冠军的试炼

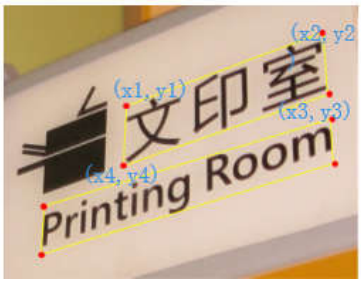
悟已往之不谏，知来者之可追

【OCR技术系列之六】文本检测CTPN的代码实现

这几天一直在用Pytorch来复现文本检测领域的CTPN论文，本文章将从数据处理、训练标签生成、神经网络搭建、损失函数设计、训练主过程编写等这几个方面来一步一步复现CTPN。CTPN算法理论可以参考这里。

训练数据处理

我们的训练选择天池ICPR2018和MSRA_TD500两个数据集，天池ICPR的数据集为网络图像，都是一些淘宝商家上传到淘宝的一些商品介绍图像，其标签方式参考了ICDAR2015的数据标签格式，即一个文本框用4个坐标来表示，即左上、右上、右下、左下四个坐标，共八个值，记作[x1 y1 x2 y2 x3 y3 x4 y4]



天池ICPR2018数据集的风格如下，字体形态格式颜色多变，多嵌套于物体之中，识别难度大：

MSRA_TD500使微软收集的一个文本检测和识别的一个数据集，里面的图像多是街景图，背景比较复杂，但文本位置比较明显，一目了然。因为MSRA_TD500的标签格式不一样，最后一个参数表示矩形框的旋转角度。

公告

昵称：Madcola
园龄：2年7个月
粉丝：1334
关注：30
+加关注

2019年8月
日 一 二 三 四 五 六
28 29 30 31 1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
1 2 3 4 5 6 7

搜索

常用链接

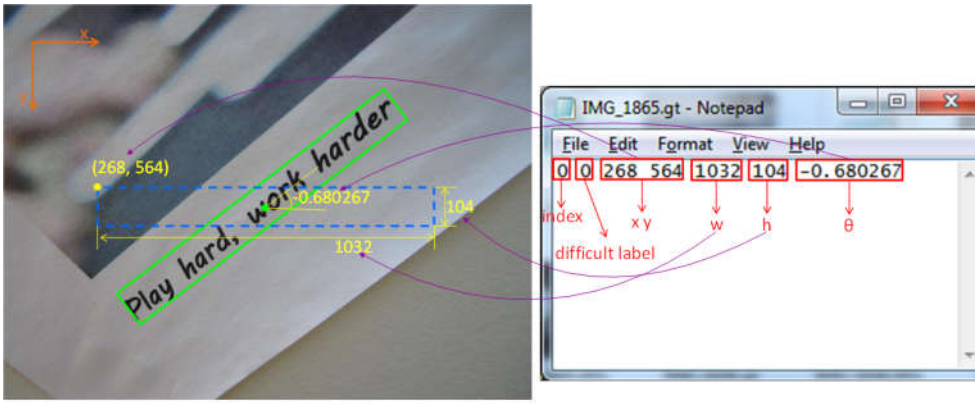
我的随笔
我的评论
我的参与
最新评论
我的标签

随笔分类(69)

C++(1)
CUDA(1)
Linux编程(12)
OCR系列(8)
opencv探索(28)
STL(2)
波折岁月(4)
工具技巧(1)
机器学习之旅(5)
深度学习(4)
数字图像处理(3)

随笔档案(69)

2019年2月 (1)
2019年1月 (1)



IMG_1865.JPG IMG_1865.gt

所以我们第一步就是将这两个数据集的标签格式统一，我的做法是将MSRA数据集格式改为ICDAR格式，方便后面的模型训练。因为MSRA_TD500采取的标签格式是[index difficulty_label x y w h angle]，所以我们需要根据这个文本框的旋转角度来求得水平文本框旋转后的4个坐标位置。实现如下：

```
"""
This file is to change MSRA_TD500 dataset format to ICDAR2015 dataset format.

MSRA_TD500 format: [index difficulty_label x y w h angle]

ICDAR2015 format: [left_top_x left_top_y right_top_X right_top_y right_bottom_x
right_bottom_y left_bottom_x left_bottom_y]

"""


import math
import cv2
import os

# 求旋转后矩形的4个坐标
def get_box_img(x, y, w, h, angle):
    # 矩形框中点(x0,y0)
    x0 = x + w/2
    y0 = y + h/2
    l = math.sqrt(pow(w/2, 2) + pow(h/2, 2))  # 即对角线的一半
    # angle小于0，逆时针转
    if angle < 0:
        a1 = -angle + math.atan(h / float(w))  # 旋转角度-对角线与底线所成的角度
        a2 = -angle - math.atan(h / float(w)) # 旋转角度+对角线与底线所成的角度
        pt1 = (x0 - l * math.cos(a2), y0 + l * math.sin(a2))
        pt2 = (x0 + l * math.cos(a1), y0 - l * math.sin(a1))
        pt3 = (x0 + l * math.cos(a2), y0 - l * math.sin(a2))  # x0+左下点旋转后在水平线上的投影，
y0-左下点在垂直线上的投影，显然逆时针转时，左下点上一和左移了。
        pt4 = (x0 - l * math.cos(a1), y0 + l * math.sin(a1))
    else:
        a1 = angle + math.atan(h / float(w))
        a2 = angle - math.atan(h / float(w))
        pt1 = (x0 - l * math.cos(a1), y0 - l * math.sin(a1))
        pt2 = (x0 + l * math.cos(a2), y0 + l * math.sin(a2))
        pt3 = (x0 + l * math.cos(a1), y0 + l * math.sin(a1))
        pt4 = (x0 - l * math.cos(a2), y0 - l * math.sin(a2))
    return [pt1[0], pt1[1], pt2[0], pt2[1], pt3[0], pt3[1], pt4[0], pt4[1]]


def read_file(path):
    result = []
    for line in open(path):
        info = []
        data = line.split(' ')
        info.append(int(data[2]))
        info.append(int(data[3]))
        info.append(int(data[4]))
        info.append(int(data[5]))
        info.append(float(data[6]))
        info.append(data[0])
```

2018年12月 (2)
2018年10月 (1)
2018年9月 (3)
2018年5月 (1)
2018年4月 (2)
2018年2月 (6)
2018年1月 (3)
2017年12月 (4)
2017年11月 (3)
2017年10月 (1)
2017年9月 (4)
2017年8月 (3)
2017年7月 (5)
2017年6月 (4)
2017年5月 (17)
2017年4月 (1)
2017年2月 (2)
2017年1月 (5)

积分与排名

积分 - 213285
排名 - 1747

最新评论

1. Re:【OCR技术系列之四】基于深度学习的文字识别（3755个汉字）
感谢博主的无私奉献
--寒冬夜行人lee

2. Re:我的2018：OCR、实习和秋招
学长好厉害！我实习内容也是在做OCR，每天照着你的好多博客看...我是18级研究生2020年毕业然后最近也在边实习边准备秋招（但是我好菜），可不可以加学长微信交流交流呀，我的微信是Dreaminice.....
--Cocoalate

3. Re:【Keras】基于SegNet和U-Net的遥感图像语义分割
@wenny-bell我也有这样的问题，请问您解决了吗？可以加下qq讨论下，2724858160...
--嗯哼！！！

4. Re:【Keras】基于SegNet和U-Net的遥感图像语义分割
@wenny-bell我也有这样的问题，请问您解决了吗？可以加下qq讨论下，2724858160...
--嗯哼！！！

5. Re:OpenCV探索之路（十一）：轮廓查找和多边形包围轮廓
请教下，int thresh_size = (100 / 4) * 2 + 1; //自适应二值化阈值这个阈值的定义是怎么给出的呢？用你设定的参数确实得到了很好的效果，但是不知道为什么这样设置.谢谢~.....
--foxlucia

