目标检测

<Excerpt in index | 首页摘要>
当前目标检测算法总结与一些实现
KeyWords Plus: **R-CNN** **Fast R-CNN** **Faster R-CNN** **YOLO**

- **Relevant blog**： Pytorch-book

  <The rest of contents | 余下全文>

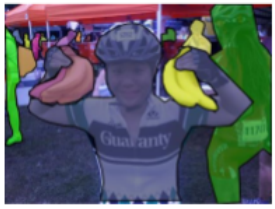
Introduction

数据库

- PASCAL VOC数据集
  检测20类物体 + 1个背景类



- COCO数据集
  80类物体检测

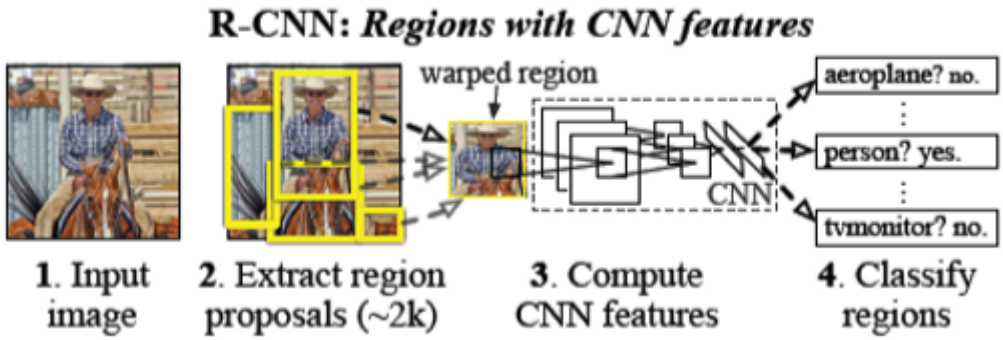

目标检测算法

R-CNN

物体检测：**搜索+分类**

首先需要产生物体可能的位置的**候选区域**，然后把这个区域送给CNN去 做**分类和检测框回归**，CNN可以使用在 ImageNet 数据集上预训练的模型。

缺陷：每个区域的分类器计算都是独立的，**速度太慢。**



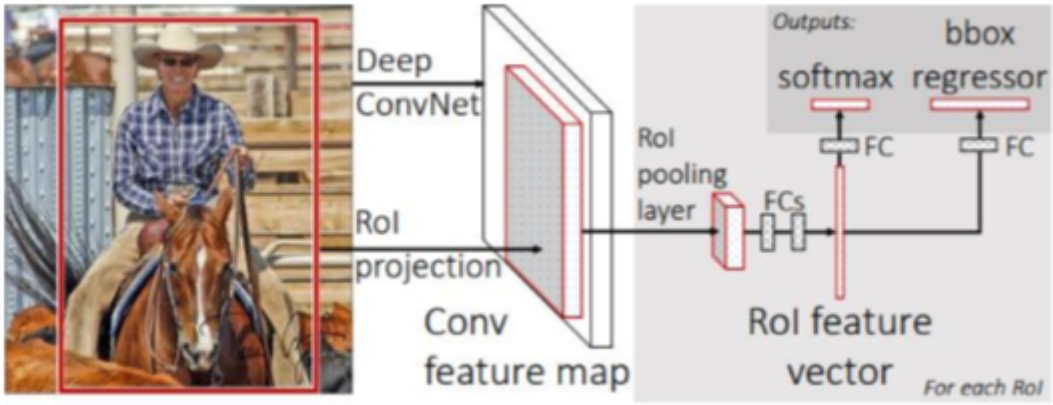
Fast R-CNN

物体检测：**搜索+分类**

输入图片直接输入到卷积网络**得到特征图**，特征计算可以被不同 的候选区域共享 .

