# 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 : |

## 2.2 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 |

# 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 |

## 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

[homedawn](https://github.com/homedawn) opened this issue on 22 Oct 2018 · 12 comments

**Open**

[**关于算法后期优化若干问题**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210)#210

[homedawn](https://github.com/homedawn) opened this Issue on 22 Oct 2018 · 12 comments

**Comments**

窗体顶端

Assignees

No one assigned

窗体底端

Labels

None yet

窗体顶端

Projects

None yet

窗体底端

窗体顶端

Milestone

No milestone

窗体底端

Notifications

窗体顶端

 Subscribe

窗体底端

You’re not receiving notifications from this thread.

6 participants

[@homedawn](https://github.com/homedawn)[@hqq1990](https://github.com/hqq1990)[@AnberLu](https://github.com/AnberLu)[@liqiang311](https://github.com/liqiang311)[@htjacky](https://github.com/htjacky)[@justein](https://github.com/justein)

[](https://github.com/homedawn)

[**homedawn**](https://github.com/homedawn)**commented**[**on 22 Oct 2018**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issue-372453352)

|  |
| --- |
| 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

 2

窗体底端

[](https://github.com/hqq1990)

[**hqq1990**](https://github.com/hqq1990)**commented**[**on 2 Nov 2018**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-435321429)

|  |
| --- |
| 您好，方便留个邮箱吗，在训练Pnet时遇到点问题想请教 |

[](https://github.com/AnberLu)

[**AnberLu**](https://github.com/AnberLu)**commented**[**on 18 Jan**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-455488315)

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

[](https://github.com/homedawn)

Author

[**homedawn**](https://github.com/homedawn)**commented**[**on 21 Jan**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-455919810)

|  |
| --- |
| 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 |

[](https://github.com/liqiang311)

[**liqiang311**](https://github.com/liqiang311)**commented**[**on 13 Feb**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-463087983)

|  |
| --- |
| [**@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。 |

[](https://github.com/homedawn)

Author

[**homedawn**](https://github.com/homedawn)**commented**[**on 13 Feb**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-463096994)

|  |
| --- |
| 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 |

[](https://github.com/liqiang311)

[**liqiang311**](https://github.com/liqiang311)**commented**[**on 13 Feb**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-463097630)

|  |
| --- |
| [**@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。 |

[](https://github.com/homedawn)

Author

[**homedawn**](https://github.com/homedawn)**commented**[**on 13 Feb**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-463098567)

|  |
| --- |
| 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 |

[](https://github.com/homedawn)

Author

[**homedawn**](https://github.com/homedawn)**commented**[**on 13 Feb**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-463103543)

|  |
| --- |
| [**@liqiang311**](https://github.com/liqiang311) |

[](https://github.com/htjacky)

[**htjacky**](https://github.com/htjacky)**commented**[**on 14 Feb**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-463525563)

|  |
| --- |
| [**@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%。 |

[](https://github.com/homedawn)

Author

[**homedawn**](https://github.com/homedawn)**commented**[**on 14 Feb**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-463526517)

|  |
| --- |
| [**@htjacky**](https://github.com/htjacky) 没有验证，目前在看减少耗时的问题 |

[](https://github.com/justein)

[**justein**](https://github.com/justein)**commented**[**21 days ago**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-468181740)

|  |
| --- |
| [**@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 |

[](https://github.com/homedawn)

Author

[**homedawn**](https://github.com/homedawn)**commented**[**21 days ago**](https://github.com/AITTSMD/MTCNN-Tensorflow/issues/210#issuecomment-468184866)

|  |
| --- |
| 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？

dear-john commented Mar 24, 2019

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？

AITTSMD commented Mar 26, 2019

@dear-john 对，注意通道顺序