OCR 项目实践

文字识别也是图像领域一个常见问题。然而，对于自然场景图像，首先要定位图像中的文字位置，然后才能进行识别。

所以一般来说，从自然场景图片中进行文字识别，需要包括 2 个步骤：

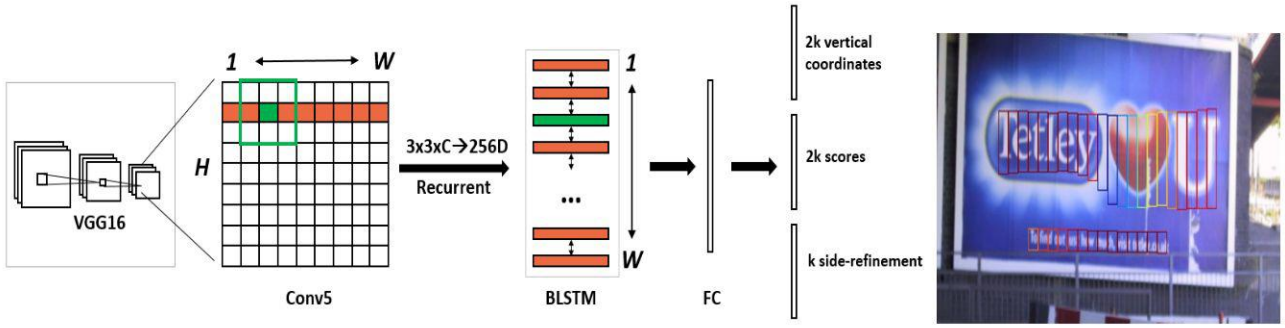
文字检测：解决的问题是哪里有文字，文字的范围有多少

文字识别：对定位好的文字区域进行识别，主要解决的问题是每个文字是什么，将图像中的文字区域进转化为字符信息。

1、CTPN 原理——文字检测

1.1、简介

CTPN 是在 ECCV 2016 提出的一种文字检测算法。CTPN 结合 CNN 与 LSTM 深度网络，能有效的检测出复杂场景的横向分布的文字，效果如下图，是目前比较好的文字检测算法。



CTPN 算法的提出，出于以下几点：(1)、假设文本是水平的；(2)、文本可以看做由每一个“字母”组成的。这里的字母可以认为是小片段。之所以有这样的想法，是因为基于通用目标检测的算法难以适应文字检测的场景，如上图中的文字，长度方面变化幅度很大。因此作者将文本在水平方向解耦，分成每一个小片，然后将文本行的检测转化为小片的检测，最后利用规则将属于同一水平行的小片组合成文本行。化繁为简。

1.2、CTPN 模型创新点

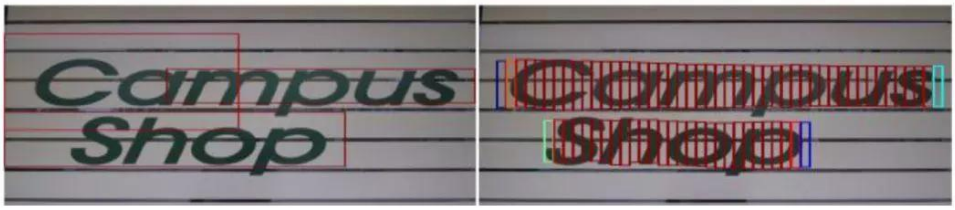
CTPN 的创新点主要由以下三点：

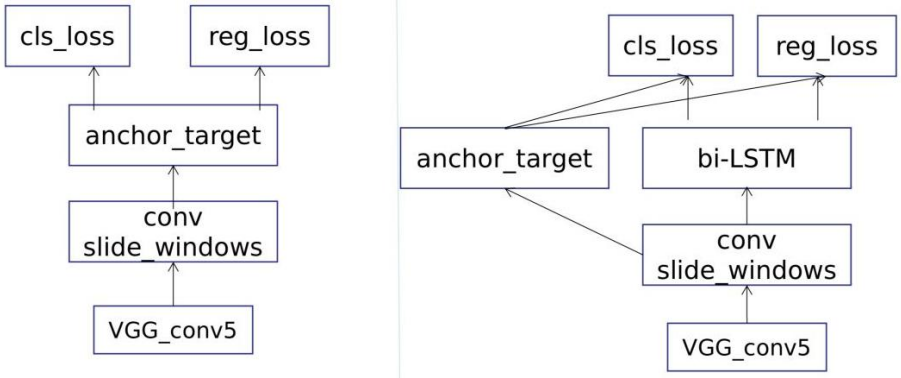
(1)、将文本行拆分为 slice 进行检测，这样在检测过程中只需要对文本的高度进行先验性的设置 anchor。

(2)、作者认为文本具有时序性，即和阅读习惯一直，从左到右。因此作者加入 RNN 获取这种语义性。

(3)、后处理算法：文本连接算法

1.3、CTPN 与 RPN 网络结构的差异





如上图所示，作图为 RPN，右图为 CTPN 的网络结构。可以看到，CTPN 本身就是 RPN，唯一不同的是加入了双向 LSTM 获取时序方向的信息，使得模型可以序列性的预测文本的小片。

