Detecting Text in Natural Image withConnectionist Text Proposal Network

目录

[论文Ideas 3](#_Toc5870424)

[整体框架 3](#_Toc5870425)

[Details 4](#_Toc5870426)

[训练 6](#_Toc5870427)

[实验结果 6](#_Toc5870428)

[附录 6](#_Toc5870429)

[6](#_Toc5870430)

[CTPN可以改进的位置 10](#_Toc5870431)

# 论文Ideas

1. 目前的文本检测方法大多采用自下而上(bottom-up)的流程, 它们通常从低级别字符或笔画检测开始，后面通常会跟随一些后续步骤：非文本组件过滤，文本行构建和文本行验证。这些自底向上的多步骤方法通常复杂，鲁棒性和可靠性较差。**本论文采用的是先检测文本区域,再找出文本线的方法。**
2. 文本检测和一般目标检测的不同——**文本检测的文本线是一个序列**（字符、字符的一部分、多字符组成的一个sequence），而不是一般目标检测中只有一个独立的目标。这既是优势，也是难点。优势体现在同一文本线上不同字符可以互相利用上下文，可以用sequence的方法比如RNN来表示。**难点**体现在要检测出一个完整的文本线，同一文本线上不同字符可能差异大，距离远，要作为一个整体检测出来难度比单个目标更大——因此，**作者认为预测文本的竖直位置（文本bounding box的上下边界）比水平位置（文本bounding box的左右边界）更容易。**
3. **RNN和CNN的无缝结合可以提高检测精度。**CNN用来提取深度特征，RNN用来序列的特征识别（2类），二者无缝结合，用在检测上性能更好。

