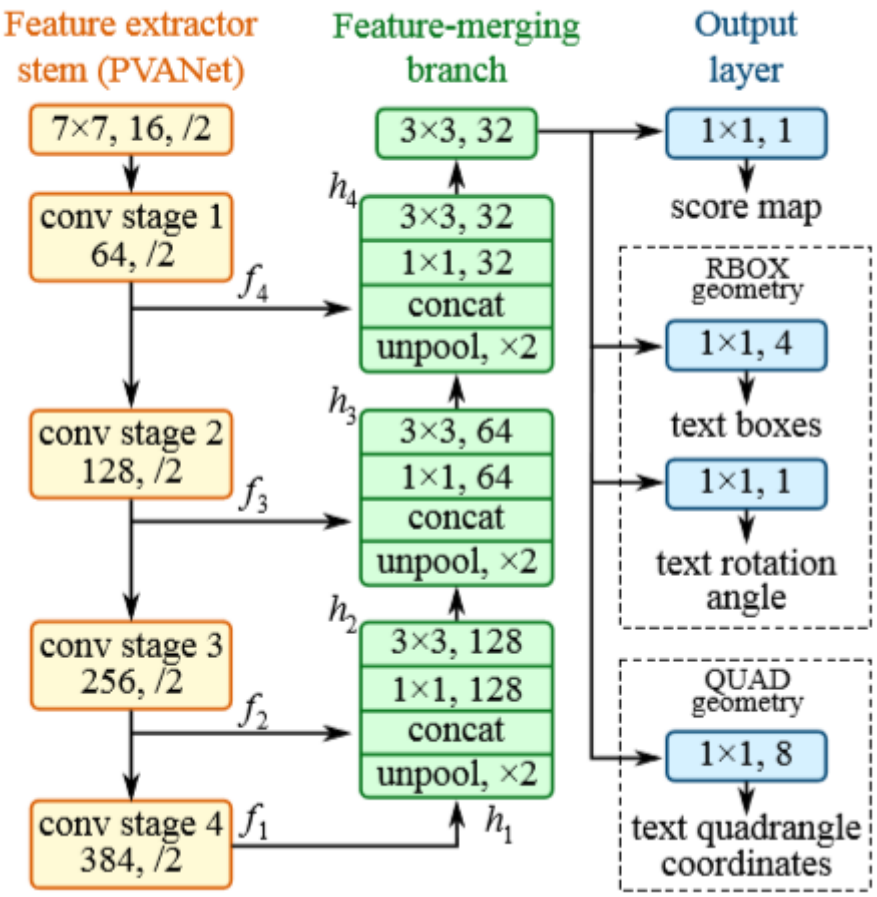
pipeline

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

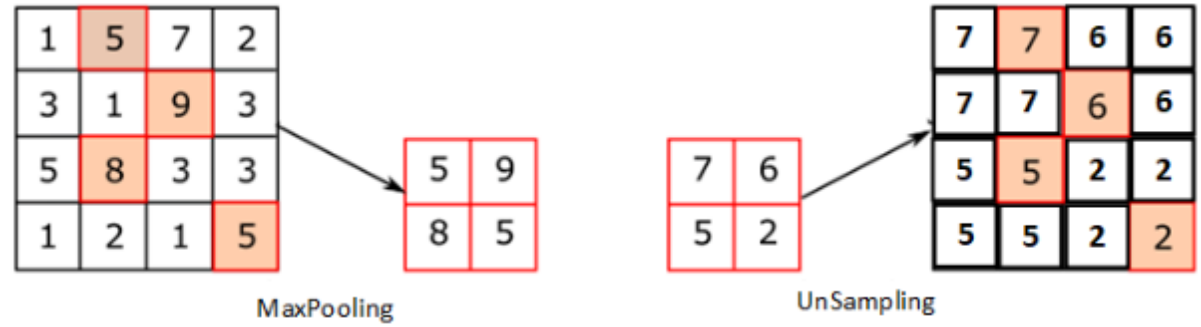
作者实验了 PVANet 和 VGG16

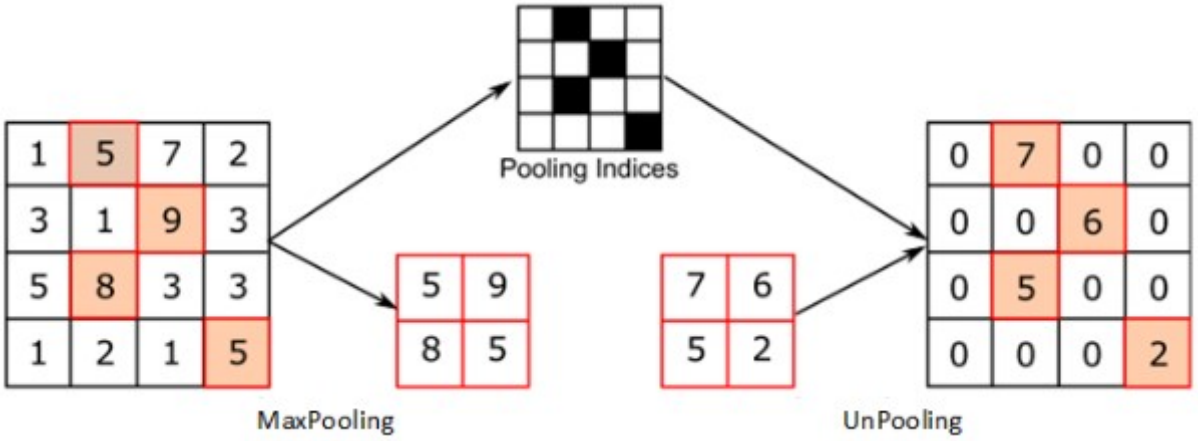
开源代码用的resnet50

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

```
def unpool(inputs):
    return tf.image.resize_bilinear(inputs, size=[tf.shape(inputs)[1]*2,  tf.shape(inputs)[2]*2])
```

模型完整的定义代码：

```
# 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
```

```
    with slim.arg_scope(resnet_v1.resnet_arg_scope(weight_decay=weight_decay)
        logits, end_points = resnet_v1.resnet_v1_50(images, is_training=is_tr

    with tf.variable_scope('feature_fusion', values=[end_points.values]):
        batch_norm_params = {'decay': 0.997, 'epsilon': 1e-5, 'scale': True,

        with slim.arg_scope([slim.conv2d], activation_fn=tf.nn.relu, normaliz
                            normalizer_params=batch_norm_params, weights_regu

            # 特征提取层输出特征
            f = [end_points['pool5'], end_points['pool4'],
                 end_points['pool3'], end_points['pool2']]
            for i in range(4):
                print('Shape of f_{} {}'.format(i, f[i].shape))

            # 合并层搭建
            g = [None, None, None, None]
            h = [None, None, None, None]
            num_outputs = [None, 128, 64, 32]
            for i in range(4):
                if i == 0:
                    h[i] = f[i]
                else:
                    c1_1 = slim.conv2d(tf.concat([g[i-1], f[i]], axis=-1), nu
                    h[i] = slim.conv2d(c1_1, num_outputs[i], 3)
                if i <= 2:
                    g[i] = unpool(h[i])
                else:
                    g[i] = slim.conv2d(h[i], num_outputs[i], 3)
                print('Shape of h_{} {}, g_{} {}'.format(i, h[i].shape, i, g[

            # 输出层搭建
            # here we use a slightly different way for regression part, we fi
            # range, and also this is do with the angle map
            F_score = slim.conv2d(g[3], 1, 1, activation_fn=tf.nn.sigmoid, no
            geo_map = slim.conv2d(g[3], 4, 1, activation_fn=tf.nn.sigmoid, no
            # angle is between [-45, 45]
            angle_map  = (slim.conv2d(g[3], 1, 1, activation_fn=tf.nn.sigmoid
            F_geometry = tf.concat([geo_map, angle_map], axis=-1)
    return F_score, F_geometry
```