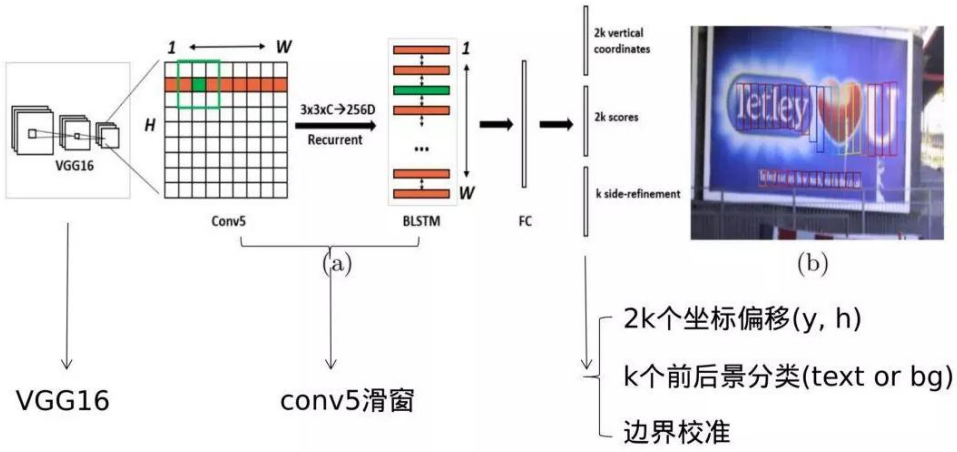
当然这里的不同之处主要有以下几点：

(1)、双向 LSTM 对文本行方向编码

(2)、标签构造方式不同：CTPN 使用水平方向的切片框作为回归目标

1.4、CTPN 网络结构

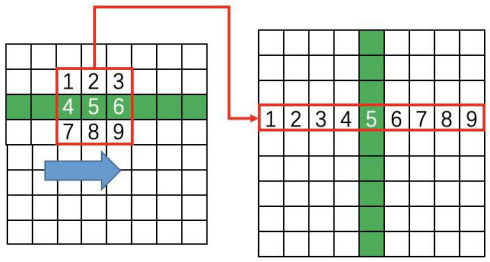
原始 CTPN 只检测横向排列的文字。CTPN 结构与 Faster R-CNN 基本类似，但是加入了 LSTM 层（CNN 学习的是感受野内的空间信息，LSTM 学习的是序列特征。对于文本序列检测，显然既需要 CNN 抽象空间特征，也需要序列特征，毕竟文字是连续的）。假设输入 N Images：



CTPN 的整体结构与流程：

1.首先通过 BackBone 架构网络 VGG16 进行特征的提取，其 Conv5 层输出 N x C x H x W 的特征图，由于 VGG16 的卷积网络中经过 4 个池化层累计的 Stride 为 16。也就是 Conv5 层输出的 Feature map 中一个像素对应原图的 16 像素。

2.然后在 Conv5 上做 3 x 3 的滑动窗口，即每个点都结合周围 3 x 3 区域特征获取一个长度为 3 x 3 x C 的特征向量。如下图所示，输出为 N x 9C x H x W 的 Feature map，该特征依然是由 CNN 学习到的空间特征。



3.之后继续对上一步输出的 Feature map 进行 Reshape 操作：

Reshape：N x 9C x H x W → (NH) x W x 9C

4.然后以 Batch = NH 且最大时间长度 T_{max}=W 的数据流输入 Bi-LSTM，学习每一行的序列特征。Bi-LSTM 输出为(N H) x W x 256，再经 Reshape 回复形状：

Reshape：(NH) x W x 256 → N x 256 x H x W

该特征既包含了空间特征，也包含了 Bi-LSTM 学习到的序列特征。

5.再然后经过“FC”层，变为 N x 512 x H x W 的特征

6.最后经过类似 Faster RCNN 的 RPN 网络，获得 Text Proposals。

Bi-LSTM 的输出输入至 FC 中，最终模型三个输出：

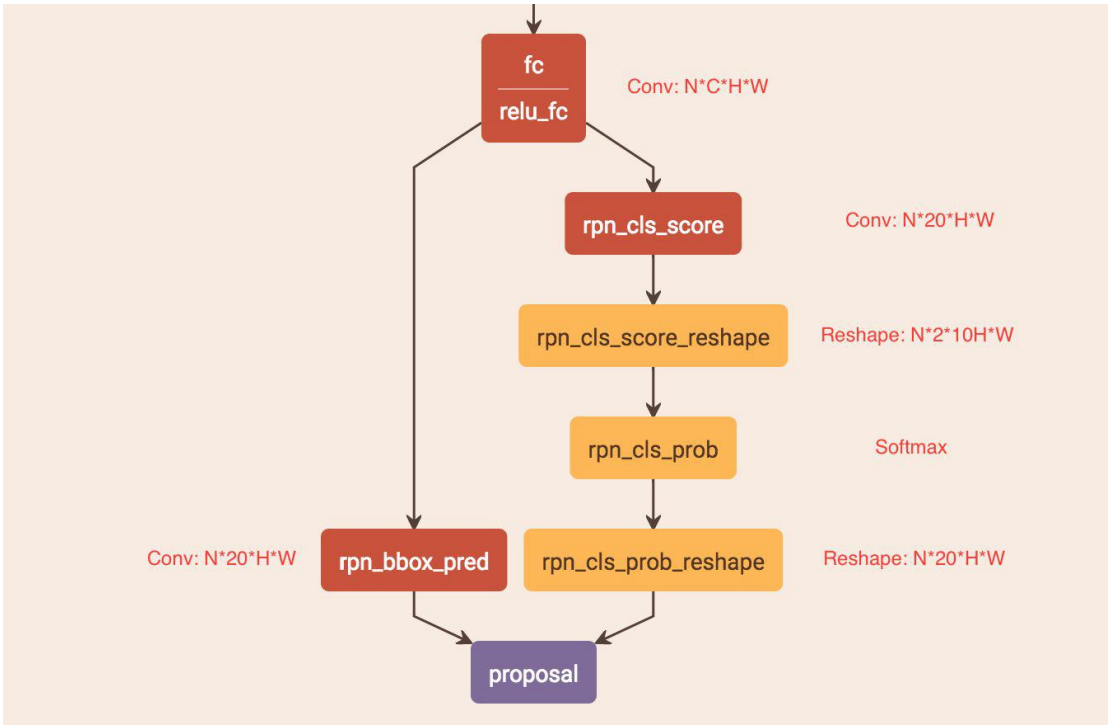
文本小片的坐标偏移(y, h)。这里作者没有对起始坐标进行预测，因为这部分在标签构造过程有固定的偏移，因此只需要知道文本的 y, h，利用固定的偏移可以构造出完整的文本行。

1.5、如何通过 FC 层输出产生 Text proposals?

