文本检测与识别

Intro to CnStd & CnOcr

Breezedeus, 2021.09

贝叶智能
Beiye.ai

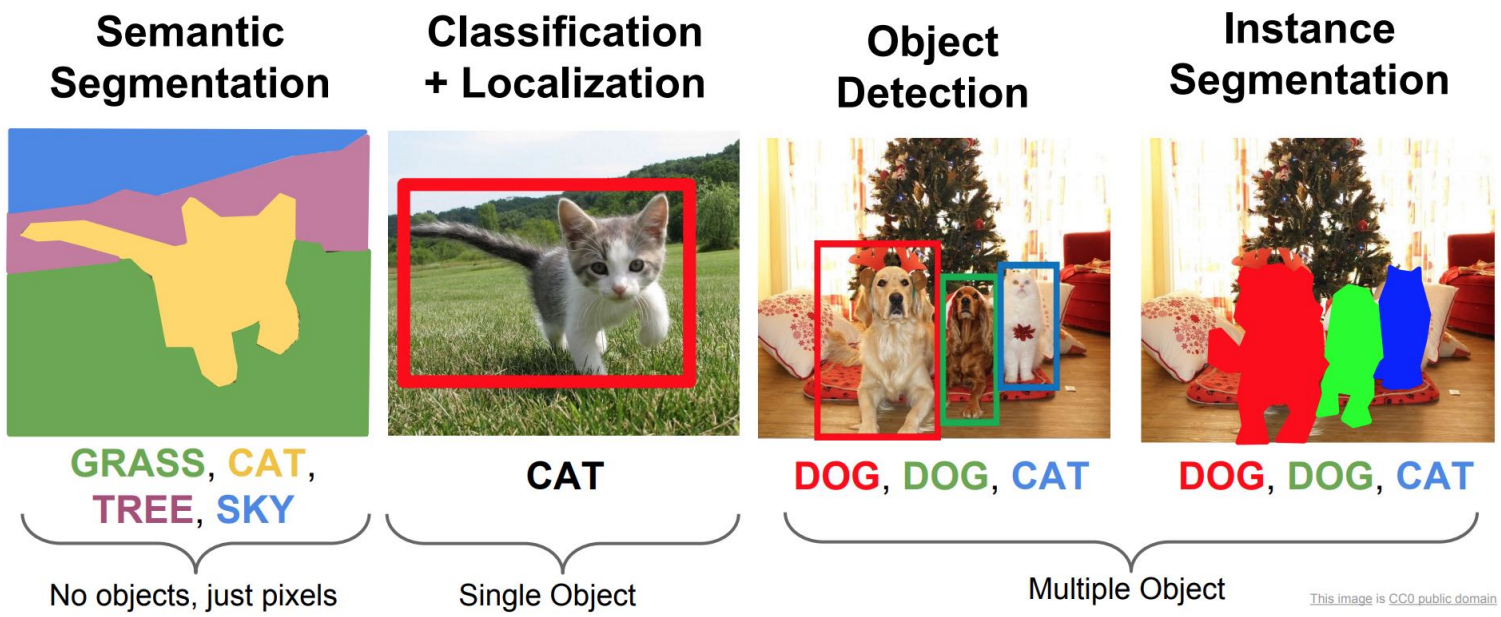
场景文字识别流程


吃派
吃饼
吃包
吃小鱼
不明真相的
吃豆腐
吃辣椒
吃点心
喝早茶
没得吃
群众
文本检测
(CnStd)
文本框
抽取
文字识别
(CnOcr)
国
哈
吃点心
没得吃
喝早茶
吃辣椒
吃豆腐
吃小鱼
不明真相的
吃包
吃派
吃饼

PART I: 场景文字检测（Scene Text Detection, STD）

- 检测图片中所在的文字位置

Segmentation, Localization and Detection Problems



- **分类+定位**：对一张图片进行分类，同时定位出所在位置
- **实例分割**区分同种类的不同实体，**语义分割**则不做区分

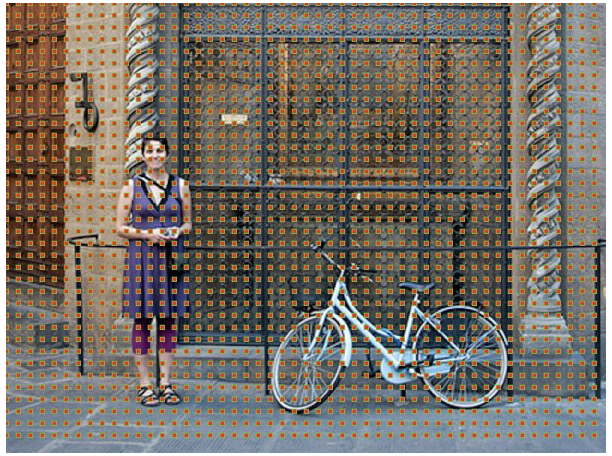
STD 模型分类

Regression-based / Segmentation-based

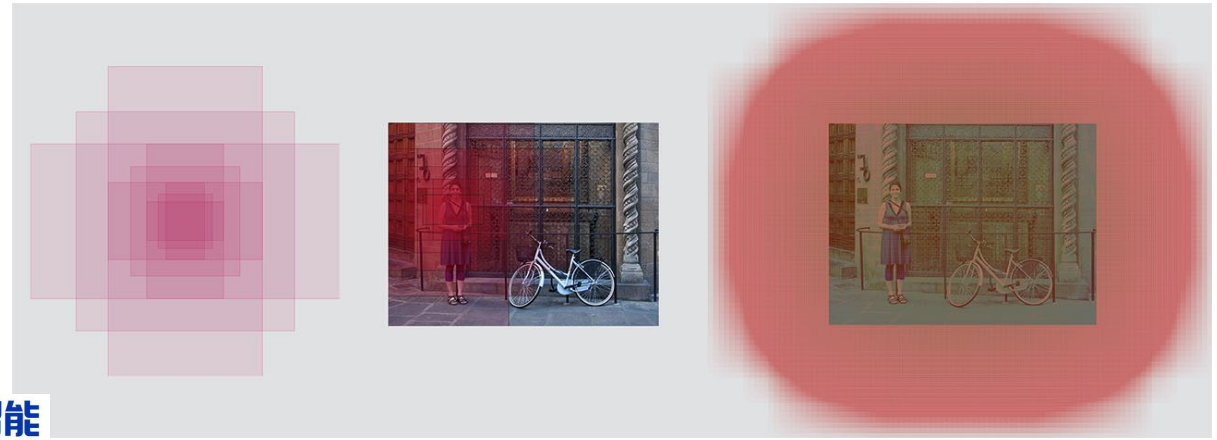
- **Regression-based models**: directly regress the bounding boxes of the text instances
  - **simple post-processing algorithms (NMS)**
  - hard to represent accurate bounding boxes for irregular shapes
- **Segmentation-based models**: combine pixel-level prediction and post-processing algorithms to get the bounding boxes
  - usually not end-to-end training
  - hard post-processing algorithms, resulting in lower inference speed
  - **easy to represent accurate bounding boxes for irregular shapes**

