boxes**.**copy**(),** np**.**transpose**(**mv**[:,**pick**]))**  total\_boxes **=** rerec**(**total\_boxes**.**copy**())**  numbox **=** total\_boxes**.**shape**[**0**]**  **if** numbox**>**0**:**  # third stage  total\_boxes **=** np**.**fix**(**total\_boxes**).**astype**(**np**.**int32**)**  dy**,** edy**,** dx**,** edx**,** y**,** ey**,** x**,** ex**,** tmpw**,** tmph **=** pad**(**total\_boxes**.**copy**(),** w**,** h**)**  tempimg **=** np**.**zeros**((**48**,**48**,**3**,**numbox**))**  **for** k **in** range**(**0**,**numbox**):**  tmp **=** np**.**zeros**((**int**(**tmph**[**k**]),**int**(**tmpw**[**k**]),**3**))**  tmp**[**dy**[**k**]-**1**:**edy**[**k**],**dx**[**k**]-**1**:**edx**[**k**],:]** **=** img**[**y**[**k**]-**1**:**ey**[**k**],**x**[**k**]-**1**:**ex**[**k**],:]**  **if** tmp**.**shape**[**0**]>**0 **and** tmp**.**shape**[**1**]>**0 **or** tmp**.**shape**[**0**]==**0 **and** tmp**.**shape**[**1**]==**0**:**  tempimg**[:,:,:,**k**]** **=** imresample**(**tmp**,** **(**48**,** 48**))**  **else:**  **return** np**.**empty**()**  tempimg **=** **(**tempimg**-**127.5**)\***0.0078125  tempimg1 **=** np**.**transpose**(**tempimg**,** **(**3**,**1**,**0**,**2**))**  **out = onet(tempimg1)**  out0 **=** np**.**transpose**(**out**[**0**])**  out1 **=** np**.**transpose**(**out**[**1**])**  out2 **=** np**.**transpose**(**out**[**2**])**  score **=** out2**[**1**,:]**  points **=** out1  ipass **=** np**.**where**(**score**>**threshold**[**2**])**  points **=** points**[:,**ipass**[**0**]]**  total\_boxes **=** np**.**hstack**([**total\_boxes**[**ipass**[**0**],**0**:**4**].**copy**(),** np**.**expand\_dims**(**score**[**ipass**].**copy**(),**1**)])**  mv **=** out0**[:,**ipass**[**0**]]**  w **=** total\_boxes**[:,**2**]-**total\_boxes**[:,**0**]+**1  h **=** total\_boxes**[:,**3**]-**total\_boxes**[:,**1**]+**1  points**[**0**:**5**,:]** **=** np**.**tile**(**w**,(**5**,** 1**))\***points**[**0**:**5**,:]** **+** np**.**tile**(**total\_boxes**[:,**0**],(**5**,** 1**))-**1  points**[**5**:**10**,:]** **=** np**.**tile**(**h**,(**5**,** 1**))\***points**[**5**:**10**,:]** **+** np**.**tile**(**total\_boxes**[:,**1**],(**5**,** 1**))-**1  **if** total\_boxes**.**shape**[**0**]>**0**:**  total\_boxes **=** bbreg**(**total\_boxes**.**copy**(),** np**.**transpose**(**mv**))**  pick **=** nms**(**total\_boxes**.**copy**(),** 0.7**,** 'Min'**)**  total\_boxes **=** total\_boxes**[**pick**,:]**  points **=** points**[:,**pick**]**    **return** total\_boxes**,** points |

#### Mtcnn的输出多人脸的后处理

##### 理解 align最大置信判据

对于一张原图输出的检测多人脸时候, 单对mtcnn的任务已经完成了. 但是对于facenet的网络训练来说,不需要多人脸去训练(**因为facenet的数据集都是一张照片标记一个人的信息,没有一张照片标记多个人的信息, 而且数据集的目标人脸都是”约定俗成”的靠近图片的中心位置**)

[相关的代码段:](#_6.1_align_操作)

src\align\align\_dataset\_mtcnn.py中.

|  |
| --- |
| **def** main**(**args**):**  ….  bounding\_boxes**,** \_ **=** align**.**detect\_face**.**detect\_face**(**img**,** minsize**,** pnet**,** rnet**,** onet**,** threshold**,** factor**)**  nrof\_faces **=** bounding\_boxes**.**shape**[**0**]**  **if** nrof\_faces **>** 0**:**  det **=** bounding\_boxes**[:,** 0**:**4**]**  img\_size **=** np**.**asarray**(**img**.**shape**)[**0**:**2**]**  **if** nrof\_faces **>** 1**:**  bounding\_box\_size **=** **(**det**[:,** 2**]** **-** det**[:,** 0**])** **\*** **(**det**[:,** 3**]** **-** det**[:,** 1**])**  img\_center **=** img\_size **/** 2  offsets **=** np**.**vstack**([(**det**[:,** 0**]** **+** det**[:,** 2**])** **/** 2 **-** img\_center**[**1**],** **(**det**[:,** 1**]** **+** det**[:,** 3**])** **/** 2 **-** img\_center**[**0**]])**  offset\_dist\_squared **=** np**.**sum**(**np**.**power**(**offsets**,** 2.0**),** 0**)**  **## 多人脸的判据.**  **index = np.argmax(bounding\_box\_size - offset\_dist\_squared \* 2.0)**  # some extra weight on the centering  det **=** det**[**index**,** **:]**  det **=** np**.**squeeze**(**det**)**  bb **=** np**.**zeros**(**4**,** dtype**=**np**.**int32**)**  #print(det[0], det[1], det[2], det[3])  bb**[**0**]** **=** np**.**maximum**(**det**[**0**]** **-** args**.**margin **/** 2**,** 0**)**  bb**[**1**]** **=** np**.**maximum**(**det**[**1**]** **-** args**.**margin **/** 2**,** 0**)**  bb**[**2**]** **=** np**.**minimum**(**det**[**2**]** **+** args**.**margin **/** 2**,** img\_size**[**1**])**  bb**[**3**]** **=** np**.**minimum**(**det**[**3**]** **+** args**.**margin **/** 2**,** img\_size**[**0**])**  #print(bb)  cropped **=** img**[**bb**[**1**]:**bb**[**3**],** bb**[**0**]:**bb**[**2**],** **:]**  scaled **=** misc**.**imresize**(**cropped**,** **(**args**.**image\_size**,** args**.**image\_size**),** interp**=**'bilinear'**)**  nrof\_successfully\_aligned **+=** 1  misc**.**imsave**(**output\_filename**,** scaled**)**  text\_file**.**write**(**'%s %d %d %d %d\n' **%** **(**output\_filename**,** bb**[**0**],** bb**[**1**],** bb**[**2**],** bb**[**3**]))** |

###### 实验

首先,需要先清掉目标文件(不然不会重新计算)

|  |
| --- |
| rm CASIA-WebFace/exp0\_182/\* -rf |

|  |
| --- |
| python src/align/align\_dataset\_mtcnn.py CASIA-WebFace/exp0 CASIA-WebFace/exp0\_182 --image\_size 182 --margin 44 **--debug true**    不需要shuffle.便于观察.  --random\_order |

###### 实验结果:

|  |
| --- |
| CASIA-WebFace/raw/0000099/052.jpg  align pic: CASIA-WebFace/raw/0000099/052.jpg  prams. img\_center [125. 125.]  prams. det [[ 79.63303906 49.66967024 190.00140981 195.39264115]  [154.71956933 28.03832471 244.80715144 142.13808279]]  prams. offsets [[ 9.81722444 74.76336038]  [ -2.46884431 -39.91179625]]  prams. bounding\_box\_size [16083.20688076 10278.97132435] |

###### 详细报告

|  |  |
| --- | --- |
| CASIA-WebFace/exp0/0000099/052.jpg  align pic: CASIA-WebFace/exp0/0000099/052.jpg  **分析数据detect\_face:**   1. **由9个数组成.** [**联合前面的Mtcnn训练的内容看**](#_Understanding_mtcnn_raw)**. 应该是x1,y1** 2. **num\*9(x1,y1,x2,y2,score,x1\_offset,y1\_offset,x2\_offset,y2\_offset)** 3. **区别是,这里(caffe的model)的offset应该不是线性距离.貌似是系数(?).** 4. **第一行为例,** 5. **其最后4个的值都在e-02次.** 6. **所以有如下的. 校验后的推断值qq1~4.**  |  | | --- | | regw **=** total\_boxes**[:,**2**]-**total\_boxes**[:,**0**]**  regh **=** total\_boxes**[:,**3**]-**total\_boxes**[:,**1**]**  qq1 **=** total\_boxes**[:,**0**]+**total\_boxes**[:,**5**]\***regw  qq2 **=** total\_boxes**[:,**1**]+**total\_boxes**[:,**6**]\***regh  qq3 **=** total\_boxes**[:,**2**]+**total\_boxes**[:,**7**]\***regw  qq4 **=** total\_boxes**[:,**3**]+**total\_boxes**[:,**8**]\***regh |   detect\_face. total\_boxes: [[ 8.30000000e+01 8.30000000e+01 1.86000000e+02 1.86000000e+02  9.99954104e-01 **3.13425176e-02 -9.66877788e-02 1.14675611e-02**  **2.93653160e-02**]  [ 1.64000000e+02 3.20000000e+01 2.37000000e+02 1.05000000e+02  9.98778403e-01 -4.18755598e-03 -7.17951432e-02 -2.79934742e-02  1.82621926e-01]  [ 1.12000000e+02 1.12000000e+02 1.84000000e+02 1.84000000e+02  9.94950771e-01 -4.09494527e-03 -2.35280469e-01 -3.43164317e-02  -6.30439371e-02]  [ 8.50000000e+01 9.80000000e+01 1.58000000e+02 1.71000000e+02  9.92114305e-01 -7.66013749e-03 -2.95226783e-01 1.35598272e-01  1.11130983e-01]  [ 6.50000000e+01 6.50000000e+01 2.09000000e+02 2.09000000e+02  9.89827991e-01 2.08121702e-01 2.39114128e-02 -1.42885260e-02  1.91108137e-02]  [ 1.78000000e+02 4.60000000e+01 2.50000000e+02 1.18000000e+02  9.74574506e-01 -3.08282748e-02 -1.22744337e-01 -6.86306655e-02  4.44093458e-02]  [ 1.12000000e+02 8.50000000e+01 1.84000000e+02 1.58000000e+02  9.72768605e-01 -4.72970419e-02 -1.08712852e-01 -4.03704867e-03  1.49357170e-01]  [ 4.00000000e+00 5.10000000e+01 5.60000000e+01 1.02000000e+02  9.65631604e-01 -2.98624113e-03 -6.56352043e-02 -1.57189697e-01  -7.06764013e-02]  [ 5.50000000e+01 1.80000000e+01 2.59000000e+02 2.22000000e+02  9.58166957e-01 1.60978079e-01 9.91562381e-03 -5.74716814e-02  -1.86594762e-02]  [ 1.18000000e+02 1.30000000e+01 2.62000000e+02 1.57000000e+02  9.54806805e-01 -3.08799185e-03 -3.31342816e-02 -1.02107763e-01  9.67335999e-02]  [ 2.00000000e+00 1.95000000e+02 2.80000000e+01 2.20000000e+02  9.47161615e-01 1.02360407e-02 -3.86864915e-02 -9.38440412e-02  5.98890446e-02]  [ 1.10000000e+01 6.80000000e+01 3.00000000e+01 8.60000000e+01  9.39644039e-01 -1.99308470e-02 -2.79780269e-01 1.39913633e-02  7.49252923e-03]  [ 1.00000000e+00 2.01000000e+02 2.00000000e+01 2.20000000e+02  9.30853724e-01 1.93698667e-02 -1.22916773e-01 -5.71118295e-03  6.71277344e-02]  [ 4.00000000e+00 3.20000000e+01 5.60000000e+01 8.40000000e+01  9.27658796e-01 -2.05240041e-01 1.12218156e-01 -2.95716017e-01  2.35284001e-01]  [ 8.50000000e+01 7.20000000e+01 1.58000000e+02 1.45000000e+02  9.26137090e-01 2.27832608e-02 -3.73607539e-02 1.27877772e-01  3.25976849e-01]  [ 1.39000000e+02 2.70000000e+01 2.41000000e+02 1.30000000e+02  9.07249928e-01 -4.14652415e-02 -1.56573236e-01 -2.49614492e-02  6.56117797e-02]  [ 1.16000000e+02 1.26000000e+02 1.68000000e+02 1.77000000e+02  8.75765026e-01 -1.74148306e-02 -9.37580541e-02 -2.22242624e-02  3.77057083e-02]  [ 2.00000000e+00 2.18000000e+02 2.80000000e+01 2.44000000e+02  8.64398658e-01 -1.08264312e-02 -9.45893675e-02 -5.77525795e-02  4.63720858e-02]  [ 3.00000000e+00 2.15000000e+02 3.90000000e+01 2.51000000e+02  8.41868043e-01 -4.58072536e-02 -2.46566199e-02 -1.68161184e-01  4.96683680e-02]  [ 3.00000000e+00 5.60000000e+01 3.90000000e+01 9.20000000e+01  8.34668100e-01 -1.02182701e-02 -6.83404431e-02 -1.11024410e-01  -2.98782699e-02]  [ 1.73000000e+02 4.20000000e+01 2.24000000e+02 9.30000000e+01  8.31634164e-01 -5.44488616e-02 -1.78418621e-01 1.64947286e-02  1.37188867e-01]  [ 1.38000000e+02 1.12000000e+02 2.11000000e+02 1.84000000e+02  8.28625202e-01 -1.14751488e-01 -5.28828651e-02 -8.60984996e-02  2.51627803e-01]  [ 8.80000000e+01 8.80000000e+01 1.40000000e+02 1.40000000e+02  8.19209099e-01 -6.79060817e-02 -7.16797262e-02 -1.51043646e-02  2.00817317e-01]  [ 1.29000000e+02 1.49000000e+02 1.65000000e+02 1.85000000e+02  8.17738116e-01 3.18847410e-03 -1.36947215e-01 6.38089702e-02  1.34276599e-01]  [ 2.18000000e+02 1.15000000e+02 2.44000000e+02 1.41000000e+02  7.63230443e-01 7.80712515e-02 -8.94545391e-02 4.20721322e-02  1.13227278e-01]  [ 2.10000000e+01 2.21000000e+02 4.00000000e+01 2.40000000e+02  7.52106845e-01 -2.20811889e-02 -9.11906958e-02 -1.29401326e-01  3.76951471e-02]  [ 1.02000000e+02 1.20000000e+02 2.04000000e+02 2.23000000e+02  7.34466195e-01 2.05935519e-02 -2.39180923e-01 -5.56785129e-02  -9.97726917e-02]  [ 2.21000000e+02 1.15000000e+02 2.40000000e+02 1.33000000e+02  7.33752787e-01 4.83403131e-02 -1.26435369e-01 4.76884171e-02  1.38486177e-01]  [ 7.00000000e+00 6.30000000e+01 3.20000000e+01 8.90000000e+01  7.23926783e-01 3.04365456e-02 -1.72067299e-01 -5.66879585e-02  -8.19441676e-02]  [ 1.69000000e+02 3.00000000e+00 2.05000000e+02 3.90000000e+01  7.16320395e-01 -5.91743849e-02 -2.42955595e-01 4.81644645e-02  1.05746478e-01]  [ 1.20000000e+02 8.30000000e+01 2.23000000e+02 1.86000000e+02  7.10307658e-01 -5.44877537e-02 -3.02043017e-02 -1.98099881e-01  7.21706077e-03]  [ 1.01000000e+02 1.48000000e+02 1.26000000e+02 1.73000000e+02  6.93565547e-01 -7.74517655e-02 -1.14602134e-01 6.16594404e-03  2.09736794e-01]  [ 1.10000000e+01 6.10000000e+01 3.00000000e+01 8.00000000e+01  6.89078689e-01 -7.35270083e-02 -1.38670892e-01 -1.36980675e-02  1.26948029e-01]  [ 1.49000000e+02 1.62000000e+02 1.85000000e+02 1.98000000e+02  6.81596279e-01 -1.05528474e-01 -2.90131420e-01 5.39351180e-02  1.85064241e-01]  [ 1.91000000e+02 5.10000000e+01 2.43000000e+02 1.02000000e+02  6.75976872e-01 4.97788936e-02 -7.34513626e-02 -2.32140496e-02  4.15287651e-02]  [ 8.00000000e+00 2.01000000e+02 2.60000000e+01 2.20000000e+02  6.63540840e-01 -1.87992118e-02 -5.64671978e-02 -9.91384163e-02  8.85773003e-02]  [ 1.10000000e+01 2.18000000e+02 3.70000000e+01 2.44000000e+02  6.06848598e-01 -6.41577542e-02 -1.06248632e-01 -8.37193280e-02  8.44048858e-02]  [ 8.20000000e+01 3.90000000e+01 1.08000000e+02 6.50000000e+01  6.04915857e-01 -4.86859865e-03 -9.47086811e-02 7.88573176e-04  1.51351869e-01]]  detect\_face. regw: [103. 73. 72. 73. 144. 72. 72. 52. 204. 144. 26. 19. 19. 52.  73. 102. 52. 26. 36. 36. 51. 73. 52. 36. 26. 19. 102. 19.  25. 36. 103. 25. 19. 36. 52. 18. 26. 26.]  detect\_face. regh: [103. 73. 72. 73. 144. 72. 73. 51. 204. 144. 25. 18. 19. 52.  73. 103. 51. 26. 36. 36. 51. 72. 52. 36. 26. 19. 103. 18.  26. 36. 103. 25. 19. 36. 51. 19. 26. 26.]  detect\_face. qq1: [ 86.22827931 163.69430841 111.70516394 84.44080996 94.9695251  175.78036422 108.59461299 3.84471546 87.83952808 117.55532917  2.26613706 10.62131391 1.36802747 -6.67248213 86.66317804  134.77054537 115.09442881 1.71851279 1.35093887 2.63214228  170.22310806 129.62314138 84.46888375 129.11478507 220.02985254  20.58045741 104.1005423 221.91846595 7.76091364 166.86972214  114.38776137 99.06370586 9.60298684 145.20097494 193.58850247  7.66161419 9.33189839 81.87341643]  detect\_face. qq2: [ 73.04115878 26.75895455 95.05980623 76.44844487 68.44324344  37.16240776 77.0639618 47.65260458 20.02278726 8.22866344  194.03283771 62.96395516 198.66458131 37.83534414 69.27266497  10.87295669 121.21833924 215.54067644 214.11236168 53.53974405  32.90065031 108.19243371 84.27265424 144.06990027 112.67418198  219.26737678 95.36436498 112.72416335 58.52625021 -5.74640143  79.88895692 145.13494666 58.36525306 151.55526888 47.25398051  199.92712324 215.23753557 36.53757429]  detect\_face. qq3: [187.1811588 234.95647639 181.52921692 167.89867386 206.94245225  245.05859208 183.7093325 47.82613575 247.275777 247.29648209  25.56005493 30.2658359 19.89148752 40.62276709 167.33507735  238.45393218 166.84433836 26.49843293 32.94619739 35.00312126  224.84123116 204.71480953 139.21457304 167.29712293 245.09387544  37.5413748 198.32079168 240.90607993 30.58280104 206.73392072  202.59571221 126.1541486 29.73973672 186.94166425 241.79286942  24.21550851 34.82329747 108.0205029 ]  detect\_face. qq4: [189.02462755 118.3314006 179.46083653 179.11256173 211.75195718  121.1974729 168.9030734 98.39550354 218.19346686 170.92963839  221.49722612 86.13486553 221.27542695 96.23476803 168.79630995  136.75801331 178.92299112 245.20567423 252.78806125 90.92438228  99.9966322 202.11720181 150.44250047 189.83395755 143.94390923  240.71620779 212.72341275 135.49275118 86.86945164 42.8068732  186.74335726 178.24341986 82.41201255 204.66231269 104.11796702  221.68296871 246.19452703 68.9351486 ]  prams. img\_center [125. 125.]  prams. det [[ 79.63303483 49.66967341 190.00140876 195.39264247]  [154.71957363 28.03833024 244.80713892 142.13806857]]  prams. offsets [[ 9.81722179 74.76335627]  [ -2.46884206 -39.91180059]]  prams. bounding\_box\_size [16083.20713894 10278.96762654]  Total number of images: 1 |