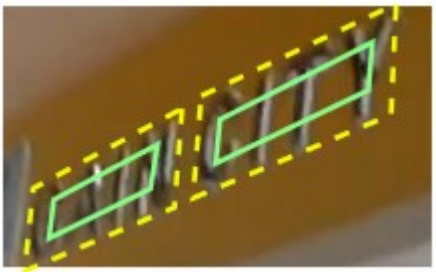
输出层

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

$$\text{For a quadrangle } \mathbf{Q} = \{p_i | i \in \{1, 2, 3, 4\}\}$$

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）

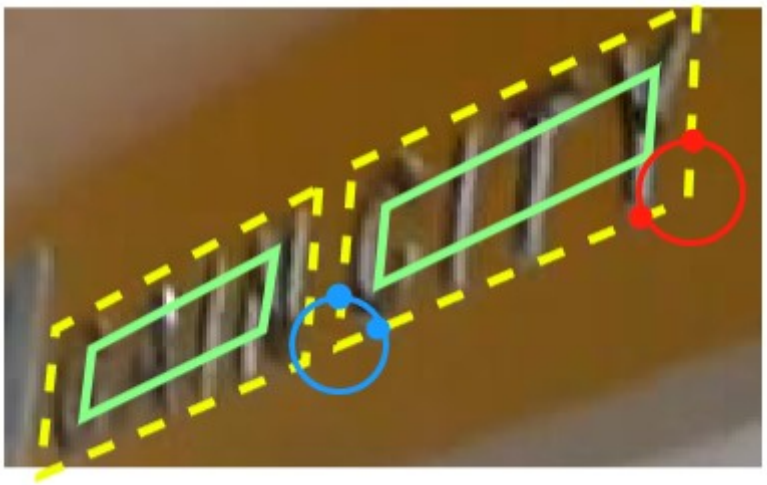
$$r_i = \min(\mathrm{D}(p_i, p_{(i \bmod 4)+1}), \mathrm{D}(p_i, p_{((i+2) \bmod 4)+1}))$$

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

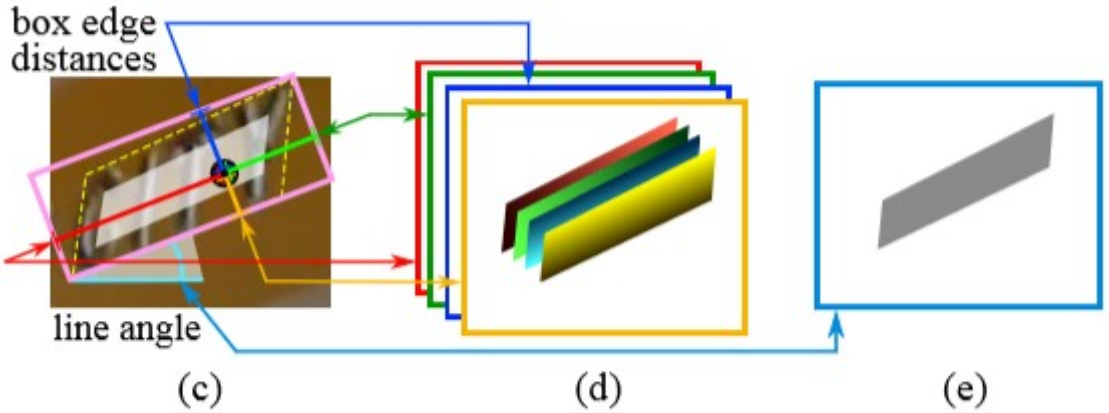
需要根据最终输出 RBOX格式 还是 QUAD格式来分别介绍

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

- [x] **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）

- [x] **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

The loss can be formulated as

$$L = L_s + \lambda_g L_g \tag{4}$$

where L_s and L_g represents the losses for the score map and the geometry, respectively, and λ_g weighs the importance between two losses. In our experiment, we set λ_g to 1.