阅读排行榜

1. 卷积神经网络CNN总结(219292)
2. 基于深度学习的目标检测技术演进：R-CNN、Fast R-CNN、Faster

```
        result.append(info)
    return result


if __name__ == '__main__':
    file_path = '/home/ljs/OCR_dataset/MSRA-TD500/test/'
    save_img_path = '../dataset/OCR_dataset/ctpn/test_im/'
    save_gt_path = '../dataset/OCR_dataset/ctpn/test_gt/'
    file_list = os.listdir(file_path)
    for f in file_list:
        if '.gt' in f:
            continue
        name = f[0:8]
        txt_path = file_path + name + '.gt'
        im_path = file_path + f
        im = cv2.imread(im_path)
        coordinate = read_file(txt_path)
        # 仿照ICDAR格式，图片名字写做img_xx.jpg，对应的标签文件写做gt_img_xx.txt
        cv2.imwrite(save_img_path + name.lower() + '.jpg', im)
        save_gt = open(save_gt_path + 'gt_' + name.lower() + '.txt', 'w')
        for i in coordinate:
            box = get_box_img(i[0], i[1], i[2], i[3], i[4])
            box = [int(box[i]) for i in range(len(box))]
            box = [str(box[i]) for i in range(len(box))]
            save_gt.write(','.join(box))
            save_gt.write('\n')
```

经过格式处理后，我们两份数据集算是整理好了。当然我们还需要对整个数据集划分为训练集和测试集，我的文件组织习惯如下：train_im, test_im文件夹装的是训练和测试图像，train_gt和test_gt装的是训练和测试标签。

```
[ljs@localhost MSRA_TD500]$ ls
test_gt  test_im  train_gt  train_im
```

训练标签生成

因为CTPN的核心思想也是基于Faster RCNN中的region proposal机制的，所以原始数据标签需要转化为anchor标签。训练数据的标签的生成的代码是最难写，因为从一个完整的文本框标签转化为一个个小尺度文本框标签确实有点难度，而且这个anchor标签的生成方式也与Faster RCNN生成方式略有不同。下面讲一讲我的实现思路：

第一步我们需要将原先每张图的bbox标签转化为每个anchor标签。为了实现该功能，我们先将一张图划分为宽度为16的各个anchor。

- 首先计算一张图可以分为多少个宽度为16的acnhor（比如一张图的宽度为w，那么水平anchor总数为w/16），再计算出我们的文本框标签中含有几个acnhor，最左和最右的anchor又是哪几个；
- 计算文本框内anchor的高度和中心是多少：此时我们可以在一个全黑的mask中把文本框label画上去（白色），然后从上往下和从下往上找到第一个白色像素点的位置作为该anchor的上下边界；
- 最后将每个anchor的位置(水平ID)、anchor中心y坐标、anchor高度存储并返回

```
def generate_gt_anchor(img, box, anchor_width=16):
    """
    calsulate ground truth fine-scale box
    :param img: input image
    :param box: ground truth box (4 point)
    :param anchor_width:
    :return: tuple (position, h, cy)
    """
    if not isinstance(box[0], float):
        box = [float(box[i]) for i in range(len(box))]
    result = []
    # 求解一个bbox下，能分解为多少个16宽度的小anchor，并求出最左和最右的小achor的id
    left_anchor_num = int(math.floor(max(min(box[0], box[6]), 0) / anchor_width))  # the left side anchor of the text box, downwards
    right_anchor_num = int(math.ceil(min(max(box[2], box[4]), img.shape[1]) / anchor_width))  # the right side anchor of the text box, upwards

    # handle extreme case, the right side anchor may exceed the image width
    if right_anchor_num * 16 + 15 > img.shape[1]:
        right_anchor_num -= 1
```

R-CNN(206008)
3. OpenCV探索之路（二十四）图像拼接和图像融合技术(88663)
4. CNN网络架构演进：从LeNet到DenseNet(58616)
5. OpenCV探索之路（二十三）：特征检测和特征匹配方法汇总(54127)
6. 【OCR技术系列之四】基于深度学习的文字识别（3755个汉字）(51233)
7. C++ STL快速入门(45935)
8. OpenCV探索之路（六）：边缘检测（canny、sobel、laplacian）(45154)
9. Linux编程之UDP SOCKET全攻略(41919)
10. OpenCV探索之路（十四）：绘制点、直线、几何图形(37784)

评论排行榜

1. 【OCR技术系列之四】基于深度学习的文字识别（3755个汉字）(82)
2. 【Keras】基于SegNet和U-Net的遥感图像语义分割(73)
3. OpenCV探索之路（二十四）图像拼接和图像融合技术(67)
4. 【OCR技术系列之八】端到端不定长文本识别CRNN代码实现(59)
5. 【Keras】从两个实际任务掌握图像分类(33)

推荐排行榜

1. 基于深度学习的目标检测技术演进：R-CNN、Fast R-CNN、Faster R-CNN(92)
2. 卷积神经网络CNN总结(61)
3. 我的2018：OCR、实习和秋招(22)
4. 【OCR技术系列之四】基于深度学习的文字识别（3755个汉字）(21)
5. 【Keras】基于SegNet和U-Net的遥感图像语义分割(20)
6. CNN网络架构演进：从LeNet到DenseNet(20)
7. OpenCV探索之路（二十四）图像拼接和图像融合技术(18)
8. 我在北京实习的四个月(15)
9. 【OCR技术系列之一】字符识别技术总览(13)
10. 读研以来的一些感想：名校好在哪里？(13)

```
    # combine the left-side and the right-side x_coordinate of a text anchor into one pair
    position_pair = [(i * anchor_width, (i + 1) * anchor_width - 1) for i in
range(left_anchor_num, right_anchor_num)]

    # 计算每个gt anchor的真实位置，其实就是求解gt anchor的上边界和下边界
    y_top, y_bottom = cal_y_top_and_bottom(img, position_pair, box)
    # 最后将每个anchor的位置(水平ID)、anchor中心y坐标、anchor高度存储并返回
    for i in range(len(position_pair)):
        position = int(position_pair[i][0] / anchor_width)  # the index of anchor box
        h = y_bottom[i] - y_top[i] + 1  # the height of anchor box
        cy = (float(y_bottom[i]) + float(y_top[i])) / 2.0  # the center point of anchor box
        result.append((position, cy, h))
    return result
```

计算anchor上下边界的方法：

```
# cal the gt anchor box's bottom and top coordinate
def cal_y_top_and_bottom(raw_img, position_pair, box):
    """
    :param raw_img:
    :param position_pair: for example:[(0, 15), (16, 31), ...]
    :param box: gt box (4 point)
    :return: top and bottom coordinates for y-axis
    """
    img = copy.deepcopy(raw_img)
    y_top = []
    y_bottom = []
    height = img.shape[0]
    # 设置图像mask，channel 0为全黑图
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            img[i, j, 0] = 0

    top_flag = False
    bottom_flag = False
    # 根据bbox四点画出文本框，channel 0下文本框为白色
    img = other.draw_box_4pt(img, box, color=(255, 0, 0))


    for k in range(len(position_pair)):
        # 从左到右遍历anchor gt，对每个anchor从上往下扫描像素，遇到白色像素点（255）就停下来，此时像素点坐标y就是该anchor gt的上边界
        # calc top y coordinate
        for y in range(0, height-1):
            # loop each anchor, from left to right
            for x in range(position_pair[k][0], position_pair[k][1] + 1):
                if img[y, x, 0] == 255:
                    y_top.append(y)
                    top_flag = True
                    break
            if top_flag is True:
                break

         # 从左到右遍历anchor gt，对每个anchor从下往上扫描像素，遇到白色像素点（255）就停下来，此时像素点坐标y就是该anchor gt的下边界
        # calc bottom y coordinate, pixel from down to top loop
        for y in range(height - 1, -1, -1):
            # loop each anchor, from left to right
            for x in range(position_pair[k][0], position_pair[k][1] + 1):
                if img[y, x, 0] == 255:
                    y_bottom.append(y)
                    bottom_flag = True
                    break
            if bottom_flag is True:
                break
        top_flag = False
        bottom_flag = False
    return y_top, y_bottom
```