#### 理解generateBoundingBox

输入的参数含义:

1. Imap是cls人脸概率
2. Pnet网络输出人脸框和人脸概率

|  |
| --- |
| **pnet\_fun = lambda img : sess.run(('pnet/conv4-2/BiasAdd:0', 'pnet/prob1:0'), feed\_dict={'pnet/input:0':img})** |

1. Out0是人脸框, out1是人脸概率.

|  |
| --- |
| out **=** pnet**(**img\_y**)**  out0 **=** np**.**transpose**(**out**[**0**],** **(**0**,**2**,**1**,**3**))**  out1 **=** np**.**transpose**(**out**[**1**],** **(**0**,**2**,**1**,**3**))**  boxes**,** \_ **=** generateBoundingBox**(**out1**[**0**,:,:,**1**].**copy**(),** out0**[**0**,:,:,:].**copy**(),** scale**,** threshold**[**0**])** |

1. Reg是人脸回归框(只有在是人脸的情况下才有意义).

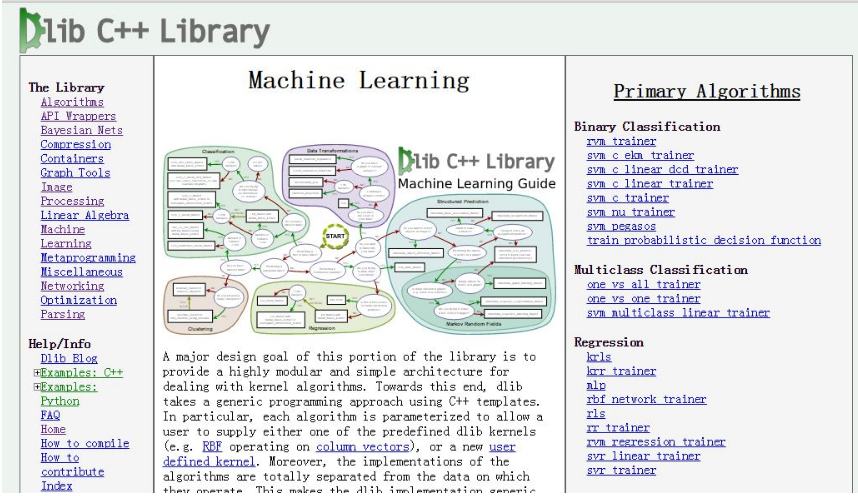
|  |
| --- |
| **def** generateBoundingBox**(**imap**,** reg**,** scale**,** t**):**  # use heatmap to generate bounding boxes  stride**=**2  cellsize**=**12  imap **=** np**.**transpose**(**imap**) ## 只有是人脸的情况下才有意义(人脸概率大于阈值).**  dx1 **=** np**.**transpose**(**reg**[:,:,**0**])**  dy1 **=** np**.**transpose**(**reg**[:,:,**1**])**  dx2 **=** np**.**transpose**(**reg**[:,:,**2**])**  dy2 **=** np**.**transpose**(**reg**[:,:,**3**])**  y**,** x **=** np**.**where**(**imap **>=** t**)**  **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 |

# 七 facenet的再理解

## 7.1 dlib face detector using HOG

### 7.1.1 何为dlib

Dlib是一个机器学习的C++库，包含了许多机器学习常用的算法。而且文档和例子都非常详细，以后准备深入学习。



Dlib的人脸检测(face detecton)使用的传统HOG算子去做的.

### 7.1.2 何为Hog,以及hog和人脸检测的关系

CVPR2005这篇文章中，HOG是用来做行人检测的。作者研究了行人检测的特征集问题，局部归一化的HOG描述子相比于现存的特征集(包括小波)有更好的表现。相比于边缘方向直方图(Edge Orientation Histograms)、SIFT描述子、形状上下文(Shape Contexts)，HOG是在网格密集的大小统一的细胞单元(dense grid of uniformly spaced cells)上进行计算，而且为了提高性能，还采用了重叠的局部对比度归一化(overlapping local contrast normalizations)。作者用行人检测(行人是大部分可见的并且基本上是直立的)进行测试，为了保证速度和简洁性，使用线性SVM作为分类器。

#### 7.1.3 dlib的face detector

依据<http://dlib.net/face_landmark_detection_ex.cpp.html>

这个是dlib上对face关键点的预测.

输出是关键点和矩形框.

有如下特点:

1. 利用histogram of Oriented Gradients(HOG)提取特征.
2. 使用到线性分类器(linear classification)做分类.
3. 使用图像金字塔(image pyramid)
4. 使用Sliding window
5. 姿态评估(pose estimation)依据如下的paper在dlib上实现的.

|  |
| --- |
| *One Millisecond Face Alignment with an Ensemble of Regression Trees by Vahid Kazemi and Josephine Sullivan, CVPR 2014* |

1. 模型是基于***iBUG 300-W face landmark dataset***做训练
2. 有pretrained模型, *shape\_predictor\_68\_face\_landmarks.dat*
3. 可以训练自己的模型.

|  |
| --- |
| http://dlib.net/train\_shape\_predictor\_ex.cpp.html |

使用的代码段

|  |
| --- |
| int **main**(int argc, char\*\* argv)  **{**  try  **{**  // This example takes in a shape model file and then a list of images to  // process. We will take these filenames in as command line arguments.  // Dlib comes with example images in the examples/faces folder so give  // those as arguments to this program.  if (argc == 1) **// 需要一个预训练模型,可用提示网址下载.**  **{**  cout << "Call this program like this:" << endl;  cout << "./face\_landmark\_detection\_ex shape\_predictor\_68\_face\_landmarks.dat faces/\*.jpg" << endl;  cout << "\nYou can get the shape\_predictor\_68\_face\_landmarks.dat file from:\n";  cout << "http://dlib.net/files/shape\_predictor\_68\_face\_landmarks.dat.bz2" << endl;  return 0;  **}**  // We need a face detector. We will use this to get bounding boxes for  // each face in an image.  frontal\_face\_detector detector = get\_frontal\_face\_detector(); **// 生成bbox**  // And we also need a shape\_predictor. This is the tool that will predict face  // landmark positions given an image and face bounding box. Here we are just  // loading the model from the shape\_predictor\_68\_face\_landmarks.dat file you gave  // as a command line argument.  shape\_predictor sp; **// 加载预训练模型**  deserialize(argv[1]) >> sp;  image\_window win, win\_faces;  // Loop over all the images provided on the command line.  for (int i = 2; i < argc; ++i)  **{**  cout << "processing image " << argv[i] << endl;  array2d<rgb\_pixel> img;  load\_image(img, argv[i]);  // Make the image larger so we can detect small faces.  pyramid\_up(img); **// 生成图像金字塔**  // Now tell the face detector to give us a list of bounding boxes  // around all the faces in the image.  std::vector<rectangle> dets = detector(img);**// 根据金字塔的图片,产生一系列的bbox. (调用模型做推断).**  cout << "Number of faces detected: " << dets.size() << endl;  // Now we will go ask the shape\_predictor to tell us the pose of  // each face we detected.  std::vector<full\_object\_detection> shapes;  for (unsigned long j = 0; j < dets.size(); ++j)  **{**  full\_object\_detection shape = sp(img, dets[j]);  cout << "number of parts: "<< shape.num\_parts() << endl;  cout << "pixel position of first part: " << shape.part(0) << endl;  cout << "pixel position of second part: " << shape.part(1) << endl;  // You get the idea, you can get all the face part locations if  // you want them. Here we just store them in shapes so we can  // put them on the screen.  shapes.push\_back(shape); **// 提取需要的”shape”信息**  **}**  // Now let's view our face poses on the screen.  win.clear\_overlay();  win.set\_image(img);  win.add\_overlay(render\_face\_detections(shapes));  // We can also extract copies of each face that are cropped, rotated upright,  // and scaled to a standard size as shown here:  dlib::array<array2d<rgb\_pixel> > face\_chips;  extract\_image\_chips(img, get\_face\_chip\_details(shapes), face\_chips);  win\_faces.set\_image(tile\_images(face\_chips));  cout << "Hit enter to process the next image..." << endl;  cin.get();  **}**  **}**  catch (exception& e)  **{**  cout << "\nexception thrown!" << endl;  cout << e.what() << endl;  **}**  **}** |

