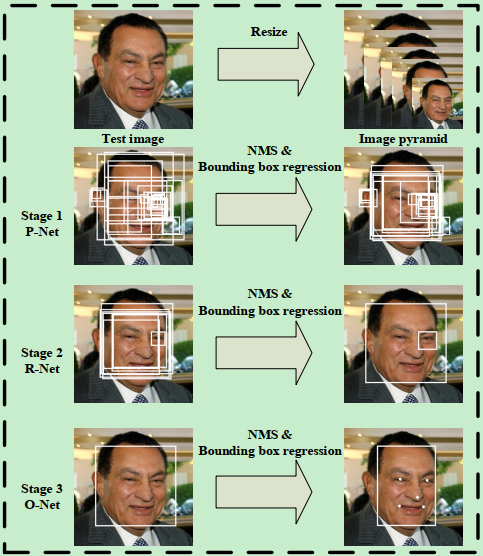
MTCNN-细节回顾

## 作用

用于人脸侦测，也就是判断某一块区域上是否有人脸，如果有人脸，那么返回人脸坐标和关键点，为后续的人脸识别做基础

## 原理

通过保持固定的图片尺寸比如(12\*12, 24\*24, 48\*48),然后使用图像金字塔对图像进行缩放，以适应不同尺寸的人脸，通过图像金字塔的缩放达到完全覆盖图像的目的，然后生成多个候选框，将候选框坐标传入下一个网络，下一个网络对上一个网络的结果坐标进行正方形化，然后通过正方形化的坐标进行抠图，将抠图后的图片进行resize再传给网络进行侦测 侦测完成之后也有多个候选框，通过置信度阈值(手动设置)筛选出一批框，将候选框进行NMS得到结果(NMS也有IOU阈值，手动设置)，总共三个级联网络 重复上面流程，最后通过O网络得到人脸框，下图为MTCNN的流程



## 优缺点

优点：

通过级联神经网络的思想，将一个大网络拆分成三个小网络，每个小网络的层数都不是很深，但是三个合并起来的深度还是可以的，类似于使用两个3\*3的卷积核替换一个5\*5的卷积核的思想，还可以获得更好的感受野，

缺点：

P网络使用了图像金子塔，会有大量图片输入，边训练边裁剪图片，导致速度变慢，训练好网络做预测，一次预测话费1.8s左右，P网络占85%左右，优化空间巨大

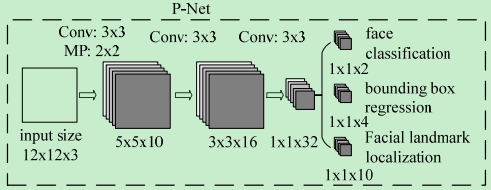
## 网络设计

* 1. 网络思想

CNN级联神经网络，多个小网络组成一个大的网络

* 1. 网络架构

**P-NET**：



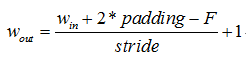
输入尺寸是12\*12\*3, 也是网络规定的最小人脸检测图片尺寸，再小也没有什么

意义，12\*12的尺寸 图片已经很小了。

网络层：

|  |
| --- |
| 一个3\*3的卷积核 步长为1 padding为1，  一个MaxPool步长为2的池化  一个3\*3的卷积核 步长为1，输入为5\*5\*10 输出为3\*3\*16  一个3\*3的卷积核 步长为1, 输入3\*3\*16 输出1\*1\*32  1\*1\*32的向量：  经过1\*1\*2的卷积，做分类问题，是否有人脸 也就是置信度  经过1\*1\*4的卷积，人脸候选框的回归，输出为相对于标签框的偏移量  经过1\*1\*10的卷积 做人脸关键点回归，输出为人脸关键点 |

**为什么padding为1？**



因为根据特征图计算公式，Win代表输入尺寸，padding代表填充，

F表示卷积核尺寸，stride表示步长

计算过程：

第一层卷积：（12+2\*0-3）/ 1 +1=10

第二层最大池化：（10-3）/ 2 +1=4

池化后的结果并不是5

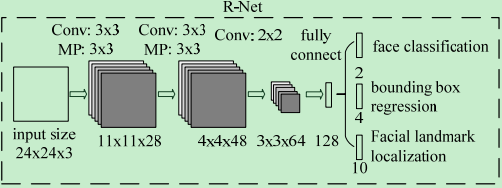
如果给输入的12\*12加一层same模式的padding之后，再来计算

第一层卷积：（12+2\*1-3）/ 1 +1=12

第二层最大池化：（12-3）/ 2 +1=5

此时才能满足上图网络架构

**R-NET：**

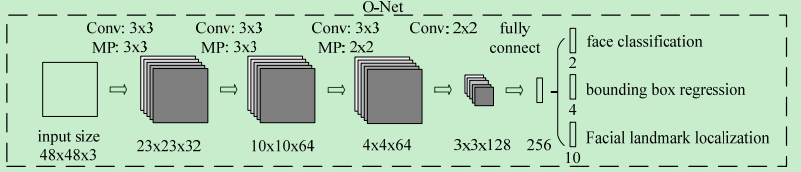
输入尺寸为24\*24\*3，

使用时：是P网络预测出来的人脸框，经过正方形化然后抠图出来的图像进行resize，

网络层：

|  |
| --- |
| 一个3\*3的卷积，步长为1  一个ReLU  一个3\*3的池化 步长为2  输入是24\*24\*3 输出是11\*11\*28  一个3\*3的卷积，步长为1  一个ReLU  一个3\*3的池化 步长为2  输入是28\*28\*3 输出是4\*4\*48  一个2\*2的卷积  一个ReLU  输入是4\*4\*48 输出是3\*3\*64  全连接层  输入是3\*3\*64 输出是128  经过ReLU  输入128 输出1 经过Sigmoid 输出置信度  输入 128 输出 4 输出偏移量 |