回归模型常用结构

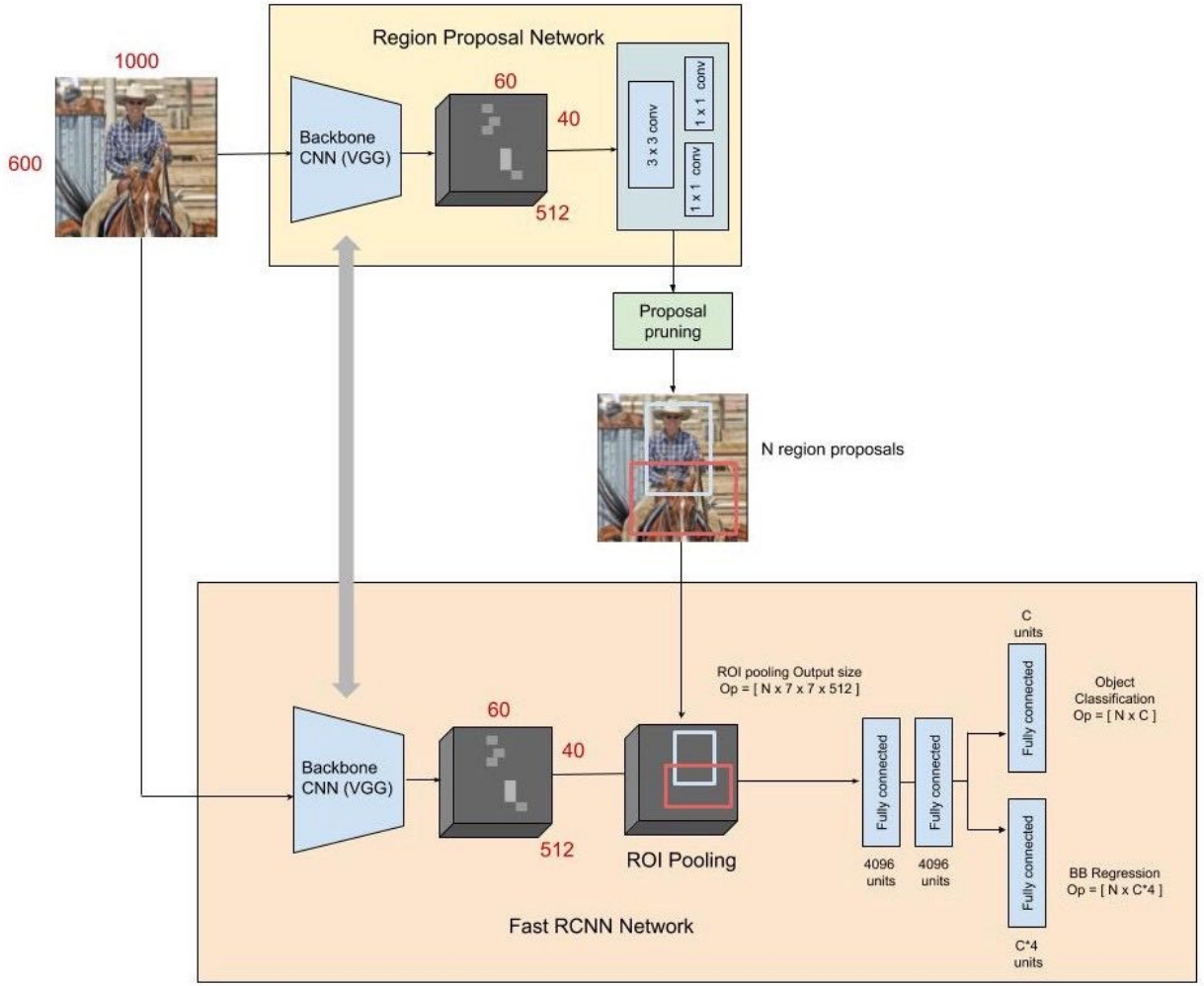
- 设定**锚点（Anchor）**位置



- 设定**候选区域（Region）**



回归模型常用结构

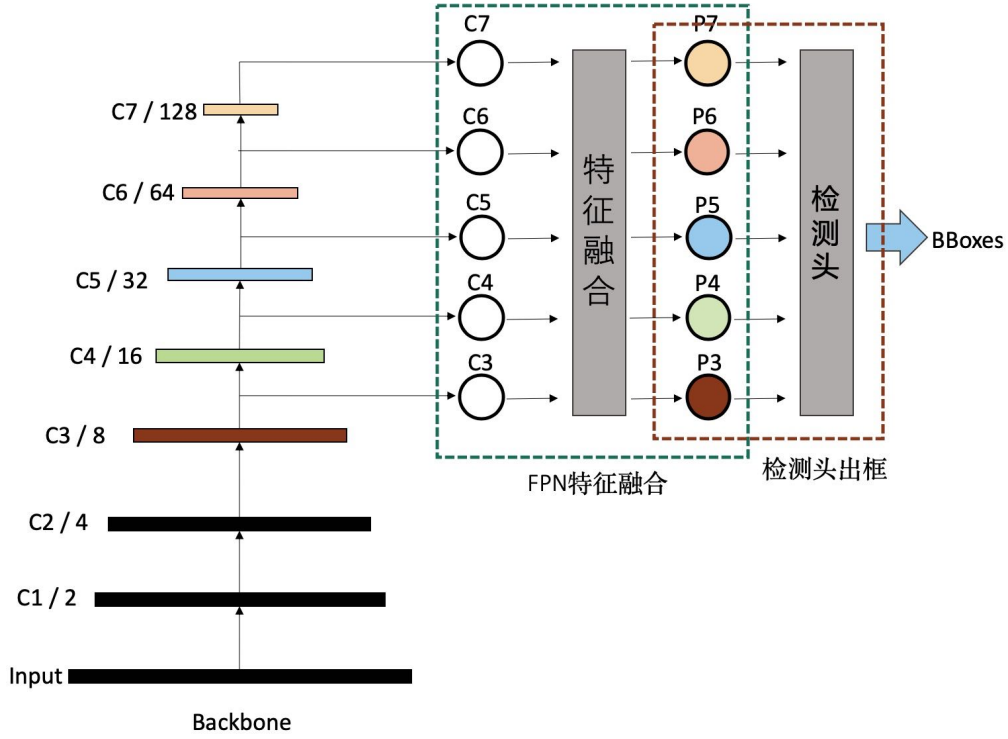

Region Proposal Network
1000
600
Backbone CNN (VGG)
60
40
512
3 x 3 conv
1 x 1 conv
1 x 1 conv
Proposal pruning
N region proposals
ROI pooling Output size
Op = [N x 7 x 7 x 512]
Backbone CNN (VGG)
60
40
512
ROI Pooling
Fully connected
Fully connected
4096 units
4096 units
C units
Fully connected
Object Classification
Op = [N x C]
Fully connected
BB Regression
Op = [N x C*4]
C*4 units
Fast RCNN Network