## 7.2 facenet使用的dlib align方法

|  |
| --- |
| **class** **AlignDlib:**  """  **##**   1. **使用上面分析的基于hog的face landmark detector方法.(HOG提特征,线性分类器,图像金字塔,sliding window.** 2. **输出”关心”的landmark(eyes,nose)以及bbox.**   Use `dlib's landmark estimation <**http://blog.dlib.net/2014/08/real-time-face-pose-estimation.html**>`\_ to align faces.  The alignment preprocess faces for input into a neural network.  Faces are resized to the same size (such as 96x96) and transformed to make landmarks (such as the eyes and nose) appear at the same location on every image.  Normalized landmarks:  .. image:: ../images/dlib-landmark-mean.png  """  #: Landmark indices corresponding to the inner eyes and bottom lip.  INNER\_EYES\_AND\_BOTTOM\_LIP **=** **[**39**,** 42**,** 57**]**  #: Landmark indices corresponding to the outer eyes and nose.  OUTER\_EYES\_AND\_NOSE **=** **[**36**,** 45**,** 33**]**  **def** \_\_init\_\_**(**self**,** facePredictor**):**  """  Instantiate an 'AlignDlib' object.  :param facePredictor: The path to dlib's  :type facePredictor: str  """  **assert** facePredictor **is** **not** **None**  #pylint: disable=no-member  [**## dlib对face landmark推断的实现.**](#_7.1.3_dlib的face_detector)  self**.**detector **=** **dlib.get\_frontal\_face\_detector()**  self**.**predictor **=** **dlib.shape\_predictor(facePredictor)** |

#### 7.2.1 align

|  |
| --- |
| #pylint: disable=dangerous-default-value  **def** align**(**self**,** imgDim**,** rgbImg**,** bb**=None,**  landmarks**=None,** landmarkIndices**=**INNER\_EYES\_AND\_BOTTOM\_LIP**,**  skipMulti**=False,** scale**=**1.0**):**  r"""align(imgDim, rgbImg, bb=None, landmarks=None, landmarkIndices=INNER\_EYES\_AND\_BOTTOM\_LIP)  Transform and align a face in an image.  :param imgDim: The edge length in pixels of the square the image is resized to. **## 输入图(是被reszie成方形的),它的边长(h,w值).**  :type imgDim: int  :param rgbImg: RGB image to process. Shape: (height, width, 3)  :type rgbImg: numpy.ndarray  :param bb: Bounding box around the face to align. \  Defaults to the largest face.  :type bb: dlib.rectangle  :param landmarks: Detected landmark locations. \  Landmarks found on `bb` if not provided.  :type landmarks: list of (x,y) tuples  :param landmarkIndices: The indices to transform to.  :type landmarkIndices: list of ints  :param skipMulti: Skip image if more than one face detected.  :type skipMulti: bool  :param scale: Scale image before cropping to the size given by imgDim.  :type scale: float  :return: The aligned RGB image. Shape: (imgDim, imgDim, 3)  :rtype: numpy.ndarray  """  **assert** imgDim **is** **not** **None**  **assert** rgbImg **is** **not** **None**  **assert** landmarkIndices **is** **not** **None**  ## 推断出给定rgbimg的bbox和landmark点.  **if** bb **is** **None:**  bb **=** [self**.**getLargestFaceBoundingBox](#_何为getLargestFaceBoundingBox)**(**rgbImg**,** skipMulti**)**  **if** bb **is** **None:**  **return**  **if** landmarks **is** **None:**  landmarks **=** self**.**findLandmarks**(**rgbImg**,** bb**)**  npLandmarks **=** np**.**float32**(**landmarks**)**  npLandmarkIndices **=** np**.**array**(**landmarkIndices**)**  ## 利用**2D 仿射变换**获得缩略图.  #pylint: disable=maybe-no-member  H **=** [**cv2.getAffineTransform**](#_何为cv2.getAffineTransform以及cv2.warpA)**(**npLandmarks**[**npLandmarkIndices**],**  imgDim **\*** MINMAX\_TEMPLATE**[**npLandmarkIndices**]\***scale **+** imgDim**\*(**1**-**scale**)/**2**)**  thumbnail **=** [**cv2.warpAffine**](#_何为cv2.getAffineTransform以及cv2.warpA)**(**rgbImg**,** H**,** **(**imgDim**,** imgDim**))**    **return** thumbnail |

##### 何为getLargestFaceBoundingBox

1. 从多人脸框中找到最大的人脸框, 公式是人脸框的面积.

**lambda rect: rect.width() \* rect.height()**

1. Self.detctor做的人脸框推断.

[其模型是前面说的get\_frontal\_face\_detector.](#_7.1.3_dlib的face_detector)

|  |
| --- |
| **def** getAllFaceBoundingBoxes**(**self**,** rgbImg**):**  """  Find all face bounding boxes in an image.  :param rgbImg: RGB image to process. Shape: (height, width, 3)  :type rgbImg: numpy.ndarray  :return: All face bounding boxes in an image.  :rtype: dlib.rectangles  """  **assert** rgbImg **is** **not** **None**  **try:**  **return** **self.detector(rgbImg, 1)**  **except** Exception **as** e**:** #pylint: disable=broad-except  **print(**"Warning: {}"**.**format**(**e**))**  # In rare cases, exceptions are thrown.  **return** **[]**  **def** getLargestFaceBoundingBox**(**self**,** rgbImg**,** skipMulti**=False):**  """  Find the largest face bounding box in an image.  :param rgbImg: RGB image to process. Shape: (height, width, 3)  :type rgbImg: numpy.ndarray  :param skipMulti: Skip image if more than one face detected.  :type skipMulti: bool  :return: The largest face bounding box in an image, or None.  :rtype: dlib.rectangle  """  **assert** rgbImg **is** **not** **None**  faces **=** self**.**getAllFaceBoundingBoxes**(**rgbImg**)**  **if** **(not** skipMulti **and** len**(**faces**)** **>** 0**)** **or** len**(**faces**)** **==** 1**:**  **return** max**(**faces**,** **key=lambda rect: rect.width() \* rect.height())**  **else:**  **return** **None** |

##### 何为cv2.getAffineTransform以及cv2.warpAffine

图像的旋转加上拉升就是图像仿射变换，仿射变化也是需要一个M矩阵就可以，但是由于仿射变换比较复杂，一般直接找很难找到这个矩阵，opencv提供了根据变换前后三个点的对应关系来自动求解M。这个函数是

**M=cv2.getAffineTransform(pos1,pos2)**

其中两个位置就是变换前后的对应位置关系。输出的就是仿射矩阵M。然后在使用函数**cv2.warpAffine()**。形象化的图如下（引用参考的）



## 7.3 facenet Loss

### 7.3.1 facenet训练骨架

Facenet loss有两种, center loss和triplet loss. 使用center loss训练时,采用train\_softmax.py

|  |
| --- |
| python src/train\_softmax.py \  --logs\_base\_dir ~/logs/facenet/ \  --models\_base\_dir ~/models/facenet/ \  --data\_dir ~/datasets/casia/casia\_maxpy\_mtcnnpy\_182 \  --image\_size 160 \  --model\_def models.inception\_resnet\_v1 \  --lfw\_dir ~/datasets/lfw/lfw\_mtcnnpy\_160 \  --optimizer RMSPROP \  --learning\_rate -1 \  --max\_nrof\_epochs 80 \  --keep\_probability 0.8 \  --random\_crop --random\_flip \  --learning\_rate\_schedule\_file  data/learning\_rate\_schedule\_classifier\_casia.txt \  --weight\_decay 5e-5 \  --center\_loss\_factor 1e-2 \  --center\_loss\_alfa 0.9 |

当使用triplet loss训练facenet时,使用train\_triplet.py.

这两者(train\_softmax.py和train\_triplet.py)都含有相似的代码结构. 是facenet train的骨架部分.

引入facenet.py, facenet.py会提供train时候的op操作以及对应的loss的计算方法.是具体网络的实现部分.

### 7.3.2 center loss的训练

在src/train\_softmax.py中.

