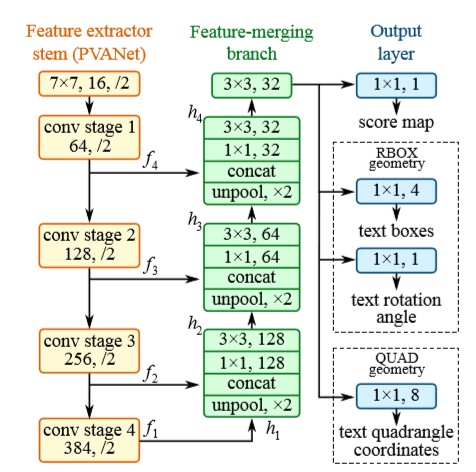


# pipeline

结构清晰，是个FCN网络，主体结构就是Unet，backbone任选



**feature extractor**

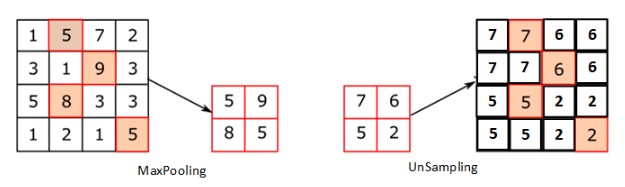
backbone任选，用ImageNet预训练即可

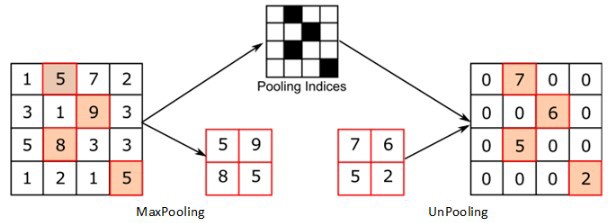
作者实验了 PVANet 和 VGG16

开源代码用的resnet50

从backbone中抽4个feature maps size分别是输入图片的 1/4，1/8，1/16，1/32

**UnPooling**





代码中用的双线性插值



模型完整的定义代码：

1

2

3

4

5

# 搭建EAST的合并层和输出层

**def** model(images, weight\_decay=1e-5, is\_training=**True**): images = mean\_image\_subtraction(images)

# 特征提取层输出

1. **with** slim.arg\_scope(resnet\_v1.resnet\_arg\_scope(weight\_decay=weight\_decay
2. logits, end\_points = resnet\_v1.resnet\_v1\_50(images, is\_training=is\_t 8

9 **with** tf.variable\_scope('feature\_fusion', values=[end\_points.values]):

10 batch\_norm\_params = {'decay': 0.997, 'epsilon': 1e-5, 'scale': **True**, 11

1. **with** slim.arg\_scope([slim.conv2d], activation\_fn=tf.nn.relu, normali
2. normalizer\_params=batch\_norm\_params, weights\_reg 14

15 # 特征提取层输出特征

1. f = [end\_points['pool5'], end\_points['pool4'],
2. end\_points['pool3'], end\_points['pool2']]
3. **for** i **in** range(4):
4. print('Shape of f\_{} {}'.format(i, f[i].shape)) 20

21 # 合并层搭建

1. g = [**None**, **None**, **None**, **None**]
2. h = [**None**, **None**, **None**, **None**]
3. num\_outputs = [**None**, 128, 64, 32]
4. **for** i **in** range(4): 26 **if** i == 0:

27 h[i] = f[i]

##### else:

1. c1\_1 = slim.conv2d(tf.concat([g[i-1], f[i]], axis=-1), n
2. h[i] = slim.conv2d(c1\_1, num\_outputs[i], 3) 31 **if** i <= 2:
3. g[i] = unpool(h[i])

##### else:

1. g[i] = slim.conv2d(h[i], num\_outputs[i], 3)
2. print('Shape of h\_{} {}, g\_{} {}'.format(i, h[i].shape, i, g 36

37 # 输出层搭建

1. # here we use a slightly different way for regression part, we f
2. # range, and also this is do with the angle map
3. F\_score = slim.conv2d(g[3], 1, 1, activation\_fn=tf.nn.sigmoid, n
4. geo\_map = slim.conv2d(g[3], 4, 1, activation\_fn=tf.nn.sigmoid, n
5. # angle is between [-45, 45]
6. angle\_map = (slim.conv2d(g[3], 1, 1, activation\_fn=tf.nn.sigmoi
7. F\_geometry = tf.concat([geo\_map, angle\_map], axis=-1)
8. **return** F\_score, F\_geometry