**O-NET：**

输入尺寸为48\*48\*3，

使用时：是R网络预测出来的人脸框，经过正方形化然后抠图出来的图像进行resize到48\*48

网络层：

|  |
| --- |
| 一个3\*3的卷积，步长为1，padding=1  一个ReLU  一个3\*3的池化 步长为2  输入是48\*48\*3 输出是23\*23\*32  一个3\*3的卷积，步长为1  一个ReLU  一个3\*3的池化 步长为2  输入是23\*23\*32 输出是10\*10\*64  一个3\*3的卷积  一个ReLU  一个2\*2的池化 步长为2  输入是4\*4\*64 输出是3\*3\*128  全连接层  输入是3\*3\*128 输出是256  经过ReLU  输入256 输出1 经过Sigmoid 输出置信度  输入 256 输出 4 输出偏移量 |

网络实现

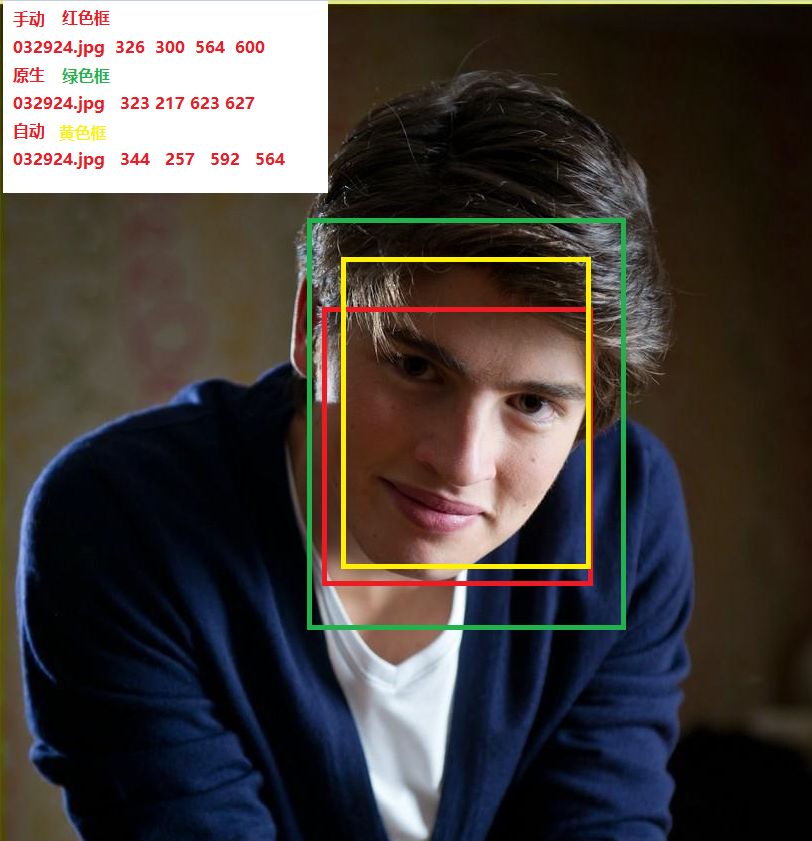
* + - 暂时只是实现了人脸边框回归，关键点还没有做

## 训练

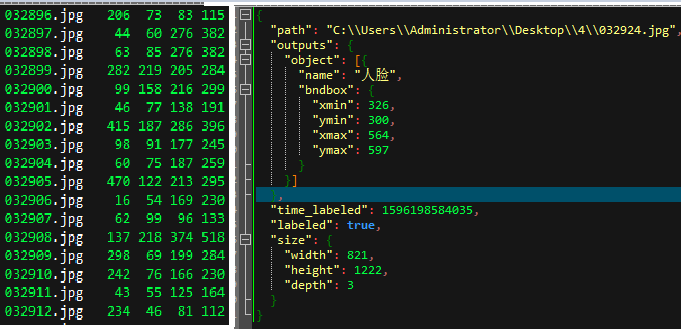
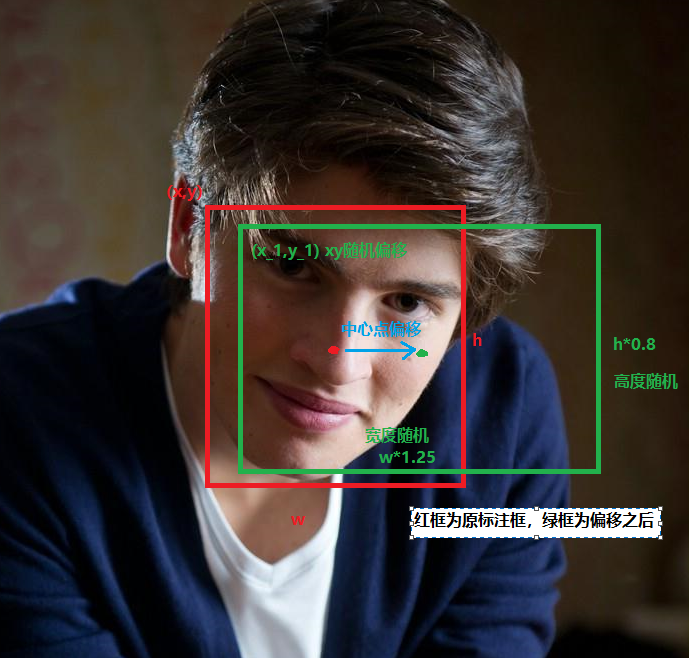
使用[Celeba](http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html)数据集，总共20w数据