CTPN 通过 CNN 和 BLSTM 学到一组“空间 + 序列”特征后，在"FC"卷积层后接入 RPN 网络。这里的 RPN 与 Faster R-CNN 类似，分为两个分支：

左边分支用于 Bounding Box Regression。由于 FC Feature map 每个点配备了 10 个 Anchor，同时只回归中心 y 坐标与高度 2 个值，所以 RPN_bboxp_red 有 20 个 Channels

右边分支用于 Softmax 分类 Anchor



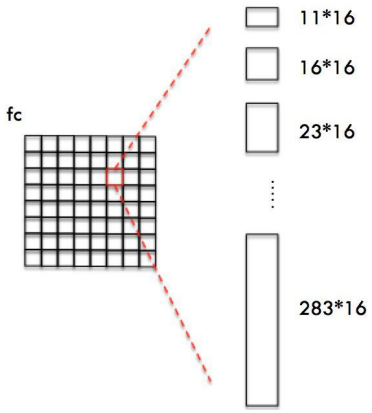
1.6、竖直 Anchor 定位文字位置

由于 CTPN 针对的是横向排列的文字检测，所以其采用了一组（10 个）等宽度的 Anchors，用于定位文字位置。Anchor 宽高为：

$$\text{widths} = [16]$$

$$\text{heights} = [11, 16, 23, 33, 48, 68, 97, 139, 198, 283]$$

需要注意，由于 CTPN 采用 VGG16 模型提取特征，那么 Conv5 Feature map 的宽高都是输入 Image 的宽高的 1/16。同时 FC 与 Conv5 width 和 height 都相等。如下图所示，CTPN 为 FC Feature map 每一个点都配备 10 个上述 Anchors。



这样设置 Anchors 是为了：

1.保证在 x 方向上，Anchor 覆盖原图每个点且不相互重叠。

2.不同文本在 y 方向上高度差距很大，所以设置 Anchors 高度为 11-283，用于覆盖不同高度的文本目标。

注意：Anchor 大小为什么对应原图尺度，而不是 conv5/fc 特征尺度？

这是因为 Anchor 是目标的候选框，经过后续分类+位置修正获得目标在原图尺度的检测框。那么这就要求 Anchor 必须是对应原图尺度！除此之外，如果 Anchor 大小对应 conv5/FC 尺度，那就要求 Bounding box regression 把很小的框回归到很大，这已经超出 Regression 小范围修正框的设计目的。

获得 Anchor 后，与 Faster R-CNN 类似，CTPN 会做如下处理：

1.Softmax 判断 Anchor 中是否包含文本，即选出 Softmax Score 大的正 Anchor；

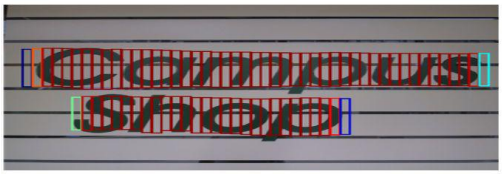
2.Bounding box regression 修正包含文本的 Anchor 的中心 y 坐标与高度。

注意，与 Faster R-CNN 不同的是，这里 Bounding box regression 不修正 Anchor 中心 x 坐标和宽度。具体回归方式如下：

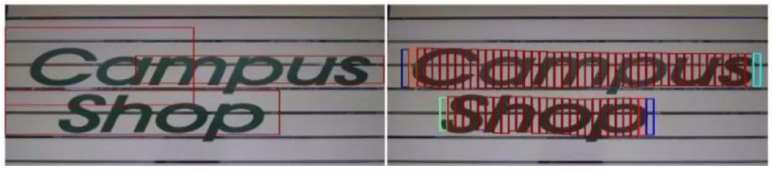
$$v_c = (c_y - c_y^a)/h^a, \qquad v_h = \log(h/h^a)$$
$$v_c^* = (c_y^* - c_y^a)/h^a, \qquad v_h^* = \log(h^*/h^a)$$

其中，v=(v_c,v_h)是回归预测的坐标，v=(v^{*}_{c},v^{*}_{h})是 Ground Truth，c^{a}_{y}和 h^{a}是 Anchor 的中心 y 坐标和高度。Bounding box regression 具体原理请参考之前文章。

Anchor 经过上述 Softmax 和 方向 bounding box regeression 处理后，会获得下图所示的一组竖直条状 text proposal。后续只需要将这些 text proposal 用文本线构造算法连接在一起即可获得文本位置。

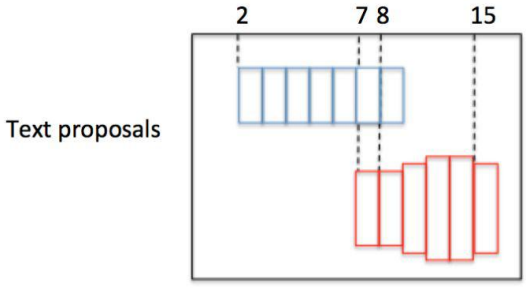


在论文中，作者也给出了直接使用 Faster R-CNN RPN 生成普通 proposal 与 CTPN LSTM+竖直 Anchor 生成 text proposal 的对比，如图 8，明显可以看到 CTPN 这种方法更适合文字检测。



1.7、文本线构造算法

在上一个步骤中，已经获得了一串或多串 text proposal，接下来就要采用文本线构造办法，把这些 text proposal 连接成一个文本检测框。



为了说明问题，假设某张图有上图所示的 2 个 text proposal，即蓝色和红色 2 组 Anchor，CTPN 采用如下算法构造文本线：

1.按照水平 x 坐标排序 Anchor;

2.按照规则依次计算每个 Anchor box_i 的 pair(box_j)，组成 pair(box_i,box_j);

3.通过 pair(box_i,box_j)建立一个 Connect graph，最终获得文本检测框.