经过上面的标签处理，我们已经将原先的标准的文本框标签转化为一个一个小尺度anchor标签，以下是标签转化后的效果：

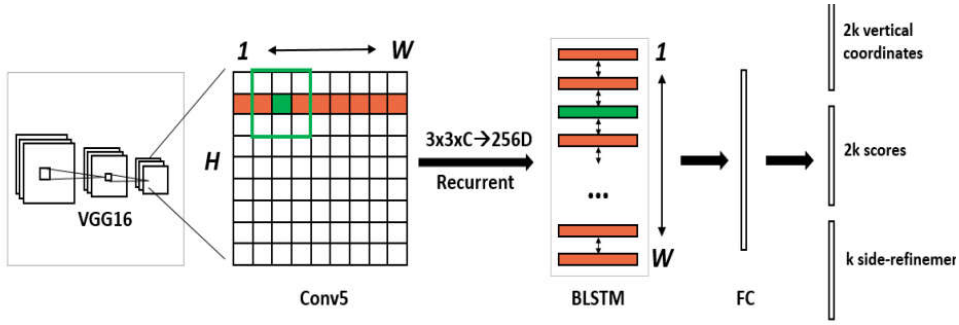
以上标签可视化后看来anchor标签做得不错，但是这里需要提出的是，我发现这种anchor生成方法是不太精准的，比如一个文本框边缘像素刚好落在一个新的anchor上，那么我们就要为这个像素分配一个16像素的anchor，显然导致了文本框标签的不准确，引入了15像素的误差，这个是需要思考的。这个问题我们先不做处理，继续下面的工作。

当然转化期间我们也遇到很多奇怪的问题，比如下图这种标签都已经超出图像范围的，我们必须做相应的特殊处理，比如限定标签横坐标的最大尺寸为图像宽度。

```
left_anchor_num = int(math.floor(max(min(box[0], box[6]), 0) / anchor_width))  # the left
side anchor of the text box, downwards
right_anchor_num = int(math.ceil(min(max(box[2], box[4]), img.shape[1]) / anchor_width))  #
the right side anchor of the text box, upwards
```

CTPN网络结构

因为CTPN用到了CNN+双向LSTM的网络结构，所以我们分步实现CTPN架构。



CNN部分CTPN采取了VGG16进行底层特征提取。

```
class VGG_16(nn.Module):
    """
    VGG-16 without pooling layer before fc layer
    """
    def __init__(self):
        super(VGG_16, self).__init__()
        self.convolution1_1 = nn.Conv2d(3, 64, 3, padding=1)
        self.convolution1_2 = nn.Conv2d(64, 64, 3, padding=1)
        self.pooling1 = nn.MaxPool2d(2, stride=2)
        self.convolution2_1 = nn.Conv2d(64, 128, 3, padding=1)
        self.convolution2_2 = nn.Conv2d(128, 128, 3, padding=1)
        self.pooling2 = nn.MaxPool2d(2, stride=2)
        self.convolution3_1 = nn.Conv2d(128, 256, 3, padding=1)
        self.convolution3_2 = nn.Conv2d(256, 256, 3, padding=1)
        self.convolution3_3 = nn.Conv2d(256, 256, 3, padding=1)
        self.pooling3 = nn.MaxPool2d(2, stride=2)
        self.convolution4_1 = nn.Conv2d(256, 512, 3, padding=1)
        self.convolution4_2 = nn.Conv2d(512, 512, 3, padding=1)
        self.convolution4_3 = nn.Conv2d(512, 512, 3, padding=1)
        self.pooling4 = nn.MaxPool2d(2, stride=2)
        self.convolution5_1 = nn.Conv2d(512, 512, 3, padding=1)
        self.convolution5_2 = nn.Conv2d(512, 512, 3, padding=1)
        self.convolution5_3 = nn.Conv2d(512, 512, 3, padding=1)

    def forward(self, x):
        x = F.relu(self.convolution1_1(x), inplace=True)
        x = F.relu(self.convolution1_2(x), inplace=True)
        x = self.pooling1(x)
        x = F.relu(self.convolution2_1(x), inplace=True)
        x = F.relu(self.convolution2_2(x), inplace=True)
        x = self.pooling2(x)
        x = F.relu(self.convolution3_1(x), inplace=True)
        x = F.relu(self.convolution3_2(x), inplace=True)
        x = F.relu(self.convolution3_3(x), inplace=True)
        x = self.pooling3(x)
        x = F.relu(self.convolution4_1(x), inplace=True)
        x = F.relu(self.convolution4_2(x), inplace=True)
        x = F.relu(self.convolution4_3(x), inplace=True)
        x = self.pooling4(x)
        x = F.relu(self.convolution5_1(x), inplace=True)
        x = F.relu(self.convolution5_2(x), inplace=True)
        x = F.relu(self.convolution5_3(x), inplace=True)
        return x
```

再实现双向LSTM，增强关联序列的信息学习。

```
class BLSTM(nn.Module):
    def __init__(self, channel, hidden_unit, bidirectional=True):
        """
        :param channel: lstm input channel num
        :param hidden_unit: lstm hidden unit
        :param bidirectional:
        """
        super(BLSTM, self).__init__()
        self.lstm = nn.LSTM(channel, hidden_unit, bidirectional=bidirectional)

    def forward(self, x):
        """
        WARNING: The batch size of x must be 1.
```

```
        """
        x = x.transpose(1, 3)
        recurrent, _ = self.lstm(x[0])
        recurrent = recurrent[np.newaxis, :, :, :]
        recurrent = recurrent.transpose(1, 3)
        return recurrent
```

这里实现多一层中间层，用于连接CNN和LSTM。将VGG最后一层卷积层输出的feature map转化为向量形式，用于接下来的LSTM训练。

```
class Im2col(nn.Module):
    def __init__(self, kernel_size, stride, padding):
        super(Im2col, self).__init__()
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding

    def forward(self, x):
        height = x.shape[2]
        x = F.unfold(x, self.kernel_size, padding=self.padding, stride=self.stride)
        x = x.reshape((x.shape[0], x.shape[1], height, -1))
        return x
```