* 1. **思路**

拿到数据集首先对数据集进行观察，发现原生标注的人脸区域偏大，对后续的人脸识别来讲，标注框过大训练出来的模型预测的框也会偏大，后续人脸识别可能会有较多噪声，下图为原生标注，使用Github训练好的mtcnn进行自动标注的人脸，和手动标注的三种效果对比：



可以看到绿色框也就是原生标注，框选区域较大，将头发衣服等无关区域也包含在内，而训练好的自动标注和手动标注效果相对较好，最好的还是手动标注，仅仅包含人脸最好，后期做人脸识别的时候噪声最少，但是成本较高，看自己选择。

* 1. **数据生成策略**
     + 确定好要手动标注数据还是自动标注之后，确定格式，原生标注为txt格式，内中为空格分割的字符串，如果自动手动标注，使用标注精灵的话，生成是json格式，每一个图片都是一个json文件，需要些python程序合并json文件，或者数据集加载直接读取json文件，建议直接整合为一个json或者一个txt，减少IO次数， 下图左边为原生标记，右边为标注精灵标注的格式
     + 
     + 1：有了标注文件之后，读取标注文件，获取xy坐标，注意两种格式的区别，原生格式里面是x,y,w,h，而标注软件是Xmin,Ymin,Xmax,Ymax，一个是右下角的坐标，一个是宽高，一定要注意
     + 2：读取标注文件之后，获取到人脸框的左上角坐标和右下角坐标，基本生成思想是：
     + 
       - 1. 获取标签框的xy坐标和宽高，计算出中心点
         2. 让中心点一定范围内随机偏移，比如坐标正负偏移20%左右
         3. 再让宽高也一定范围内随机增长减少，改变生成框的大小
         4. 根据随机以后的中心点+随机以后的宽高，得到新的生成框
         5. 使用新生成的框和标签框进行IOU，

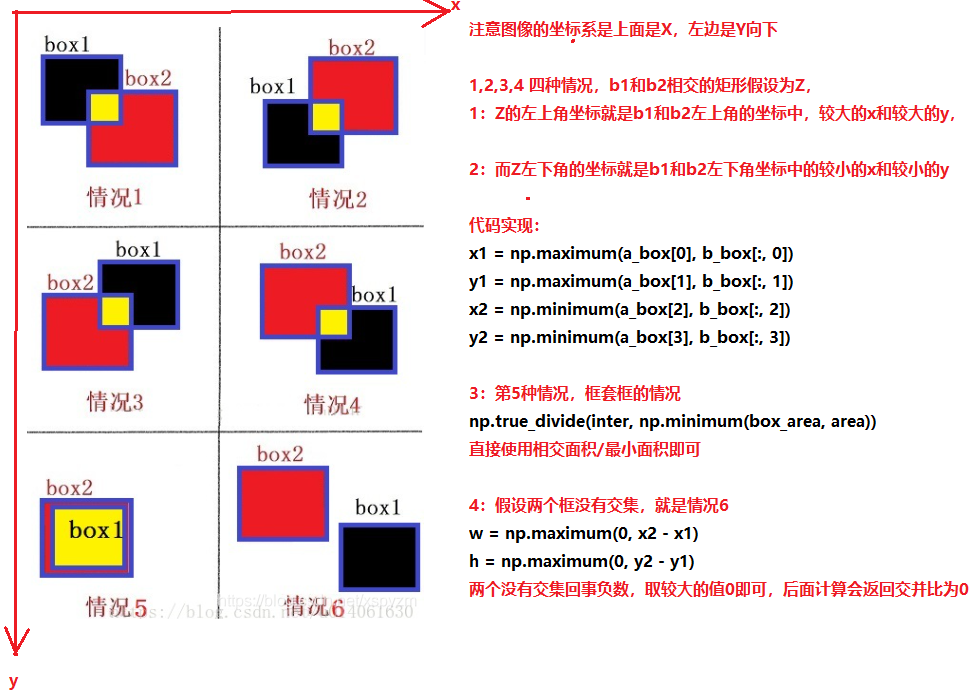
大于0.7为正样本

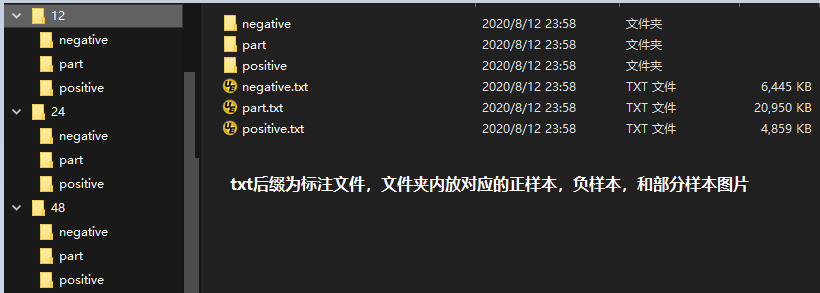
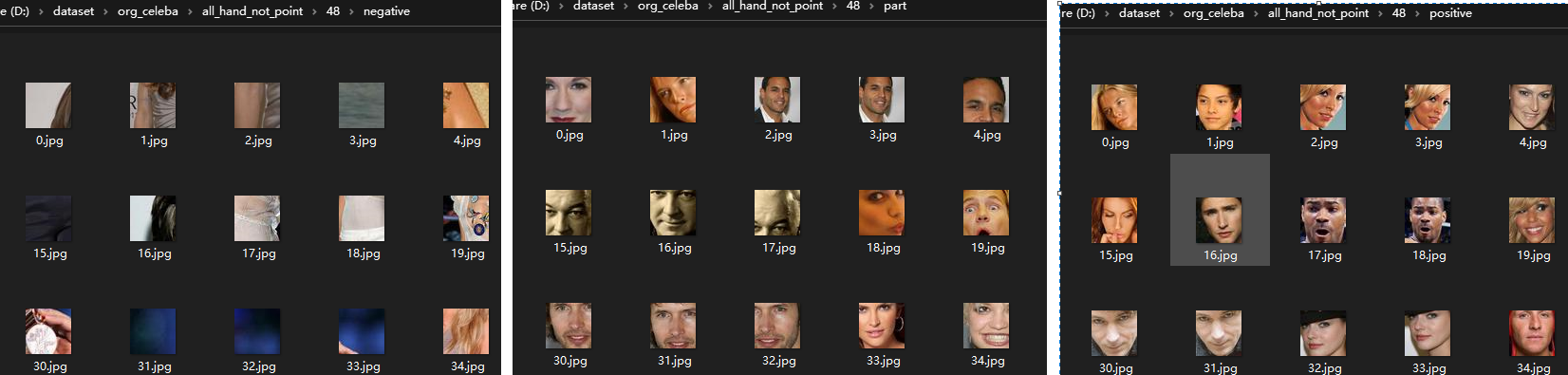
大于0.4小于0.5为Part样本