# 整体框架

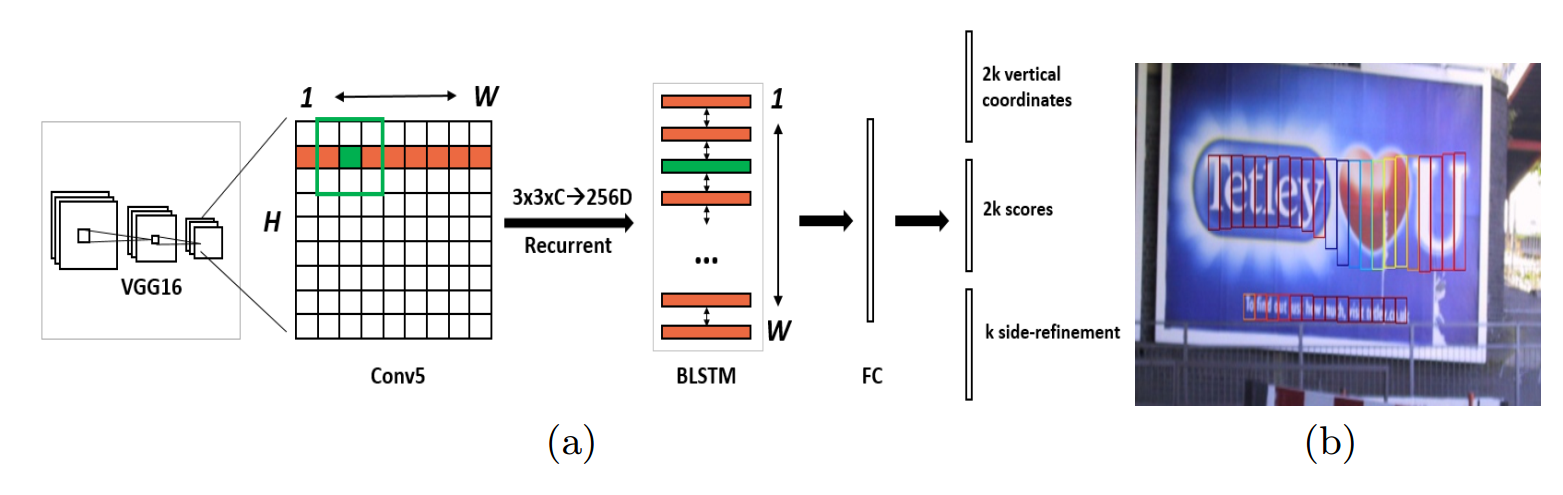


Fig.1

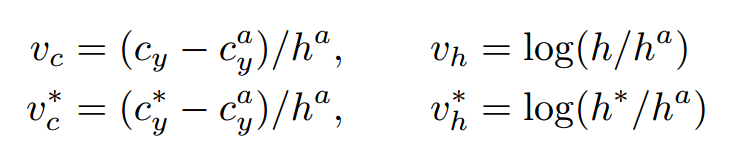
**基本流程如上图：**

1. 用VGG16的Conv5得到feature map（W\*H\*C）。
2. 在Conv5的feature map的每个位置上取（3\*3\*C）的窗口的特征，这些特征将用于预测该位置k个anchor（anchor的定义和Faster RCNN类似）对应的类别信息，位置信息。
3. 将每一行的所有窗口对应的3\*3\*C的特征（W\*3\*3\*C）输入到RNN（BLSTM）中，得到W\*256的输出。
4. 将RNN的W\*256输入到512维的fc层。
5. fc层特征输入到三个分类或者回归层中。第二个2k scores 表示的是k个anchor的类别信息（是字符或不是字符）。第一个2k vertical coordinate和第三个k side-refinement是用来回归k个anchor的位置信息。2k vertical coordinate表示的是bounding box的高度和中心的y轴坐标（可以决定上下边界），k个side-refinement表示的bounding box的水平平移量。这边注意，只用了3个参数表示回归的bounding box，因为这里默认了每个anchor的width是16，且不再变化（VGG16的conv5的stride是16）。回归出来的box如Fig.1中那些红色的细长矩形，它们的宽度是一定的。
6. 用简单的文本线构造算法，把分类得到的文字的proposal（图Fig.1（b）中的细长的矩形）合并成文本线

## Details

1. Detecting Text in Fine-scale Proposals

* k个anchor尺度和长宽比设置：宽度都是16，建议k = 10，高度从11~273（每次除于0.7）
* 回归的高度和bounding box的中心的y坐标如下，带\*的表示是groundTruth，带a的表示是anchor