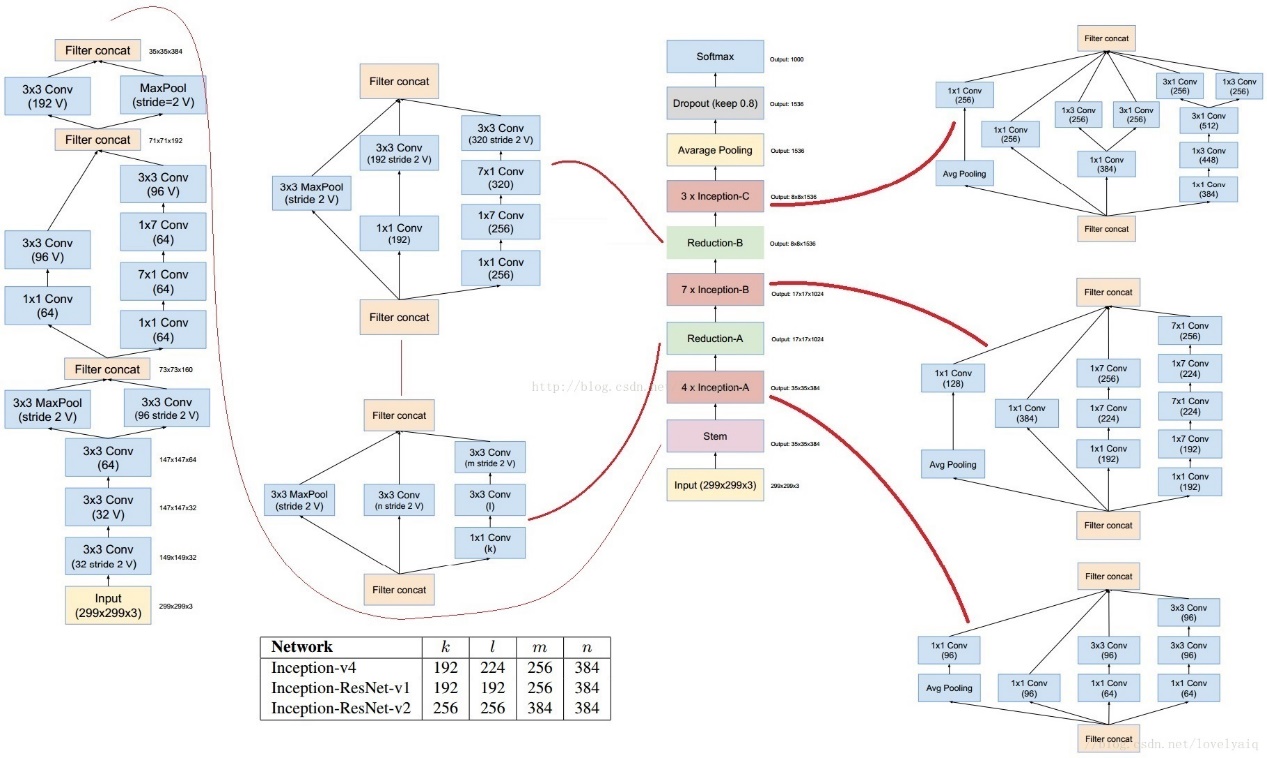
|  |
| --- |
| **def** main**(**args**):**  ##   1. 使用inception\_resnet\_v1作为back bone   **network = importlib.import\_module(args.model\_def)**  subdir **=** datetime**.**strftime**(**datetime**.**now**(),** '%Y%m%d-%H%M%S'**)**  log\_dir **=** os**.**path**.**join**(**os**.**path**.**expanduser**(**args**.**logs\_base\_dir**),** subdir**)**  **if** **not** os**.**path**.**isdir**(**log\_dir**):** # Create the log directory if it doesn't exist  os**.**makedirs**(**log\_dir**)**  model\_dir **=** os**.**path**.**join**(**os**.**path**.**expanduser**(**args**.**models\_base\_dir**),** subdir**)**  **if** **not** os**.**path**.**isdir**(**model\_dir**):** # Create the model directory if it doesn't exist  os**.**makedirs**(**model\_dir**)**  # Store some git revision info in a text file in the log directory  src\_path**,** \_ **=** os**.**path**.**split**(**os**.**path**.**realpath**(**\_\_file\_\_**))**  facenet**.**store\_revision\_info**(**src\_path**,** log\_dir**,** ' '**.**join**(**sys**.**argv**))**  np**.**random**.**seed**(**seed**=**args**.**seed**)**  random**.**seed**(**args**.**seed**)**  **train\_set = facenet.get\_dataset(args.data\_dir) ## 数据集**  **if** args**.**filter\_filename**:**  train\_set **=** filter\_dataset**(**train\_set**,** os**.**path**.**expanduser**(**args**.**filter\_filename**),**  args**.**filter\_percentile**,** args**.**filter\_min\_nrof\_images\_per\_class**)**  nrof\_classes **=** len**(**train\_set**)**  **print(**'Model directory: %s' **%** model\_dir**)**  **print(**'Log directory: %s' **%** log\_dir**)**  pretrained\_model **=** **None**  **if** args**.**pretrained\_model**:**  pretrained\_model **=** os**.**path**.**expanduser**(**args**.**pretrained\_model**)**  **print(**'Pre-trained model: %s' **%** pretrained\_model**)**  **if** **args.lfw\_dir: ## 在train的过程中会evaluation,eval是在lfw上做的**  **print(**'LFW directory: %s' **%** args**.**lfw\_dir**)**  # Read the file containing the pairs used for testing  pairs **=** lfw**.**read\_pairs**(**os**.**path**.**expanduser**(**args**.**lfw\_pairs**))**  # Get the paths for the corresponding images  lfw\_paths**,** actual\_issame **=** lfw**.**get\_paths**(**os**.**path**.**expanduser**(**args**.**lfw\_dir**),** pairs**,** args**.**lfw\_file\_ext**)**  **with** tf**.**Graph**().**as\_default**():**  tf**.**set\_random\_seed**(**args**.**seed**)**  **global\_step** **=** tf**.**Variable**(**0**,** trainable**=False)**  # Get a list of image paths and their labels  image\_list**,** label\_list **=** facenet**.**get\_image\_paths\_and\_labels**(**train\_set**)**  **assert** len**(**image\_list**)** **>** 0**,** 'The dataset should not be empty'  # Create a queue that produces indices into the image\_list and label\_list  labels **=** ops**.**convert\_to\_tensor**(**label\_list**,** dtype**=**tf**.**int32**)**  range\_size **=** array\_ops**.**shape**(**labels**)[**0**]**  index\_queue **=** tf**.**train**.**range\_input\_producer**(**range\_size**,** num\_epochs**=None,**  shuffle**=True,** seed**=None,** capacity**=**32**)**  index\_dequeue\_op **=** index\_queue**.**dequeue\_many**(**args**.**batch\_size **\*** args**.**epoch\_size**,** 'index\_dequeue'**)**  learning\_rate\_placeholder **=** tf**.**placeholder**(**tf**.**float32**,** name**=**'learning\_rate'**)**  batch\_size\_placeholder **=** tf**.**placeholder**(**tf**.**int32**,** name**=**'batch\_size'**)**  phase\_train\_placeholder **=** tf**.**placeholder**(**tf**.**bool**,** name**=**'phase\_train'**)**  image\_paths\_placeholder **=** tf**.**placeholder**(**tf**.**string**,** shape**=(None,** 1**),** name**=**'image\_paths'**)**  **labels\_placeholder =** tf**.**placeholder**(**tf**.**int64**,** shape**=(None,** 1**),** name**=**'labels'**)**  input\_queue **=** data\_flow\_ops**.**FIFOQueue**(**capacity**=**100000**,**  dtypes**=[**tf**.**string**,** tf**.**int64**],**  shapes**=[(**1**,),** **(**1**,)],**  shared\_name**=None,** name**=None)**  enqueue\_op **=** input\_queue**.**enqueue\_many**([**image\_paths\_placeholder**,** labels\_placeholder**],** name**=**'enqueue\_op'**)**  nrof\_preprocess\_threads **=** 4  **images\_and\_labels** **=** **[] ## 是个list,存img和label对.**  **for** \_ **in** range**(**nrof\_preprocess\_threads**):**  filenames**,** label **=** input\_queue**.**dequeue**()**  images **=** **[]**  **for** filename **in** tf**.**unstack**(**filenames**):**  file\_contents **=** tf**.**read\_file**(**filename**)**  **image = tf.image.decode\_png(file\_contents)**  **if** args**.**random\_rotate**:**  image **=** tf**.**py\_func**(**facenet**.**random\_rotate\_image**,** **[**image**],** tf**.**uint8**)**  **if** args**.**random\_crop**:**  image **=** tf**.**random\_crop**(**image**,** **[**args**.**image\_size**,** args**.**image\_size**,** 3**])**  **else:**  image **=** tf**.**image**.**resize\_image\_with\_crop\_or\_pad**(**image**,** args**.**image\_size**,** args**.**image\_size**)**  **if** args**.**random\_flip**:**  image **=** tf**.**image**.**random\_flip\_left\_right**(**image**)**  # pylint: disable=no-member  image**.**set\_shape**((**args**.**image\_size**,** args**.**image\_size**,** 3**))**  images**.**append**(**tf**.**image**.**per\_image\_standardization**(**image**))**  images\_and\_labels**.**append**([**images**,** label**])**  **## 拿到batch img和label**  **image\_batch, label\_batch** **=** tf**.**train**.**batch\_join**(**  images\_and\_labels**,** batch\_size**=**batch\_size\_placeholder**,**  shapes**=[(**args**.**image\_size**,** args**.**image\_size**,** 3**),** **()],** enqueue\_many**=True,**  capacity**=**4 **\*** nrof\_preprocess\_threads **\*** args**.**batch\_size**,**  allow\_smaller\_final\_batch**=True)**  image\_batch **=** tf**.**identity**(**image\_batch**,** 'image\_batch'**)**  image\_batch **=** tf**.**identity**(**image\_batch**,** 'input'**)**  label\_batch **=** tf**.**identity**(**label\_batch**,** 'label\_batch'**)**  **print(**'Total number of classes: %d' **%** nrof\_classes**)**  **print(**'Total number of examples: %d' **%** len**(**image\_list**))**  **print(**'Building training graph'**)**  # Build the inference graph  **## 是inception resnet v1 网络推断**  prelogits**,** \_ **=** **network.inference(**image\_batch**,** args**.**keep\_probability**,**  phase\_train**=**phase\_train\_placeholder**,** bottleneck\_layer\_size**=**args**.**embedding\_size**,**  weight\_decay**=**args**.**weight\_decay**)**  logits **=** **slim.fully\_connected(**prelogits**,** len**(**train\_set**),** activation\_fn**=None,**  weights\_initializer**=**tf**.**truncated\_normal\_initializer**(**stddev**=**0.1**),**  weights\_regularizer**=**slim**.**l2\_regularizer**(**args**.**weight\_decay**),**  scope**=**'Logits'**,** reuse**=False)**  **## 生成用以描述人脸的embeddings(类似向量).**  **embeddings = tf.nn.l2\_normalize(prelogits, 1, 1e-10, name='embeddings')**  # Add center loss  **if** args**.**center\_loss\_factor **>** 0.0**:**  prelogits\_center\_loss**,** \_ **= facenet.center\_loss(**prelogits**,** label\_batch**,** args**.**center\_loss\_alfa**,** nrof\_classes**)**  tf**.**add\_to\_collection**(tf.GraphKeys.REGULARIZATION\_LOSSES,** prelogits\_center\_loss **\*** args**.**center\_loss\_factor**)**  learning\_rate **=** tf**.**train**.**exponential\_decay**(**learning\_rate\_placeholder**,** global\_step**,**  args**.**learning\_rate\_decay\_epochs **\*** args**.**epoch\_size**,** args**.**learning\_rate\_decay\_factor**,** staircase**=True)**  tf**.**summary**.**scalar**(**'learning\_rate'**,** learning\_rate**)**  # Calculate the average cross entropy loss across the batch  **cross\_entropy = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(**  labels**=**label\_batch**,** logits**=**logits**,** name**=**'cross\_entropy\_per\_example'**)**  cross\_entropy\_mean **=** tf**.**reduce\_mean**(**cross\_entropy**,** name**=**'cross\_entropy'**)**  tf**.**add\_to\_collection**(**'losses'**,** cross\_entropy\_mean**)**  # Calculate the total losses  **## total loss**   1. **Cross entry 的loss. 是预测值logits和label\_batch的loss** 2. **正则化loss** 3. **正则化loss包含有center loss.**   regularization\_losses **=** tf**.**get\_collection**(**tf**.**GraphKeys**.**REGULARIZATION\_LOSSES**)**  **total\_loss = tf.add\_n([cross\_entropy\_mean] + regularization\_losses, name='total\_loss')**  # Build a Graph that trains the model with one batch of examples and updates the model parameters  **## 传入loss, 训练的目标操作**  **train\_op = facenet.train(**total\_loss**,** global\_step**,** args**.**optimizer**,**  learning\_rate**,** args**.**moving\_average\_decay**,** tf**.**global\_variables**(),** args**.**log\_histograms**)**  # Create a saver  saver **=** tf**.**train**.**Saver**(**tf**.**trainable\_variables**(),** max\_to\_keep**=**3**)**  # Build the summary operation based on the TF collection of Summaries.  summary\_op **=** tf**.**summary**.**merge\_all**()**  # Start running operations on the Graph.  gpu\_options **=** tf**.**GPUOptions**(**per\_process\_gpu\_memory\_fraction**=**args**.**gpu\_memory\_fraction**)**  sess **=** tf**.**Session**(**config**=**tf**.**ConfigProto**(**gpu\_options**=**gpu\_options**,** log\_device\_placement**=False))**  sess**.**run**(**tf**.**global\_variables\_initializer**())**  sess**.**run**(**tf**.**local\_variables\_initializer**())**  summary\_writer **=** tf**.**summary**.**FileWriter**(**log\_dir**,** sess**.**graph**)**  coord **=** tf**.**train**.**Coordinator**()**  tf**.**train**.**start\_queue\_runners**(**coord**=**coord**,** sess**=**sess**)**  **with** sess**.**as\_default**():**  **if** pretrained\_model**:**  **print(**'Restoring pretrained model: %s' **%** pretrained\_model**)**  saver**.**restore**(**sess**,** pretrained\_model**)**  # Training and validation loop  **print(**'Running training'**)**  epoch **=** 0  **while** epoch **<** args**.**max\_nrof\_epochs**:**  step **=** sess**.**run**(**global\_step**,** feed\_dict**=None)**  epoch **=** step **//** args**.**epoch\_size  # Train for one epoch  train**(**args**,** sess**,** epoch**,** image\_list**,** label\_list**,** index\_dequeue\_op**,** enqueue\_op**,** image\_paths\_placeholder**,** labels\_placeholder**,**  learning\_rate\_placeholder**,** phase\_train\_placeholder**,** batch\_size\_placeholder**,** global\_step**,**  total\_loss**,** train\_op**,** summary\_op**,** summary\_writer**,** regularization\_losses**,** args**.**learning\_rate\_schedule\_file**)**  # Save variables and the metagraph if it doesn't exist already  save\_variables\_and\_metagraph**(**sess**,** saver**,** summary\_writer**,** model\_dir**,** subdir**,** step**)**  # Evaluate on LFW  **if args.lfw\_dir: ## 在LFW上做eval**  evaluate**(**sess**,** enqueue\_op**,** image\_paths\_placeholder**,** labels\_placeholder**,** phase\_train\_placeholder**,** batch\_size\_placeholder**,**  embeddings**,** label\_batch**,** lfw\_paths**,** actual\_issame**,** args**.**lfw\_batch\_size**,** args**.**lfw\_nrof\_folds**,** log\_dir**,** step**,** summary\_writer**)**  sess**.**close**()**  **return** model\_dir |

#### 何为inception\_resnet\_v1

##### 网络结构上理解

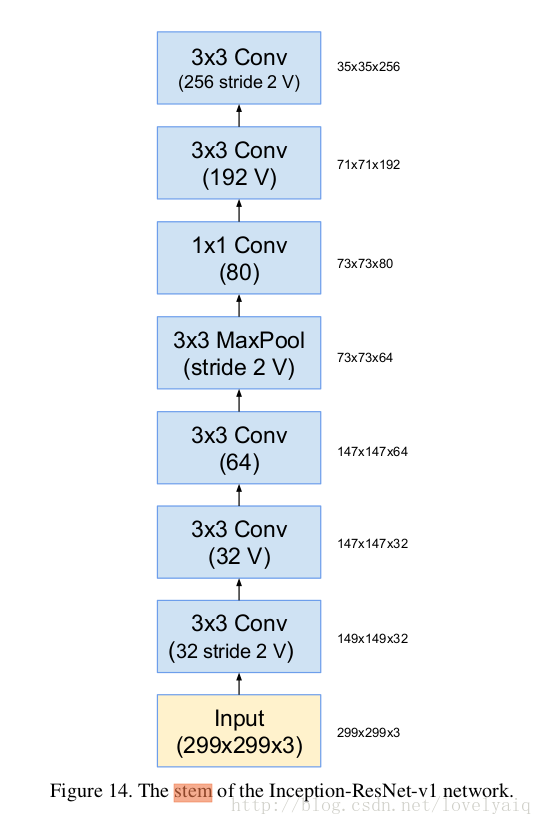
可以从下面几条理解:

1. 主结构是串行的convs. 以图像进入,以softmax结束.
2. 含有类似结构: inceptionA,B,C, reduction-A,B.

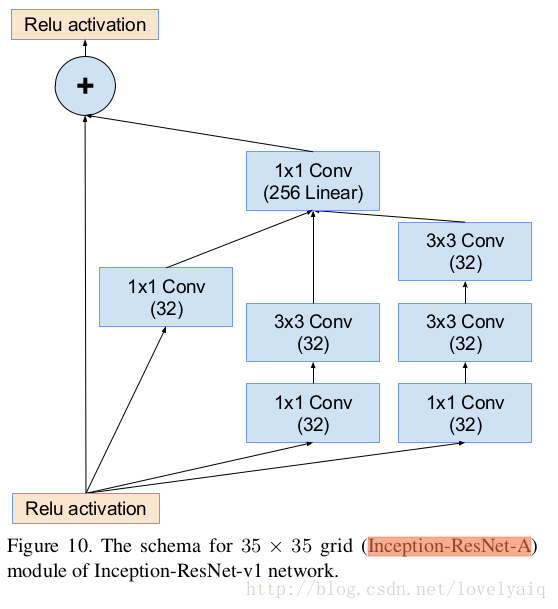


**Stem结构**

起到一个升维效果(作用).由卷积组成.



**InceptionA**



1. 采用”多尺度”(多个filter size的conv对同一个输入做卷积).
2. 多尺度输出在chn上concat.
3. 在3x3的卷积前加入1x1的卷积,其作用是为了降维.
4. 1\*1的卷积的作用:
   1. 由于是1x1的,所有没有考虑输入局部信息间的关系.
   2. 降维作用
   3. 在输入上(前一层)上面加入的非线性.提高网络表达能力.