主要贡献：避免了对相同区域的特征重复提取，**大大提高了检测 器的速度**

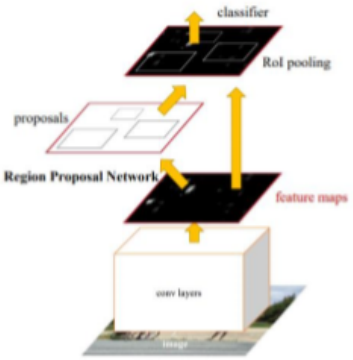
缺陷：**候选区域提取成了瓶颈。**

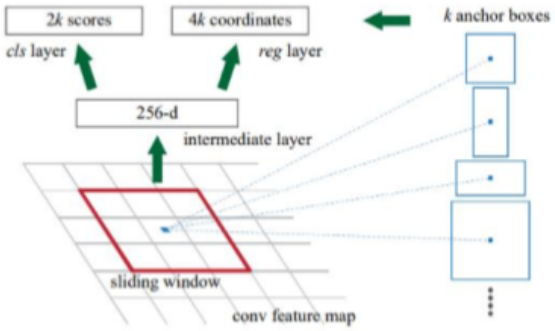


Faster R-CNN

物体检测：**搜索+分类**

引入了一个叫`Region Proposal Network(RPN)`的结构，所谓 RPN就是一个CNN，用来预测哪些地方可能有物体，并回归出 物体的位置.



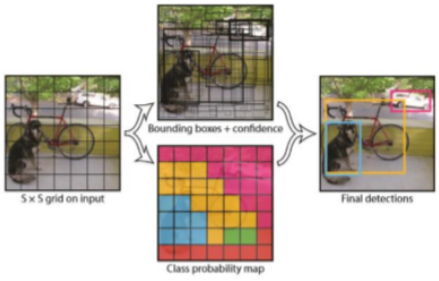


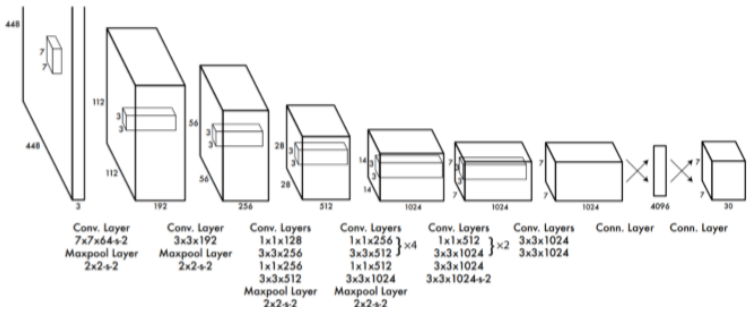
Region Proposal Network(RPN)