下面详细解释。假设每个 Anchor index 如绿色数字，同时每个 Anchor Softmax score 如黑色数字。

文本线构造算法通过如下方式建立每个 Anchor box_i 的 pair(box_i,box_j)：

正向寻找：

1.沿水平正方向，寻找和 **box_i** 水平距离小于 50 的候选 Anchor;

2.从候选 Anchor 中，挑出与 **box_i** 竖直方向 **$overlap_v$** > 0.7 的 Anchor;

3.挑出符合条件 2 中 Softmax score 最大的 **box_j**

再反向寻找：

1.沿水平负方向，寻找和 **box_j** 水平距离小于 50 的候选 Anchor;

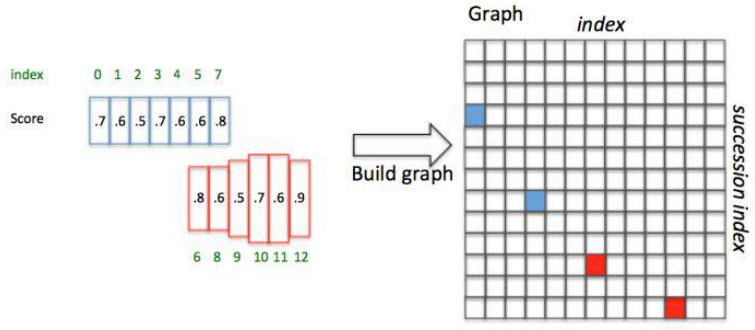
2.从候选 Anchor 中，挑出与 **box_j** 竖直方向 **$overlap_v$** > 0.7 的 Anchor;

3.挑出符合条件 2 中 Softmax score 最大的 **box_k**

最后对比 $score_i$ 和 $score_k$:

1.如果 **$score_i$ >= $score_k$**，则这是一个最长连接，那么设置 Graph(i,j)=True;

2.如果 **$score_i$ < $score_k$**，说明这不是一个最长的连接（即该连接肯定包含在另外一个更长的连接中）。



举例说明，如上图，Anchor 已经按照 x 顺序排列好，并具有图中的 Softmax score（这里的 score 是随便给出的，只用于说明文本线构造算法）：

对于 i=3 的 **box_3**，向前寻找 50 像素，满足 **$overlap_v$** > 0.7 且 **score** 最大的是 **box_7**，即 j=7；**box_7** 反向寻找，满足 **$overlap_v$** > 0.7 且 score 最大的是 **box_3**，即 k=3。由于 **$score_3$ >= $score_3$**，**pair(box_3,box_7)**是最长连接，那么设置 Graph(3,7)=True

对于 **box_4** 正向寻找得到 **box_7**；**box_7** 反向寻找得到 **box_3**，但是 **$score_4$ < $score_3$**，即 **pair(box_4,box_3)**不是最长连接，包含在 **pair(box_3,box_7)**中。

然后，这样就建立了一个 N x N 的 Connect graph（其中 N 是正 Anchor 数量）。遍历 Graph：

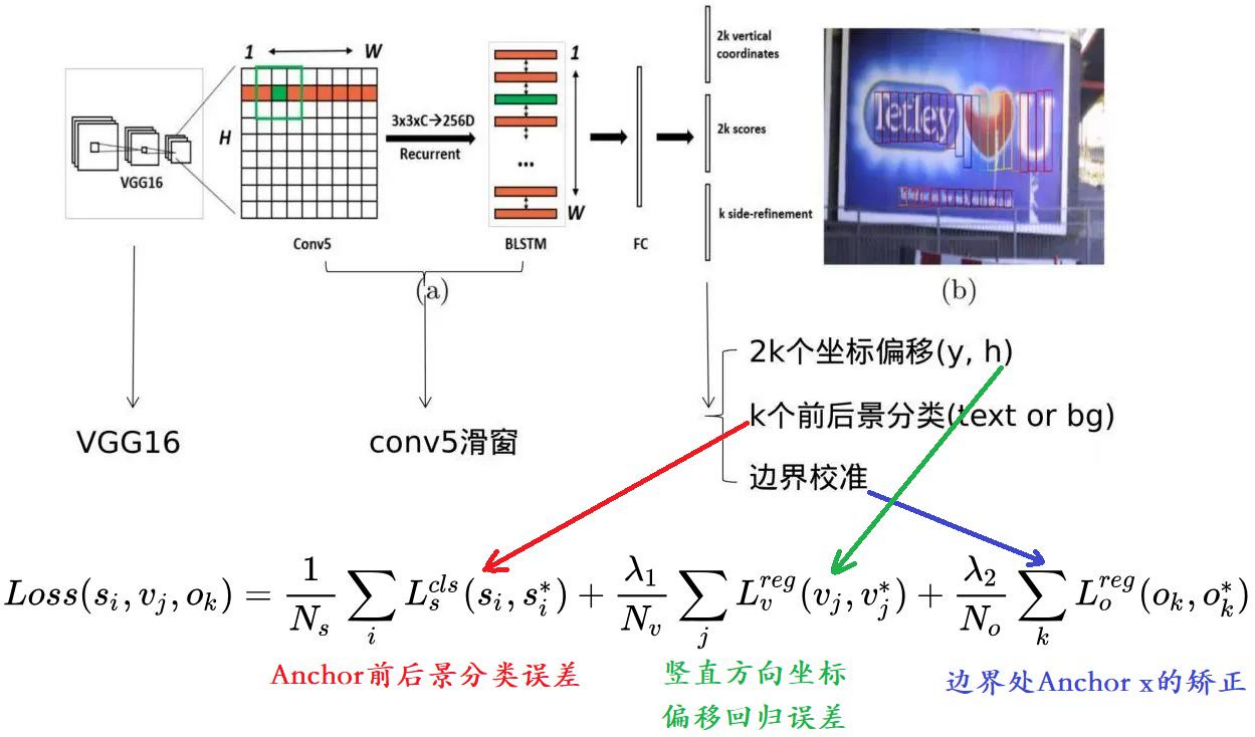
1.Graph(0,3)=True 且 Graph(3,7)=True，所以 Anchor index 1→3→7 组成一个文本，即蓝色文本区域。