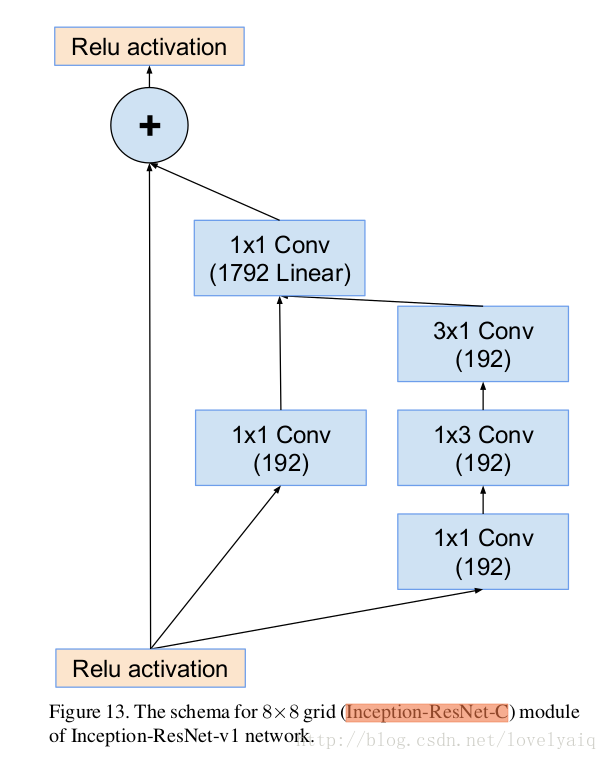
**Inception-B**



理解inceptionB:

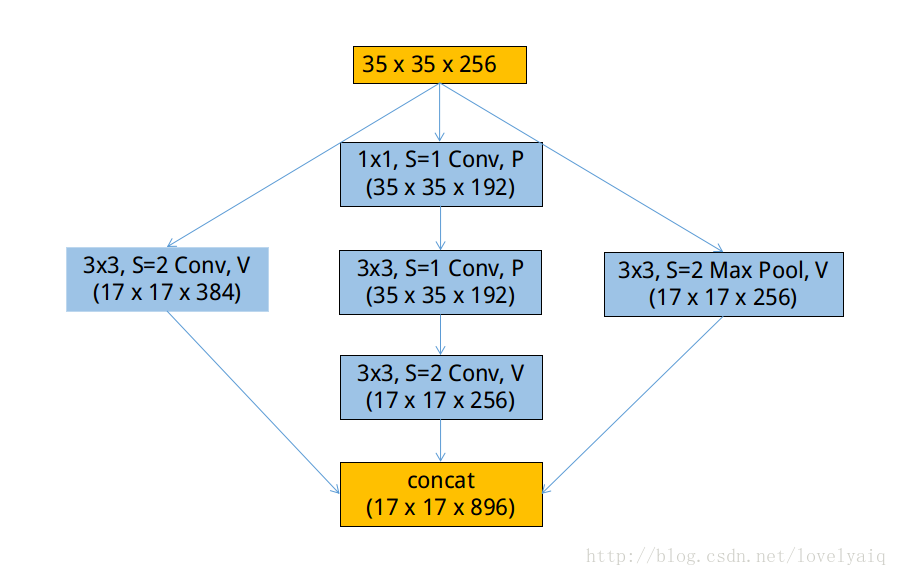
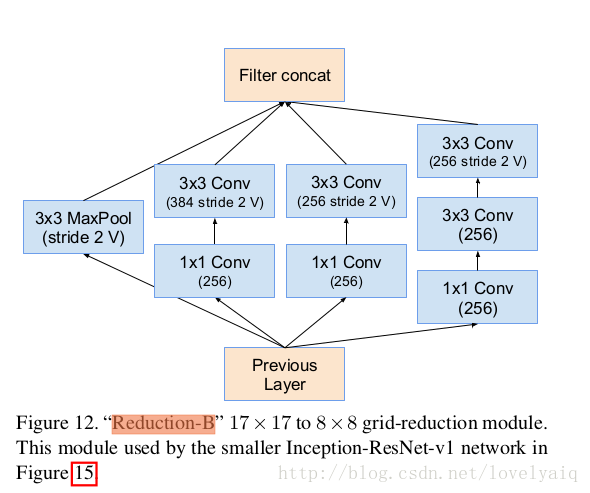
1. 多尺度,在chn上concat.
2. 1x1卷积降维.
3. 1x7和7x1的非对称卷积,减少计算量和参数量
   1. 达到7x7的卷积效果,但是参数量和计算量(乘法)都减小很多.
   2. 实验上看,1xn和nx1在n<20的时候表现较好.

**InceptionC**



1. 多尺度.
2. 1x3和3x1的非对称卷积.
3. 1x1卷积升维.

**ReductionA和ReductionB**

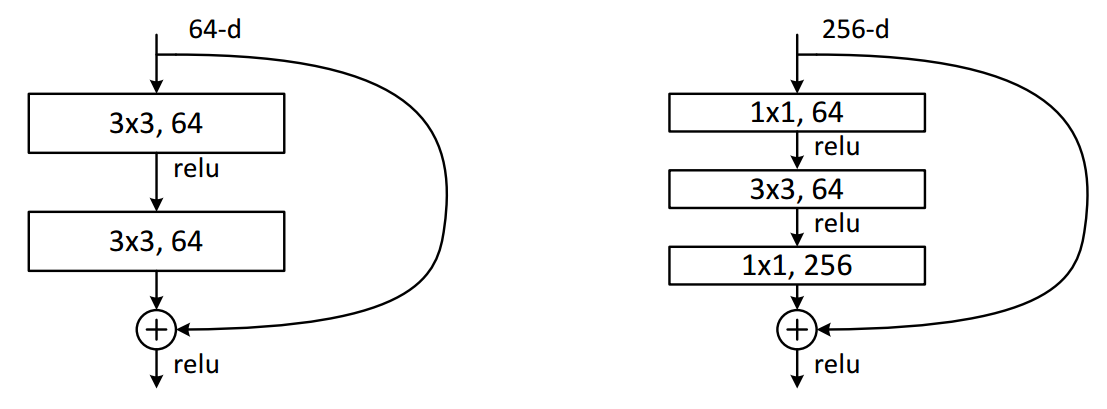
1. 左边是reductionA,右边是reductionB
2. 都采用多尺度.

##### 代码上理解

|  |
| --- |
| **def** inception\_resnet\_v1**(**inputs**,** is\_training**=True,**  dropout\_keep\_prob**=**0.8**,**  bottleneck\_layer\_size**=**128**,**  reuse**=None,**  scope**=**'InceptionResnetV1'**):**  """Creates the Inception Resnet V1 model.  Args:  inputs: a 4-D tensor of size [batch\_size, height, width, 3].  num\_classes: number of predicted classes.  is\_training: whether is training or not.  dropout\_keep\_prob: float, the fraction to keep before final layer.  reuse: whether or not the network and its variables should be reused. To be  able to reuse 'scope' must be given.  scope: Optional variable\_scope.  Returns:  **logits**: the logits outputs of the model.  **end\_points**: the set of end\_points from the inception model.  """  end\_points **=** **{}**    **with** tf**.**variable\_scope**(**scope**,** 'InceptionResnetV1'**,** **[**inputs**],** reuse**=**reuse**):**  **with** slim**.**arg\_scope**([**slim**.**batch\_norm**,** slim**.**dropout**],**  is\_training**=**is\_training**):**  **with** slim**.**arg\_scope**([**slim**.**conv2d**,** slim**.**max\_pool2d**,** slim**.**avg\_pool2d**],**  stride**=**1**,** padding**=**'SAME'**):**    # 149 x 149 x 32  net **=** slim**.**conv2d**(**inputs**,** 32**,** 3**,** stride**=**2**,** padding**=**'VALID'**,**  scope**=**'Conv2d\_1a\_3x3'**)**  end\_points**[**'Conv2d\_1a\_3x3'**]** **=** net  # 147 x 147 x 32  net **=** slim**.**conv2d**(**net**,** 32**,** 3**,** padding**=**'VALID'**,**  scope**=**'Conv2d\_2a\_3x3'**)**  end\_points**[**'Conv2d\_2a\_3x3'**]** **=** net  # 147 x 147 x 64  net **=** slim**.**conv2d**(**net**,** 64**,** 3**,** scope**=**'Conv2d\_2b\_3x3'**)**  end\_points**[**'Conv2d\_2b\_3x3'**]** **=** net  # 73 x 73 x 64  net **=** slim**.**max\_pool2d**(**net**,** 3**,** stride**=**2**,** padding**=**'VALID'**,**  scope**=**'MaxPool\_3a\_3x3'**)**  end\_points**[**'MaxPool\_3a\_3x3'**]** **=** net  # 73 x 73 x 80  net **=** slim**.**conv2d**(**net**,** 80**,** 1**,** padding**=**'VALID'**,**  scope**=**'Conv2d\_3b\_1x1'**)**  end\_points**[**'Conv2d\_3b\_1x1'**]** **=** net  # 71 x 71 x 192  net **=** slim**.**conv2d**(**net**,** 192**,** 3**,** padding**=**'VALID'**,**  scope**=**'Conv2d\_4a\_3x3'**)**  end\_points**[**'Conv2d\_4a\_3x3'**]** **=** net  # 35 x 35 x 256  net **=** slim**.**conv2d**(**net**,** 256**,** 3**,** stride**=**2**,** padding**=**'VALID'**,**  scope**=**'Conv2d\_4b\_3x3'**)**  end\_points**[**'Conv2d\_4b\_3x3'**]** **=** net    # 5 x Inception-resnet-A  net **=** slim**.**repeat**(**net**,** 5**,** block35**,** scale**=**0.17**)**    # Reduction-A  **with** tf**.**variable\_scope**(**'Mixed\_6a'**):**  net **=** reduction\_a**(**net**,** 192**,** 192**,** 256**,** 384**)**  end\_points**[**'Mixed\_6a'**]** **=** net    # 10 x Inception-Resnet-B  net **=** slim**.**repeat**(**net**,** 10**,** block17**,** scale**=**0.10**)**    # Reduction-B  **with** tf**.**variable\_scope**(**'Mixed\_7a'**):**  net **=** reduction\_b**(**net**)**  end\_points**[**'Mixed\_7a'**]** **=** net    # 5 x Inception-Resnet-C  net **=** slim**.**repeat**(**net**,** 5**,** block8**,** scale**=**0.20**)**  net **=** block8**(**net**,** activation\_fn**=None)**    **with** tf**.**variable\_scope**(**'Logits'**):**  end\_points**[**'PrePool'**]** **=** net  #pylint: disable=no-member  net **=** slim**.**avg\_pool2d**(**net**,** net**.**get\_shape**()[**1**:**3**],** padding**=**'VALID'**,**  scope**=**'AvgPool\_1a\_8x8'**)**  net **=** slim**.**flatten**(**net**)**    net **=** slim**.**dropout**(**net**,** dropout\_keep\_prob**,** is\_training**=**is\_training**,**  scope**=**'Dropout'**)**    end\_points**[**'PreLogitsFlatten'**]** **=** net    **net = slim.fully\_connected(net, bottleneck\_layer\_size, activation\_fn=None,**  **scope='Bottleneck', reuse=False)**    **return** net**,** end\_points |

1. 基本上是前面结构的代码实现.
2. 最后的输出含有一个bottleneck 层(layer).

##### Understanding bottleneck



Deeper Bottleneck Architectures(间层DBA).为何会引入这个DBA层呢?

|  |
| --- |
| Because of concerns on the training time that we can afford, we modify the building block as a bottleneck design. |

其含义是,考虑到GPU计算能力,为了加速训练,采用了bottleneck来降低计算和参数量.

ResNet的bottleneck是指使用1x1卷积，主要是为了**通过降通道数量，来降卷积的参数和计算量**。这中间会有信息损失，但影响不大，因为毕竟是负责残差的计算。而且，增加的深度也是一个弥补。

对比Inception V3的bottleneck问题.

Inceptionv3主要是针对池化操作来说，因为池化的信息损失太多了。所以需要做卷积分支来补充信息。(两者本质上并不一样。)

#### 何为embadding

在src/train\_softmax.py中.

|  |
| --- |
| **def** main**(**args**):**  …  embeddings **=** tf**.**nn**.**l2\_normalize**(prelogits,** 1**,** 1e-10**,** name**=**'embeddings'**)** |

采用tf.nn.l2\_normalize.

1. 依据的是inception\_resnet\_v1的输出(bottleneck输出).
2. 做归一化的维度是dim=1.

tf.nn.l2\_normalize(x, dim, epsilon=1e-12, name=None)

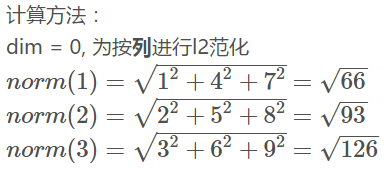
上式：

x为输入的向量；

dim为l2范化的维数，dim取值为0或0或1；

epsilon的范化的最小值边界；

|  |
| --- |
| **import** tensorflow **as** tf  input\_data **=** tf**.**constant**([[**1.0**,**2**,**3**],[**4.0**,**5**,**6**],[**7.0**,**8**,**9**]])**  output **=** tf**.**nn**.**l2\_normalize**(**input\_data**,** dim **=** 0**)**  **with** tf**.**Session**()** **as** sess**:**  **print** sess**.**run**(**input\_data**)**  **print** sess**.**run**(**output**)** |

1. 先计算dim0上(一行的norm值).
2. 然后对dim0的每个元素做归一化.

|  |
| --- |
| [[1./norm(1), 2./norm(2) , 3./norm(3) ]  [4./norm(1) , 5./norm(2) , 6./norm(3) ] =  [7./norm(1) , 8./norm(2) , 9./norm(3) ]]  [[0.12309149 0.20739034 0.26726127]  [0.49236596 0.51847583 0.53452253]  [0.86164045 0.82956135 0.80178374]] |

实验结果是:

|  |
| --- |
| [[ 1. 2. 3.]  [ 4. 5. 6.]  [ 7. 8. 9.]]  [[ 0.12309149 0.20739034 0.26726127]  [ 0.49236596 0.51847583 0.53452253]  [ 0.86164045 0.82956135 0.80178374]] |

**Embeding作用:**

1. One hot编码中会有大量稀疏成员.
2. 在train过程中,经embeding后的(生成)的向量,都会得到更新.
3. 图片(人脸框)---->embedding---->vector
4. 用以计算center Loss, 描述(用于计算)人脸间的间隔.
5. 当facenet应用时候,会使用人脸向量做识别,分类.

### 7.3.3 understanding center Loss

在src/facenet.py中实现(生成) center loss.

代码段如下:

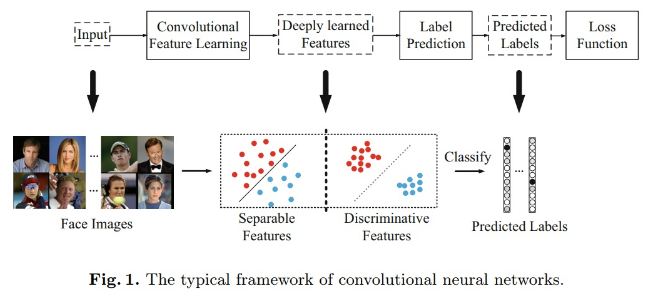
|  |
| --- |
| **def** center\_loss**(**features**,** label**,** alfa**,** nrof\_classes**):**  """Center loss based on the paper "A Discriminative Feature Learning Approach for Deep Face Recognition"  (http://ydwen.github.io/papers/WenECCV16.pdf)  """  nrof\_features **=** features**.**get\_shape**()[**1**]**  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**)**  loss **=** tf**.**reduce\_mean**(**tf**.**square**(**features **-** centers\_batch**))**  **return** loss**,** centers |

#### 何为center Loss?

要达成下图的discriminative Loss的效果. 将类间距离更大.(用以辅助softmax loss).

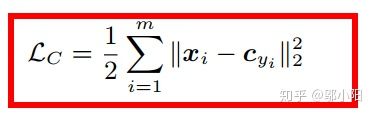
首先, 利用softmax loss分类.

然后,利用center loss对同类压缩(压缩效果).