最后将以上三部分拼接成一个完整的CTPN网络：底层使用VGG16做特征提取->lstm序列信息学习->output每个anchor分数，h, y, side_refinement

```
class CTPN(nn.Module):
    def __init__(self):
        super(CTPN, self).__init__()
        self.cnn = nn.Sequential()
        self.cnn.add_module('VGG_16', VGG_16())
        self.rnn = nn.Sequential()
        self.rnn.add_module('im2col', Net.Im2col((3, 3), (1, 1), (1, 1)))
        self.rnn.add_module('blstm', BLSTM(3 * 3 * 512, 128))
        self.FC = nn.Conv2d(256, 512, 1)
        self.vertical_coordinate = nn.Conv2d(512, 2 * 10, 1)  # 最终输出2K个参数（k=10），10表示anchor的尺寸个数，2个参数分别表示anchor的h和dy
        self.score = nn.Conv2d(512, 2 * 10, 1)  # 最终输出是2K个分数（k=10），2表示有无字符，10表示anchor的尺寸个数
        self.side_refinement = nn.Conv2d(512, 10, 1)  # 最终输出1K个参数（k=10），该参数表示该anchor的水平偏移，用于精修文本框水平边缘精度，，10表示anchor的尺寸个数

    def forward(self, x, val=False):
        x = self.cnn(x)
        x = self.rnn(x)
        x = self.FC(x)
        x = F.relu(x, inplace=True)
        vertical_pred = self.vertical_coordinate(x)
        score = self.score(x)
        if val:
            score = score.reshape((score.shape[0], 10, 2, score.shape[2], score.shape[3]))
            score = score.squeeze(0)
            score = score.transpose(1, 2)
            score = score.transpose(2, 3)
            score = score.reshape((-1, 2))
            #score = F.softmax(score, dim=1)
            score = score.reshape((10, vertical_pred.shape[2], -1, 2))
            vertical_pred = vertical_pred.reshape((vertical_pred.shape[0], 10, 2, vertical_pred.shape[2], vertical_pred.shape[3]))
        side_refinement = self.side_refinement(x)
        return vertical_pred, score, side_refinement
```

损失函数设计

CTPN的LOSS分为三部分：

- h,y的regression loss，用的是SmoothL1Loss；
- score的classification loss，用的是CrossEntropyLoss；
- side refinement loss，用的是用的是SmoothL1Loss。

$$L(\mathbf{s}_i, \mathbf{v}_j, \mathbf{o}_k) = \frac{1}{N_s}\sum_i L_s^{cl}(\mathbf{s}_i, \mathbf{s}_i^*) + \frac{\lambda_1}{N_v}\sum_j L_v^{re}(\mathbf{v}_j, \mathbf{v}_j^*) + \frac{\lambda_2}{N_o}\sum_k L_o^{re}(\mathbf{o}_k, \mathbf{o}_k^*$$

先定义好一些固定参数

```
class CTPN_Loss(nn.Module):
    def __init__(self, using_cuda=False):
        super(CTPN_Loss, self).__init__()
        self.Ns = 128
        self.ratio = 0.5
        self.lambda1 = 1.0
        self.lambda2 = 1.0
        self.Ls_cls = nn.CrossEntropyLoss()
        self.Lv_reg = nn.SmoothL1Loss()
        self.Lo_reg = nn.SmoothL1Loss()
        self.using_cuda = using_cuda
```

首先设计classification loss

```
        cls_loss = 0.0
        if self.using_cuda:
            for p in positive_batch:
                cls_loss += self.Ls_cls(score[0, p[2] * 2: ((p[2] + 1) * 2), p[1],
p[0]].unsqueeze(0),
                                        torch.LongTensor([1]).cuda())
            for n in negative_batch:
                cls_loss += self.Ls_cls(score[0, n[2] * 2: ((n[2] + 1) * 2), n[1],
n[0]].unsqueeze(0),
                                        torch.LongTensor([0]).cuda())
        else:
            for p in positive_batch:
                cls_loss += self.Ls_cls(score[0, p[2] * 2: ((p[2] + 1) * 2), p[1],
p[0]].unsqueeze(0),
                                        torch.LongTensor([1]))
            for n in negative_batch:
                cls_loss += self.Ls_cls(score[0, n[2] * 2: ((n[2] + 1) * 2), n[1],
n[0]].unsqueeze(0),
                                        torch.LongTensor([0]))
        cls_loss = cls_loss / self.Ns
```

然后是vertical coordinate regression loss，反映的是y和h的偏差

```
        # calculate vertical coordinate regression loss
        v_reg_loss = 0.0
        Nv = len(vertical_reg)
        if self.using_cuda:
            for v in vertical_reg:
                v_reg_loss += self.Lv_reg(vertical_pred[0, v[2] * 2: ((v[2] + 1) * 2), v[1],
v[0]].unsqueeze(0),
                                          torch.FloatTensor([v[3],
v[4]]).unsqueeze(0).cuda())
        else:
            for v in vertical_reg:
                v_reg_loss += self.Lv_reg(vertical_pred[0, v[2] * 2: ((v[2] + 1) * 2), v[1],
v[0]].unsqueeze(0),
                                          torch.FloatTensor([v[3], v[4]]).unsqueeze(0))
        v_reg_loss = v_reg_loss / float(Nv)
```

最后计算side refinement regression loss，用于修正边缘精度

```
        # calculate side refinement regression loss
        o_reg_loss = 0.0
        No = len(side_refinement_reg)
        if self.using_cuda:
            for s in side_refinement_reg:
                o_reg_loss += self.Lo_reg(side_refinement[0, s[2]: s[2] + 1, s[1],
s[0]].unsqueeze(0),
                                          torch.FloatTensor([s[3]]).unsqueeze(0).cuda())
        else:
            for s in side_refinement_reg:
                o_reg_loss += self.Lo_reg(side_refinement[0, s[2]: s[2] + 1, s[1],
```

```
s[0]].unsqueeze(0),
                                        torch.FloatTensor([s[3]]).unsqueeze(0))
        o_reg_loss = o_reg_loss / float(No)
```

当然最后还有个total loss，汇总整个训练过程中的loss

```
loss = cls_loss + v_reg_loss * self.lambda1 + o_reg_loss * self.lambda2
```