回归模型常用结构

CTPN (2016)

分割模型常用结构

1. **Backbone** 生成特征阶段
   - 生成不同尺度下的 feature maps
2. **Neck** 特征融合阶段
   - 融合不同尺度下的 feature maps
3. **Head** 输出bounding box
   - 输出最终的预测结果



DBNet 介绍

- 分割模型的常见流程（蓝色箭头）
- **DBNet**（红色箭头）：二值化图的计算方式可导，故可在训练中直接优化loss
  - 旷视 2019 年工作
  - **DB**: Differentiable Binarization

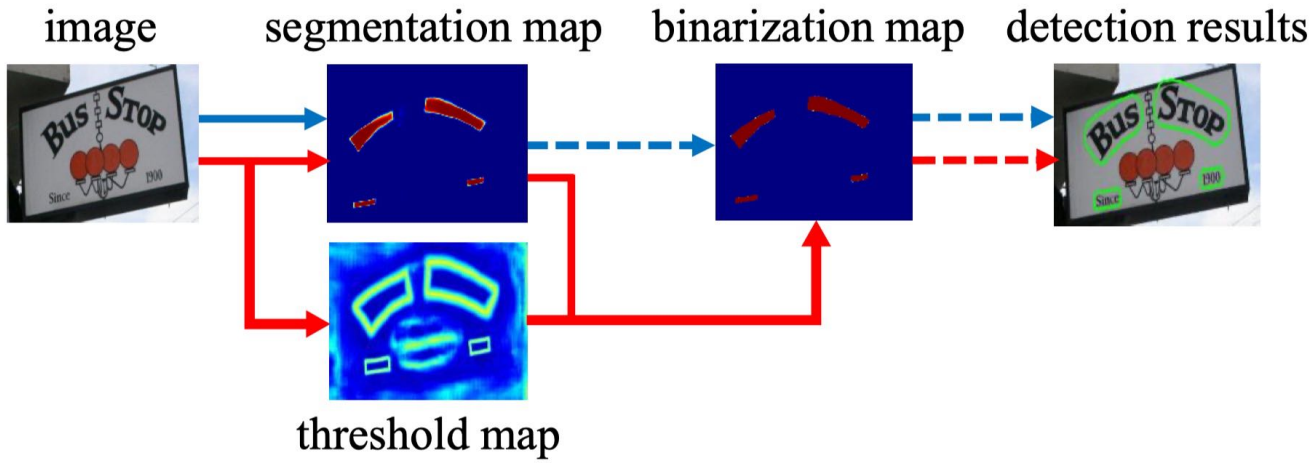
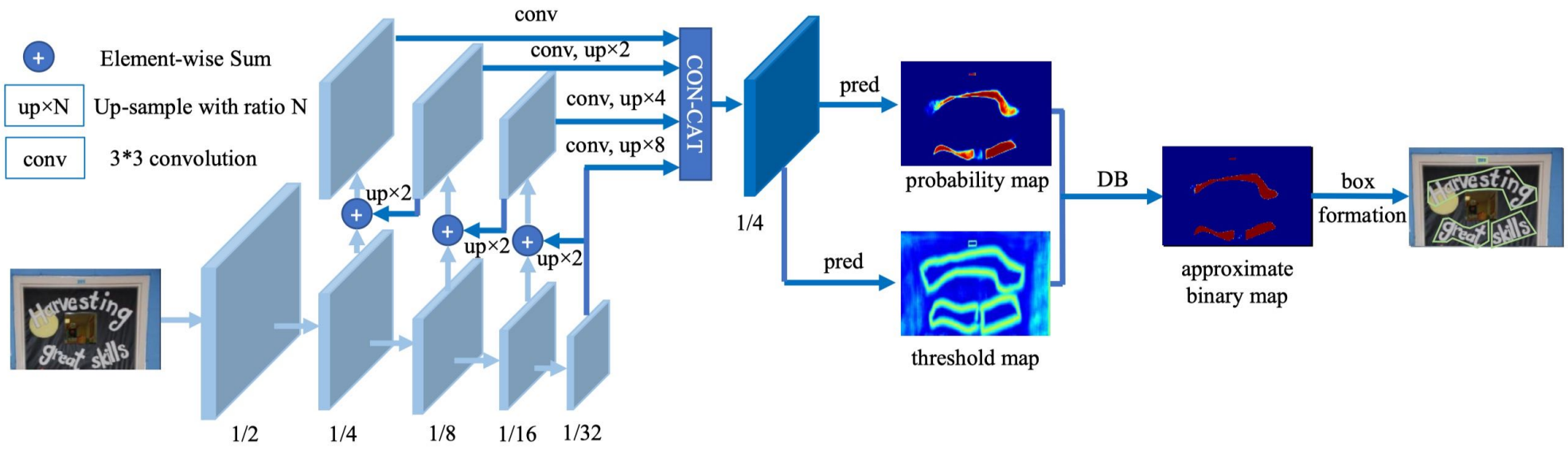


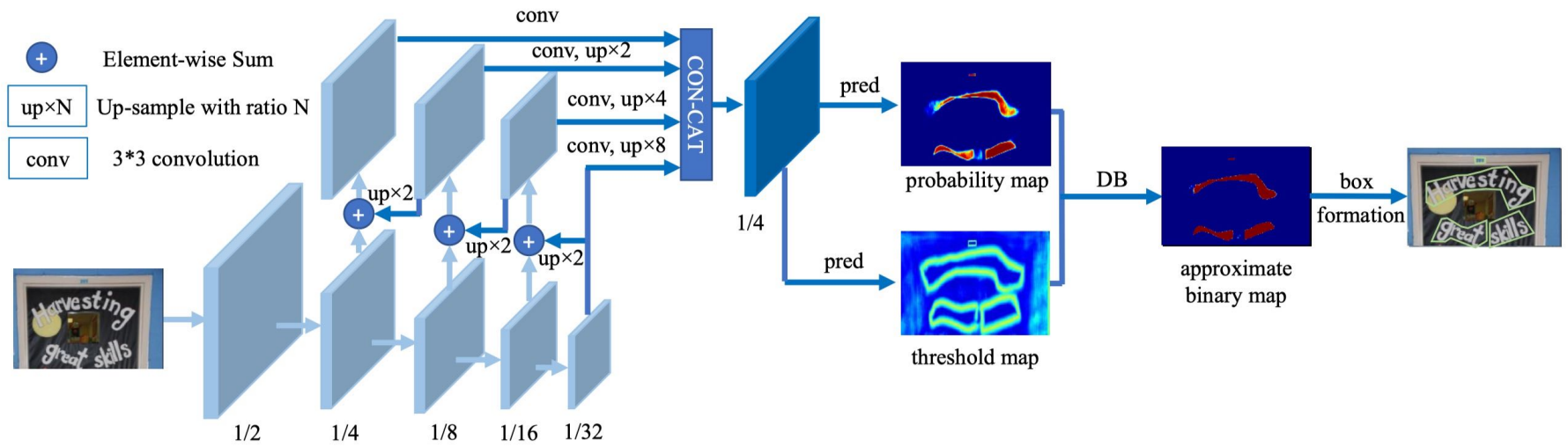
Figure 2: Traditional pipeline (blue flow) and our pipeline (red flow). Dashed arrows are the inference only operators; solid arrows indicate differentiable operators in both training and inference.