小于0.29的为负样本 我设置为0.05并且使用两种iou模式（最小化和交并比）

* + - * 1. 通过上面的判断，将符合要求的坐标从原图上crop下来进行保存，
        2. 计算偏移量，通过标注框的xy坐标-新框的xy坐标，然后除以新框随机出来的边长得到一个四个偏移量，分别是针对标注框x,y x1,y1的偏移量
        3. 同时保存文件名和对应的坐标，生成新的标注文件
        4. 根据上面的方法得到的正样本和part样本是没有问题的，但是负样本过少
        5. 单独生成负样本，让x,y的坐标在最小边长和原图宽高之间进行随机，增大了随机的可能性，增加偏移量，让生成框和标签框的重合度更低获取到部分样本

**IOU的解释：**



* 1. **生成效果和调试**
     + 生成效果应该是生成三个文件夹，每个文件夹中有一个尺寸的正负部分样本图片例如下图结构示例：
     + 
     + 因为12\*12尺寸太小，为了看到效果，所以下面都是选用48\*48展示
     + 从左至右分别是 负样本效果：Part样本效果：正样本效果
     + 
     + 样本标注文件，内容展示，从上到下，顺序分别是，负样本，part样本 正样本
     + 

## 训练日志

[训练日志文档连接](MTCNN%20训练日志.docx)



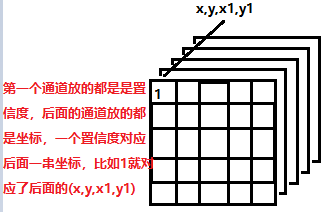
## 使用&侦测思路

1. **P-Net思路**：

1：因为是CNN，没有限制图片的输入尺寸，首先获取输入图像的最短边长用于下面计算图像金字塔

2：图像格式转换为张量，并升维，将CHW转换为NCHW,增加批次维度

3：输入P网络得到输出，两个输出，一个是一个通道的输出是置信度，一个是4个通道的输出是偏移量，结构类似下图：



4：现在就要从这样的结构中将对应的置信度和x,y的坐标提取出来

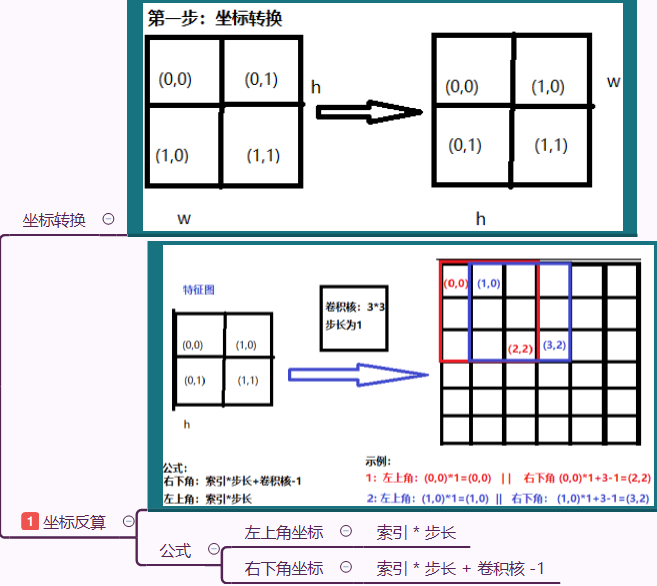
5：假设输出尺寸为200,400，那么置信度的形状就是(1,1,200,400),而偏移量的形状就是(1,4,200,400), 提取置信度形状为（200\*400）,提取偏移量形状为（4,200,400），