训练过程设计

训练:优化器我们选择SGD，learning rate我们设置了两个，前N个epoch使用较大的lr，后面的epoch使用较小的lr以更好地收敛。训练过程我们定义了4个loss，分别是total_cls_loss，total_v_reg_loss， total_o_reg_loss，total_loss（前面三个loss相加）。

```
    net = Net.CTPN() # 获取网络结构
    for name, value in net.named_parameters():
        if name in no_grad:
            value.requires_grad = False
        else:
            value.requires_grad = True
    # for name, value in net.named_parameters():
    #     print('name: {0}, grad: {1}'.format(name, value.requires_grad))
    net.load_state_dict(torch.load('./lib/vgg16.model'))
    # net.load_state_dict(model_zoo.load_url(model_urls['vgg16']))
    lib.utils.init_weight(net)
    if using_cuda:
        net.cuda()
    net.train()
    print(net)

    criterion = Loss.CTPN_Loss(using_cuda=using_cuda)  # 获取loss

    train_im_list, train_gt_list, val_im_list, val_gt_list = create_train_val()  # 获取训练、测试数据
    total_iter = len(train_im_list)
    print("total training image num is %s" % len(train_im_list))
    print("total val image num is %s" % len(val_im_list))

    train_loss_list = []
    test_loss_list = []

    # 开始迭代训练
    for i in range(epoch):
        if i >= change_epoch:
            lr = lr_behind
        else:
            lr = lr_front
        optimizer = optim.SGD(net.parameters(), lr=lr, momentum=0.9, weight_decay=0.0005)
        #optimizer = optim.Adam(net.parameters(), lr=lr)
        iteration = 1
        total_loss = 0
        total_cls_loss = 0
        total_v_reg_loss = 0
        total_o_reg_loss = 0
        start_time = time.time()

        random.shuffle(train_im_list)  # 打乱训练集
        # print(random_im_list)
        for im in train_im_list:
            root, file_name = os.path.split(im)
            root, _ = os.path.split(root)
            name, _ = os.path.splitext(file_name)
            gt_name = 'gt_' + name + '.txt'

            gt_path = os.path.join(root, "train_gt", gt_name)

            if not os.path.exists(gt_path):
                print('Ground truth file of image {0} not exists.'.format(im))
                continue

            gt_txt = lib.dataset_handler.read_gt_file(gt_path)  # 读取对应的标签
            #print("processing image %s" % os.path.join(img_root1, im))
            img = cv2.imread(im)
```

```
            if img is None:
                iteration += 1
                continue

            img, gt_txt = lib.dataset_handler.scale_img(img, gt_txt)  # 图像和标签做归一化
            tensor_img = img[np.newaxis, :, :, :]
            tensor_img = tensor_img.transpose((0, 3, 1, 2))
            if using_cuda:
                tensor_img = torch.FloatTensor(tensor_img).cuda()
            else:
                tensor_img = torch.FloatTensor(tensor_img)

            vertical_pred, score, side_refinement = net(tensor_img)  # 正向计算，获取预测结果
            del tensor_img

            # transform bbox gt to anchor gt for training
            positive = []
            negative = []
            vertical_reg = []
            side_refinement_reg = []

            visual_img = copy.deepcopy(img)  # 该图用于可视化标签

            try:
                # loop all bbox in one image
                for box in gt_txt:
                    # generate anchors from one bbox
                    gt_anchor, visual_img = lib.generate_gt_anchor.generate_gt_anchor(img,
box, draw_img_gt=visual_img)  # 获取图像的anchor标签
                    positive1, negative1, vertical_reg1, side_refinement_reg1 =
lib.tag_anchor.tag_anchor(gt_anchor, score, box) # 计算预测值反映在anchor层面的数据
                    positive += positive1
                    negative += negative1
                    vertical_reg += vertical_reg1
                    side_refinement_reg += side_refinement_reg1
            except:
                print("warning: img %s raise error!" % im)
                iteration += 1
                continue

            if len(vertical_reg) == 0 or len(positive) == 0 or len(side_refinement_reg) == 0:
                iteration += 1
                continue

            cv2.imwrite(os.path.join(DRAW_PREFIX, file_name), visual_img)
            optimizer.zero_grad()
            # 计算误差
            loss, cls_loss, v_reg_loss, o_reg_loss = criterion(score, vertical_pred,
side_refinement, positive,
                                                               negative, vertical_reg,
side_refinement_reg)
            # 反向传播
            loss.backward()
            optimizer.step()
            iteration += 1
            # save gpu memory by transferring loss to float
            total_loss += float(loss)
            total_cls_loss += float(cls_loss)
            total_v_reg_loss += float(v_reg_loss)
            total_o_reg_loss += float(o_reg_loss)

            if iteration % display_iter == 0:
                end_time = time.time()
                total_time = end_time - start_time
                print('Epoch: {2}/{3}, Iteration: {0}/{1}, loss: {4}, cls_loss: {5},
v_reg_loss: {6}, o_reg_loss: {7}, {8}'.
                      format(iteration, total_iter, i, epoch, total_loss / display_iter,
total_cls_loss / display_iter,
                             total_v_reg_loss / display_iter, total_o_reg_loss /
display_iter, im))

                logger.info('Epoch: {2}/{3}, Iteration: {0}/{1}'.format(iteration,
total_iter, i, epoch))
                logger.info('loss: {0}'.format(total_loss / display_iter))
                logger.info('classification loss: {0}'.format(total_cls_loss / display_iter))
```

```
                logger.info('vertical regression loss: {0}'.format(total_v_reg_loss /
display_iter))
                logger.info('side-refinement regression loss: {0}'.format(total_o_reg_loss /
display_iter))

                train_loss_list.append(total_loss)

                total_loss = 0
                total_cls_loss = 0
                total_v_reg_loss = 0
                total_o_reg_loss = 0
                start_time = time.time()

            # 定期验证模型性能
            if iteration % val_iter == 0:
                net.eval()
                logger.info('Start evaluate at {0} epoch {1} iteration.'.format(i,
iteration))
                val_loss = evaluate.val(net, criterion, val_batch_size, using_cuda, logger,
val_im_list)
                logger.info('End evaluate.')
                net.train()
                start_time = time.time()
                test_loss_list.append(val_loss)

            # 定期存储模型
            if iteration % save_iter == 0:
                print('Model saved at ./model/ctpn-{0}-{1}.model'.format(i, iteration))
                torch.save(net.state_dict(), './model/ctpn-msra_ali-{0}-{1}.model'.format(i,
iteration))

        print('Model saved at ./model/ctpn-{0}-end.model'.format(i))
        torch.save(net.state_dict(), './model/ctpn-msra_ali-{0}-end.model'.format(i))

        # 画出loss的变化图
        draw_loss_plot(train_loss_list, test_loss_list)
```

缩放图像具有一定规则：首先要保证文本框label的最短边也要等于600。我们通过 `scale = float(shortest_side)/float(min(height, width))` 来求得图像的缩放系数，对原始图像进行缩放。同时我们也要对我们的label也要根据该缩放系数进行缩放。

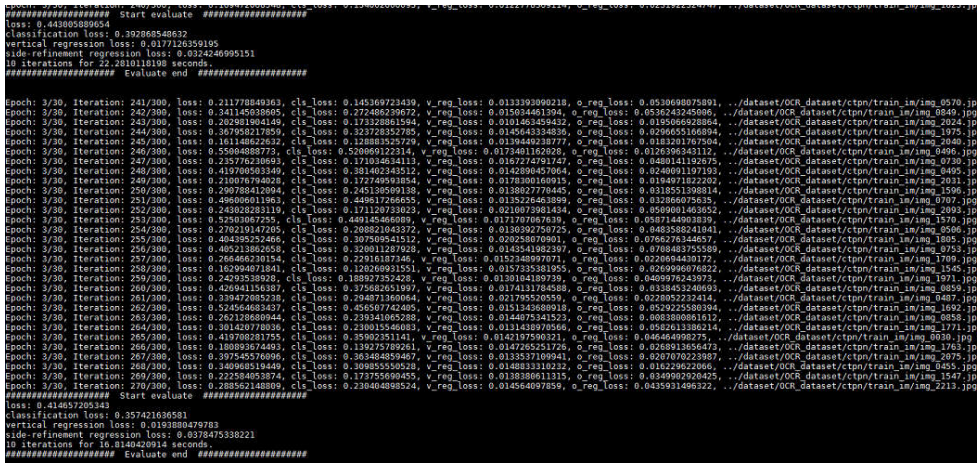
```
def scale_img(img, gt, shortest_side=600):
    height = img.shape[0]
    width = img.shape[1]
    scale = float(shortest_side)/float(min(height, width))
    img = cv2.resize(img, (0, 0), fx=scale, fy=scale)
    if img.shape[0] < img.shape[1] and img.shape[0] != 600:
        img = cv2.resize(img, (600, img.shape[1]))
    elif img.shape[0] > img.shape[1] and img.shape[1] != 600:
        img = cv2.resize(img, (img.shape[0], 600))
    elif img.shape[0] != 600:
        img = cv2.resize(img, (600, 600))
    h_scale = float(img.shape[0])/float(height)
    w_scale = float(img.shape[1])/float(width)
    scale_gt = []
    for box in gt:
        scale_box = []
        for i in range(len(box)):
            # x坐标
            if i % 2 == 0:
                scale_box.append(int(int(box[i]) * w_scale))
            # y坐标
            else:
                scale_box.append(int(int(box[i]) * h_scale))
        scale_gt.append(scale_box)
    return img, scale_gt
```