其中和分别对应着预测坐标与ground truth坐标。

是anchor box的轴的中心与anchor box的高,可以从输入图像中预先计算的。

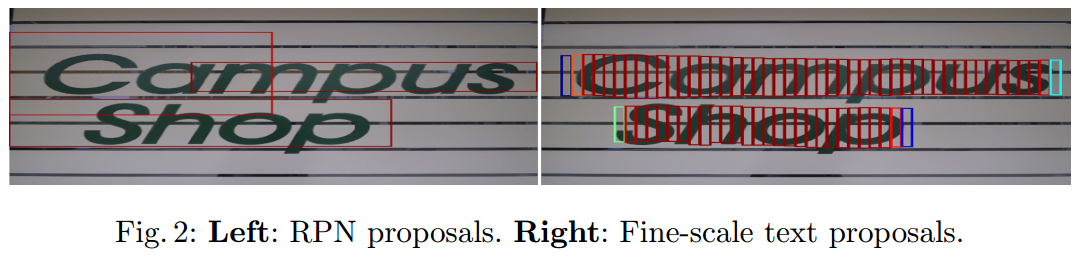
是预测出来的anchor box 轴的中心与预测出的anchor box的高。

是ground truth的y轴中心与高。

* score阈值设置：0.7 （+NMS）

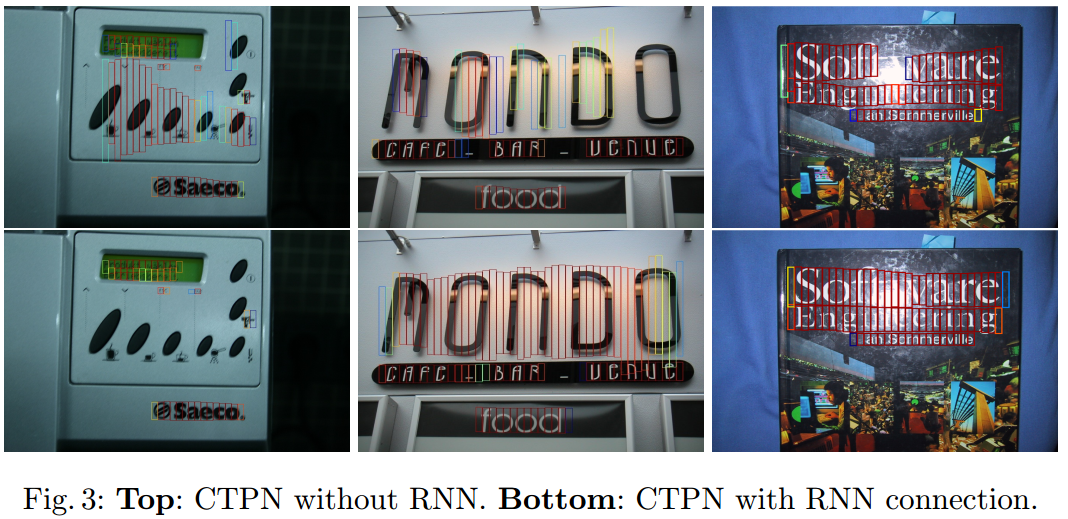
ps. 负样本(negative anchors)的定义与Ground Truth Box IoU小于0.5的anchor.

* 一般的RPN和采用本文的方法检测出的效果对比



1. Recurrent Connectionist Text Proposals

* RNN类型：BLSTM（双向LSTM），每个LSTM有128个隐含层
* RNN输入：每个滑动窗口的3\*3\*C的特征（可以拉成一列），同一行的窗口的特征形成一个序列
* RNN输出：每个窗口对应256维特征
* 使用RNN和不适用RNN的效果对比，CTPN是本文的方法（Connectionist Text Proposal Network）

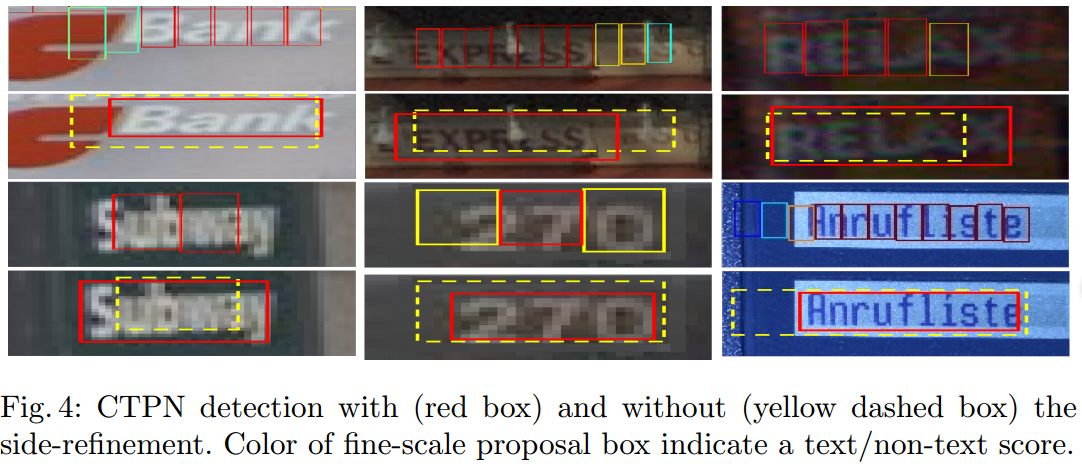


1. Side-refinement

* 文本线构造算法（多个细长的proposal合并成一条文本线）
  + 主要思想：每两个相近的proposal组成一个pair，合并不同的pair直到无法再合并为止（没有公共元素）
  + 判断两个proposal，Bi和Bj组成pair的条件：
    - ， 且。（表示是的最好邻居）
    - 条件1：是的邻居中距离Bi最近的，且该距离小于50个像素
    - 条件2：和的vertical overlap大于0.7
* 固定要regression的box的宽度和水平位置会导致predict的box的水平位置不准确，所以作者引入了side-refinement，用于水平位置的regression。是最接近水平边到当前锚点的预测的x坐标.是x轴的GT边缘坐标. 是x轴的锚点的中心. 是固定的锚点宽度, = 16

https://images2015.cnblogs.com/blog/1058268/201701/1058268-20170112212912588-2132674610.png