2.Graph(6,10)=True 且 Graph(10,12)=True，所以 Anchor index 6→10→12 组成另外一个文本，即红色文本区域。

这样就通过 Text proposals 确定了文本检测框。

1.8、CTPN 的训练策略

该 Loss 分为 3 个部分



$$Loss(s_i, v_j, o_k) = \frac{1}{N_s}\sum_i L_s^{cls}(s_i, s_i^*) + \frac{\lambda_1}{N_v}\sum_j L_v^{reg}(v_j, v_j^*) + \frac{\lambda_2}{N_o}\sum_k L_o^{reg}(o_k, o_k^*)$$

Anchor前后景分类误差　竖直方向坐标偏移回归误差　边界处Anchor x的矫正

Anchor 前后景分类误差：该Loss用于监督学习每个Anchor中是否包含文本。s^{*}_{i}={0,1}表示是否是Groud truth。

竖直方向坐标偏移回归误差：该 Loss 用于监督学习每个包含为本的 Anchor 的 Bouding box regression y 方向 offset，类似于 Smooth L1 loss。其中 v_j 是 s_i 中判定为有文本的 Anchor，或者与 Groud truth vertical IoU>0.5。

边界处 Anchor x 的矫正误差：该 Loss 用于监督学习每个包含文本的 Anchor 的 Bouding box regression x 方向 offset，与 y 方向同理。前两个 Loss 存在的必要性很明确，但这个 Loss 有何作用作者没有解释（从训练和测试的实际效果看，作用不大）

说明一下，在 Bounding box regression 的训练过程中，其实只需要注意被判定成正的 Anchor，不需要去关心杂乱的负 Anchor。这与 Faster R-CNN 类似。

1.9、CTPN 小结

1.由于加入 LSTM，所以 CTPN 对水平文字检测效果超级好。

2.因为 Anchor 设定的原因，CTPN 只能检测横向分布的文字，小幅改进加入水平 Anchor 即可检测竖直文字。但是由于框架限定，对不规则倾斜文字检测效果非常一般。

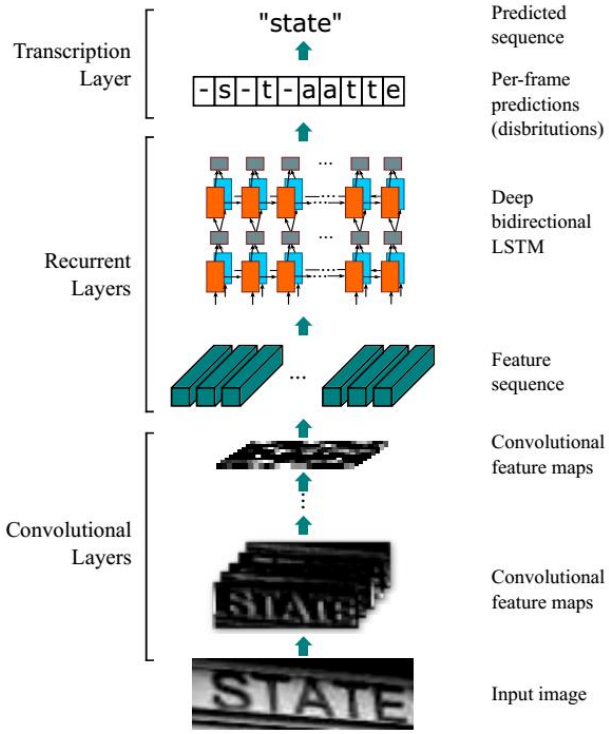
3.CTPN 加入了双向 LSTM 学习文字的序列特征，有利于文字检测。但是引入 LSTM 后，在训练时很容易梯度爆炸，需要小心处理。

2、CRNN 网络

现今基于深度学习的端到端 OCR 技术有两大主流技术：CRNN OCR 和 attention OCR。其实这两大方法主要区别在于最后的输出层（翻译层），即怎么将网络学习到的序列特征信息转化为最终的识别结果。这两大主流技术在其特征学习阶段都采用了 CNN+RNN 的网络结构，CRNN OCR 在对齐时采取的方式是 CTC 算法，而 attention OCR 采取的方式则是 attention 机制。本部分主要介绍应用更为广泛的 CRNN 算法。

2.1、CRNN 介绍

CRNN 全称为 Convolutional Recurrent Neural Network，主要用于端到端地对不定长的文本序列进行识别，不用先对单个文字进行切割，而是将文本识别转化为时序依赖的序列学习问题，就是基于图像的序列识别。