YOLO

物体检测：**一步到位的检测器**

利用整张图作为网络的输入，直接在输出层回归 `bounding box` 的 位置和 `bounding box`所属的类别

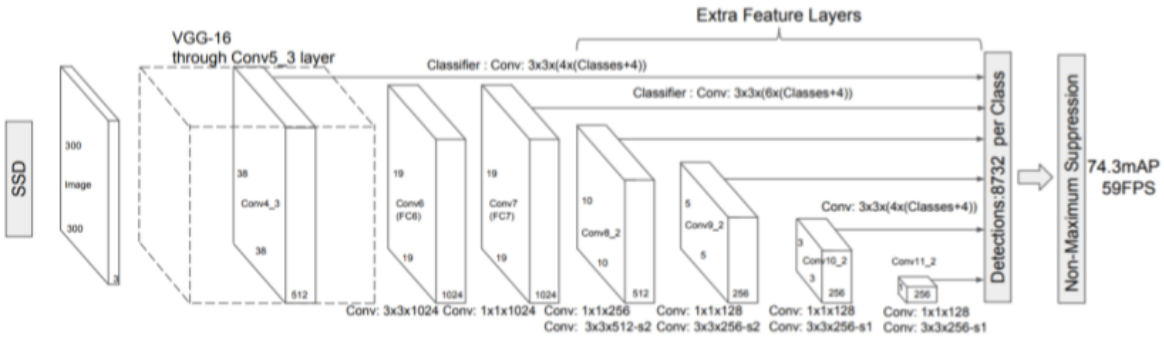




SSD

物体检测：**一步到位的检测器**

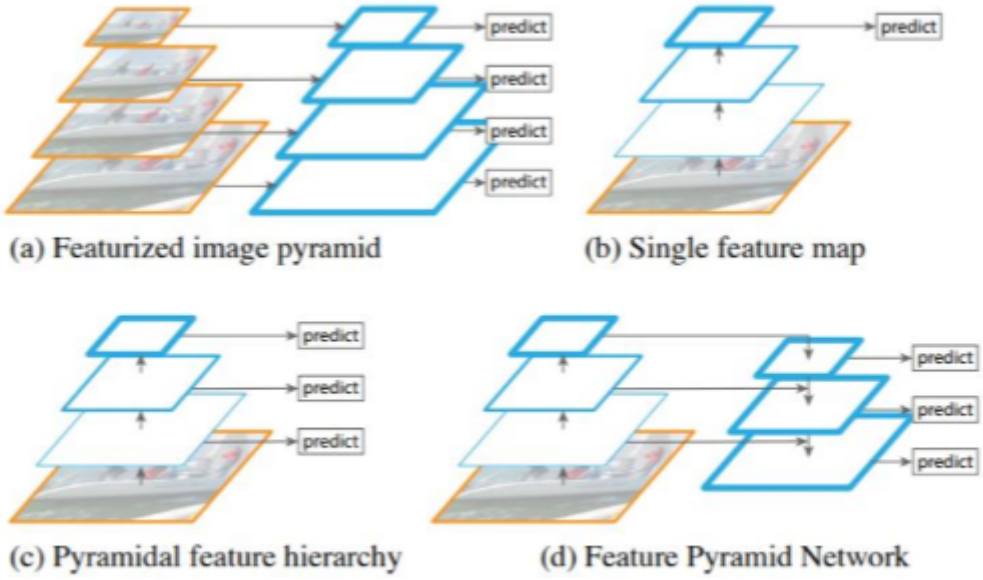
`SSD`可以看作是一个强化版的`RPN`，多尺度检测，**比较浅的输出层的分辨率比较高**，用来检测小一些 的物体，比较深的层分辨率低，检测大的物体。



FPN

物体检测：**多尺度检测**

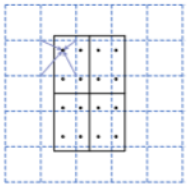
很好的**利用了各个卷积特征图**.预测是在不同特征层独立进行的。



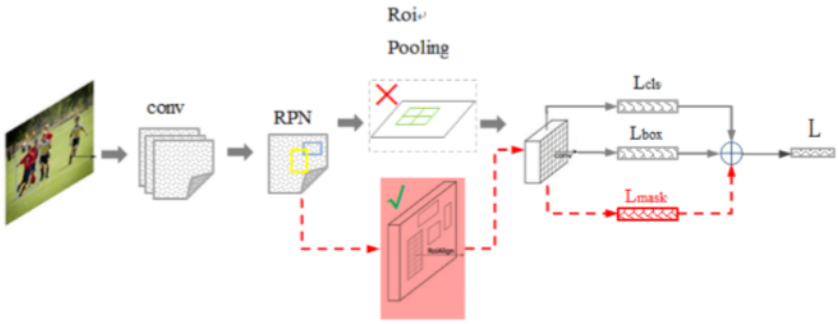
Mask R-CNN

将`Roi Pooling`层替换成了`RoiAlign`,解决了仅通过`Pooling`直 接采样带来的`misalignment`非对齐问题

在 Faster-RCNN 的基础上添加一个分支网络，在实现目标检测 的同时，把**目标像素分割出来。**



RoIAlign



Faster R-CNN

安装faster-rcnn

Caffe安装中多版本protoc选择问题
Ubuntu 16.04 在Conda沙盒环境下安装Caffe(Python2.7.15 + Protobuf2.6.1 + GPU)
Ubuntu14.04下安装protobuf 2.6.1
Caffe框架的机器学习——安装与问题锦集