#### Center loss再理解.

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.

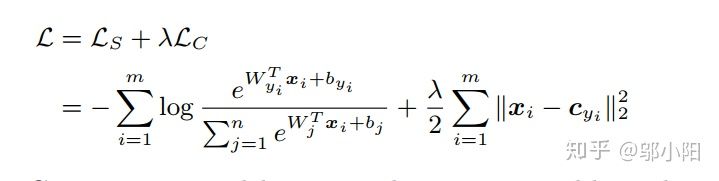


1. 距离LC求导如下:



* 1. 某个类别j的中心Cj只依赖于该类的特征计算得来.

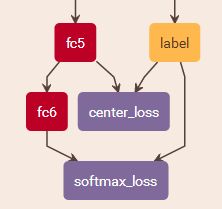
1. 训练总体loss是softmax Loss和center Loss之和.



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.

#### Center Loss 代码实现

|  |
| --- |
| **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**)**  loss **=** tf**.**reduce\_mean**(**tf**.**square**(**features **-** centers\_batch**))**  **return** loss**,** centers |

##### 实验

|  |
| --- |
| python src/train\_softmax.py --logs\_base\_dir logs/facenet/ --models\_base\_dir models/facenet/ --data\_dir CASIA-WebFace/exp0\_182 --image\_size 160 --model\_def models.inception\_resnet\_v1 --lfw\_dir lfw/lfw\_mtcnnpy\_160 --optimizer RMSPROP --learning\_rate -1 --max\_nrof\_epochs 80 --keep\_probability 0.8 --random\_crop --random\_flip --learning\_rate\_schedule\_file data/learning\_rate\_schedule\_classifier\_casia.txt --weight\_decay 5e-5 --center\_loss\_factor 1e-2 --center\_loss\_alfa 0.9 |

代码段:

|  |
| --- |
| **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)  """  zeros\_tsr **=** tf**.**zeros**([**2**,** 3**])**  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)**  **centers\_1 = centers**  label **=** tf**.**reshape**(**label**,** **[-**1**])**  centers\_batch **=** **tf.gather(**centers**,** label**)**  diff **=** **(**1 **-** alfa**)** **\*** **(**centers\_batch **-** features**)**  centers **=** tf**.**scatter\_sub**(**centers**,** label**,** diff**)**  **centers\_2 = centers**  loss **=** tf**.**reduce\_mean**(**tf**.**square**(**features **-** centers\_batch**))**  **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 |

理解tf.gather

|  |
| --- |
| # Scalar indices (output is rank(params) - 1).     output[a\_0, ..., a\_n, b\_0, ..., b\_n] =       params[a\_0, ..., a\_n, indices, b\_0, ..., b\_n]      # **Vector indices (output is rank(params)).**     output[a\_0, ..., a\_n, i, b\_0, ..., b\_n] =       params[a\_0, ..., a\_n, indices[i], b\_0, ..., b\_n]      # Higher rank indices (output is rank(params) + rank(indices) - 1).     output[a\_0, ..., a\_n, i, ..., j, b\_0, ... b\_n] =       params[a\_0, ..., a\_n, indices[i, ..., j], b\_0, ..., b\_n] |

##### 实验结果

|  |
| --- |
| Epoch: [0][314/1000] Time 0.314 Loss 4.278 RegLoss 4.278  [features,label,nrof\_classes, centers\_1, centers\_batch, diff, centers\_2, loss, ]**[90 128][90][1][1 128][90 128][90 128][1 128][0.999902427]**  Epoch: [0][315/1000] Time 0.306 Loss 4.277 RegLoss 4.277  [features,label,nrof\_classes, centers\_1, centers\_batch, diff, centers\_2, loss, ][90 128][90][1][1 128][90 128][90 128][1 128][0.999900758]  Features是 **[90 128]**  **Label是[90]** , 应该是90个**(行,90个样本)**, 每个值是0或者1.  Nrof\_classes是1. (是否是人脸)  Centers初始化的shape是 [1 128] 1个分类(样本,因为是Cyi的, 同一类只有一个类内中心,一个中心是128维的), 128个维度(类似2维坐标,这里是128维坐标).  **Centers\_batch是 [90 128]**的. 一个batch有90个样本, 每个样本属于某类的特征图(几个样本要是同一类的,可以使用同一类的类内中心).  Diff是[ 90 128] (每个样本所属类的类内中心梯度下降的迭代算子.)  preview  Centers\_2是[1 128] 每个样本所属类的类内中心的梯度迭代后的值.  Loss是求和. |

#### 何为label?

##### 从数据集角度

输入的图是mtcnn 识别出的矩形框,然后resize到特定尺寸的.

Label是mtcnn识别的矩形框.

再次理解datasets

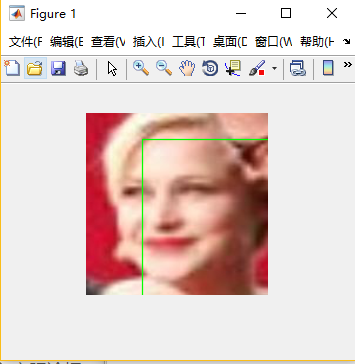
Facenet的dataset的可视化如下图.

Mtcnn的align输出的结果是(含有矩形框):

|  |
| --- |
| CASIA-WebFace/exp0\_182/0000099/052.png 57 27 212 217 |

将这个人脸矩形框在原图上描绘出来是:

052.png是[182 182 3]的尺寸(这个是mtcnn align参数).



在facenet实际使用上.有以下几个约定:

1. 读入的图已经是经mtcnn预处理的人脸图. 如上图所示.
2. 并没有使用mtcnn预处理人脸图的人脸框信息(如上图绿色矩形框).
3. Facenet应该是默认整张图(182x182)都是人脸.
4. Facenet重点去做人脸基础上的识别
5. 所谓识别,就是把相同人脸的向量距离缩短(类内距离缩小).
6. 不同人脸的”向量距离”拉长(类间距离扩大).

##### 从代码角度

在train\_softmax.py中

|  |
| --- |
| ***python train\_softmax.py …. --data\_dir CASIA-WebFace/exp0\_182***  train\_set **=** facenet**.**get\_dataset**(**args**.**data\_dir**)**  # Get a list of image paths and their labels  image\_list**,** label\_list **=** **facenet.get\_image\_paths\_and\_labels(**train\_set**)**  **assert** len**(**image\_list**)** **>** 0**,** 'The dataset should not be empty'  # Create a queue that produces indices into the image\_list and label\_list  labels **=** **ops.convert\_to\_tensor(**label\_list**,** dtype**=**tf**.**int32**)** |

###### 何为get\_image\_paths\_and\_labels

|  |
| --- |
| **def** get\_image\_paths\_and\_labels**(**dataset**):**  image\_paths\_flat **=** **[]**  labels\_flat **=** **[]**  **for** i **in** range**(**len**(**dataset**)):**  image\_paths\_flat **+=** dataset**[**i**].**image\_paths  labels\_flat **+=** **[i]** **\*** len**(**dataset**[**i**].**image\_paths**)**  **## label[i]是, 第i个成员, 设置值为i, 个数是同人脸的图片个数.**  **return** image\_paths\_flat**,** labels\_flat |

实验:

|  |
| --- |
| python src/train\_softmax.py --logs\_base\_dir logs/facenet/ --models\_base\_dir models/facenet/ --data\_dir CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/ --image\_size 160 --model\_def models.inception\_resnet\_v1 --lfw\_dir lfw/lfw\_mtcnnpy\_160 --optimizer RMSPROP --learning\_rate -1 --max\_nrof\_epochs 80 --keep\_probability 0.8 --random\_crop --random\_flip --learning\_rate\_schedule\_file data/learning\_rate\_schedule\_classifier\_casia.txt --weight\_decay 5e-5 --center\_loss\_factor 1e-2 --center\_loss\_alfa 0.9 > dataset.log |

实验结果:

|  |
| --- |
| 实验结果也证明下面的分析(猜想部分).   1. 第0组(0000045)里边的人脸框是属于同一个人的.共有15个人脸框,生成的label的值是1, 共有15个label. 2. 第1组,含有256个人脸框(属于同一个人的人脸),label大小是256,label值是2. 3. 第2组,含有69个人脸框, label大小是69,值是3. 4. 这样确保 5. 同人脸的label相同 6. 不同人脸的label不同. 7. 人脸目标数训练时候是确定的, 但是实际使用的时候,会有新的人脸.   Facenet训练的目的不是说为了能够识别出具体的人脸的分类(区别于voc等20分类的具体的分类)  Facenet训练的目的是训练出一个embedding层.  一个将人脸映射(压缩,抽象)成embedding向量的函数f(x).  这个embedding层会对每个人脸框都”归纳”成一个向量表示.  当做人脸识别的时候, 将一组向量(一组人脸的表示)”聚类”, 之间的距离小于某个阈值可认为是同一张人脸.  get\_image\_paths\_and\_labels.dataset[0].image\_paths:['CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/006.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/014.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/009.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/002.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/008.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/012.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/013.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/015.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/011.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/003.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/007.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/010.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/005.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/004.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000045/001.png']  get\_image\_paths\_and\_labels.0-th'lens:15 labels:[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  …  get\_image\_paths\_and\_labels.1-th'lens:256 labels:[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]  …  get\_image\_paths\_and\_labels.dataset[2].image\_paths:['CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/006.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/070.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/038.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/063.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/040.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/064.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/047.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/028.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/014.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/009.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/002.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/062.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/067.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/051.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/069.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/008.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/050.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/012.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/043.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/013.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/058.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/049.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/031.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/061.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/015.png','CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/033.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/024.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/048.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/035.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/016.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/021.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/011.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/027.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/057.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/054.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/042.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/034.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/032.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/055.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/045.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/003.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/022.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/068.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/018.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/029.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/056.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/065.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/007.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/059.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/010.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/053.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/017.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/025.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/020.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/046.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/026.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/005.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/039.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/066.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/041.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/071.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/004.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/001.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/052.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/019.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/023.png','CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/060.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/044.png', 'CASIA-WebFace/casia\_maxpy\_mtcnnpy\_182/0000100/030.png']get\_image\_paths\_and\_labels.2-th'lens:69 labels:[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,2, 2, 2] |

猜想:

应该是一组int值的list.

[0][0]…num\_face#0..

[1][1]……num\_face#1 (比如90个,等).

[2][2]….

当计算到cross entropy时候. 会去向 [0]….[m] 计算Loss.并让Loss减小.

当Loss减小时,会让embedding层输出”正相关”的分离.

如下的代码印证了,前面的约束.

|  |
| --- |
| **class** **ImageClass():**  "Stores the paths to images for a given class"  **def** \_\_init\_\_**(**self**,** name**,** image\_paths**):**  self**.**name **=** name  self**.**image\_paths **=** image\_paths    **def** \_\_str\_\_**(**self**):**  **return** self**.**name **+** ', ' **+** str**(**len**(**self**.**image\_paths**))** **+** ' images'    **def** \_\_len\_\_**(**self**):**  **return** len**(**self**.**image\_paths**)**  **def** get\_dataset**(**paths**):**  dataset **=** **[]**  **for** path **in** paths**.**split**(**':'**):**  path\_exp **=** os**.**path**.**expanduser**(**path**)**  classes **=** **os.listdir(path\_exp) ## 从*CASIA-WebFace/exp0\_182*中列出当前目录的文件名(同一个人脸的矩形框图片).**  classes**.**sort**() ## 排序这些文件(同一个人脸的矩形图片).**  nrof\_classes **=** len**(**classes**)**  **for** i **in** range**(**nrof\_classes**): ## 对于每一个人脸框,生成对应的image\_paths.**  class\_name **=** classes**[**i**]**  facedir **=** os**.**path**.**join**(**path\_exp**,** class\_name**)**  **if** os**.**path**.**isdir**(**facedir**):**  images **=** os**.**listdir**(**facedir**)**  image\_paths **=** **[**os**.**path**.**join**(**facedir**,**img**)** **for** img **in** images**]**  dataset**.**append**(**ImageClass**(**class\_name**,** image\_paths**))**    **return** dataset |

### 7.3.4 cross entropy Loss

理解点:

1. Label是[0],[0],..[1]…[m]等值.表示第几个人脸(相同值是同一人脸.
2. 预测值是bottleneck.输出是len(train\_set)维度特征.

|  |
| --- |
| # Calculate the average cross entropy loss across the batch  cross\_entropy **=** **tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(**  labels**=**label\_batch**,** logits**=**logits**,** name**=**'cross\_entropy\_per\_example'**)**  cross\_entropy\_mean **=** tf**.**reduce\_mean**(**cross\_entropy**,** name**=**'cross\_entropy'**)**  tf**.**add\_to\_collection**(**'losses'**,** cross\_entropy\_mean**)** |

预测值的代码段:

|  |
| --- |
| # Build the inference graph  prelogits**,** \_ **=** **network.inference(**image\_batch**,** args**.**keep\_probability**,** phase\_train**=**phase\_train\_placeholder**,** bottleneck\_layer\_size**=**args**.**embedding\_size**,** weight\_decay**=**args**.**weight\_decay**) ## 做网络推断, 此时prelogits是128维,center loss也是使用的这个.**  logits **=** slim**.**fully\_connected**(**prelogits**,** len**(**train\_set**),** activation\_fn**=None,** weights\_initializer**=**tf**.**truncated\_normal\_initializer**(**stddev**=**0.1**),** weights\_regularizer**=**slim**.**l2\_regularizer**(**args**.**weight\_decay**),** scope**=**'Logits'**,** reuse**=False) ## cross entropy的预测值需要在prelogits基础上接一个输出是len(train\_set)的fc层.**  https://pic4.zhimg.com/80/v2-b2f979b42da7e6862c655cfb1f7162a6_hd.jpg |

## 7.4 理解embadding和loss关系

Facenet中使用loss是一个综合.包括:

1. 同人脸的Cross entropy loss
2. 辅助的center loss,用以压缩类内距离
3. 正则化loss

Embedding输出是做人脸识别的”向量”依据.

当loss收敛时, embedding表现也会好.之间有依赖关系.(loss的小,能够使得embedding表现很好.)