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

$$\begin{aligned} L_s &= \text{balanced-xent}(\hat{\mathbf{Y}}, \mathbf{Y}^*) \\ &= -\beta \mathbf{Y}^* \log \hat{\mathbf{Y}} - (1-\beta)(1-\mathbf{Y}^*) \log(1-\hat{\mathbf{Y}}) \end{aligned} \tag{5}$$

where $\hat{\mathbf{Y}} = F_s$ is the prediction of the score map, and $\mathbf{Y}^*$ is the ground truth. The parameter β is the balancing factor between positive and negative samples, given by

$$\beta = 1 - \frac{\sum_{y^* \in \mathbf{Y}^*} y^*}{|\mathbf{Y}^*|}. \tag{6}$$

classbalanced cross-entropy 出自Holistically-nested edge detection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

Dice系数是一种集合相似度度量函数，通常用于计算两个样本的相似度，取值范围在[0,1]：

$$s = \frac{2|X \bigcap Y|}{|X| + |Y|}$$

其中 |X∩Y| 是X和Y之间的交集，|X|和|Y|分表表示X和Y的元素的个数，其中，分子的系数为2，是因为分母存在重复计算X和Y之间的共同元素的原因。

Dice Loss:

$$d = 1 - \frac{2|X \bigcap Y|}{|X| + |Y|}$$

代码实现：

```
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）

$$L_{\mathrm{AABB}} = -\log \mathrm{IoU}(\hat{\mathbf{R}}, \mathbf{R}^*) = -\log \frac{|\hat{\mathbf{R}} \cap \mathbf{R}^*|}{|\hat{\mathbf{R}} \cup \mathbf{R}^*|}$$

在假设角度完全一致的前提下，

可以很容易得到 $|\hat{\mathbf{R}} \cap \mathbf{R}^*|$ 的宽和高为:

$$\begin{aligned} w_{\mathbf{i}} &= \min(\hat{d}_2, d_2^*) + \min(\hat{d}_4, d_4^*) \\ h_{\mathbf{i}} &= \min(\hat{d}_1, d_1^*) + \min(\hat{d}_3, d_3^*) \end{aligned} \tag{8}$$

where d_1, d_2, d_3 and d_4 represents the distance from a pixel to the top, right, bottom and left boundary of its corresponding rectangle, respectively. The union area is given by

$$|\hat{\mathbf{R}} \cup \mathbf{R}^*| = |\hat{\mathbf{R}}| + |\mathbf{R}^*| - |\hat{\mathbf{R}} \cap \mathbf{R}^*|. \tag{9}$$

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

$$L_\theta(\hat{\theta}, \theta^*) = 1 - \cos(\hat{\theta} - \theta^*).$$

因此RBOX的总体loss为

$$L_{\mathrm{g}} = L_{\mathrm{AABB}} + \lambda_\theta L_\theta.$$

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

all coordinate values of $\mathbf{Q}$ be an ordered set

$$\mathrm{C}_{\mathbf{Q}} = \{x_1, y_1, x_2, y_2, \ldots, x_4, y_4\}$$

于是Loss可以被表示为：

$$\begin{aligned} L_{\mathrm{g}} &= L_{\mathrm{QUAD}}(\hat{\mathbf{Q}}, \mathbf{Q}^*) \\ &= \min_{\tilde{\mathbf{Q}} \in P_{\mathbf{Q}^*}} \sum_{\substack{c_i \in \mathrm{C}_{\mathbf{Q}}, \\ \tilde{c}_i \in \mathrm{C}_{\tilde{\mathbf{Q}}}}} \frac{\mathrm{smoothed}_{L1}(c_i - \tilde{c}_i)}{8 \times N_{\mathbf{Q}^*}} \end{aligned}$$

计算loss的完整代码

```
# rbox对应的loss
def loss(y_true_cls, y_pred_cls, y_true_geo, y_pred_geo, training_mask):
    # 计算score map的loss，cls_weight为了平衡score map loss和rbox loss
    cls_weight = 0.01
    classification_loss = cls_weight*dice_coefficient(y_true_cls, y_pred_cls,

    # 根据rbox信息，计算四边形最小外接矩形的面积
    d1_gt, d2_gt, d3_gt, d4_gt, theta_gt = tf.split(value=y_true_geo, num_or_
    d1_pred, d2_pred, d3_pred, d4_pred, theta_pred = tf.split(value=y_pred_ge
    area_gt = (d1_gt + d3_gt) * (d2_gt + d4_gt)
    area_pred = (d1_pred + d3_pred) * (d2_pred + d4_pred)