DBNet 架构

- **DBNet** 架构： Backbone(ResNet) + Neck(FPN) + Prob/Thresh Heads



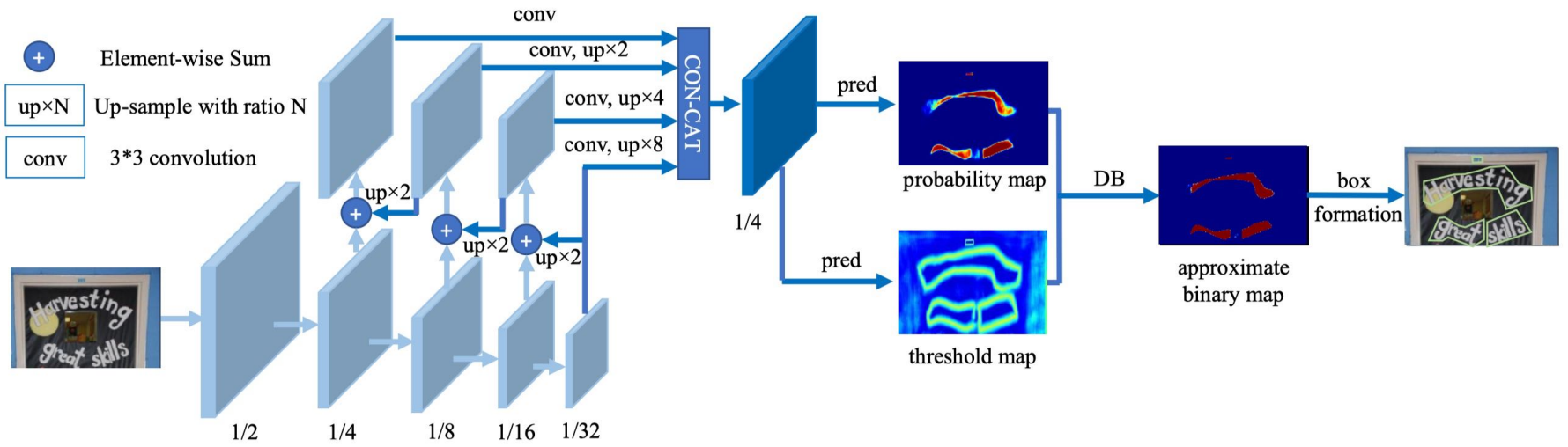
DBNet 架构



Backbone(ResNet)

- 把原图逐渐压缩为 $\frac{1}{4}$、$\frac{1}{8}$、$\frac{1}{16}$ 和 $\frac{1}{32}$ 大小
- 不一定非要 ResNet，各种逐层压缩的模型（backbone）都 ok

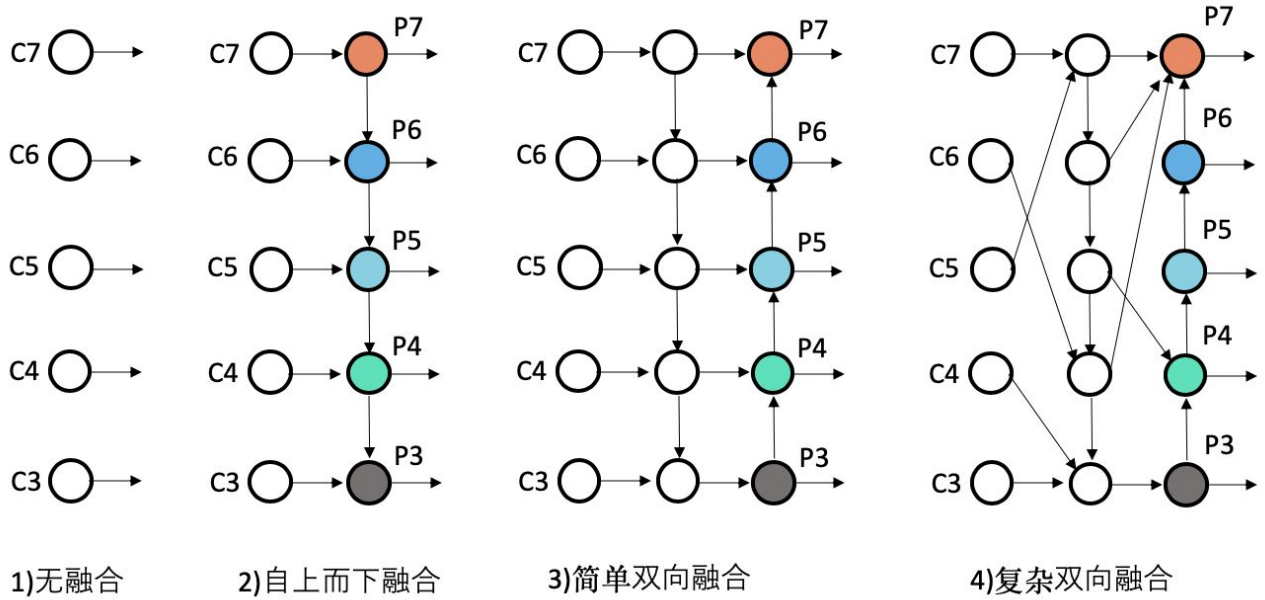
DBNet 架构



Neck (Feature Pyramid Network, FPN)

- 把之前的 $\frac{1}{4}$、$\frac{1}{8}$、$\frac{1}{16}$ 和 $\frac{1}{32}$ 大小的maps，从后往前加一遍，然后全部 upsample 到原图的 $\frac{1}{4}$ 大小
- 最后这些 $\frac{1}{4}$ 大小的maps，直接在channel维度拼接为 **4倍** 厚度的 map，作为后续模块的输入