6：筛选大于置信度阈值的第一通道内的坐标

7：通过坐标拿到置信度的值，将置信度的值和坐标，缩放比例，偏移量一起进行反算

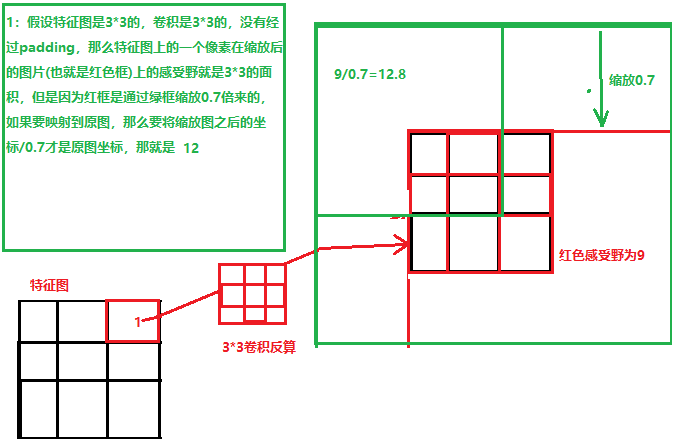
8：反算过程如下 重点

start\_index其实就是最后一层网络输出的特征图上面的坐标，现在要将特征图上的坐标反算到上一层，那么就使用下面的公式计算



也就是对应下面代码中的第一部分，\_x1,\_y1\_x2,\_y2的计算过程，

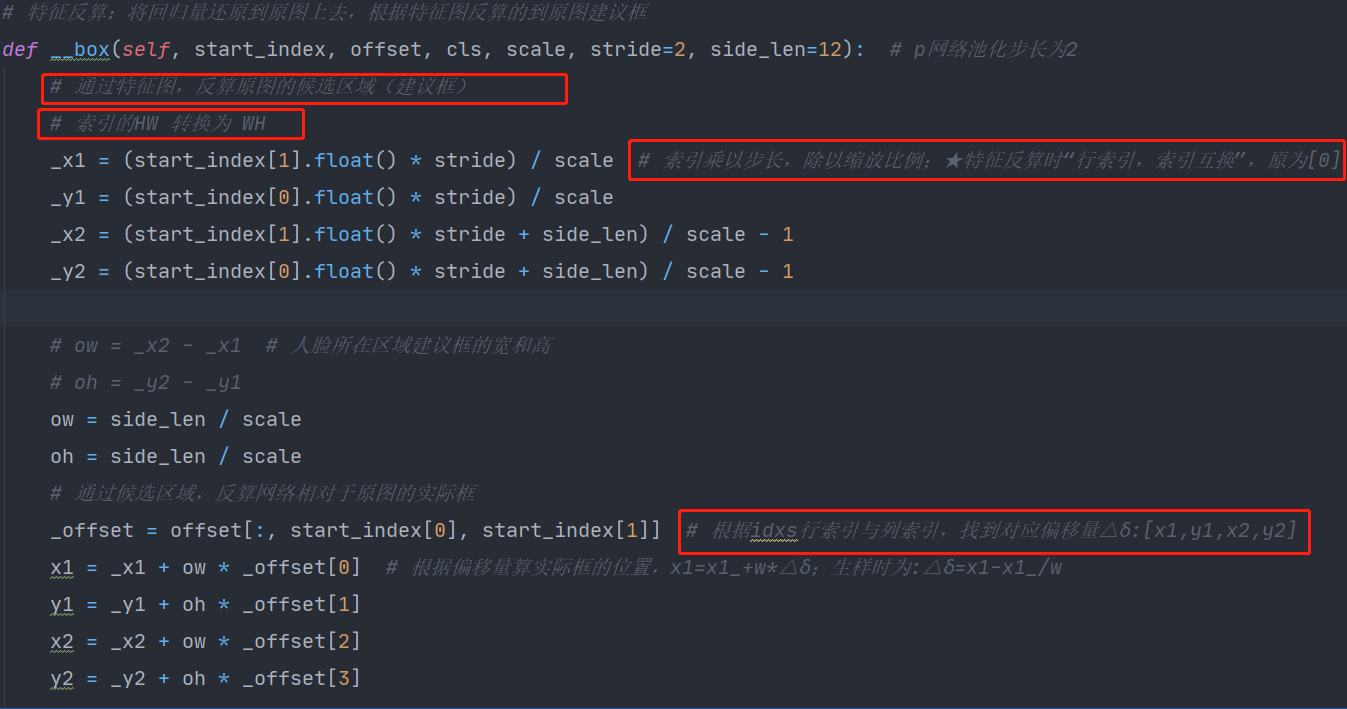
后面的缩放比例的原理如下图所示



然后通过传入的边长和缩放比例计算宽高，这里为什么要重新计算宽高，就是因为反算回去之后坐标都发生了变化，宽高也会发生变化，所以要重新计算，而偏移量为什么不需要重新计算，因为他是一个偏移量是一个比值，缩放比例同时变化，偏移量是不变的

\_offset的计算，就是用start\_index中的索引(也就是筛选后的所以)拿到形状为(4,h,w)的偏移量中和start\_index对应的偏移量，

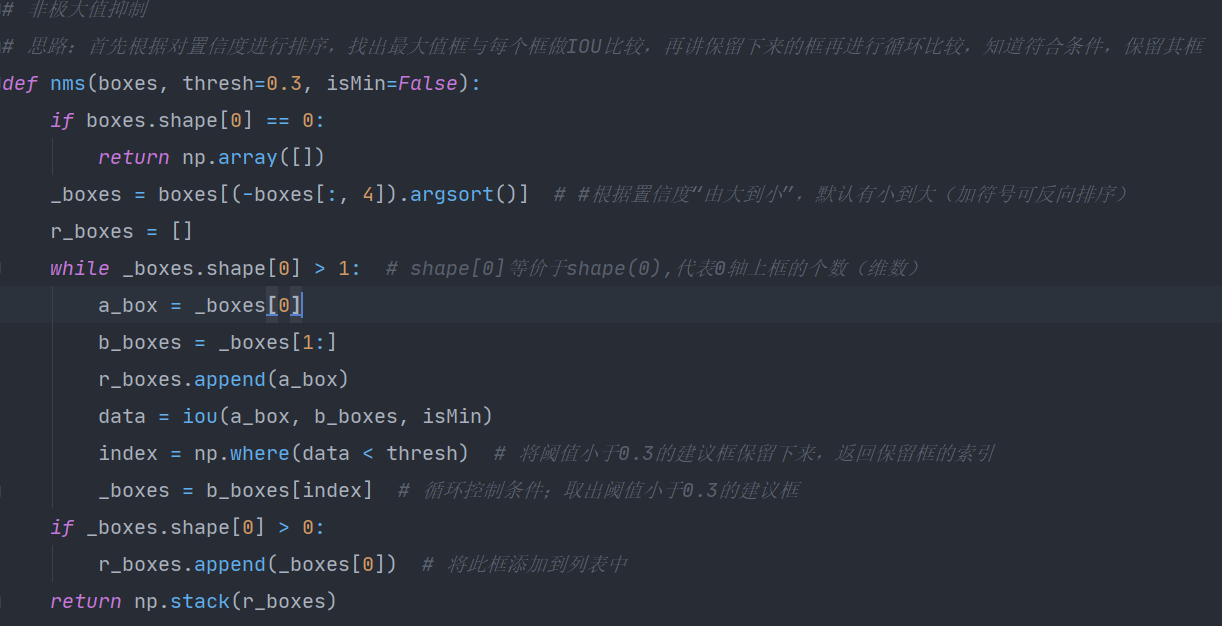
拿到偏移量之后通过上面计算出来的坐标+边长\*偏移量 得到反算后的原图坐标，将原图坐标和置信度组装 返回