重新编译`caffe`即可！重新编译之前一定记得`make clean`！！！

安装流程

环境要求

– `Ubuntu16.04, cuda8`

• 安装依赖库

```
        sudo apt-get install libprotobuf-dev libleveldb-dev libsnappy-dev
    libopencv-dev libhdf5-serial-dev protobuf-compiler libatlas-base-dev git
    python-dev python-setuptools cython python-numpy python-pip
        sudo apt-get install --no-install-recommends libboost-all-dev
        pip install scikit-image
        pip install easydict
        pip install Cython
        pip install PyYAML
        pip install opencv-python
```

• 安装 cuda8 https://developer.nvidia.com/cuda-80-ga2-download-archive

• 下载 faster rcnn 代码

```
    git clone --recursive https://github.com/rbgirshick/py-faster-rcnn.git
```

• 进入 py-faster-rcnn 代码目录

```
    cd py-faster-rcnn
```

• 编译 lib 库模块

```
    cd lib && make && cd ..
```

• 编译 caffe 和 pycaffe http://caffe.berkeleyvision.org/installation.html - compilation

```
    cd caffe-fast-rcnn
    cp Makefile.config.example Makefile.config
```

• 修改 Makefile.config 内容

`WITH_PYTHON_LAYER:= 1`

• make -j8

• 编译 pycaffe

注意事项

protoc版本问题

`编译caffe需要的protoc版本是protobuf-2.6.1。`

若想用`/usr/bin/protoc`则:

命令 whereis protoc 可以查看哪些路径下安装了protoc
命令which protoc 可以查看默认选用protoc的路径
命令 protoc --version 可以查看当前protoc版本

指定protoc的版本可以在 Makefile文件内修改
在Makefile 中修改这两句：
$(Q)protoc --proto_path=$(PROTO_SRC_DIR) --cpp_out=$(PROTO_BUILD_DIR) $<
$(Q)protoc --proto_path=$(PROTO_SRC_DIR) --python_out=$(PY_PROTO_BUILD_DIR) $<
为
$(Q)/usr/bin/protoc --proto_path=$(PROTO_SRC_DIR) --cpp_out=$(PROTO_BUILD_DIR) $<
$(Q)/usr/bin/protoc --proto_path=$(PROTO_SRC_DIR) --python_out=$(PY_PROTO_BUILD_DIR) $<
即把开头的"protoc"补全路径即可 (/usr/bin/protoc即为自己向指定给的版本路径)
注：这种修改不会影响系统默认的protoc版本，只会在caffe编译的时候调用相应的proto版本

若使用`anaconda`中的，则：

由于我的$HOME/anaconda2/bin中有合适的版本，所以我是打算直接用这里的protobuf，通过查资料，了解到sudo的时候，默认的环境变量在/etc/sudoers文件中，一般来说sudo会重置$PATH，而sudoers文件有这样一行

```
1 Defaults    secure_path = /sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin
```

所以 sudo的时候$PATH就是从这里读取，因此我只需添加我的路径就可以了，注意要把路径添加在其他版本路径的前面，位置决定了读取顺序

```
1 #Defaults    secure_path="/usr/local/sbin:/usr/local/bin:\$HOME/anaconda2/bin:/usr/sbin:/usr/bin:/sb
```