DBNet 架构



Neck 演进：FPN、PAN、BiFPN、…

- 高效融合不同尺度信息（参考资料：1、2）

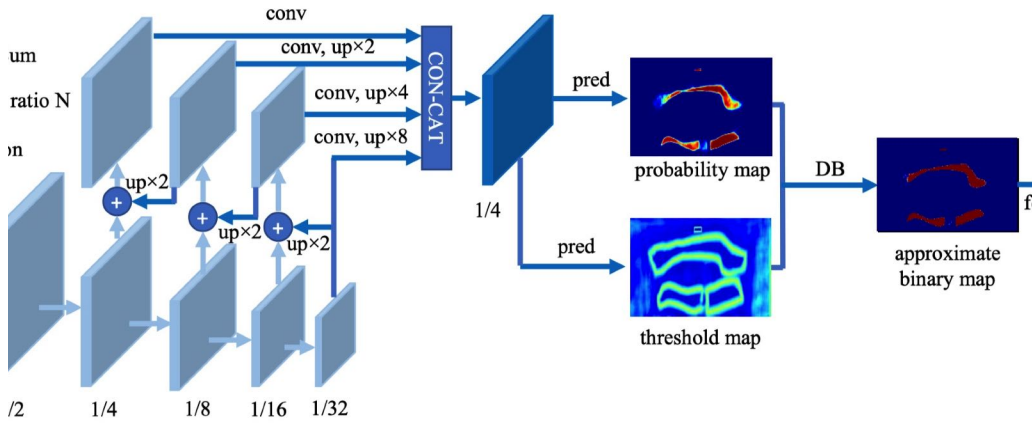
DBNet 架构

Prob/Thresh Heads

- 利用两次逆卷积操作，把输入的 $\frac{1}{4}$ 大小的 map，恢复为原图大小的输出
- 两个头使用了完全相同的结构
- 基于概率map P 和 阈值 map T，获得二值化预测结果：