* 使用side-refinement的效果对比

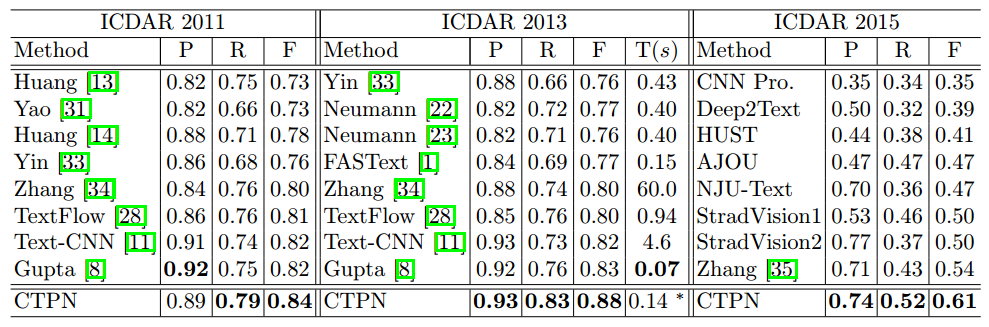


# 训练

* Optimizer: SGD
* 训练标签: 对于文本/非文本分类，二值标签分配给每个正（文本）锚点或负（非文本）锚点。它通过计算与实际边界框的重叠（除以锚点位置）来定义。正锚点被定义为：（i）与任何实际边界框具有>0.7的重叠；或者（ii）与实际边界框具有最高重叠。通过条件（ii），即使是非常小的文本模式也可以分为正锚点。这对于检测小规模文本模式至关重要，这是CTPN的主要优势之一。这不同于通用目标检测，通用目标检测中条件（ii）的影响可能不显著。负锚点定义为与所有实际边界框具有<0.5的重叠。
* 训练数据: 在训练过程中，每个小批量样本从单张图像中随机收集。每个小批量数据的锚点数量固定为，正负样本的比例为1：1。如果正样本的数量少于64，则会用小图像块填充负样本。

# 实验结果

* 时间：0.14s with GPU
* ICDAR2011，ICDAR2013，ICDAR2015库上检测结果

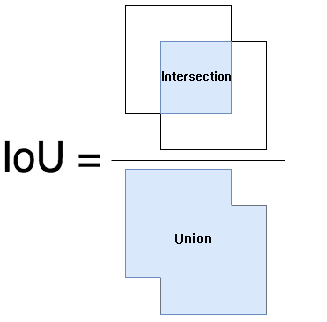


# 附录

## 

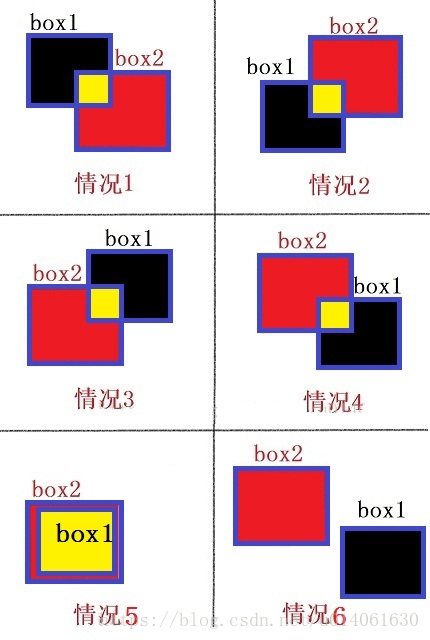
1. 的简介及原理解析

的全称为交并比（Intersection over Union），通过这个名称我们大概可以猜到 的计算方法。计算的是 “预测的边框” 和 “真实的边框” 的交集和并集的比值。