注意sudoers文件很重要，修改前记得备份

secure_path是即时生效的，这时候再次运行sudo使用的版本就变成了我需要的3.0.0版本。

安装

```
conda install -c anaconda protobuf=2.6.1
```

数据集

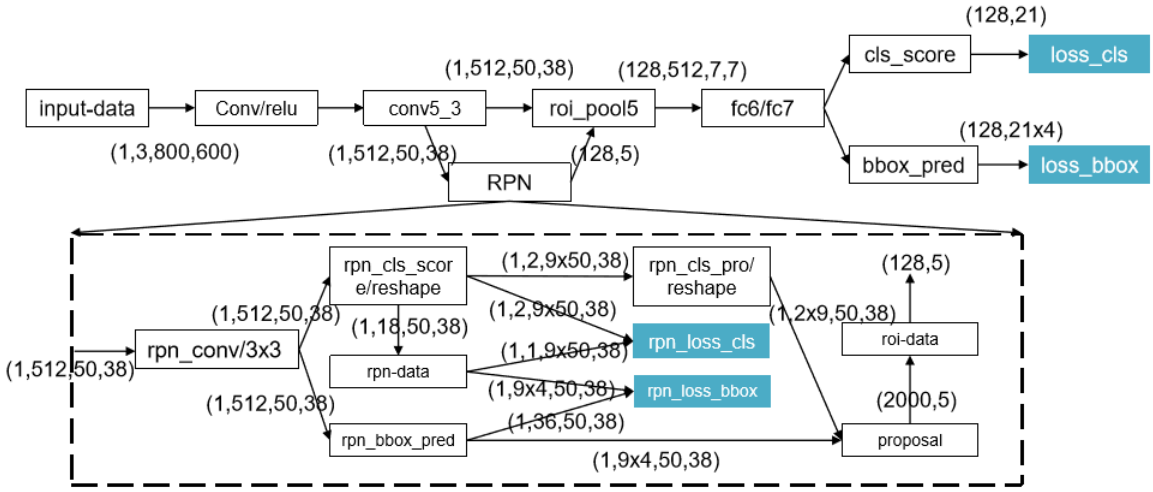
采用`PASCAL VOC`数据集
20类物体检测 + 1类背景
VOCdevkit目录

```
└── VOCdevkit       #根目录
    └── VOC2012     #不同年份的数据集，这里只下载了2012的，还有2007等其它年份的
        ├── Annotations         #存放xml文件，与JPEGImages中的图片一一对应，解释
图片的内容等等
```

```
        ├── ImageSets           #该目录下存放的都是txt文件，txt文件中每一行包含一
个图片的名称，末尾会加上±1表示正负样本
        │   ├── Action
        │   ├── Layout
        │   ├── Main
        │   └── Segmentation
        ├── JPEGImages          #存放源图片
        ├── SegmentationClass   #存放的是图片，分割后的效果，见下文的例子
        └── SegmentationObject  #存放的是图片，分割后的效果，见下文的例子
```