验证集评估：

```
def val(net, criterion, batch_num, using_cuda, logger):
    img_root = '../dataset/OCR_dataset/ctpn/test_im'
    gt_root = '../dataset/OCR_dataset/ctpn/test_gt'
    img_list = os.listdir(img_root)
```

```
    total_loss = 0
    total_cls_loss = 0
    total_v_reg_loss = 0
    total_o_reg_loss = 0
    start_time = time.time()
    for im in random.sample(img_list, batch_num):
        name, _ = os.path.splitext(im)
        gt_name = 'gt_' + name + '.txt'
        gt_path = os.path.join(gt_root, gt_name)
        if not os.path.exists(gt_path):
            print('Ground truth file of image {0} not exists.'.format(im))
            continue

        gt_txt = Dataset.port.read_gt_file(gt_path, have_BOM=True)
        img = cv2.imread(os.path.join(img_root, im))
        img, gt_txt = Dataset.scale_img(img, gt_txt)
        tensor_img = img[np.newaxis, :, :, :]
        tensor_img = tensor_img.transpose((0, 3, 1, 2))
        if using_cuda:
            tensor_img = torch.FloatTensor(tensor_img).cuda()
        else:
            tensor_img = torch.FloatTensor(tensor_img)

        vertical_pred, score, side_refinement = net(tensor_img)
        del tensor_img
        positive = []
        negative = []
        vertical_reg = []
        side_refinement_reg = []
        for box in gt_txt:
            gt_anchor = Dataset.generate_gt_anchor(img, box)
            positive1, negative1, vertical_reg1, side_refinement_reg1 =
Net.tag_anchor(gt_anchor, score, box)
            positive += positive1
            negative += negative1
            vertical_reg += vertical_reg1
            side_refinement_reg += side_refinement_reg1

        if len(vertical_reg) == 0 or len(positive) == 0 or len(side_refinement_reg) == 0:
            batch_num -= 1
            continue

        loss, cls_loss, v_reg_loss, o_reg_loss = criterion(score, vertical_pred,
side_refinement, positive,
                                                           negative, vertical_reg,
side_refinement_reg)
        total_loss += loss
        total_cls_loss += cls_loss
        total_v_reg_loss += v_reg_loss
        total_o_reg_loss += o_reg_loss
    end_time = time.time()
    total_time = end_time - start_time
    print('####################  Start evaluate  ####################')
    print('loss: {0}'.format(total_loss / float(batch_num)))
    logger.info('Evaluate loss: {0}'.format(total_loss / float(batch_num)))

    print('classification loss: {0}'.format(total_cls_loss / float(batch_num)))
    logger.info('Evaluate vertical regression loss: {0}'.format(total_v_reg_loss /
float(batch_num)))

    print('vertical regression loss: {0}'.format(total_v_reg_loss / float(batch_num)))
    logger.info('Evaluate side-refinement regression loss: {0}'.format(total_o_reg_loss /
float(batch_num)))

    print('side-refinement regression loss: {0}'.format(total_o_reg_loss / float(batch_num)))
    logger.info('Evaluate side-refinement regression loss: {0}'.format(total_o_reg_loss /
float(batch_num)))

    print('{1} iterations for {0} seconds.'.format(total_time, batch_num))
    print('####################  Evaluate end  ####################')
    print('\n')
```

训练过程：



训练效果与预测效果

测试效果：输入一张图片，给出最后的检测结果

```
def infer_one(im_name, net):
    im = cv2.imread(im_name)
    im = lib.dataset_handler.scale_img_only(im)  # 归一化图像
    img = copy.deepcopy(im)
    img = img.transpose(2, 0, 1)
    img = img[np.newaxis, :, :, :]
    img = torch.Tensor(img)
    v, score, side = net(img, val=True)  # 送入网络预测
    result = []
    # 根据分数获取有文字的anchor
    for i in range(score.shape[0]):
        for j in range(score.shape[1]):
            for k in range(score.shape[2]):
                if score[i, j, k, 1] > THRESH_HOLD:
                    result.append((j, k, i, float(score[i, j, k, 1].detach().numpy())))

    # nms过滤
    for_nms = []
    for box in result:
        pt = lib.utils.trans_to_2pt(box[1], box[0] * 16 + 7.5, anchor_height[box[2]])
        for_nms.append([pt[0], pt[1], pt[2], pt[3], box[3], box[0], box[1], box[2]])
    for_nms = np.array(for_nms, dtype=np.float32)
    nms_result = lib.nms.cpu_nms(for_nms, NMS_THRESH)

    out_nms = []
    for i in nms_result:
        out_nms.append(for_nms[i, 0:8])

    # 确定哪几个anchors是属于一组的
    connect = get_successions(v, out_nms)
    # 将一组anchors合并成一条文本线
    texts = get_text_lines(connect, im.shape)

    for box in texts:
        box = np.array(box)
        print(box)
        lib.draw_image.draw_ploy_4pt(im, box[0:8])

    _, basename = os.path.split(im_name)
    cv2.imwrite('./infer_'+basename, im)
```

推断时提到了 `get_successions` 用于获取一个预测文本行里的所有anchors，换句话说，我们得到的很多预测有字符的anchor，但是我们怎么知道哪些acnhors可以组成一个文本线呢？所以我们需要实现一个anchor合并算法，这也是CTPN代码实现中最为困难的一步。

CTPN论文提到，文本线构造法如下：文本行构建很简单，通过将那些text/no-text score > 0.7的连续的text proposals相连接即可。文本行的构建如下。

- 首先，为一个proposal Bi定义一个邻居（Bj）：Bj->Bi，其中：

1. Bj在水平距离上离Bi最近

2. 该距离小于50 pixels

- 它们的垂直重叠(vertical overlap) > 0.7

一看理论很简单，但是一到自己实现就困难重重了。真是应了那句“纸上得来终觉浅，绝知此事要躬行”啊！

`get_successions` 传入的参数是v代表每个预测anchor的h和y信息，anchors代表每个anchors的四个顶点坐标信息。

```
def get_successions(v, anchors=[]):
    texts = []
    for i, anchor in enumerate(anchors):
        neighbours = []  # 记录每组的anchors
        neighbours.append(i)
        center_x1 = (anchor[2] + anchor[0]) / 2
        h1 = get_anchor_h(anchor, v)  # 获取该anchor的高度
        # find i's neighbour
        # 遍历余下的anchors，找出邻居
        for j in range(i + 1, len(anchors)):
            center_x2 = (anchors[j][2] + anchors[j][0]) / 2 # 中心点x坐标
            h2 = get_anchor_h(anchors[j], v)
            # 如果这两个Anchor间的距离小于50，而且他们的它们的垂直重叠(vertical overlap)大于一定阈值，那就是邻居
            if abs(center_x1 - center_x2) < NEIGHBOURS_MIN_DIST and \
                    meet_v_iou(max(anchor[1], anchors[j][1]), min(anchor[3], anchors[j][3]), h1, h2):  # less than 50 pixel between each anchor
                neighbours.append(j)

        if len(neighbours) != 0:
            texts.append(neighbours)

    # 通过上面的步骤，我们已经把每一个anchor的邻居都找到并加入了对应的集合中了，现在我们
    # 通过一个循环来不断将每个小组合并
    need_merge = True
    while need_merge:
        need_merge = False
        # ok, we combine again.
        for i, line in enumerate(texts):
            if len(line) == 0:
                continue
            for index in line:
                for j in range(i+1, len(texts)):
                    if index in texts[j]:
                        texts[i] += texts[j]
                        texts[i] = list(set(texts[i]))
                        texts[j] = []
                        need_merge = True

    result = []
    #print(texts)
    for text in texts:
        if len(text) < 2:
            continue
        local = []
        for j in text:
            local.append(anchors[j])
        result.append(local)
    return result
```