9：通过反算返回原图上的候选框，然后在做完图像金字塔之后，将所有的反算候选框做NMS,阈值自己定义，这里阈值定义为0.5，因为P网络精度不高，置信度并不是完全可信，所以阈值不能太高

**NMS思路和代码：**

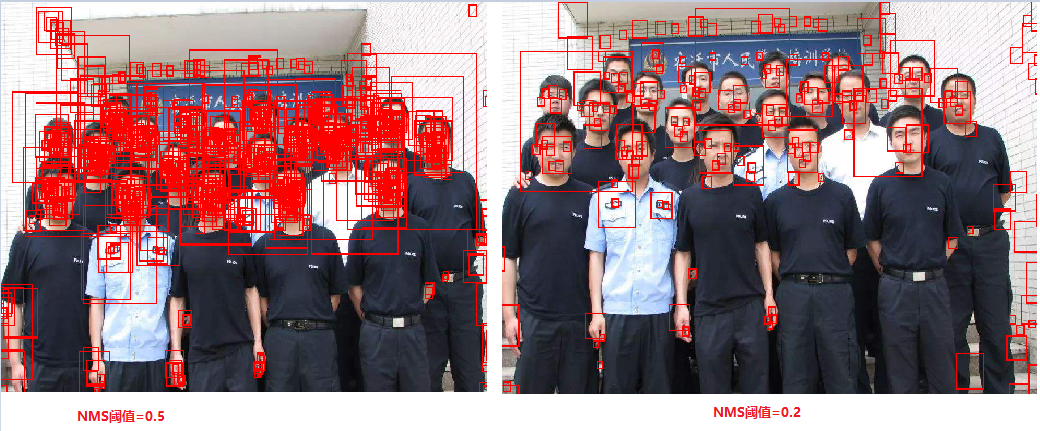
根据上一步反算得到的所有的框和置信度，根据置信度排序，排序后，计算iou，去掉超过iou阈值的框，说明重合度过高了，保留重合度较低的框，给下一个网络



P网络NMS效果展示：

左图阈值为0.5 右图为0.2，

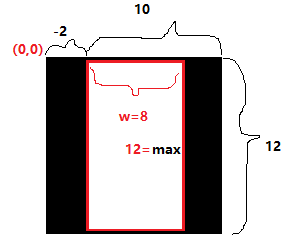
但是在P网络中应该用0.5的阈值，因为P网络只是第一层网络，精度不高，过低的阈值舍弃过多的候选框，可能造成误判，影响后面网络的精度