$$\hat{B}_{i,j} = \frac{1}{1 + e^{-k(P_{i,j} - T_{i,j})}}$$

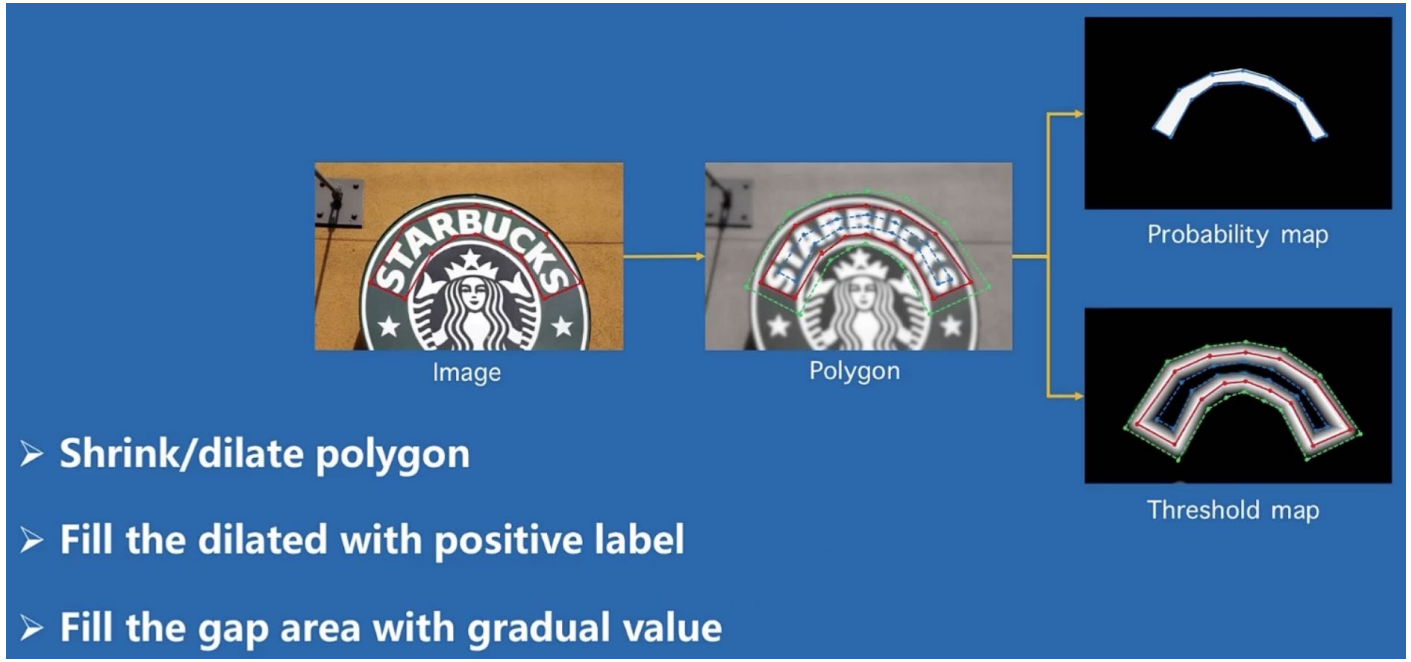
  - k 为超参数，可取 $k = 50$



DBNet 训练样本

训练 labels 的生成

- 使用以下方式产生训练使用的 prob & threshold maps



DBNet 训练Loss

训练 Loss

$$L = L_s + \alpha \times L_b + \beta \times L_t$$

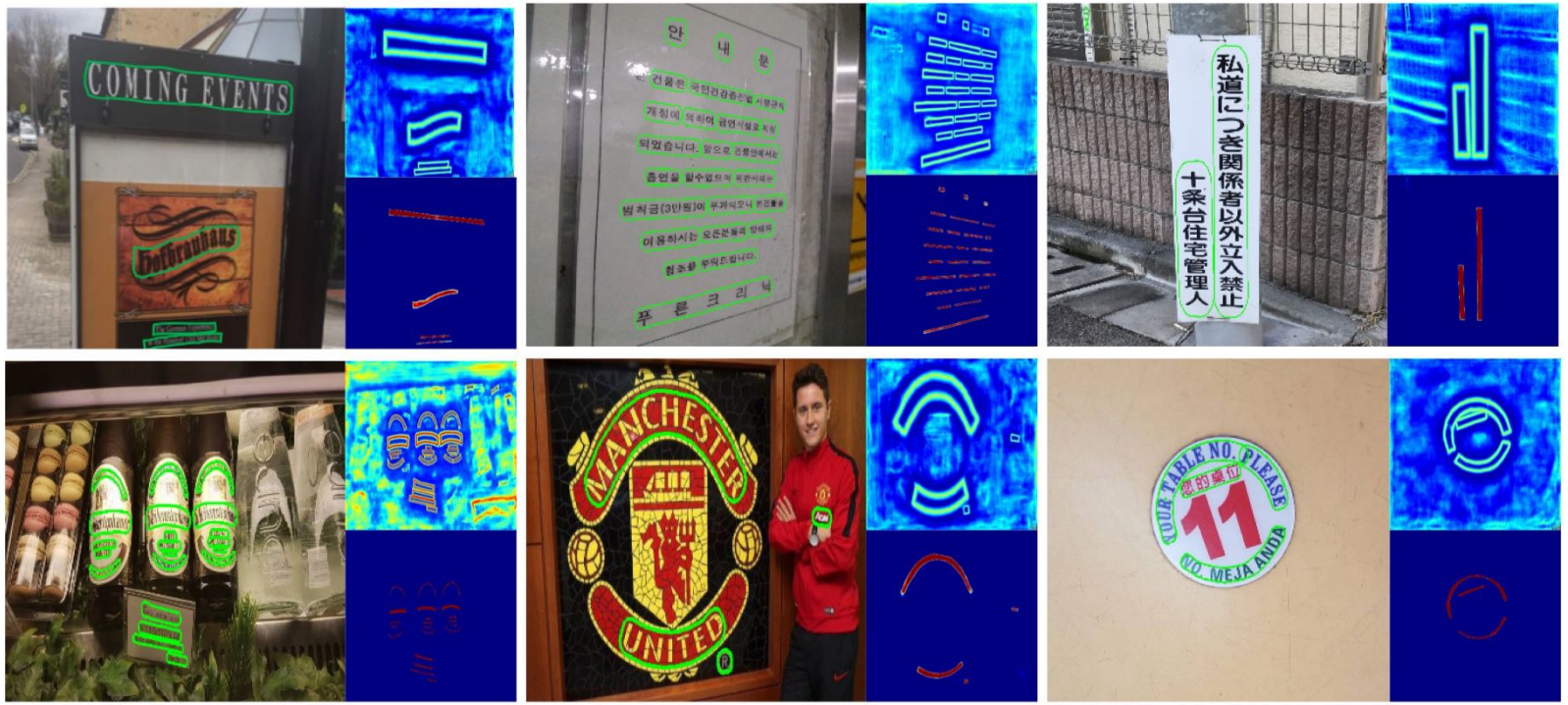
其中 α 和 β 为超参数。由三部分组成：

- 基于概率 map 的 $L_s = \sum_{i \in S_l} y_i \log x_i + (1 - y_i) \log (1 - x_i)$
  - S_l 是抽取的集合，保证正负样本比例为 `1:3`
- 基于二值化 map 的 L_b，形式与 L_s 相同
- 基于阈值 map 的 `L1` 距离 $L_t = \sum_{i \in R_d} |y_i^* - x_i^*|$
  - R_d 为膨胀后的文本框中的像素集合

DBNet 效果展示

官方样例

- 可识别任意形状文本



CnStd 介绍

CnStd：Python 3 下的场景文字检测（Scene Text Detection，简称STD）工具包，支持中文、英文等语言的文字检测，自带多个训练好的检测模型

V0.1 → V1.0：

- MXNet → **PyTorch**
- PSENet → DBNet

CnStd 使用

- **安装**

```
pip install cnstd
```

> 注：cnstd 依赖 **opencv**，可能需要额外安装opencv

- **使用**

```
from cnstd import CnStd
std = CnStd()
box_info_list = std.detect('examples/taobao.jpg')
```

- **在线 Demo**

CnStd 使用

自带模型

模型名称	迭代次数	参数规模	模型文件大小	测试集精度（IoU）	平均推断耗时（秒/张）	下载方式
db_resnet18	29	12.3 M	47 M	0.554	1.04	自动
db_resnet34	33	22.5 M	86 M	0.6144	1.58	自动
db_mobilenet_v3	30	4.2 M	16 M	0.5949	0.83	下载链接

> 首次使用 **cnstd** 时，系统会自动从 贝叶智能 下载zip格式的模型压缩文件

CnStd 中：

- Backbone 支持不同尺寸的模型：**ResNet**、**MobilenetV3**、**ShufflenetV2**
- Neck 支持结构：**fpn**、**pan**

CnStd 使用

初始化（详见 CnStd文档）

```
class CnStd(object):
    """
    场景文字检测器（Scene Text Detection）。虽然名字中有个"Cn"（Chinese），但其实也可以轻松识别英文的。
    """

    def __init__(
        self,
        model_name: str = 'db_resnet18',  # 模型名称
        model_epoch: Optional[int] = None,  # 模型迭代次数
        *,
        auto_rotate_whole_image: bool = False,  # 是否自动对整张图片进行旋转调整
        rotated_bbox: bool = True,  # 是否支持检测带角度的文本框
        context: str = 'cpu',  # 预测使用的机器资源
        model_fp: Optional[str] = None,  # 如果不使用系统自带的模型，可以通过此参数直接指定所使用的模型文件
        root: Union[str, Path] = data_dir(),  # 模型文件所在的根目录
        **kwargs,
    ):
```