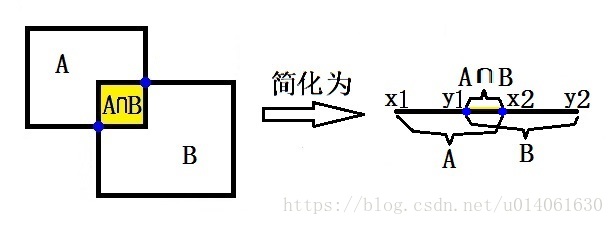
开始计算之前，我们首先进行分析下交集和并集到底应该怎么计算：我们首先需要计算交集，然后并集通过两个边框的面积的和减去交集部分即为并集，因此 的计算的难点在于交集的计算。

为了计算交集，你脑子里首先想到的方法应该是：考虑两个边框的相对位置，然后按照相对位置（左上，左下，右上，右下，包含，互不相交）分情况讨论，来计算交集。



上图就是你的直觉，这样想没有错。但计算一个交集，就要分多种情况讨论，要是程序真的按照这逻辑编写就太搞笑了。因此对这个问题进行进一步地研究显得十分有必要。

让我们重新思考一下两个框交集的计算。两个框交集的计算的实质是两个集合交集的计算，因此我们可以将两个框的交集的计算简化为：



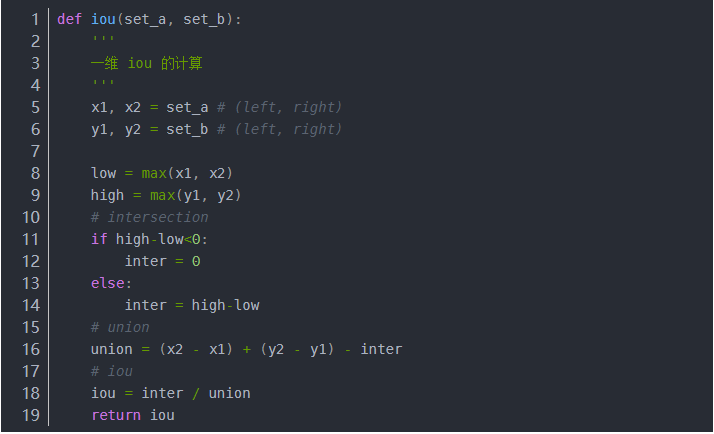
通过简化，我们可以清晰地看到，交集计算的关键是交集上下界点（图中蓝点）的计算。

我们假设集合A为, 集合 B, 为然后我们来求AB交集的上下界限。

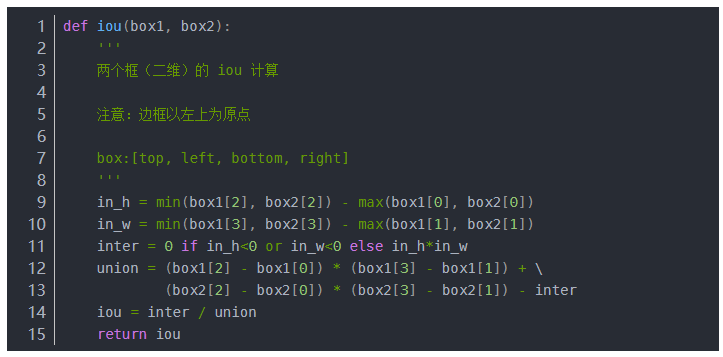
交集计算的逻辑

* 交集下界
* 交集上界
* 如果小于0,则说明集合A和集合B没有交集

下面使用Python来实现两个一维集合的 的计算：



上面，我们计算了两个一维集合的，将上面的程序进行扩展，即可得到两个框 计算的程序。



2. 基于TensorFlow的实现

上节介绍了，及其的计算，下面我们给出其在 TensorFlow 上的实现：

## 对CTPN的思考

CTPN的单字符效果不是很好,如模型测试文书的结果.

以下为思考出的几个可以改进的方向:

1. 对CTPN的text proposal的参数进行调整,将proposal的最小值微调.
2. 对CTPN的LSTM进行改进,使用CNN或者transformer加快其速度.
3. CTPN后接Dense Net + CTC提高模型精确度