1. **R-Net思路：**

1：拿到P网络传入的候选框之后，需要做正方形化，

正方形化原理：



找到中心点然后沿着最大边长的两边进行扩充，

依据上图，假设一个宽高为w=8,h=12的矩形，最大边长为h=12，x,y=0,

首先计算左上角x,y，代入下面代码计算得到

0+8\*0.5-12\*0.5=-2 x需要向左移动2

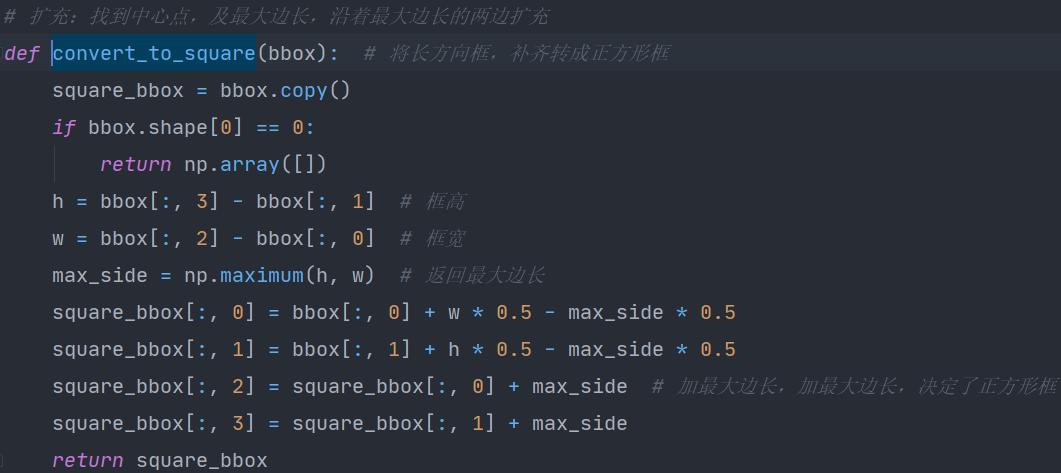
0+12\*0.5-12\*0.5=0 y因为本来就是12所以不用动

然后计算右下角x,y，代入下面代码计算得到：

-2+12=10

0+12=12

然后得到一个正方形区域



2：将正方形的坐标抠图，然后进行resize至 24\*24,再进行堆叠放入Rnet得到输出，

3：大致流程和P网络相同，通过阈值筛选出索引坐标，

然后通过遍历索引坐标获取到偏移坐标，注意这里是拿到索引从P网络传入的box中获取到坐标，因为p网络传入的候选框就是R网络的建议框

4：拿到xy坐标进行反算，xy坐标加上边长\*偏移量，然后加入需要计算

的候选框

5：最后进行NMS

1. **O\_Net思路：**

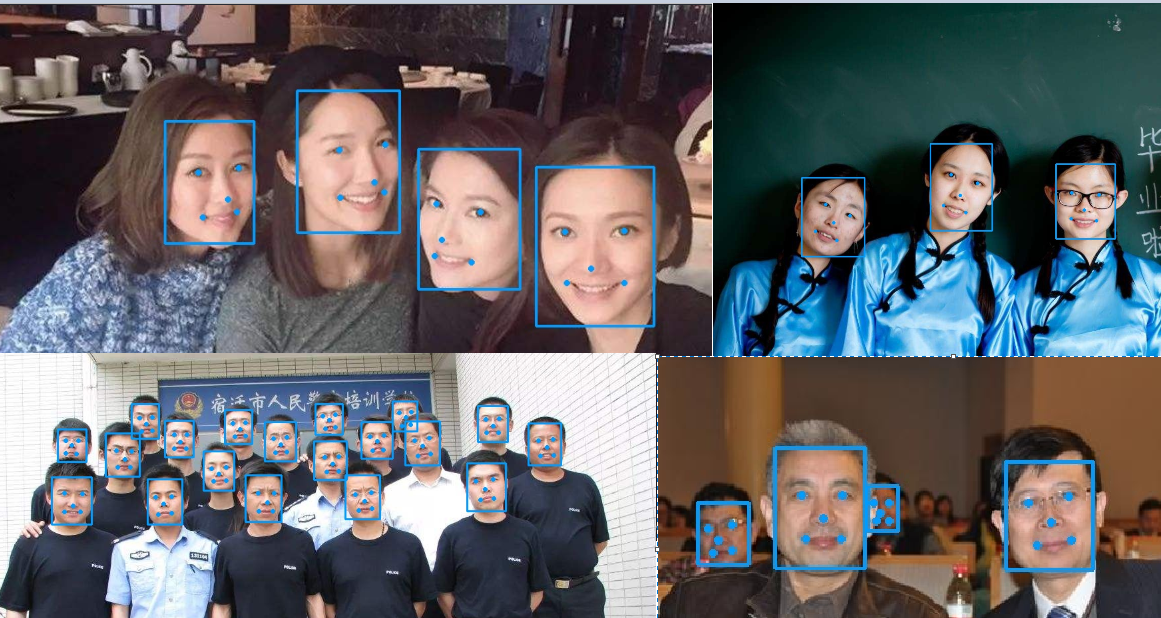
思路和R网络一致，只是输入是R网络的输入

## 效果

最终效果如下图，因为使用了手动标注数据，所以人脸框偏小，方便后期做人脸识别

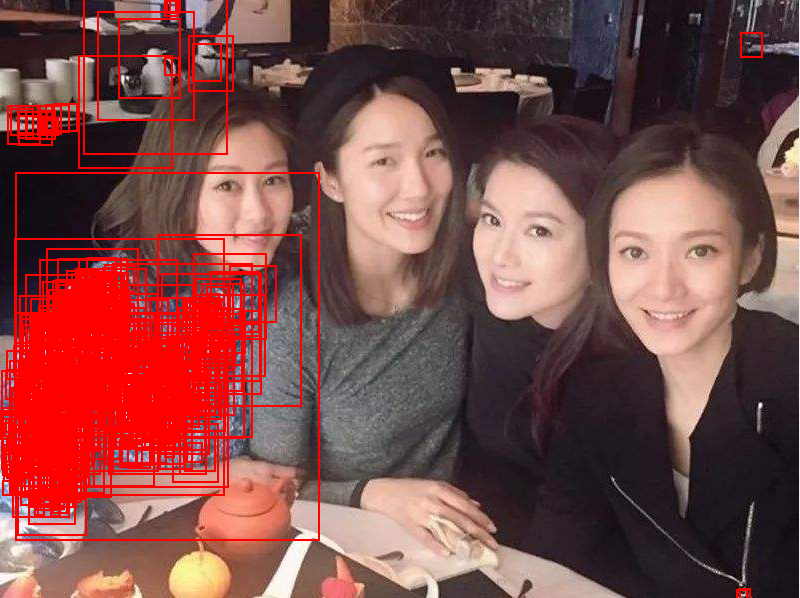


下面是github上面别人训练好的模型预测出来的效果：效果差距不大，上面做的人脸框更小一点



## 问题和解决：

* **1：预测框一直在非人脸区域**



解决：

原因是加载数据集的时候，读取图片使用了CV2读取图片，而CV2读取读片默认是BGR格式的，而训练需要使用RGB格式，所以使用了cv2读取一定要转换RGB

* **2：多个置信度为1的框**

解决：使用增强正样本，减少正样本的偏移量来解决，不到半张脸也能是置信度为1

的情况

* **3：每个文件夹中的图片数量对dataloader影响很大，超过20w之后会变得很慢**

解决：

减少同一个文件夹的图片，可以分批训练，拆分到多个文件夹或多个标注文件

* **4：MTCNN不能增加BN层 ，否则会出现大量无用候选框导致无法进行iou和nms**

解决：去掉BN层，原因是什么？

* **5：因为P网络侦测出来的框太多，导致整个侦测速度变慢，现在继续训练提高精度，P网络损失为0.005时侦测一张图片有200个候选框，但是损失为0.01的时候有4000候选框，大量的框运算较慢拖慢了速度，所以继续训练提高精度尝试**