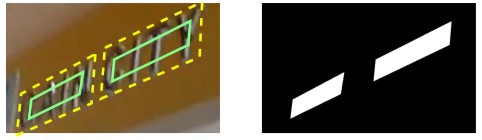
**输出层**

输出层分为两部分

Score map 和 Geometry map

**Score map**

Score map 负责预测文字区域，相当于对某个像素点是否为文字进行打分，输出一个二 值图

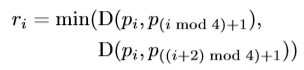


制作ground truth标签的时候，取值为1 的positive区域对应四边形label往内收一圈 这是基于pixel方法的常见技巧，主要是防止预测文本行直接发生粘连

具体到算法



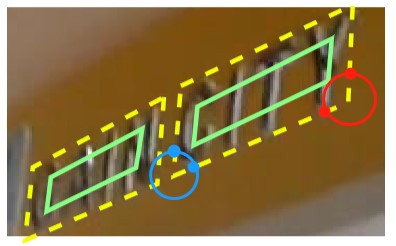
pi对应按照顺时针有序排列的四边形的顶点 首先为每个顶点计算一个向内收缩的基准半径（其实就是相邻两边较短的那个）



然后先收缩两条长边，再收缩两条短边 收缩大小就是基准距离\*0.3

以上是paper的描述，我理解每条边收缩后得到两个新的端点，8个点一连就得到了收缩 后的矩形

具体还要看下代码，不过这块并不复杂，感觉怎么实现都可以



**Geometry map**

需要根据最终输出 RBOX格式 还是 QUAD格式来分别介绍

**构造RBOX格式的输出**

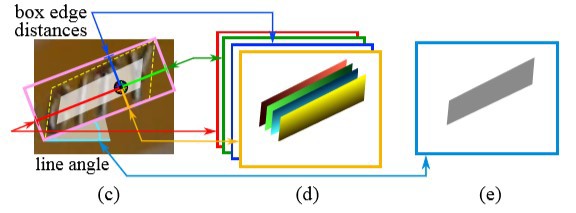
score map上对应positive的区域的每个pixel需要分别预测到RBOX上下左右4条边的距离

（对应4个channel）+ 一个角度

**如何判断上下左右？**



输出可以直接用下图表达



对于标注格式为QUAD（例如ICDAR2015），但是想使用RBOX格式构造GroundTruth的 情况的处理方法

就是先求一个能够包裹原四边形的具有最小面积的旋转矩形，然后进行后面的操作 如图中黄色虚线四边形框 到 粉色实线旋转矩形框 所示意的那样。

**构造QUAD格式的输出**

score map上对应positive的区域的每个pixel需要分别预测到4个顶点的坐标偏移（因此是

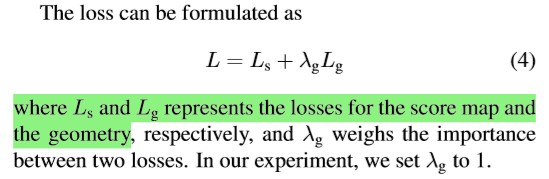
8个channel）

**有个同样的问题，如何判断哪个channel对应哪个点？**



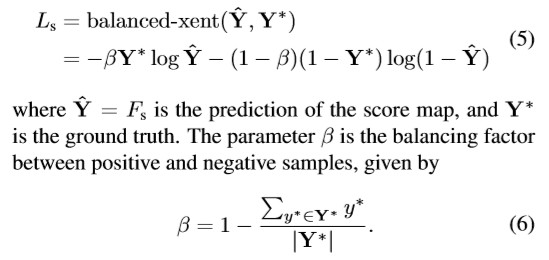
**损失函数**

总的损失函数



**Loss for Score Map**

Score map的loss设计其实和分割问题一样，逐像素点计算交叉熵肯定是基本方案。 几乎所有的检测/分割的pipeline都要小心的处理正负样本不均衡的问题，这里也不例外 作者用的是 类平衡交叉熵 classbalanced cross-entropy



classbalanced cross-entropy 出自[Holistically-nested edge detection](https://arxiv.org/pdf/1504.06375.pdf)这篇文 章， 主要用于FCN,U-net等分割，边缘检测的网络，用于对像素级别的二分类 样本不平衡进行优化。