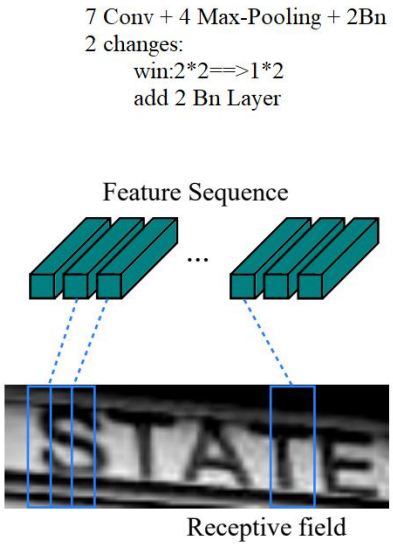
整个 CRNN 网络结构包含三部分，从下到上依次为：

1.CNN（卷积层）：使用深度 CNN，对输入图像提取特征，得到特征图；

2.RNN（循环层）：使用双向 RNN（BLSTM）对特征序列进行预测，对序列中的每个特征向量进行学习，并输出预测标签（真实值）分布；

3.CTC loss（转录层）：使用 CTC 损失，把从循环层获取的一系列标签分布转换成最终的标签序列。

2.2、CNN 卷积层结构



Type	Configurations
Transcription	-
Bidirectional-LSTM	#hidden units:256
Bidirectional-LSTM	#hidden units:256
Map-to-Sequence	-
Convolution	#maps:512, k:2 × 2, s:1, p:0
MaxPooling	Window:1 × 2, s:2
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
BatchNormalization	-
Convolution	#maps:512, k:3 × 3, s:1, p:1
MaxPooling	Window:1 × 2, s:2
Convolution	#maps:256, k:3 × 3, s:1, p:1
Convolution	#maps:256, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:128, k:3 × 3, s:1, p:1
MaxPooling	Window:2 × 2, s:2
Convolution	#maps:64, k:3 × 3, s:1, p:1
Input	W × 32 gray-scale image

这里有一个很精彩的改动，一共有四个最大池化层，但是最后两个池化层的窗口尺寸由 2x2 改为 1x2，也就是图片的高度减半了四次（除以 2^4），而宽度则只减半了两次（除以 2^2），这是因为文本图像多数都是高较小而宽较长，所以其 feature map 也是这种高小宽长的矩形形状，如果使用 1×2 的池化窗口可以尽量保证不丢失在宽度方向的信息，更适合英文字母识别（比如区分 i 和 l）。

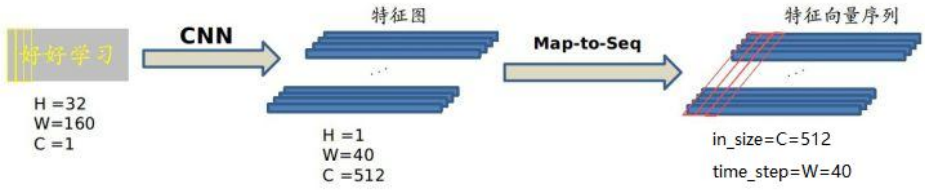
CRNN 还引入了 BatchNormalization 模块，加速模型收敛，缩短训练过程。

输入图像为灰度图像（单通道）；高度为 32，这是固定的，图片通过 CNN 后，高度就变为 1，这点很重要；宽度为 160，宽度也可以为其他的值，但需要统一，所以输入 CNN 的数据尺寸为 (channel, height, width)=(1, 32, 160)。

CNN 的输出尺寸为 (512, 1, 40)。即 CNN 最后得到 512 个特征图，每个特征图的高度为 1，宽度为 40。

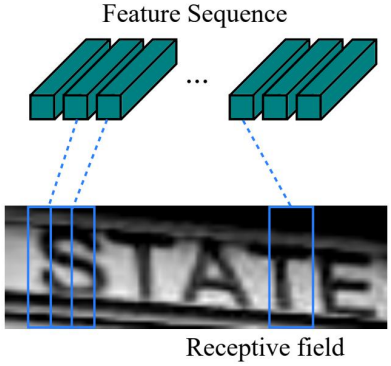
2.3、Map-to-Sequence

不能直接把 CNN 得到的特征图送入 RNN 进行训练的，需要进行一些调整，根据特征图提取 RNN 需要的特征向量序列。



现在需要从 CNN 模型产生的特征图中提取特征向量序列，每一个特征向量（如上图中的一个红色框）在特征图上按列从左到右生成，每一列包含 512 维特征，这意味着第 i 个特征向量是所有的特征图第 i 列像素的连接，这些特征向量就构成一个序列。

由于卷积层，最大池化层和激活函数在局部区域上执行，因此它们是平移不变的。因此，特征图的每列（即一个特征向量）对应于原始图像的一个矩形区域（称为感受野），并且这些矩形区域与特征图上从左到右的相应列具有相同的顺序。特征序列中的每个向量关联一个感受野。如下图所示：



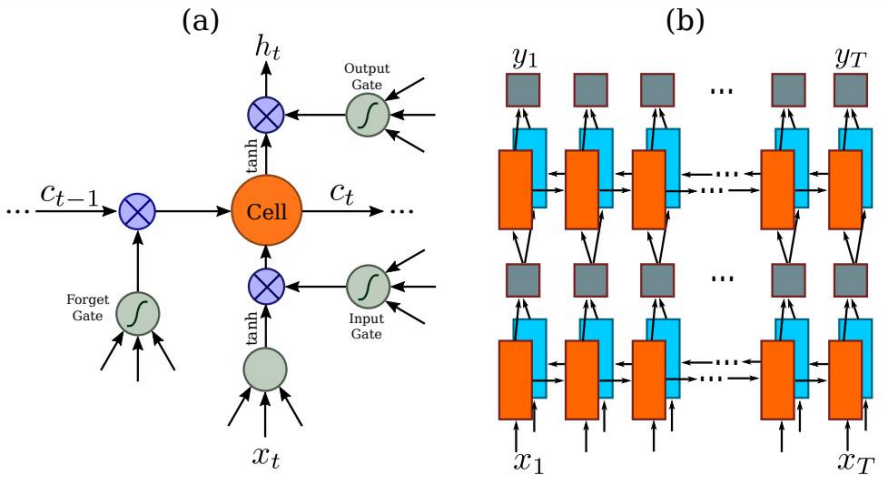
这些特征向量序列就作为循环层的输入，每个特征向量作为 RNN 在一个时间步（time step）的输入。