当我们得到一个文本框的anchors组合后，接下来要做的就是将组内的anchors串联成一个文本框。

`get_text_lines` 函数做的就是这个功能。

```
def get_text_lines(text_proposals, im_size, scores=0):
    """
    text_proposals:boxes

    """
    text_lines = np.zeros((len(text_proposals), 8), np.float32)

    for index, tp_indices in enumerate(text_proposals):
        text_line_boxes = np.array(tp_indices)  # 每个文本行的全部小框
```

```
        #print(text_line_boxes)
        #print(type(text_line_boxes))
        #print(text_line_boxes.shape)
        X = (text_line_boxes[:, 0] + text_line_boxes[:, 2]) / 2  # 求每一个小框的中心x，y坐标
        Y = (text_line_boxes[:, 1] + text_line_boxes[:, 3]) / 2
        #print(X)
        #print(Y)

        z1 = np.polyfit(X, Y, 1)  # 多项式拟合，根据之前求的中心店拟合一条直线（最小二乘）

        x0 = np.min(text_line_boxes[:, 0])  # 文本行x坐标最小值
        x1 = np.max(text_line_boxes[:, 2])  # 文本行x坐标最大值

        offset = (text_line_boxes[0, 2] - text_line_boxes[0, 0]) * 0.5  # 小框宽度的一半

        # 以全部小框的左上角这个点去拟合一条直线，然后计算一下文本行x坐标的极左极右对应的y坐标
        lt_y, rt_y = fit_y(text_line_boxes[:, 0], text_line_boxes[:, 1], x0 + offset, x1 -
offset)
        # 以全部小框的左下角这个点去拟合一条直线，然后计算一下文本行x坐标的极左极右对应的y坐标
        lb_y, rb_y = fit_y(text_line_boxes[:, 0], text_line_boxes[:, 3], x0 + offset, x1 -
offset)

        #score = scores[list(tp_indices)].sum() / float(len(tp_indices))  # 求全部小框得分的均值
作为文本行的均值

        text_lines[index, 0] = x0
        text_lines[index, 1] = min(lt_y, rt_y)  # 文本行上端 线段 的y坐标的小值
        text_lines[index, 2] = x1
        text_lines[index, 3] = max(lb_y, rb_y)  # 文本行下端 线段 的y坐标的大值
        text_lines[index, 4] = scores  # 文本行得分
        text_lines[index, 5] = z1[0]  # 根据中心点拟合的直线的k，b
        text_lines[index, 6] = z1[1]
        height = np.mean((text_line_boxes[:, 3] - text_line_boxes[:, 1]))  # 小框平均高度
        text_lines[index, 7] = height + 2.5

    text_recs = np.zeros((len(text_lines), 9), np.float32)
    index = 0
    for line in text_lines:
        b1 = line[6] - line[7] / 2  # 根据高度和文本行中心线，求取文本行上下两条线的b值
        b2 = line[6] + line[7] / 2
        x1 = line[0]
        y1 = line[5] * line[0] + b1  # 左上
        x2 = line[2]
        y2 = line[5] * line[2] + b1  # 右上
        x3 = line[0]
        y3 = line[5] * line[0] + b2  # 左下
        x4 = line[2]
        y4 = line[5] * line[2] + b2  # 右下
        disX = x2 - x1
        disY = y2 - y1
        width = np.sqrt(disX * disX + disY * disY)  # 文本行宽度

        fTmp0 = y3 - y1  # 文本行高度
        fTmp1 = fTmp0 * disY / width
        x = np.fabs(fTmp1 * disX / width)  # 做补偿
        y = np.fabs(fTmp1 * disY / width)
        if line[5] < 0:
            x1 -= x
            y1 += y
            x4 += x
            y4 -= y
        else:
            x2 += x
            y2 += y
            x3 -= x
            y3 -= y
        # clock-wise order
        text_recs[index, 0] = x1
        text_recs[index, 1] = y1
        text_recs[index, 2] = x2
        text_recs[index, 3] = y2
        text_recs[index, 4] = x4
        text_recs[index, 5] = y4
        text_recs[index, 6] = x3
        text_recs[index, 7] = y3
```

```
        text_recs[index, 8] = line[4]
        index = index + 1

    text_recs = clip_boxes(text_recs, im_size)

    return text_recs
```

检测效果和总结

首先看一下训练出来的模型的文字检测效果，为了便于观察，我把anchor和最终合并好的文本框一并画出：

下面再看看一些比较好的文字检测效果吧：

在实现过程中的一些总结和想法：

1. CTPN对于带旋转角度的文本的检测效果不好，其实这是CTPN的算法特点决定的：一个个固定宽度的四边形是很难合并出一个准确的文本框，比如一些anchors很难组成一组，即使组成一组了也很难精确恢复成完整的精确的文本矩形框（推断阶段的缺点）。当然啦，对于水平排布的文本检测，个人认为这个算法思路还是很奏效的。
2. CTPN中的side-refinement其实作用不大，如果我们检测出来的文本是直接拿出识别，这个side-refinement优化的几个像素差别其实可以忽略；
3. CTPN的中间步骤有点多：从anchor标签的生成到中间计算loss再到最后推断的文本线生成步骤，都会引入一定的误差，这个缺点也是EAST论文中所提出的。训练的步骤越简洁，中间过程越少，精度更有保障。
4. CTPN的算法得出的效果可以看出，准确率低但召回率高。这种基于16像素的anchor识别感觉对于一些大的非文字图标（比如路标）误判率相当高，这是源于其anchor的宽度实在太小了，尽管使用了lstm关联周围anchor，但是我还是认为有点“一叶障目”的感觉。所以CTPN对于过大或过小的文字检测效果不会太好。
5. CTPN是个比较老的算法了（2016年），其思路在当年还是很创新的，但是也有很多弊端。现在提出的新方法已经基本解决了这些不足之处，比如EAST，PixelNet都是一些很优秀的新算法。

CTPN的完整实现可以参考我的Github

分类： OCR系列

Madcola
关注 - 30
粉丝 - 1334
+加关注

8 0

« 上一篇：【OCR技术系列之五】自然场景文本检测技术综述（CTPN, SegLink, EAST）
» 下一篇：我的2018：OCR、实习和秋招

posted @ 2018-12-02 17:59 Madcola 阅读(6904) 评论(32) 编辑 收藏

评论列表

#1楼 2018-12-19 21:06 **笠翁**

你好，感谢公布了这么多有用的代码，想问下，可否给一份文中的两个训练集？再次感谢：diegoinsh[at]163.com

支持(0) 反对(0)

#2楼[楼主] 2018-12-20 09:59 **Madcola**

@ 笠翁
文末有github地址，数据集里面有

支持(0) 反对(0)

#3楼 2018-12-26 17:47 **乌羽**

非常感谢，两篇文章对于刚接触OCR的人非常有意义。
关于“一个文本框边缘像素刚好落在一个新的anchor上，那么我们就要为这个像素分配一个16像素的anchor，显然导致了文本框标签的不准确，引入了15像素的误差”这个问题，请问有什么思路么

支持(0) 反对(0)

#4楼 2019-01-21 11:02 **张博的博客**

厉害，感谢你开源

支持(0) 反对(0)