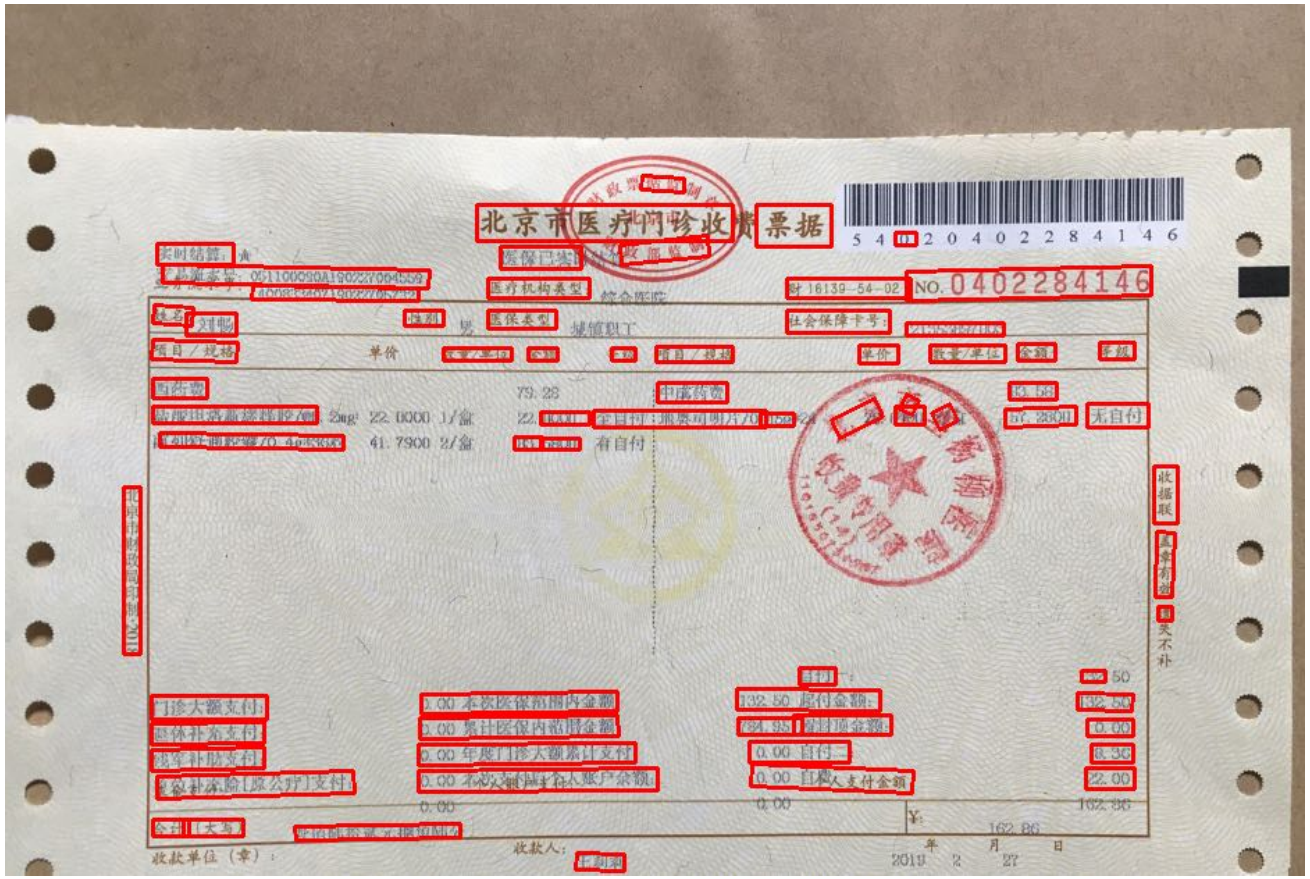
CnStd 使用

类函数 `CnStd.detect()` （详见 CnStd文档）

```
def detect(
    self,
    img_list: Union[
        str,
        Path,
        Image.Image,
        np.ndarray,
        List[Union[str, Path, Image.Image, np.ndarray]],
    ],  # 支持对单个图片或者多个图片（列表）的检测
    resized_shape: Tuple[int, int] = (768, 768),  # (height, width)，检测前会先把原始图片 resize 到此大小
    preserve_aspect_ratio: bool = True,  # 对原始图片 resize 时是否保持高宽比不变
    min_box_size: int = 8,  # 过滤掉高度或者宽度小于此值的文本框
    box_score_thresh: float = 0.3,  # 过滤掉得分低于此值的文本框
    batch_size: int = 20,  # 处理图片多个图片时每批处理的图片数量
    **kwargs,
) -> Union[Dict[str, Any], List[Dict[str, Any]]]:
```

CnStd 效果

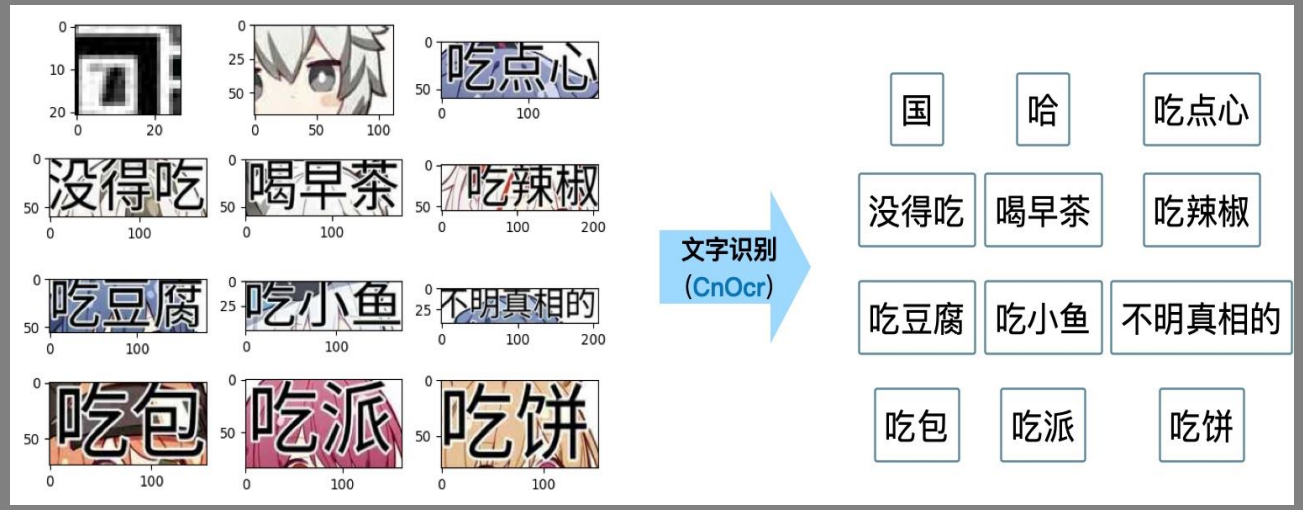
示例


北京市医疗门诊收费票据
NO.0402284146


11:24
自选 市场 基金 组合
新加坡 沪深港通 期货 外汇 指数 债券
全部A股
恒而达 300946 69.55 +20.00%
华辰装备 300809 30.38 +19.98%
利亚德 300296 10.45 +19.84%
聚光科技 300203 15.61 +18.62%
泰林生物 300813 76.00 +17.72%
创业板
恒而达 300946 69.55 +20.00%
华辰装备 300809 30.38 +19.98%
利亚德 300296 10.45 +19.84%
聚光科技 300203 15.61 +18.62%
泰林生物 300813 76.00 +17.72%
已上市新股
行情 交易 资讯 牛牛圈 我的