理论上:

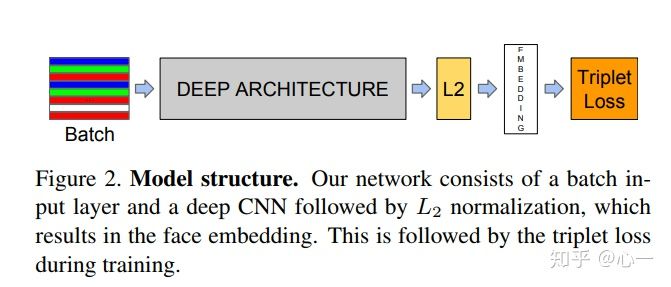
1. 可能需要一个label描述embadding
2. 用网络推断逼近embadding,使之更小.

实际上:

1. 采用inception输出的特征,经embadding层转成人脸向量.(该向量是人脸识别应用的依据)
2. Inception输出层,经一系列网络层,做cross entropy和center loss.

7.4.1 embedding在理解

所谓的内嵌(embedding)，可以理解成一种映射关系，就是把原有的特征空间转换为新的特征空间。



## 7.5 facenet的训练目的

人脸目标数训练时候是确定的, 但是实际使用的时候,会有新的人脸.

Facenet训练的目的不是说为了能够识别出具体的人脸的分类(区别于voc等20分类的具体的分类)

Facenet训练的目的是训练出一个embedding层.一个将人脸映射(压缩,抽象)成embedding向量的函数f(x).这个embedding层会对每个人脸框都”归纳”成一个向量表示.当做人脸识别的时候, 将一组向量(一组人脸的表示)”聚类”, 之间的距离小于某个阈值可认为是同一张人脸.

## 7.6 facenet的应用

### 7.6.1 在lfw上eval

|  |
| --- |
| **def** evaluate**(**sess**,** enqueue\_op**,** image\_paths\_placeholder**,** labels\_placeholder**,** phase\_train\_placeholder**,** batch\_size\_placeholder**,**  embeddings**,** labels**,** image\_paths**,** actual\_issame**,** batch\_size**,** nrof\_folds**,** log\_dir**,** step**,** summary\_writer**):**  start\_time **=** time**.**time**()**  # Run forward pass to calculate embeddings  **print(**'Runnning forward pass on LFW images'**)**  # Enqueue one epoch of image paths and labels  labels\_array **=** np**.**expand\_dims**(**np**.**arange**(**0**,** len**(**image\_paths**)),** 1**)**  image\_paths\_array **=** np**.**expand\_dims**(**np**.**array**(**image\_paths**),** 1**)**  **## 先读取lfw数据.**   1. **同时会把lfw的数据填充到facenet的节点上.** 2. **图像输入节点image\_paths\_array** 3. **图像输入label节点 labels\_placeholder.**   **sess.run(**enqueue\_op**,** **{**image\_paths\_placeholder**:** image\_paths\_array**,** labels\_placeholder**:** labels\_array**})**  embedding\_size **=** embeddings**.**get\_shape**()[**1**]**  nrof\_images **=** len**(actual\_issame)** **\*** 2 **## issame是list, 一个tile是2 张图, 所以实际的图片个数需要乘以2.**  **assert** nrof\_images **%** batch\_size **==** 0**,** 'The number of LFW images must be an integer multiple of the LFW batch size'  nrof\_batches **=** nrof\_images **//** batch\_size  ## 生成emb array如下.   1. Img行 2. 128向量维度     emb\_array **=** np**.**zeros**((**nrof\_images**,** embedding\_size**))**  lab\_array **=** np**.**zeros**((**nrof\_images**,))**  **for** \_ **in** range**(**nrof\_batches**):**  feed\_dict **=** **{**phase\_train\_placeholder**:** **False,** batch\_size\_placeholder**:** batch\_size**}**  emb**,** lab **=** **sess.run([embeddings, labels], feed\_dict=feed\_dict) ## 利用facenet推断出emb值. 此处的label只是为了做一个assert检查.并不会作用到结果上.**  lab\_array**[**lab**]** **=** lab  emb\_array**[**lab**]** **=** emb  **assert** np**.**array\_equal**(**lab\_array**,** np**.**arange**(**nrof\_images**))** **==** **True,** 'Wrong labels used for evaluation, possibly caused by training examples left in the input pipeline'  \_**,** \_**,** accuracy**,** val**,** val\_std**,** far **=** [**lfw.evaluate**](#_何为lfw.evaluate)**(**emb\_array**,** actual\_issame**,** nrof\_folds**=**nrof\_folds**)**  **print(**'Accuracy: %1.3f+-%1.3f' **%** **(**np**.**mean**(**accuracy**),** np**.**std**(**accuracy**)))**  **print(**'Validation rate: %2.5f+-%2.5f @ FAR=%2.5f' **%** **(**val**,** val\_std**,** far**))**  lfw\_time **=** time**.**time**()** **-** start\_time  # Add validation loss and accuracy to summary  summary **=** tf**.**Summary**()**  # pylint: disable=maybe-no-member  summary**.**value**.**add**(**tag**=**'lfw/accuracy'**,** simple\_value**=**np**.**mean**(**accuracy**))**  summary**.**value**.**add**(**tag**=**'lfw/val\_rate'**,** simple\_value**=**val**)**  summary**.**value**.**add**(**tag**=**'time/lfw'**,** simple\_value**=**lfw\_time**)**  summary\_writer**.**add\_summary**(**summary**,** step**)**  **with** open**(**os**.**path**.**join**(**log\_dir**,** 'lfw\_result.txt'**),** 'at'**)** **as** f**:**  f**.**write**(**'%d\t%.5f\t%.5f\n' **%** **(**step**,** np**.**mean**(**accuracy**),** val**))** |

#### 何为lfw data pairs

在data/pairs.txt中

|  |
| --- |
| 10 300  Abel\_Pacheco 1 4  Akhmed\_Zakayev 1 3  Akhmed\_Zakayev 2 3  Amber\_Tamblyn 1 2  Anders\_Fogh\_Rasmussen 1 3  Anders\_Fogh\_Rasmussen 1 4  Angela\_Bassett 1 5  Angela\_Bassett 2 5  Angela\_Bassett 3 4  Ann\_Veneman 3 5  Ann\_Veneman 6 10  Ann\_Veneman 10 11  Anthony\_Fauci 1 2  Antony\_Leung 1 2  Antony\_Leung 2 3  Anwar\_Ibrahim 1 2  Augusto\_Pinochet 1 2  Barbara\_Brezigar 1 2  Benjamin\_Netanyahu 1 4  Benjamin\_Netanyahu 4 5  Bernard\_Law 2 3  Bernard\_Law 3 4  Bertrand\_Bonello 1 2  …  cott\_Wolf 2 Troy\_Polamalu 1  Sergei\_Alexandrovitch\_Ordzhonikidze 1 Yolanda\_King 1  Shane\_Loux 1 Val\_Ackerman 1  Shawn\_Marion 1 Shirley\_Jones 1  Slobodan\_Milosevic 2 Sok\_An 1 |

解析代码段:

Src/lfw.py中

1. 从第1行开始(忽略”10 300 “那行(第0行))
2. 一行中有多组参数,需要组成多组参数对.
3. 一行中人名后接多组参数的解析成一对.是属于不匹配的样本.

|  |
| --- |
| **def** read\_pairs**(**pairs\_filename**):**  pairs **=** **[]**  **with** open**(**pairs\_filename**,** 'r'**)** **as** f**:**  **for** line **in** f**.**readlines**()[**1**:]:**  pair **=** line**.**strip**().**split**()**  pairs**.**append**(**pair**)**  **return** np**.**array**(**pairs**)** |

解析的pairs如下:

|  |
| --- |
| pair: ['Zhu\_Rongji', '1', '3']  pair: ['Zhu\_Rongji', '2', '8']  …  pair: ['Scott\_Wolf', '2', 'Troy\_Polamalu', '1']  pair: ['Sergei\_Alexandrovitch\_Ordzhonikidze', '1', 'Yolanda\_King', '1']  pair: ['Shane\_Loux', '1', 'Val\_Ackerman', '1']  pair: ['Shawn\_Marion', '1', 'Shirley\_Jones', '1']  pair: ['Slobodan\_Milosevic', '2', 'Sok\_An', '1'] |

Pairs如何使用

|  |
| --- |
| **def** get\_paths**(**lfw\_dir**,** pairs**,** file\_ext**):**  nrof\_skipped\_pairs **=** 0  path\_list **=** **[]**  issame\_list **=** **[]**  **for** pair **in** pairs**:**  **if** len**(**pair**)** **==** 3**: ## 对于一个人名后接2个参数的.(这两个参数属于同一个人名),属于人脸匹配的样本.**  path0 **=** os**.**path**.**join**(**lfw\_dir**,** pair**[**0**],** pair**[**0**]** **+** '\_' **+** '%04d' **%** int**(**pair**[**1**])+**'.'**+**file\_ext**)**  path1 **=** os**.**path**.**join**(**lfw\_dir**,** pair**[**0**],** pair**[**0**]** **+** '\_' **+** '%04d' **%** int**(**pair**[**2**])+**'.'**+**file\_ext**)**  issame **=** **True**  **elif** len**(**pair**)** **==** 4**: ## 对于4组参数的, 是两组人脸. 它们属于不匹配的样本 issame标记为False. 用意后面计算accuracy等指标参数.**  path0 **=** os**.**path**.**join**(**lfw\_dir**,** pair**[**0**],** pair**[**0**]** **+** '\_' **+** '%04d' **%** int**(**pair**[**1**])+**'.'**+**file\_ext**)**  path1 **=** os**.**path**.**join**(**lfw\_dir**,** pair**[**2**],** pair**[**2**]** **+** '\_' **+** '%04d' **%** int**(**pair**[**3**])+**'.'**+**file\_ext**)**  issame **=** **False**  **if** os**.**path**.**exists**(**path0**)** **and** os**.**path**.**exists**(**path1**):** # Only add the pair if both paths exist  path\_list **+=** **(**path0**,**path1**)**  issame\_list**.**append**(**issame**)**  **else:**  nrof\_skipped\_pairs **+=** 1  **if** nrof\_skipped\_pairs**>**0**:**  **print(**'Skipped %d image pairs' **%** nrof\_skipped\_pairs**)**    **return** path\_list**,** issame\_list |

#### 何为lfw.evaluate

评估lfw

1. embedding是[img 128]的.
2. 我们知道lfw的issame\_list是按照pairs组织的.
3. 这里对于imgs个图片(issame\_list长度的2倍)
4. 利用embedding, 计算出绘制Roc曲线需要TP, FP
5. 利用embedding, 计算far.

|  |
| --- |
| **def** evaluate**(**embeddings**,** actual\_issame**,** nrof\_folds**=**10**):**  # Calculate evaluation metrics  thresholds **=** np**.**arange**(**0**,** 4**,** 0.01**)**  embeddings1 **=** embeddings**[**0**::**2**] ## 偶数行取出来**  embeddings2 **=** embeddings**[**1**::**2**] ## 奇数行取出来.**  tpr**,** fpr**,** accuracy **=** [**facenet.calculate\_roc**](#_何为facenet_calculate_roc)**(**thresholds**,** embeddings1**,** embeddings2**,**  np**.**asarray**(**actual\_issame**),** nrof\_folds**=**nrof\_folds**)**  thresholds **=** np**.**arange**(**0**,** 4**,** 0.001**)**  val**,** val\_std**,** far **=** [**facenet.calculate\_val**](#_何为facenet.calculate_val)**(**thresholds**,** embeddings1**,** embeddings2**,**  np**.**asarray**(**actual\_issame**),** 1e-3**,** nrof\_folds**=**nrof\_folds**)**  **return** tpr**,** fpr**,** accuracy**,** val**,** val\_std**,** far |

#### 何为facenet calculate\_roc

|  |
| --- |
| **def** calculate\_roc**(**thresholds**,** embeddings1**,** embeddings2**,** actual\_issame**,** nrof\_folds**=**10**):**  **assert(**embeddings1**.**shape**[**0**]** **==** embeddings2**.**shape**[**0**])**  **assert(**embeddings1**.**shape**[**1**]** **==** embeddings2**.**shape**[**1**])**  nrof\_pairs **=** min**(**len**(**actual\_issame**),** embeddings1**.**shape**[**0**])**  nrof\_thresholds **=** len**(**thresholds**)**  k\_fold **=** KFold**(**n\_splits**=**nrof\_folds**,** shuffle**=False)**  **## 引入k折网格搜索交叉验证.**   1. **K折找到acc最高的thrd.** 2. **每一对(pairs)都计算128维的方差(某种程度的距离).** 3. **计算不同阈值下的tp,fp.** 4. **Tp,Fp就可以做roc曲线了.**     tprs **=** np**.**zeros**((**nrof\_folds**,**nrof\_thresholds**))**  fprs **=** np**.**zeros**((**nrof\_folds**,**nrof\_thresholds**))**  accuracy **=** np**.**zeros**((**nrof\_folds**))**    diff **=** np**.**subtract**(**embeddings1**,** embeddings2**)**  dist **=** np**.**sum**(**np**.**square**(**diff**),**1**) ## dist**  indices **=** np**.**arange**(**nrof\_pairs**)**    **for** fold\_idx**,** **(**train\_set**,** test\_set**)** **in** enumerate**(**k\_fold**.**split**(**indices**)):**    # Find the best threshold for the fold  acc\_train **=** np**.**zeros**((**nrof\_thresholds**))**  **for** threshold\_idx**,** threshold **in** enumerate**(**thresholds**):**  \_**,** \_**,** acc\_train**[**threshold\_idx**]** **=** calculate\_accuracy**(**threshold**,** dist**[**train\_set**],** actual\_issame**[**train\_set**])**  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**])**    tpr **=** np**.**mean**(**tprs**,**0**)**  fpr **=** np**.**mean**(**fprs**,**0**)**  **return** tpr**,** fpr**,** accuracy |

#### Understanding actual\_issame

1. actural\_issame是lfw中same的照片对
   1. 一组验证样本包含2个照片, 相同的照片和不同的照片.
   2. 相同的照片, 会标记该验证样本的”label”为same**(issame为True)**
   3. 不同的照片, issame标记为False.
2. 每个样本是照片对(pairs). 对于每一个样本.
   1. 某次样本i的照片对(A和B) 都会经facenet计算出一组embedding输出(embedding1,和embedding2)
   2. Embedding输出可以理解成两张照片在某个维度上的表现.
   3. 怎么描述这两张照片的相似程度呢?
      1. 可以计算两张照片在embedding维空间的距离.
      2. 然后引入一个阈值T, 如果照片对之间的距离小于该阈值T,则可以认为两张照片属于同一人.在predict\_issame中标记为True.