    # 计算最小外接矩形的交集
    w_union = tf.minimum(d2_gt, d2_pred) + tf.minimum(d4_gt, d4_pred)
    h_union = tf.minimum(d1_gt, d1_pred) + tf.minimum(d3_gt, d3_pred)
    area_intersect = w_union * h_union

    # 计算最小外接矩形的并集
    area_union = area_gt + area_pred - area_intersect

    # 计算rbox loss
    L_AABB = -tf.log((area_intersect + 1.0)/(area_union + 1.0))
    L_theta = 1 - tf.cos(theta_pred - theta_gt)
    tf.summary.scalar('geometry_AABB', tf.reduce_mean(L_AABB * y_true_cls * t
    tf.summary.scalar('geometry_theta', tf.reduce_mean(L_theta * y_true_cls *
    L_g = L_AABB + 20 * L_theta

```

```
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

by the scores of two given quadrangles. To be specific, if $a = \text{WEIGHTEDMERGE}(g, p)$, then $a_i = V(g)g_i + V(p)p_i$ and $V(a) = V(g)+V(p)$, where a_i is one of the coordinates of a subscripted by i, and $V(a)$ is the score of geometry a.

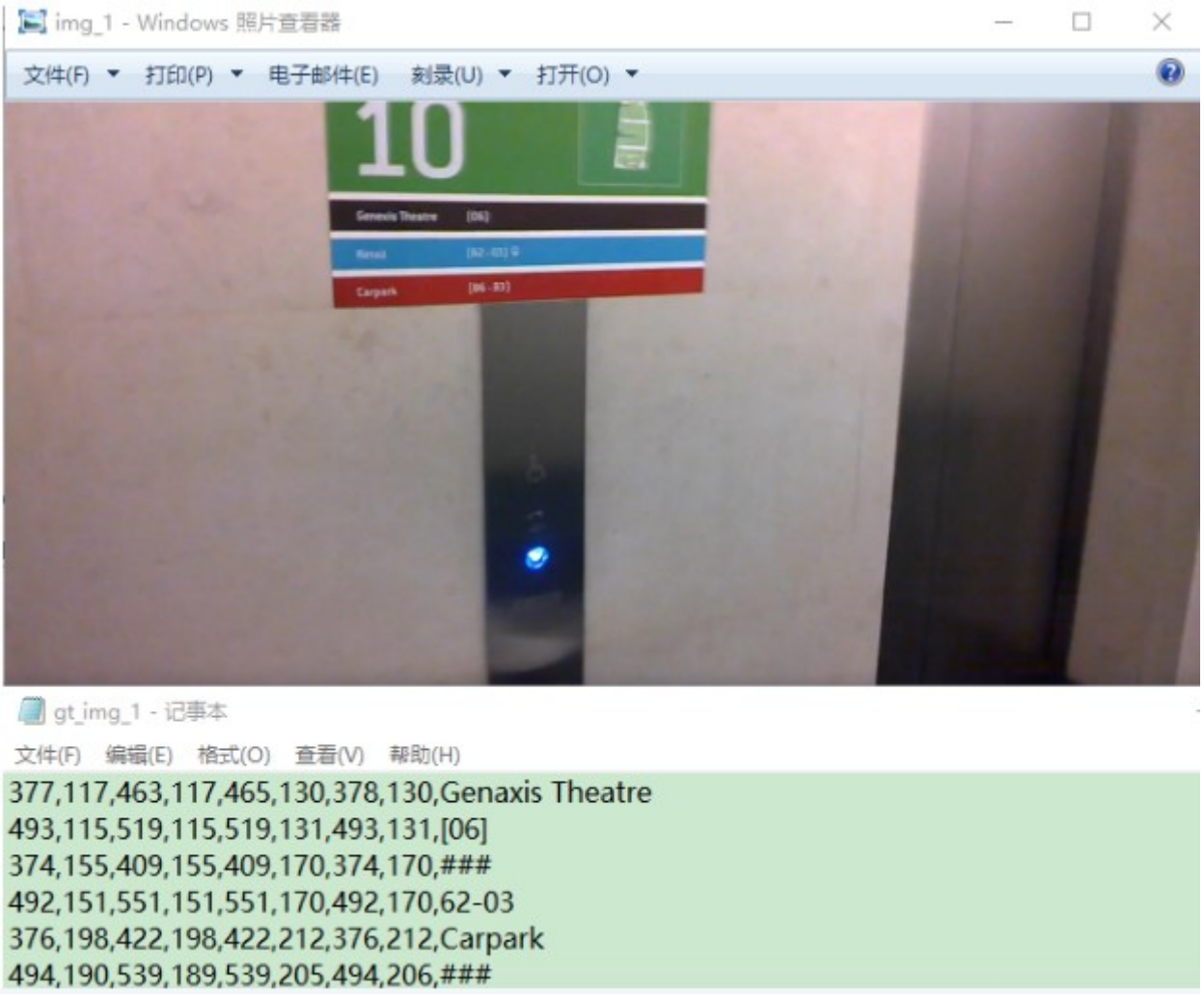
完整的后处理算法流程：

Algorithm 1 Locality-Aware NMS

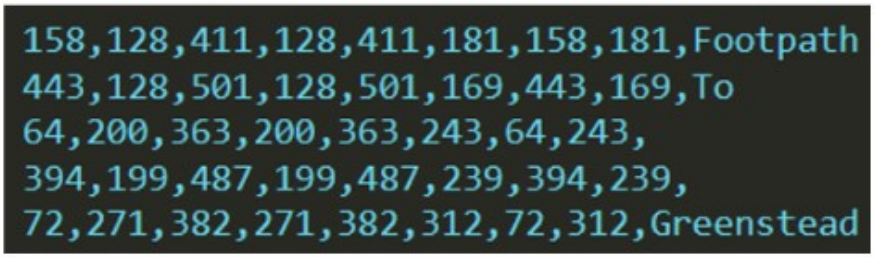
```
function NMSLocality(geometries)
    S ← ∅, p ← ∅
    for g ∈ geometries in row first order do
        if p ≠ ∅ ∧ ShouldMerge(g, p) then
            p ← WeightedMerge(g, p)
        else
            if p ≠ ∅ then
                S ← S ∪ {p}
            end if
            p ← g
        end if
    end for
    if p ≠ ∅ then
        S ← S ∪ {p}
    end if
    return StandardNMS(S)
end function
```