#5楼 2019-01-28 16:31 **狮子森林**

请问，在统一微软训练集标签时，a1 = -angle + math.atan(h / float(w)) # 旋转角度-对角线与底线所成的角度
a2 = -angle - math.atan(h / float(w)) # 旋转角度+对角线与底线所成的角度，
其中math.atan(h / float(w))应该是math.atan(float(w)/h)对么？？

支持(0) 反对(0)

#6楼 2019-01-31 11:59 **静悟生慧**

楼主写的非常棒，感谢！ 最后总结中： 5. EAST是个比较老的算法了（2016年）... 这里应该是笔误了 CTPN

支持(0) 反对(0)

#7楼[楼主] 2019-01-31 15:07 **Madcola**

@ 静悟生慧
谢谢提醒，已改正：）

支持(0) 反对(0)

#8楼 2019-02-27 18:50 **心冷2080**

大佬 太社会了 简直大佬

支持(0) 反对(0)

#9楼 2019-03-20 20:57 **zyh的打怪历程**

博主你好，一直在学习你的博客，很有收获，很感谢你。我想问一下infer.py里import的lib.nms在你分享的GitHub上没有这个文件，我看你引用的那篇文章是用了cython写的，我直接把nms.pyx文件下载过来，出现了很多问题，一直没搞定。能否麻烦你分享一下nms.py文件，我想在我的机器上训练学习一下，谢谢了

支持(2) 反对(0)

#10楼[楼主] 2019-03-21 15:11 **Madcola**

@ zyh的打怪历程
在CTPN文件夹下make就可以了，需要先编译的才可以用

支持(0) 反对(0)

#11楼 2019-03-24 11:09 **芝士条**

博主你好，我看训练时一个照片处理时间约3秒，这是否可以理解成很慢，有很大的优化空间？第一次跑OCR代码，对这块没什么概念，请多指教。

支持(0) 反对(0)

#12楼 2019-03-31 18:33 **征服天堂jj**

调试了两天的程序终于跑起来了，太感动了，感谢大佬奉献的源码

支持(0) 反对(0)

#13楼 2019-04-08 09:09 **征服天堂jj**

麻烦能问一下博主这个数据集怎么制作吗，或者可以使用什么软件制作啊，四个坐标八个参数手动标太难搞了

支持(0) 反对(0)

#14楼 2019-04-08 14:53 **Keep_exercising**

@ zyh的打怪历程
你的问题解决了嘛, 我现在也遇到同样的问题

支持(0) 反对(0)

#15楼 2019-04-08 17:08 **zyh的打怪历程**

@ Keep_exercising
我使用的楼主给的docker，然后往容器里下载安装里cython，把楼主GitHub上给的放进去，能运行了。

支持(0) 反对(0)

#16楼 2019-04-09 18:40 **DLKKILL**

博主您好，想请教您一个问题
最近我再做银行卡号的识别，想通过神经网络进行文本定位，但是我没有太多的完整银行卡图片数据，但有不少截取出来的卡号图片，请问我用这些图片训练可以么

支持(0) 反对(0)

#17楼[楼主] 2019-04-10 09:13 **Madcola**

@ 不可以的

支持(0) 反对(0)

#18楼 2019-04-10 14:55 **DLKKILL**

@ skyfsm

引用

@ 不可以的

好的，谢谢博主解答
那个在打扰一下，如果我使用自然场景文本图片数据训练网络训练出来的网络迁移到银行卡上，效果会不会很差

支持(0) 反对(0)

#19楼[楼主] 2019-04-10 16:21 **Madcola**

@ DLKKILL
我觉得不会很差，毕竟银行卡场景很简单

支持(0) 反对(0)

#20楼 2019-04-10 16:25 **DLKKILL**

@ Madcola
但是银行卡种类很多呀，而且受拍摄条件影响，可能会产生反光，模糊等各种情况，而且银行卡号使用的字体和正常字体不太

一样

支持(0) 反对(0)

#21楼[楼主] 2019-04-10 20:07 Madcola

@ DLKKILL
不知道你的具体场景是什么，现在银行卡识别都是受限的场景下的拍照，比如时要有个拍摄指导框指导你拍摄的，得到的银行卡图像不会太差

支持(0) 反对(0)

#22楼 2019-04-10 22:50 DLKKILL

主要是现在手里我有的测试数据图片很少而且还都不太好==，另外其实还有个问题，银行卡号一般都是靠突起来显示数字，他和背景颜色并没有太大的区别 。我曾经试过用传统的图像处理办法来识别文字区域。效果只在一两种银行卡上还可以。所有现在考虑想用深度学习来试一试。深度学习小白一个，十分感谢博主耐心指点orz

支持(0) 反对(0)

#23楼 2019-04-11 15:36 十四行诗r

为什么我想试着训练楼主提供的数据集，结果显示读入图片是空呢？我在win10，py3.6下运行的train.py

支持(0) 反对(0)

#24楼 2019-04-11 16:07 十四行诗r

我下载楼主提供的GitHub源码，可以检测图片，想试着调通楼主的train.py,一直不成功，结果在调试的时候，在输入train.py177行img = cv2.imread（im），读入为空导致无法训练。。有没有朋友调通train,py文件，小白，希望得到解答，谢谢楼主提供的源码。

支持(0) 反对(0)

#25楼 2019-04-13 11:42 十四行诗r

训练不收敛，8个epoch了，损失还是降不下，有朋友遇到过类似问题吗？求解决办法

支持(0) 反对(0)

#26楼 2019-04-16 20:06 jakonson

楼主你好，请问你的大框平均得分那一处你给注释掉了，请问要怎么计算平均得分，还有VGG16输出来的分数为什么大于1，望解答

支持(0) 反对(0)

#27楼 2019-05-05 15:59 mzw0509

@ zyh的打怪历程
楼主给的镜像我下载不下来啊，总是报错。
能提供一下你下载的镜像包吗?
982503758@qq.com,谢谢!

支持(0) 反对(0)

#28楼 2019-05-27 10:11 jingwanli

请问一下楼主，我运行了您开源的ctpn代码，ctpn检测一张图片的时间有点长。我最开始因为是nms的原因，我测试了gpu_nm以及cython_nms，但发现测试的时间差异不大。我想请问一下楼主哪些还能优化，谢谢!

支持(0) 反对(0)

#29楼 2019-06-04 15:31 牧童祭歌

epoch:1/10训练了15h,感觉好慢啊，不知道有没有相似的情况。

支持(0) 反对(0)

#30楼 2019-06-05 15:47 **牧童祭歌**

@ zyh的打怪历程
你好，我也遇到和你相同的问题，缺少lib.nms（尝试make但是不得其门），可以把你的lib.nms发我一份吗?
(1584280029@qq.com)

支持(0) 反对(0)

#31楼 2019-06-10 11:25 **y_shan**

您好博主，请问下博主分享的这份代码是需要在linux下运行才行吗?

支持(0) 反对(0)

#32楼 2019-06-27 11:31 **牧童祭歌**

新人求教：博主在train.py中有这么一句代码
vertical_pred, score, side_refinement = net(tensor_img)
并把score在此句使用
positive1, negative1, vertical_reg1, side_refinement_reg1 = lib.tag_anchor.tag_anchor(gt_anchor, score, box)
在函数内部用的是 height = score.shape[2]
width = score.shape[3]他们的值固定都是37,37，
想知道为什么？不太理解这不分?

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

相关博文：
· ICDAR2015数据处理及训练
· FasterR-CNN:TowardsReal-TimeObjectDetectionwithRegionProposalNetworks论文理解
· 【OCR技术系列之五】自然场景文本检测技术综述（CTPN,SegLink,EAST）
· rcnn系列
· Faster RCNN 学习笔记