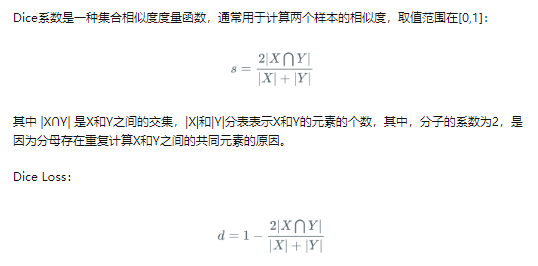
思想就是引入新的权值β，实现正负样本loss的平衡，从而实现对不同正负样本 的平衡。

这里作者将β设置为1-文本区域占比，相当于文本约小，就加越多的权重。

#### 代码中使用的是dice loss



Dice系数与Dice Loss



代码实现：

1

2

3

4

5

6

7

**def** dice\_coefficient(y\_true\_cls, y\_pred\_cls, training\_mask): eps = 1e-5

intersection = tf.reduce\_sum(y\_true\_cls \* y\_pred\_cls \* training\_mask) union = tf.reduce\_sum(y\_true\_cls \* training\_mask) + tf.reduce\_sum(y\_pred loss = 1. - (2 \* intersection / union) tf.summary.scalar('classification\_dice\_loss', loss)

**return** loss

关于dice\_coefficient loss更多的细节

<https://www.aiuai.cn/aifarm1159.html>

**Loss for Geometries** 衡量图形预测好坏的loss设计上，有两个需要考虑的： 1.loss能衡量出形状/坐标的准确程度

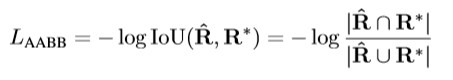
2.具有一定程度的尺度不变性，不能对大文本框有“偏袒”

还是分 RBOX 和 QUAD 两部分讨论

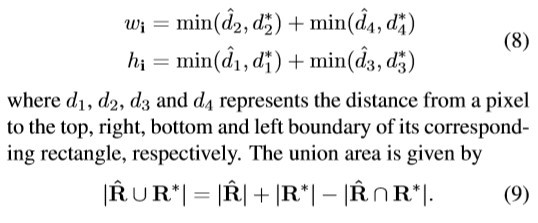
**Loss for RBOX**

RBOX有AABB和angle两部分

AABB部分使用IOU作为loss （理由是IOU本身具有尺度不变性）



在假设角度完全一致的前提下， 可以很容易得到 的 宽 和 高 为：



因此IOU很容易就通过每个pixel处输出的四个距离 d1，d2，d3，d4计算得到 预测角度的loss的计算公式为



因此RBOX的总体loss为



这里又有一个预先设置的超参数，作者取10，代码里是20

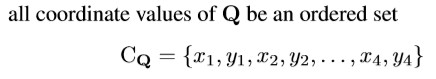
需要注意的是，形状Loss的计算成立是建立在预测的旋转矩形框的角度完全正确的前提 下的。

这块的逻辑我个人认为不太完备，有点绕，而且我感觉这应该也是给角度的权重很大的 原因。

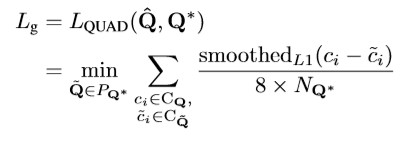
**Loss for QUAD**

QUAD预测的是8个坐标偏移，所以基本思路就是 L1 Loss + 尺度的正则化 作者使用 smoothed L1 Loss，然后用四边形的最短边进行归一化。

首先定义



于是Loss可以被表示为：



**计算loss的完整代码**

* 1. # rbox对应的loss
  2. **def** loss(y\_true\_cls, y\_pred\_cls, y\_true\_geo, y\_pred\_geo, training\_mask):
  3. # 计算score map的loss，cls\_weight为了平衡score map loss和rbox loss
  4. cls\_weight = 0.01
  5. classification\_loss = cls\_weight\*dice\_coefficient(y\_true\_cls, y\_pred\_cls 6