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

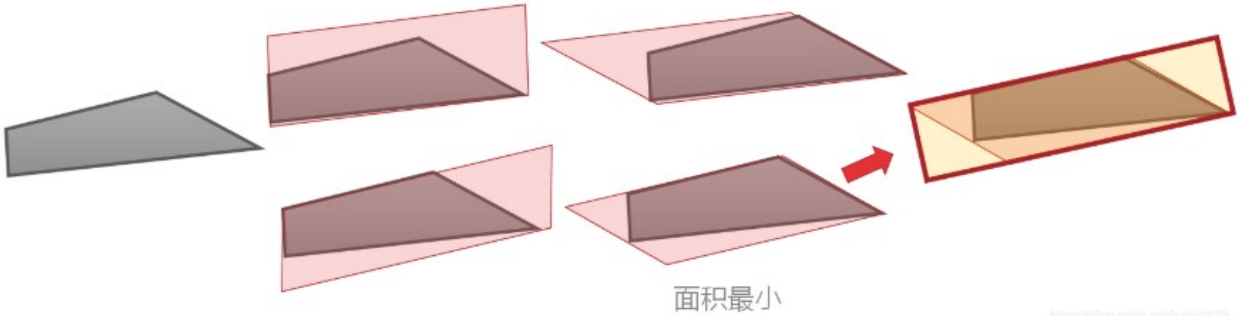
- [ ] **看下收缩的具体实现 TODO**

- [x] **产生四边形的最小外接矩形：**

icdar.py generate_rbox函数

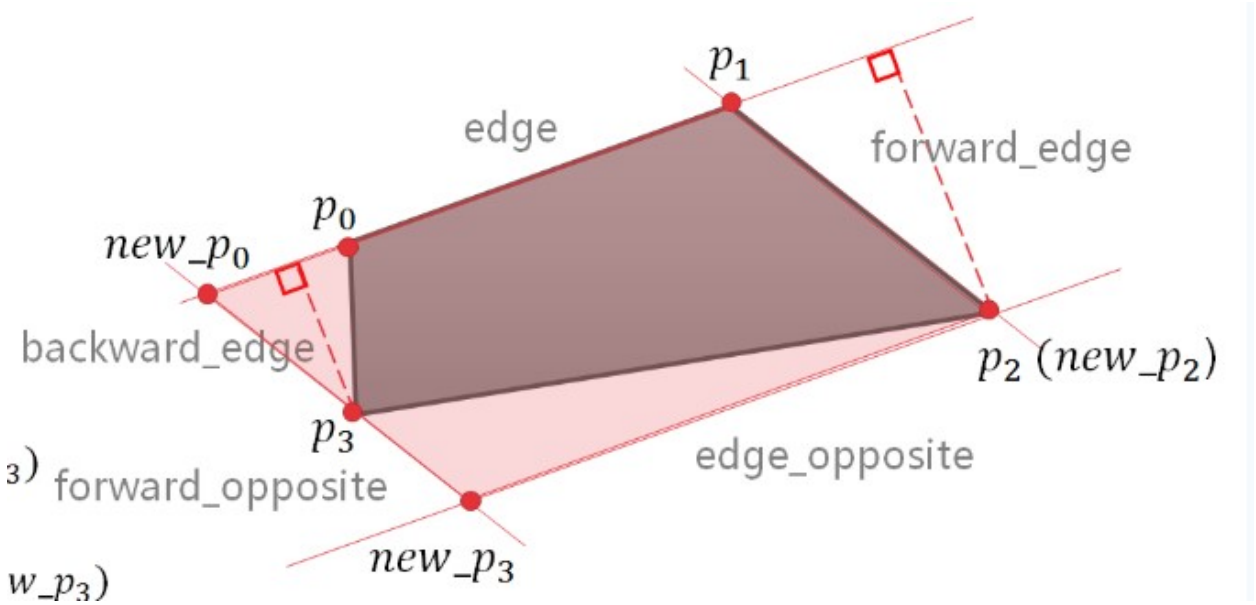
它的功能是把一个任意四边形转成包含四个顶点的最小外接矩形。
该方法首先是**以四边形的任意两个相邻的边为基础**，求出包含四个顶点的**最小平行四边形**，总共有4个

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

```python
def sort_rectangle(poly):
    # sort the four coordinates of the polygon, points in poly should be sort
    # First find the lowest point
    p_lowest = np.argmax(poly[:, 1])
    if np.count_nonzero(poly[:, 1] == poly[p_lowest, 1]) == 2:
        # 底边平行于X轴，那么p0为左上角 - if the bottom line is parallel to x-a
        p0_index = np.argmin(np.sum(poly, axis=1))
        p1_index = (p0_index + 1) % 4
        p2_index = (p0_index + 2) % 4
        p3_index = (p0_index + 3) % 4
        return poly[[p0_index, p1_index, p2_index, p3_index]], 0.
    else:
        # 找到最低点右边的点 - find the point that sits right to the lowest po
        p_lowest_right = (p_lowest - 1) % 4
        p_lowest_left = (p_lowest + 1) % 4
        angle = np.arctan(-(poly[p_lowest][1] - poly[p_lowest_right][1])/(pol
        # assert angle > 0
        if angle <= 0:
            print(angle, poly[p_lowest], poly[p_lowest_right])
        if angle/np.pi * 180 > 45:
            # 这个点为p2 - this point is p2
            p2_index = p_lowest
            p1_index = (p2_index - 1) % 4
            p0_index = (p2_index - 2) % 4
            p3_index = (p2_index + 1) % 4
            return poly[[p0_index, p1_index, p2_index, p3_index]], -(np.pi/2
        else:
            # 这个点为p3 - this point is p3
            p3_index = p_lowest
            p0_index = (p3_index + 1) % 4
            p1_index = (p3_index + 2) % 4
            p2_index = (p3_index + 3) % 4
            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%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/