2.4、RNN

因为 RNN 有梯度消失的问题，不能获取更多上下文信息，所以 CRNN 中使用的是 LSTM，LSTM 的特殊设计允许它捕获长距离依赖。

LSTM 是单向的，它只使用过去的信息。然而，在基于图像的序列中，两个方向的上下文是相互有用且互补的。将两个 LSTM，一个向前和一个向后组合到一个双向 LSTM 中。此外，可以堆叠多层双向 LSTM，深层结构允许比浅层抽象更高层次的抽象。

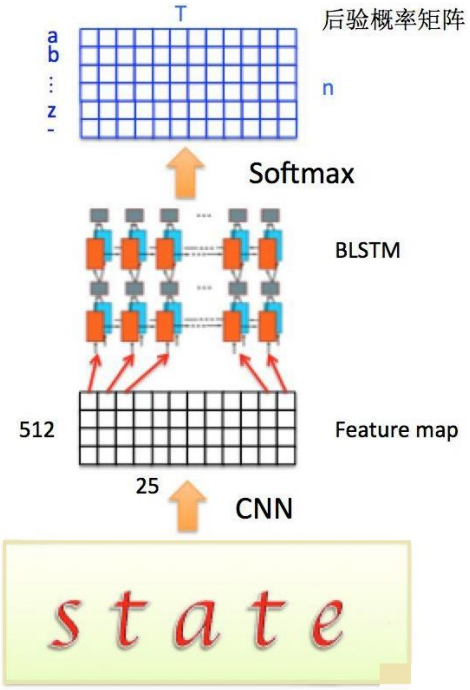
这里采用的是两层各 256 单元的双向 LSTM 网络：



通过上面一步，我们得到了 40 个特征向量，每个特征向量长度为 512，在 LSTM 中一个时间步就传入一个特征向量进行分类，这里一共有 40 个时间步。

我们知道一个特征向量就相当于原图中的一个小矩形区域，RNN 的目标就是预测这个矩形区域为哪个字符，即根据输入的特征向量，进行预测，得到所有字符的 softmax 概率分布，这是一个长度为字符类别数的向量，作为 CTC 层的输入。

因为每个时间步都会有一个输入特征向量 x^T，输出一个所有字符的概率分布 y^T，所以输出为 40 个长度为字符类别数的向量构成的后验概率矩阵。如下图所示：



然后将这个后验概率矩阵传入转录层。

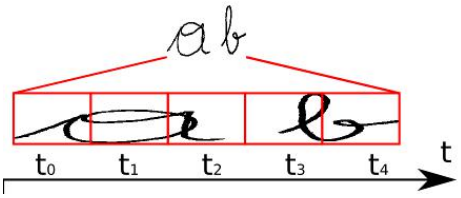
2.5、CTC Loss

这算是 CRNN 最难的地方，这一层为转录层，转录是将 RNN 对每个特征向量所做的预测转换成标签序列的过程。数学上，转录是根据每帧预测找到具有最高概率组合的标签序列。

端到端 OCR 识别的难点在于怎么处理不定长序列对齐的问题！OCR 可建模为时序依赖的文本图像问题，然后使用 CTC（Connectionist Temporal Classification, CTC）的损失函数来对 CNN 和 RNN 进行端到端的联合训练。

2.5.1、序列合并机制

我们现在要将 RNN 输出的序列翻译成最终的识别结果，RNN 进行时序分类时，不可避免地会出现很多冗余信息，比如一个字母被连续识别两次，这就需要一套去冗余机制。



比如我们要识别上面这个文本，其中 RNN 中有 5 个时间步，理想情况下 t0, t1, t2 时刻都应映射为“a”，t3, t4 时刻都应映射为“b”，然后将这些字符序列连接起来得到“aaabb”，我们再将连续重复的字符合并成一个，那么最终结果为“ab”。

这似乎是个比较好的方法，但是存在一个问题，如果是 book，hello 之类的词，合并连续字符后就会得到 bok 和 helo，这显然不行，所以 CTC 有一个 blank 机制来解决这个问题。

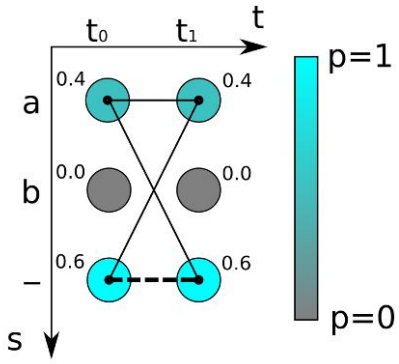
我们以“-”符号代表 blank，RNN 输出序列时，在文本标签中的重复的字符之间插入一个“-”，比如输出序列为“bbooo-ookk”，则最后将被映射为“book”，即有 blank 字符隔开的话，连续相同字符就不进行合并。

即对字符序列先删除连续重复字符，然后从路径中删除所有“-”字符，这个称为解码过程，而编码则是由神经网络来实现。引入 blank 机制，我们就可以很好地解决重复字符的问题。