PART II: 光学文字识别（Optical Character Recognition，OCR）

- 识别文本框中的内容

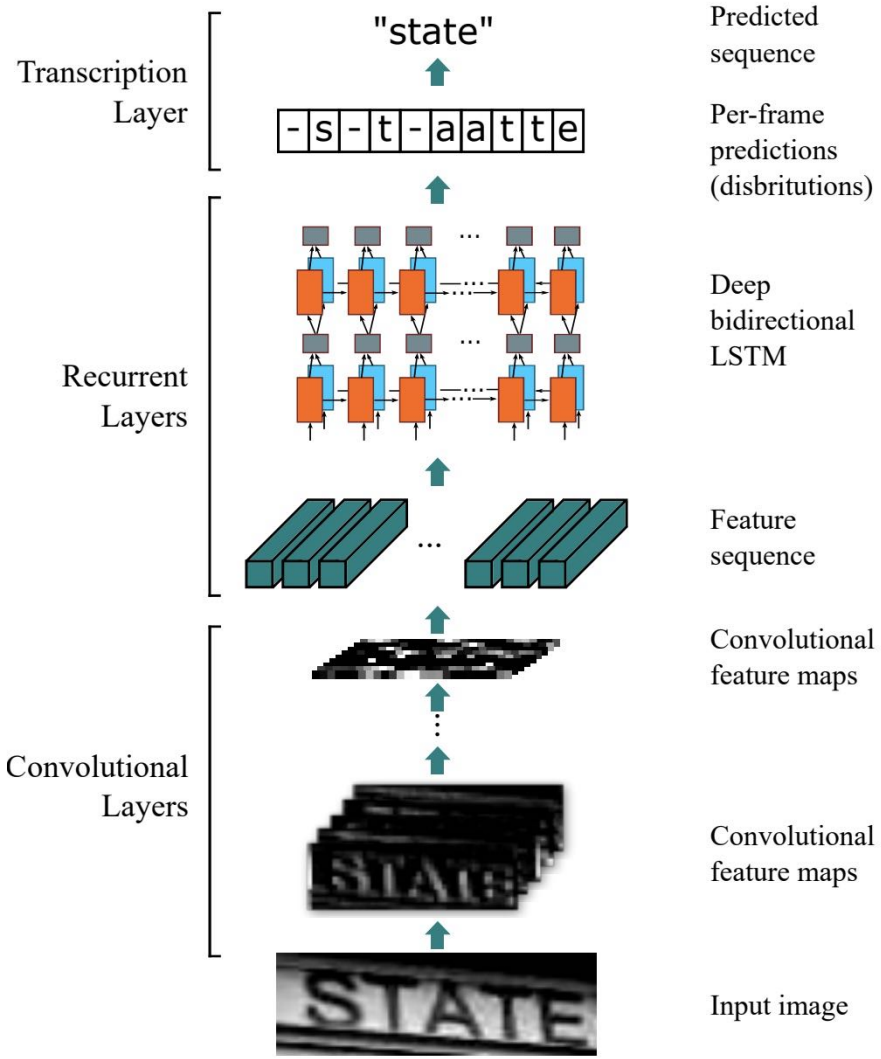


CRNN 介绍

CRNN (2015): Conv + RNN + CTC

- 使用 Conv 进行局部编码
- 使用 RNN 进行全局编码
- 使用 CTC Loss 进行训练

- 参考资料：1、2、3



CnOcr 介绍

CnOcr： Python 3 下的文字识别（Optical Character Recognition，简称OCR）工具包，支持**中文**、**英文**的常见字符识别，自带多个训练好的识别模型

V1.2 → V2.0：

- MXNet → **PyTorch**
- 优化了能识别的字符集合
- 重新生成训练数据，优化了对英文的识别效果
- 训练数据中加入了真实场景文字数据，优化了对场景文字的识别效果

CnOcr 使用

- 安装

```
pip install cnocr
```

- 使用

```
from cnocr import CnOcr

ocr = CnOcr()
res = ocr.ocr('examples/multi-line_cn1.png')
print("Predicted Chars:", res)
```

- 在线 Demo

CnOcr 使用

CRNN for CnOcr: DenseNet + GRU/LSTM/FC + CTC

自带模型

模型名称	局部编码模型	序列编码模型	模型大小	迭代次数	测试集准确率
densenet-s-gru	densenet-s	gru	11 M	11	95.5%
densenet-s-fc	densenet-s	fc	8.7 M	39	91.9%

> 首次使用 **cnocr** 时，系统会自动从 贝叶智能 下载zip格式的模型压缩文件

CnOcr 使用

初始化（详见 CnOcr文档）

```
class CnOcr(object):
    def __init__(
        self,
        model_name: str = 'densenet-s-fc'  # 模型名称
        model_epoch: Optional[int] = None,  # 模型迭代次数
        *,
        cand_alphabet: Optional[Union[Collection, str]] = None,  # 待识别字符的候选集合
        context: str = 'cpu',  # 预测使用的机器资源，['cpu', 'gpu', 'cuda']
        model_fp: Optional[str] = None,  # 如果不使用系统自带的模型，可以通过此参数直接指定所使用的模型文件
        root: Union[str, Path] = data_dir(),  # 模型文件所在的根目录
        **kwargs,
    ):
```

CnOcr 使用

- 类函数 `CnOcr.ocr(img_fp)` （详见 CnOcr文档）
  - 可以对包含**多行**或**单行**文字的图片进行文字识别

```
def ocr(
    self, img_fp: Union[str, Path, torch.Tensor, np.ndarray]
) -> List[Tuple[List[str], float]]:
```

- 类函数 `CnOcr.ocr_for_single_line(img_fp)` （详见 CnOcr文档）
  - 只能对包含**单行**文字的图片进行文字识别

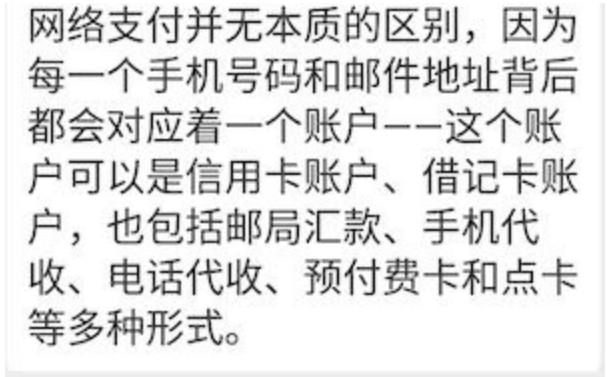
```
def ocr_for_single_line(
    self, img_fp: Union[str, Path, torch.Tensor, np.ndarray]
) -> Tuple[List[str], float]:
```

- 类函数 `CnOcr.ocr_for_single_lines(img_list)` （详见 CnOcr文档）
  - 可以对多个**单行**文字图片进行批量预测

```
def ocr_for_single_lines(
    self, img_list: List[Union[str, Path, torch.Tensor, np.ndarray]]
) -> List[Tuple[List[str], float]]:
```

CnOcr 效果

示例



返回结果：

```
Predicted Chars: [
(['网', '络', '支', '付', '并', '无', '本', '质', '的', '区', '别', ', ', '因', '为'], 0.8677546381950378),
(['每', '一', '个', '手', '机', '号', '码', '和', '邮', '件', '地', '址', '背', '后'], 0.6706454157829285),
(['都', '会', '对', '应', '着', '一', '个', '账', '户', '一', '一', '这', '个', '账'], 0.5052655935287476),
(['户', '可', '以', '是', '信', '用', '卡', '账', '户', '、', '借', '记', '卡', '账'], 0.7785991430282593),
(['户', ', ', '也', '包', '括', '邮', '局', '汇', '款', '、', '手', '机', '代'], 0.37458470463752747),
(['收', '、', '电', '话', '代', '收', '、', '预', '付', '费', '卡', '和', '点', '卡'], 0.7326119542121887),
(['等', '多', '种', '形', '式', '。'], 0.14462216198444366)]
```

应对通用场景的识别：CnStd ⊕ CnOcr

- **cnocr** 针对的是排版简单的印刷体文字图片（如截图），内置的文字检测和分行模块无法处理复杂的场景文字图片
- 对于一般的场景图片（如照片、票据等），需要先利用 **cnstd** 定位到文字位置，然后再使用 **cnocr** 进行文本识别

```
from cnstd import CnStd
from cnocr import CnOcr

std = CnStd()
cn_ocr = CnOcr()

box_infos = std.detect('examples/taobao.jpg')

for box_info in box_infos['detected_texts']:
    cropped_img = box_info['cropped_img']
    ocr_res = cn_ocr.ocr_for_single_line(cropped_img)
    print('ocr result: %s' % str(ocr_out))
```

未来工作

- 模型轻量化，效果持续优化
- 降低特殊场景下的精调代价
- 功能增强：表格、文档排版、数学公式、表情符
- 支持特殊场景的使用：车牌、身份证、票据

Thanks

贝叶智能
Beiye.ai