详情参考Pascal Voc数据集详细分析

网络框架



整体网络结构

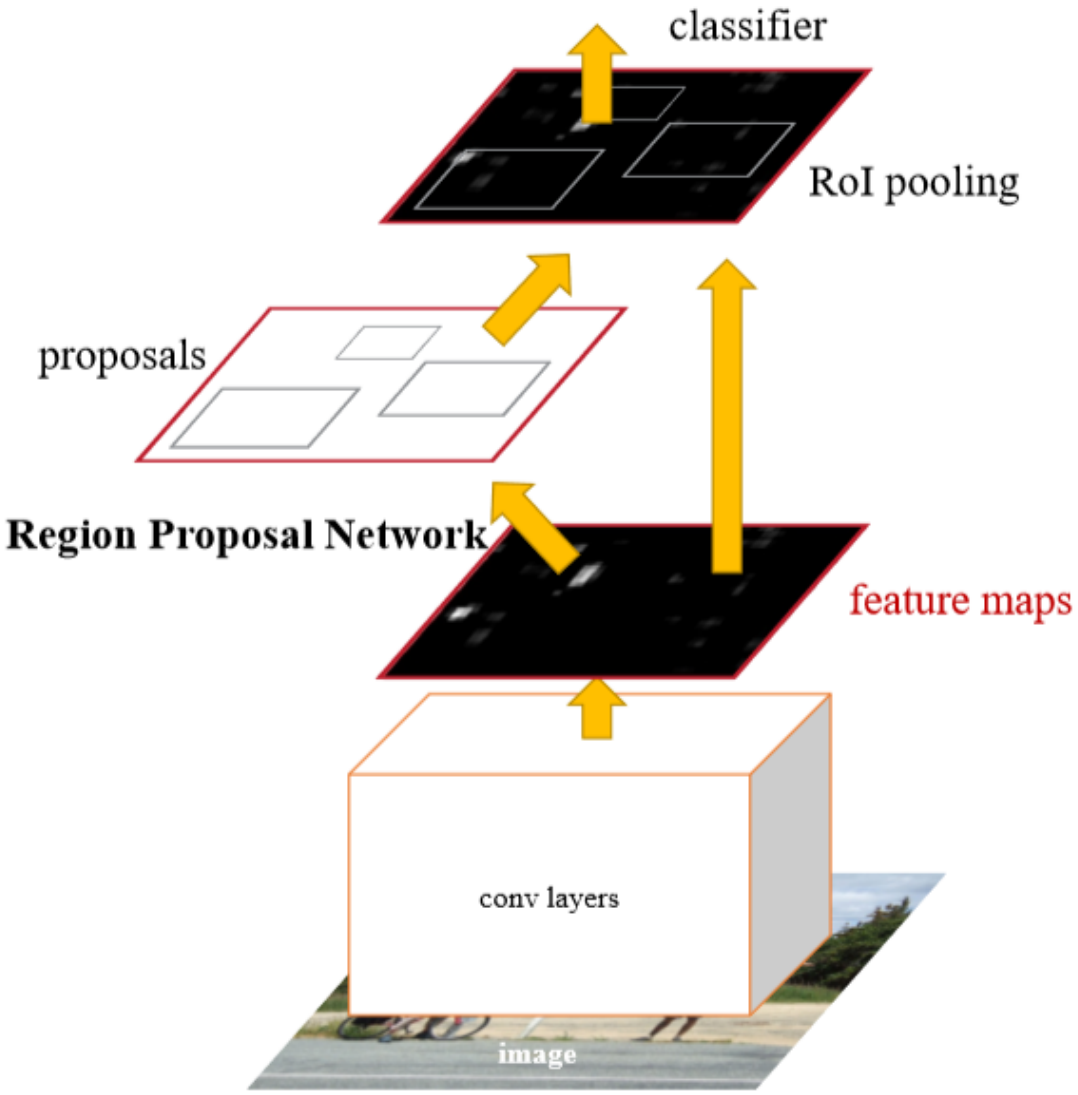
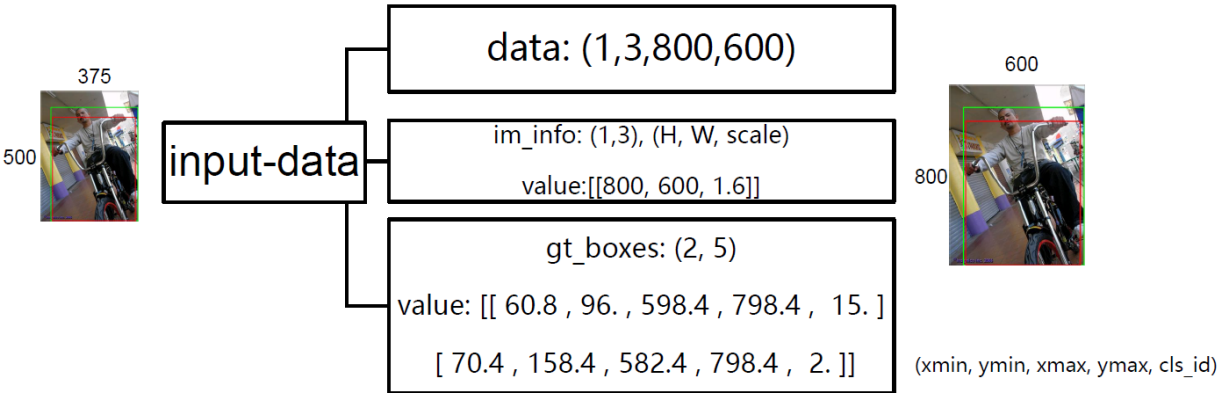