相同的文本标签可以有多个不同的字符对齐组合，例如，“aa-b”和“aabb”以及“-abb”都代表相同的文本(“ab”)，但是与图像的对齐方式不同。更总结地说，一个文本标签存在一条或多条的路径。

2.5.2、训练阶段

在训练阶段，我们需要根据这些概率分布向量和相应的文本标签得到损失函数，从而训练神经网路模型，下面来看看如何得到损失函数的。



其中黑细线是代表文本“a”的路径，而粗虚线是代表空文本的路径

如上图，对于最简单的时序为 2 的字符识别，有两个时间步长(t0，t1)和三个可能的字符为“ a ”，“ b ”和“-”，我们得到两个概率分布向量，如果采取最大概率路径解码的方法，则“--”的概率最大，即真实字符为空的概率为0.6*0.6=0.36。

但是为字符“ a ”的情况有多种对齐组合，“aa”, “a-“和“-a”都是代表“ a ”，所以，输出“ a ”的概率应该为三种之和：

$$0.4 * 0.4 + 0.4 * 0.6 + 0.6 * 0.4 = 0.16 + 0.24 + 0.24 = 0.64$$

所以“ a ”的概率比空“”的概率高！如果标签文本为“a”，则通过计算图像中为“a”的所有可能的对齐组合（或者路径）的分数之和来计算损失函数。

所以对于 RNN 给定输入概率分布矩阵为 $y=\{y^1,y^2,...,y^T\}$，T 是序列长度，最后映射为标签文本 l 的总概率为：

$$p(\mathbf{l}|\mathbf{y}) = \sum_{\boldsymbol{\pi}:\mathcal{B}(\boldsymbol{\pi})=\mathbf{l}} p(\boldsymbol{\pi}|\mathbf{y}),$$

其中 B(π)代表从序列到序列的映射函数 B 变换后是文本 l 的所有路径集合，而π则是其中的一条路径。每条路径的概率为各个时间步中对应字符的分数的乘积。

我们就是需要训练网络使得这个概率值最大化，类似于普通的分类，CTC 的损失函数定义为概率的负最大似然函数，为了计算方便，对似然函数取对数。

通过对损失函数的计算，就可以对之前的神经网络进行反向传播，神经网络的参数根据所使用的优化器进行更新，从而找到最可能的像素区域对应的字符。

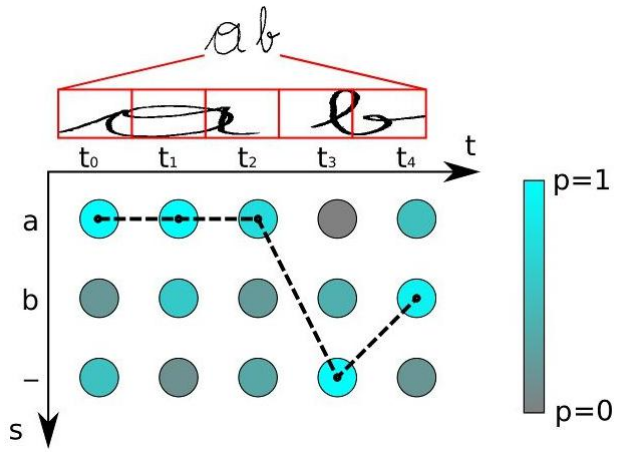
这种通过映射变换和所有可能路径概率之和的方式使得 CTC 不需要对原始的输入字符序列进行准确的切分。

2.5.3、测试阶段

在测试阶段，过程与训练阶段有所不同，我们用训练好的神经网络来识别新的文本图像。这时候我们事先不知道任何文本，如果我们像上面一样将每种可能文本的所有路径计算出来，对于很长的时间步和很长的字符序列来说，这个计算量是非常庞大的，这不是一个可行的方案。

我们知道 RNN 在每一个时间步的输出为所有字符类别的概率分布，即一个包含每个字符分数的向量，我们取其中最大概率的字符作为该时间步的输出字符，然后将所有时间步得到一个字符进行拼接得到一个序列路径，即最大概率路径，再根据上面介绍的合并序列方法得到最终的预测文本结果。

在输出阶段经过 CTC 的翻译，即将网络学习到的序列特征信息转化为最终的识别文本，就可以对整个文本图像进行识别。



比如上面这个图，有 5 个时间步，字符类别有“a”, “b” and “-” (blank)，对于每个时间步的概率分布，我们都取分数最大的字符，所以得到序列路径“aaa-b”，先移除相邻重复的字符得到“a-b”，然后去除 blank 字符得到最终结果：“ab”。

2.5.4、CRNN 小结

预测过程中，先使用标准的 CNN 网络提取文本图像的特征，再利用 BLSTM 将特征向量进行融合以提取字符序列的上下文特征，然后得到每列特征的概率分布，最后通过转录层(CTC)进行预测得到文本序列。

利用 BLSTM 和 CTC 学习到文本图像中的上下文关系，从而有效提升文本识别准确率，使得模型更加鲁棒。

在训练阶段，CRNN 将训练图像统一缩放为 160×32（w×h）；在测试阶段，针对字符拉伸会导致识别率降低的问题，CRNN 保持输入图像尺寸比例，但是图像高度还是必须统一为 32 个像素，卷积特征图的尺寸动态决定 LSTM 的时序长度（时间步长）。

参考：

https://zhuanlan.zhihu.com/p/28133530
https://zhuanlan.zhihu.com/p/43534801
https://arxiv.org/pdf/1507.05717.pdf
https://www.cnblogs.com/skyfsm/p/10335717.html
https://zhuanlan.zhihu.com/p/43534801
http://noahsnail.com/
https://www.jianshu.com/p/109231be4a24