#### 计算TPR,FPR.

|  |
| --- |
| **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**)))##负判正**  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  **考虑下负判正的情况**   1. **np.logical\_and是逻辑与** 2. **np.logical\_not是取反** 3. **下图左侧蓝框是结果.右上是真值,可见正判正是1个.** 4. **当计算FP(负判正)时,需要对真值取反,得到一个”负判正”的”期望”结果.**   **然后,让预测值和取反真值做逻辑与,就找到”负判正”那个样本了.** |

#### 何为facenet.calculate\_val

|  |
| --- |
| **def** calculate\_val**(**thresholds**,** embeddings1**,** embeddings2**,** actual\_issame**,** far\_target**,** nrof\_folds**=**10**):**  **assert(**embeddings1**.**shape**[**0**]** **==** embeddings2**.**shape**[**0**])**  **assert(**embeddings1**.**shape**[**1**]** **==** embeddings2**.**shape**[**1**])**  nrof\_pairs **=** min**(**len**(**actual\_issame**),** embeddings1**.**shape**[**0**])**  nrof\_thresholds **=** len**(**thresholds**)**  k\_fold **=** KFold**(**n\_splits**=**nrof\_folds**,** shuffle**=False)**    val **=** np**.**zeros**(**nrof\_folds**)**  far **=** np**.**zeros**(**nrof\_folds**)**    diff **=** np**.**subtract**(**embeddings1**,** embeddings2**)**  dist **=** np**.**sum**(**np**.**square**(**diff**),**1**)**  indices **=** np**.**arange**(**nrof\_pairs**)**    **for** fold\_idx**,** **(**train\_set**,** test\_set**)** **in** enumerate**(**k\_fold**.**split**(**indices**)):**    # Find the threshold that gives FAR = far\_target  far\_train **=** np**.**zeros**(**nrof\_thresholds**)**  **for** threshold\_idx**,** threshold **in** enumerate**(**thresholds**):**  \_**,** far\_train**[**threshold\_idx**]** **=** calculate\_val\_far**(**threshold**,** dist**[**train\_set**],** actual\_issame**[**train\_set**])**  **## 涉及一个插值**   1. **对train set交叉验证,找到所有k折的far值.** 2. **目的是找到一个阈值(threshold)能让这个阈值的far值是期望值(far\_target).** 3. **如果不满足,需要插值出一个threshold来(某种程度的近似).**   **if** np**.**max**(**far\_train**)>=far\_target:**  f **=** interpolate**.**interp1d**(**far\_train**,** thresholds**,** kind**=**'slinear'**)**  threshold **=** f**(**far\_target**)**  **else:**  threshold **=** 0.0  **##　最后在test set上验证**  val**[**fold\_idx**],** far**[**fold\_idx**]** **=** calculate\_val\_far**(**threshold**,** dist**[**test\_set**],** actual\_issame**[**test\_set**])**    val\_mean **=** np**.**mean**(**val**)**  far\_mean **=** np**.**mean**(**far**)**  val\_std **=** np**.**std**(**val**)**  **return** val\_mean**,** val\_std**,** far\_mean |

|  |
| --- |
| **def** calculate\_val\_far**(**threshold**,** dist**,** actual\_issame**):**  predict\_issame **=** np**.**less**(**dist**,** threshold**)**  true\_accept **=** np**.**sum**(**np**.**logical\_and**(**predict\_issame**,** actual\_issame**))**  false\_accept **=** np**.**sum**(**np**.**logical\_and**(**predict\_issame**,** np**.**logical\_not**(**actual\_issame**)))**  n\_same **=** np**.**sum**(**actual\_issame**)**  n\_diff **=** np**.**sum**(**np**.**logical\_not**(**actual\_issame**))**  val **=** float**(**true\_accept**)** **/** float**(**n\_same**)**  far **=** float**(**false\_accept**)** **/** float**(**n\_diff**)**  **return** val**,** far |

### 7.6.2 facenet的compare

在src/compare.py中

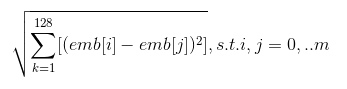
使用

|  |
| --- |
| python src/compare.py \  ~/models/facenet/20170512-110547/ \  ./test\_imgs/1.jpg ./test\_imgs/2.jpg ./test\_imgs/3.jpg |

1. 可以显示三者图片的距离.

代码段:

1. 加载模型
2. Input节点是”input:0”, 把输入(3张人脸图)填充到输入节点(“input:0”)
3. Output节点是”embeddings:0”
4. 运行facenet推断,得到3张人脸的embedding向量(128维) .
5. 距离公式如下:



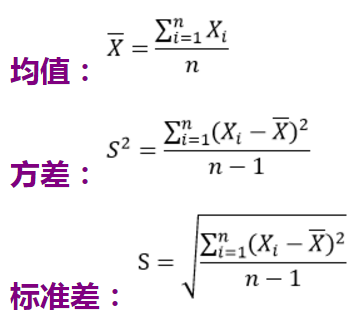
|  |
| --- |
| **def** main**(**args**):**  ##　利用mtcnn在原图上做align.(从原图中crop出人脸来,然后resize).   1. 默认的resize到160x160的. 2. Align最后还会经**facenet.prewhiten**函数做”归一化”   images **=** **load\_and\_align\_data(**args**.**image\_files**,** args**.**image\_size**,** args**.**margin**,** args**.**gpu\_memory\_fraction**)**  **with** tf**.**Graph**().**as\_default**():**  **with** tf**.**Session**()** **as** sess**:**  # Load the model  facenet**.**load\_model**(**args**.**model**)**  # Get input and output tensors  images\_placeholder **=** tf**.**get\_default\_graph**().**get\_tensor\_by\_name**(**"input:0"**)**  embeddings **=** tf**.**get\_default\_graph**().**get\_tensor\_by\_name**(**"embeddings:0"**)**  phase\_train\_placeholder **=** tf**.**get\_default\_graph**().**get\_tensor\_by\_name**(**"phase\_train:0"**)**  # Run forward pass to calculate embeddings  feed\_dict **=** **{**images\_placeholder**:** images**,** phase\_train\_placeholder**:** **False}**  emb **=** sess**.**run**(**embeddings**,** feed\_dict**=**feed\_dict**)**  nrof\_images **=** len**(**args**.**image\_files**)**  **print(**'Images:'**)**  **for** i **in** range**(**nrof\_images**):**  **print(**'%1d: %s' **%** **(**i**,** args**.**image\_files**[**i**]))**  **print(**''**)**  # Print distance matrix  **print(**'Distance matrix'**)**  **print(**' '**,** end**=**''**)**  **for** i **in** range**(**nrof\_images**):**  **print(**' %1d ' **%** i**,** end**=**''**)**  **print(**''**)**  **for** i **in** range**(**nrof\_images**):**  **print(**'%1d ' **%** i**,** end**=**''**)**  **for** j **in** range**(**nrof\_images**):**  dist **=** **np.sqrt(np.sum(np.square(np.subtract(emb[i, :], emb[j, :]))))**    **print(**' %1.4f ' **%** dist**,** end**=**''**)**  **print(**''**)** |

**load\_and\_align\_data代码段**

|  |
| --- |
| **def** load\_and\_align\_data**(**image\_paths**,** image\_size**,** margin**,** gpu\_memory\_fraction**):**  minsize **=** 20 # minimum size of face  threshold **=** **[**0.6**,** 0.7**,** 0.7**]** # three steps's threshold  factor **=** 0.709 # scale factor  **print(**'Creating networks and loading parameters'**)**  **with** tf**.**Graph**().**as\_default**():**  gpu\_options **=** tf**.**GPUOptions**(**per\_process\_gpu\_memory\_fraction**=**gpu\_memory\_fraction**)**  sess **=** tf**.**Session**(**config**=**tf**.**ConfigProto**(**gpu\_options**=**gpu\_options**,** log\_device\_placement**=False))**  **with** sess**.**as\_default**():**  pnet**,** rnet**,** onet **=** **align.detect\_face.create\_mtcnn(**sess**,** **None) ## mtcnn “骨架”网络**  nrof\_samples **=** len**(**image\_paths**)**  img\_list **=** **[None]** **\*** nrof\_samples  **for** i **in** range**(**nrof\_samples**):**  img **=** misc**.**imread**(**os**.**path**.**expanduser**(**image\_paths**[**i**]))**  img\_size **=** np**.**asarray**(**img**.**shape**)[**0**:**2**]**  bounding\_boxes**,** \_ **=** **align.detect\_face.detect\_face(**img**,** minsize**,** pnet**,** rnet**,** onet**,** threshold**,** factor**) ## mtcnn做推断.**  det **=** np**.**squeeze**(**bounding\_boxes**[**0**,** 0**:**4**])**  bb **=** np**.**zeros**(**4**,** dtype**=**np**.**int32**)**  bb**[**0**]** **=** np**.**maximum**(**det**[**0**]** **-** margin **/** 2**,** 0**)**  bb**[**1**]** **=** np**.**maximum**(**det**[**1**]** **-** margin **/** 2**,** 0**)**  bb**[**2**]** **=** np**.**minimum**(**det**[**2**]** **+** margin **/** 2**,** img\_size**[**1**])**  bb**[**3**]** **=** np**.**minimum**(**det**[**3**]** **+** margin **/** 2**,** img\_size**[**0**])**  cropped **=** img**[**bb**[**1**]:**bb**[**3**],** bb**[**0**]:**bb**[**2**],** **:]**  aligned **=** **misc.imresize(**cropped**,** **(**image\_size**,** image\_size**),** interp**=**'bilinear'**) ## 把人脸框resize到目标尺寸.**  prewhitened **=** **facenet.prewhiten(**aligned**)## 归一化**  img\_list**[**i**]** **=** prewhitened  images **=** np**.**stack**(**img\_list**)**  **return** images |

**归一化实现:**

1. np.mean计算矩阵的均值(矩阵所有元素和的均值)
2. np.std计算矩阵的标准差:



1. 求出最大值(作为归一化的分母).

|  |
| --- |
| **def** prewhiten**(**x**):**  mean **=** np**.**mean**(**x**)**  std **=** np**.**std**(**x**)**  std\_adj **=** np**.**maximum**(**std**,** 1.0**/**np**.**sqrt**(**x**.**size**))**  y **=** np**.**multiply**(**np**.**subtract**(**x**,** mean**),** 1**/**std\_adj**)**  **return** y |

# 八 sequeezenet下的facenet

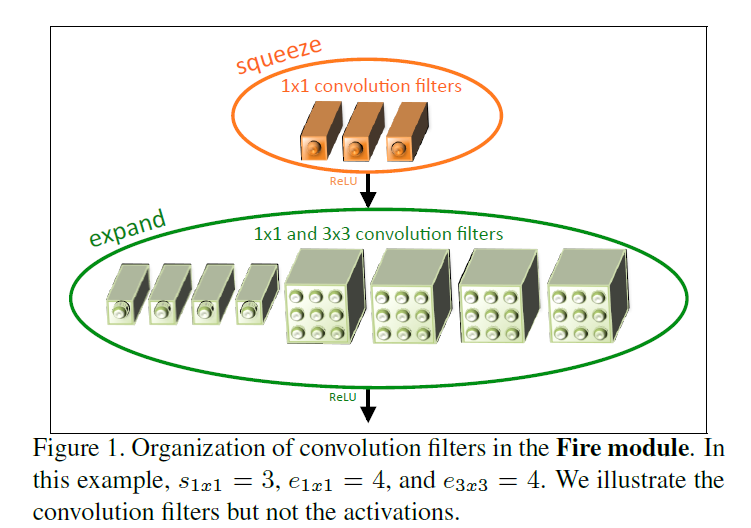
**何为sequeezenet?**

SqueezeNet是由UC Berkeley和Stanford研究人员一起完成的。SqueezeNet**设计目标不是为了得到最佳的CNN识别精度**，而**是希望简化网络复杂度**，同时达到public网络的识别精度。所以SqueezeNet主要是为了降低CNN模型参数数量而设计的。

Squeezenet特点(减少复杂度的几个方面):

1. 替换部分3x3到1x1. 仅在fire module的expand阶段保留部分3x3.(有部分为1x1)
2. 减少3x3卷积的输入feature map数量.
3. 采用将一个3x3的卷积,拆成两个连接的卷积. 这个结构是fire module
4. 减少pooling层.

**何为Fire module**

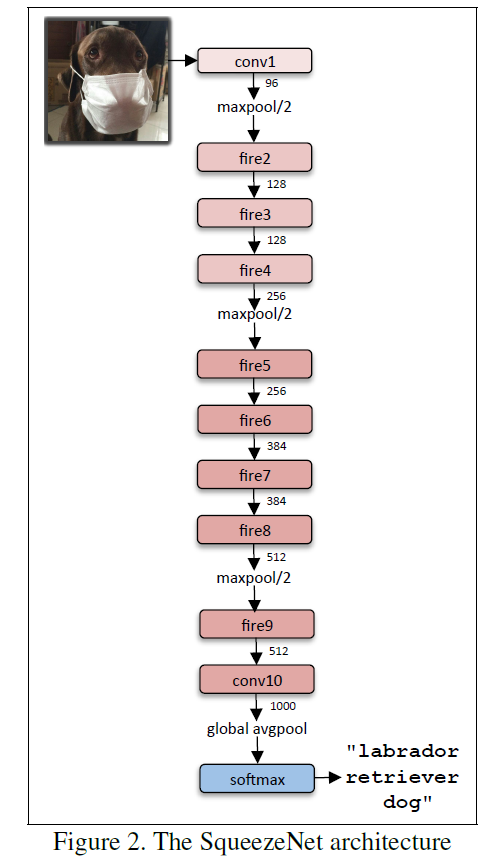


为了减少3x3的conv层的特征图数量.

引入fire module

1. 分成squeeze和expand两个部分.
2. Squeeze部分都是1x1的卷积
3. Expand有一部分是1x1卷积一部分是3x3的.(没有完全用1x1的取代3x3卷积).

总体结构来看:



看图就很明朗了，总共有9层fire module，中间穿插一些max pooling，最后是global avg pooling代替了fc层（参数大大减少）。**在开始和最后还有两层最简单的单层conv层，保证输入输出大小可掌握。**

Squeeze的性能

1. 和alexnet对比.
2. 精度上相差无几.
3. 参数量(模型size)减小越50倍.(倒数第三行), 经压缩技术后,可以减少461倍.