7 # 根据rbox信息，计算四边形最小外接矩形的面积

1. d1\_gt, d2\_gt, d3\_gt, d4\_gt, theta\_gt = tf.split(value=y\_true\_geo, num\_or
2. d1\_pred, d2\_pred, d3\_pred, d4\_pred, theta\_pred = tf.split(value=y\_pred\_g
3. area\_gt = (d1\_gt + d3\_gt) \* (d2\_gt + d4\_gt)
4. area\_pred = (d1\_pred + d3\_pred) \* (d2\_pred + d4\_pred) 12

13 # 计算最小外接矩形的交集

1. w\_union = tf.minimum(d2\_gt, d2\_pred) + tf.minimum(d4\_gt, d4\_pred)
2. h\_union = tf.minimum(d1\_gt, d1\_pred) + tf.minimum(d3\_gt, d3\_pred)
3. area\_intersect = w\_union \* h\_union 17

18 # 计算最小外接矩形的并集

19 area\_union = area\_gt + area\_pred - area\_intersect 20

1. # 计算rbox loss
2. L\_AABB = -tf.log((area\_intersect + 1.0)/(area\_union + 1.0))
3. L\_theta = 1 - tf.cos(theta\_pred - theta\_gt)
4. tf.summary.scalar('geometry\_AABB', tf.reduce\_mean(L\_AABB \* y\_true\_cls \*
5. tf.summary.scalar('geometry\_theta', tf.reduce\_mean(L\_theta \* y\_true\_cls
6. L\_g = L\_AABB + 20 \* L\_theta 27

28

**return** tf.reduce\_mean(L\_g \* y\_true\_cls \* training\_mask) + classification

## 训练

ADAM

batch=24

learning rate 1e-3 decay 1/10 to 1e-5 sample 512\*512 crop to build a batch

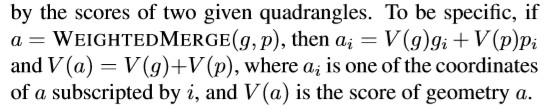
**后处理：Locality-AwareNMS**

因为当前设计的pipeline下，每个像素点都预测了一个文本框 即使使用阈值去掉“负样本”后，框的个数也非常多，正常进行NMS会非常慢 因此需要在进行标准的NMS前需要先把文本框的个数降下来

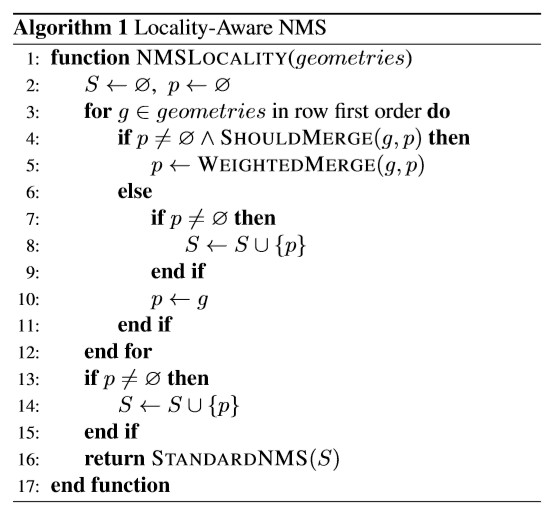
因为位置上临近的像素点本身就很可能是在同一个文本框，所以可以对临近的像素点所 对应的文本框先进行聚合，最后再统一把剩下的框进行NMS