解决：

训练网络出来效果之后，发现和同学的速度差别较大，最后定位到是因为P网络

返回了4000多个候选框，而同学只是返回了200多候选框，预测可能是因为P

网络损失没有下去导致的预测框过多

* **6：为什么需要对图片做“金字塔”变换？**

由于各种原因，图片中的人脸的尺度有大有小，让识别算法不被目标尺度影响一直是挑战；**目标检测本质上来说上目标区域内特征与模板权重的点乘操作**；那么如果模板尺度与目标尺度匹配，自然会有很高的检测效果

**缺点就是：慢**。

1. 生成图片金字塔慢；
2. 每种尺度的图片都需要输入进模型，相当于执行了多次的模型推断流程。

* **7：MTCNN算法可以接受任意尺度的图片，为什么？**  
   因为第一阶段的P-NET是一个全卷积网络（Fully Convolutional Networks）

卷积、池化、非线性激活都是一些可以接受任意尺度矩阵的运算，

但全连接运算是需要规定输入。如果网络中有全连接层，

则输入的图片尺度(一般)需固定；如果没有全连接层，图片尺度可以是任意的。

当然也有既包含全连接层也能接受任意尺度图片的结构：Kaiming He等提出的**SPP（Spatial Pyramid Pooling，空间金字塔池化）**就是做这个的(https://arxiv.org/abs/1406.4729)。本文不作过多说明

* **8：设置合适的最小人脸尺寸和缩放因子为什么可以优化计算效率？缩放因子为什么官方选择0.709?**

我们已经知道，第一阶段会多次缩放原图得到图片金字塔，目的是为了让缩放后图片中的人脸与P-NET训练时候的图片尺度(12px \* 12px)接近。怎么实现呢？先把原图等比缩放`12/minsize`，再按缩放因子`factor`(例如0.5)用上一次的缩放结果不断缩放，直至最短边小于或等于12。根据上述算法，minsize越大，生成的“金字塔”层数越少，resize和pnet的计算量越小

**为什么缩放因子factor官方选择0.709？**图片金字塔缩放时，默认把宽，高都变为原来的1/2，缩放后面积变为原来的1/4；如果认为1/4的缩放幅度太大，你会怎么办？把面积缩放为原来的1/2。对的，这是很直观的想法，所以这里的缩放因子0.709 ≈ sqrt(2)/2，这样宽高变为原来的sqrt(2)/2，面积就变为原来的1/2。并且从比MTCNN更早提出的级联人脸检测CVPR2015\_cascade CNN（http://users.eecs.northwestern.edu/~xsh835/assets/cvpr2015\_cascnn.pdf）的实现中也能找到端倪

* **9： 为什么把图片输入模型的时候要对每个元素做(x – 127.5)/128的操作？**

归一化操作，加快收敛速度。

由于图片每个像素点上是[0, 255]的数，都是非负数，对每个像素点做(x – 127.5)/128，可以把[0, 255]映射为(-1, 1)。具体的理论原因可以自行搜索，但实践中发现，有正有负的输入，收敛速度更快。训练时候输入的图片需要先做这样的预处理，推断的时候也需要做这样的预处理才行。

* **10： P-NET网络的输出怎么还原成原图上人脸区域坐标？**

就是反算的逻辑，通过特征图上的点反算到缩放后的图 然后再除以缩放比例

* **11：什么叫边框回归？在MTCNN怎么利用边框回归的结果？为什么可以这样做？**
* <https://blog.csdn.net/zijin0802034/article/details/77685438>，
* 对于窗口一般使用四维向量(x,y,w,h)来表示，分别表示窗口的中心点坐标和宽高。我们的目标是寻找一种映射，使得输入原始的窗口 P(下图红框) 经过映射的结果跟真实窗口G(下图绿框)更接近。



我们学习的就是映射关系dx(P),dy(P),dw(P),dh(P)。而上面的这几个公式，就是在做线性变换。学习这些映射关系的过程就是在线性回归求解参数矩阵的问题。所以称之为**边框回归**。

* **12：哪些步骤是影响MTCNN的计算效率的关键？以及有哪些优化思路？**

**三个阶段的时间占比如何？**

MTCNN的推断是CPU密集型运算，如果是图片超过1080，生成图像金字塔的过程可能是流程中最耗时的过程。

因为金字塔结构，第一阶段需要地计算很多尺度的图片。但超过1080的图片中，你需要识别的最小人脸真的需要12\*12吗？minsize是不是也变大了？另外，如果你事先已经知道图片中人脸的大小，是不是可以调整一下minsize？结合你的实践场景可以思考下。以耗时最大的第一阶段为主要优化的关键点，以下说明一些我尝试过的办法：

* 第一阶段受输入图片尺寸影响较大，可以让minsize随图片尺寸而改变，大图用大minsize
* 图片金字塔的生成过程，对上一次的resize结果进行resize而不是对全图resize
* 并行化第一阶段，收益不大，不如少resize几次。



* 

MTCNN的第一阶段，图像金字塔会反反复复地很多次调用一个很浅层的P-NET网络，导致数据会反反复复地从内存COPY到显存，又从显存COPY到内存，而这个复制操作消耗很大，甚至比计算本身耗时。

这里对性能方面的研究做一些总结：

* MTCNN的推断流程性能优化从第一阶段入手，关键是降低迭代次数，可以利用minsize；
* MTCNN的推断流程中，模型计算耗时没有想象中那么大，反而可能是不断显存和内存之间来回复制数据导致效率不高；

## 总结：