这个过程作者起了个名字 局部感知的NMS，其实和NMS没什么关系，具体看算法细节就 可以发现，操作更应该被叫做投票

合并两个pixel对应的文本框的方法



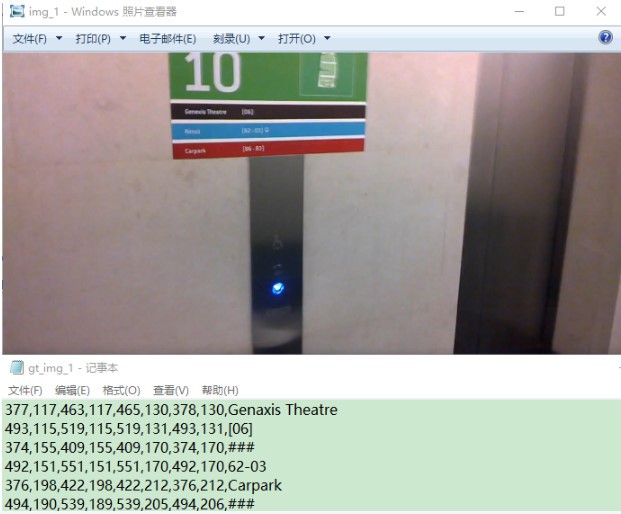
完整的后处理算法流程：



后处理这部分的逻辑也觉得有点奇怪，为什么不求连通区域？

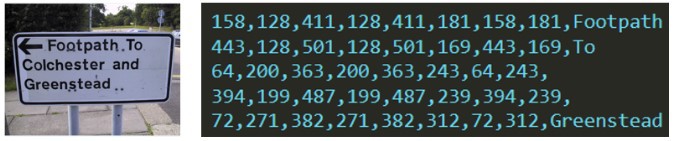
**ICDAR2015数据格式**





总共有1000张图，每张图对应1个txt，txt中每一行的前8个数字分别是文本框的左上x1，

左上y1，右上x2，右上y2，右下x3，右下y3，左下x4，左下y4。最后一列有的是英文有 的是数字，有的是###，不是###的表示文本框里面的内容，因为该数据集是英文的，所 以基本上是英文和数字。是###的表示标记的文本比较模糊，难以辨认。



有的文本框的内容是没有的，只有框的信息

**看下收缩的具体实现 TODO**



**产生四边形的最小外接矩形：**

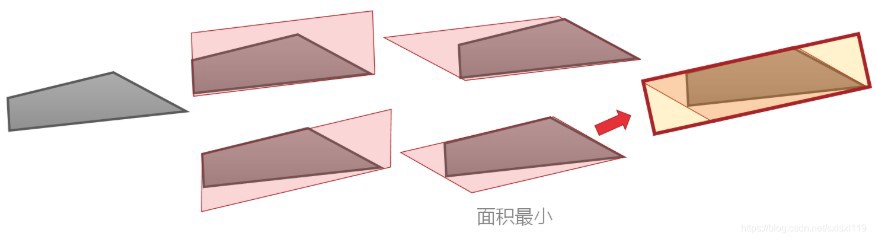


icdar.py generate\_rbox函数

它的功能是把一个任意四边形转成包含四个顶点的最小外接矩形。

该方法首先是**以四边形的任意两个相邻的边为基础**，求出包含四个顶点的**最小 平行四边形**，总共有4个

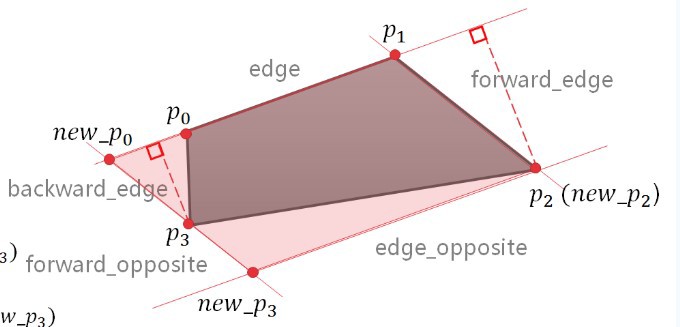
然后选择面积最小的平行四边形，将其转换为矩形



#### 产生单个平行四边形的方法：



这里我们假设要求的是以边（p0, p1）和边（p1, p2）作为参考边的平行四边 形。边（p0, p1）设为edge，边（p1, p2）设为forward\_edge，边p0和p3设 为backward\_edge。首先第一步，先求出点p2和p3到边edge的距离，求出比 较大的那个点，图中p2距离更远，因此选择p2。然后过点p2做一条平行于边 edge的直线，该边我们定义为edge\_opposite。现在，我们就有了平行四边形 的三条边，接下来了来画最后一条边。采用同样的方法对比点p0和p3到直线 forward\_edge的距离，选择距离更远的点，图中是p3，然后过点p3做直线平 行于forward\_edge，最后这条直线称为forward\_opposite。到这里，四条边 都画出来了，分别是edge，forward\_edge，edge\_opposite，和 forward\_opposite，最后根据直线的交点更新4个顶点位置。



**如何判断上下左右？以及计算angle？**



产生最小外接矩形后，外接矩形4个顶点除了要按顺时针排列外 我们还要对每个点的含义做一个约定，才能让模型正确的学习

我们定义 p0-p1，p1-p2，p2-p3，p3-p0 分别代表 上，右，下，左 四个方向的边 具体如何判断上下左右 和 计算angle 需要仔细阅读 sort\_rectangle() 函数

* 1. **def** sort\_rectangle(poly):
  2. # sort the four coordinates of the polygon, points in poly should be sor
  3. # First find the lowest point
  4. p\_lowest = np.argmax(poly[:, 1])
  5. **if** np.count\_nonzero(poly[:, 1] == poly[p\_lowest, 1]) == 2:
  6. # 底边平行于X轴, 那么p0为左上角 - if the bottom line is parallel to x-a
  7. p0\_index = np.argmin(np.sum(poly, axis=1))
  8. p1\_index = (p0\_index + 1) % 4
  9. p2\_index = (p0\_index + 2) % 4
  10. p3\_index = (p0\_index + 3) % 4
  11. **return** poly[[p0\_index, p1\_index, p2\_index, p3\_index]], 0.

##### else:

* 1. # 找到最低点右边的点 - find the point that sits right to the lowest po
  2. p\_lowest\_right = (p\_lowest - 1) % 4
  3. p\_lowest\_left = (p\_lowest + 1) % 4
  4. angle = np.arctan(-(poly[p\_lowest][1] - poly[p\_lowest\_right][1])/(po
  5. # assert angle > 0
  6. **if** angle <= 0:
  7. print(angle, poly[p\_lowest], poly[p\_lowest\_right])
  8. **if** angle/np.pi \* 180 > 45:
  9. # 这个点为p2 - this point is p2
  10. p2\_index = p\_lowest
  11. p1\_index = (p2\_index - 1) % 4
  12. p0\_index = (p2\_index - 2) % 4
  13. p3\_index = (p2\_index + 1) % 4
  14. **return** poly[[p0\_index, p1\_index, p2\_index, p3\_index]], -(np.pi/2

##### else:

* 1. # 这个点为p3 - this point is p3
  2. p3\_index = p\_lowest
  3. p0\_index = (p3\_index + 1) % 4
  4. p1\_index = (p3\_index + 2) % 4
  5. p2\_index = (p3\_index + 3) % 4
  6. **return** poly[[p0\_index, p1\_index, p2\_index, p3\_index]], angle

#### QUAD，如何判断哪个channel对应哪个点？



同理上一个问题的解释

筛选的不错的资料：

<https://blog.csdn.net/sxlsxl119/article/details/103934957>

<https://www.cnblogs.com/lillylin/p/9954981.html>

[https://zhaopeng0103.github.io/%E6%96%87%E6%9C%AC%E6%A3%80%E6%B5%8 B/CVPR2017-](https://zhaopeng0103.github.io/%E6%96%87%E6%9C%AC%E6%A3%80%E6%B5%8B/CVPR2017-%E6%97%B7%E8%A7%86%E6%8F%90%E5%87%BA%E8%87%AA%E7%84%B6%E5%9C%BA%E6%99%AF%E6%96%87%E6%9C%AC%E6%A3%80%E6%B5%8B%E6%96%B9%E6%B3%95%EF%BC%9AEAST/)

[%E6%97%B7%E8%A7%86%E6%8F%90%E5%87%BA%E8%87%AA%E7%84%B6% E5%9C%BA%E6%99%AF%E6%96%87%E6%9C%AC%E6%A3%80%E6%B5%8B%E 6%96%B9%E6%B3%95%EF%BC%9AEAST/](https://zhaopeng0103.github.io/%E6%96%87%E6%9C%AC%E6%A3%80%E6%B5%8B/CVPR2017-%E6%97%B7%E8%A7%86%E6%8F%90%E5%87%BA%E8%87%AA%E7%84%B6%E5%9C%BA%E6%99%AF%E6%96%87%E6%9C%AC%E6%A3%80%E6%B5%8B%E6%96%B9%E6%B3%95%EF%BC%9AEAST/)