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

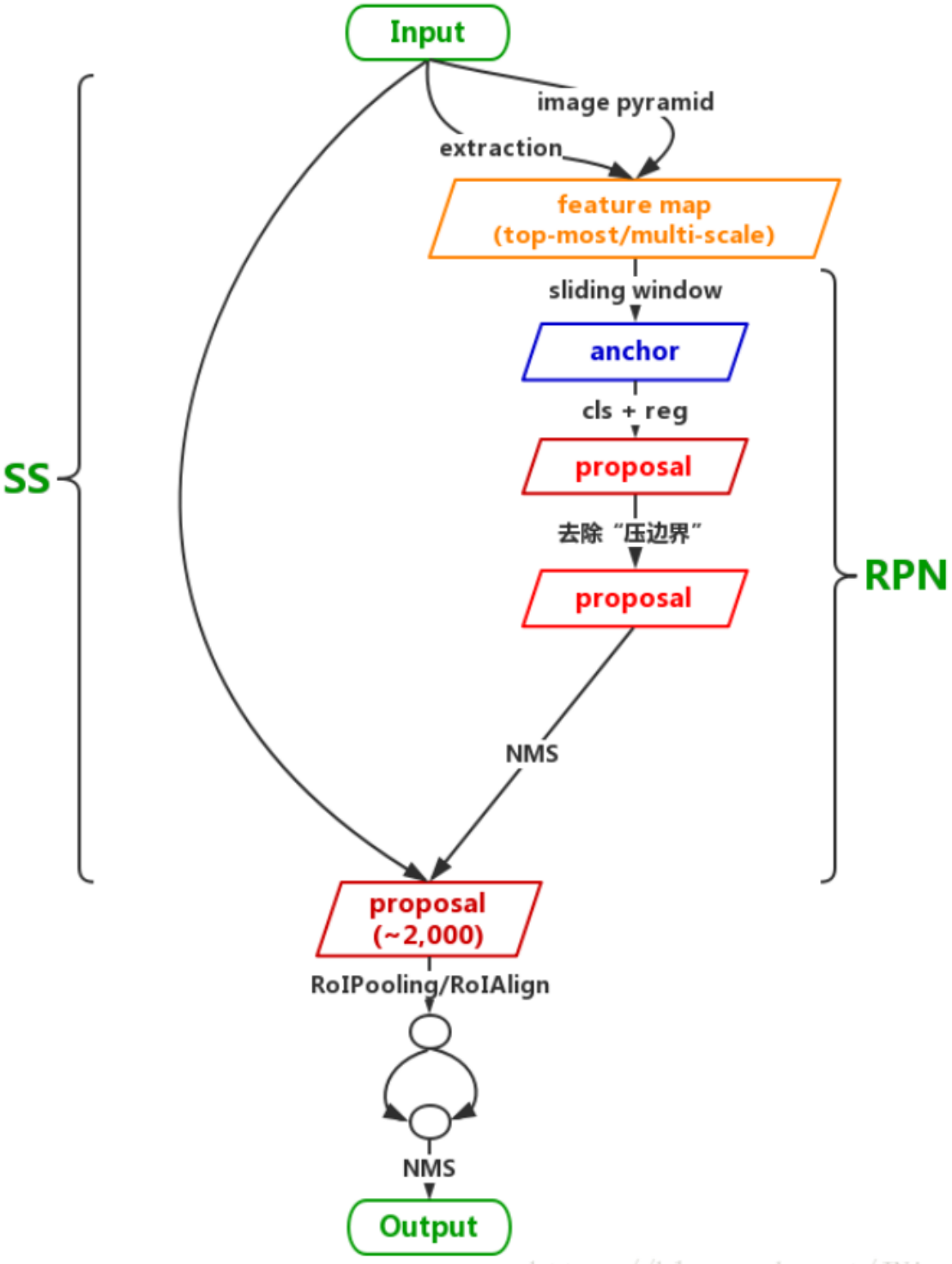
Input-data



得到`resize`后的图片，`boxes`：框的坐标和类别

RPN卷积

二分类检测器



RPN的整体流程框架

Anchor box

– base_size :16

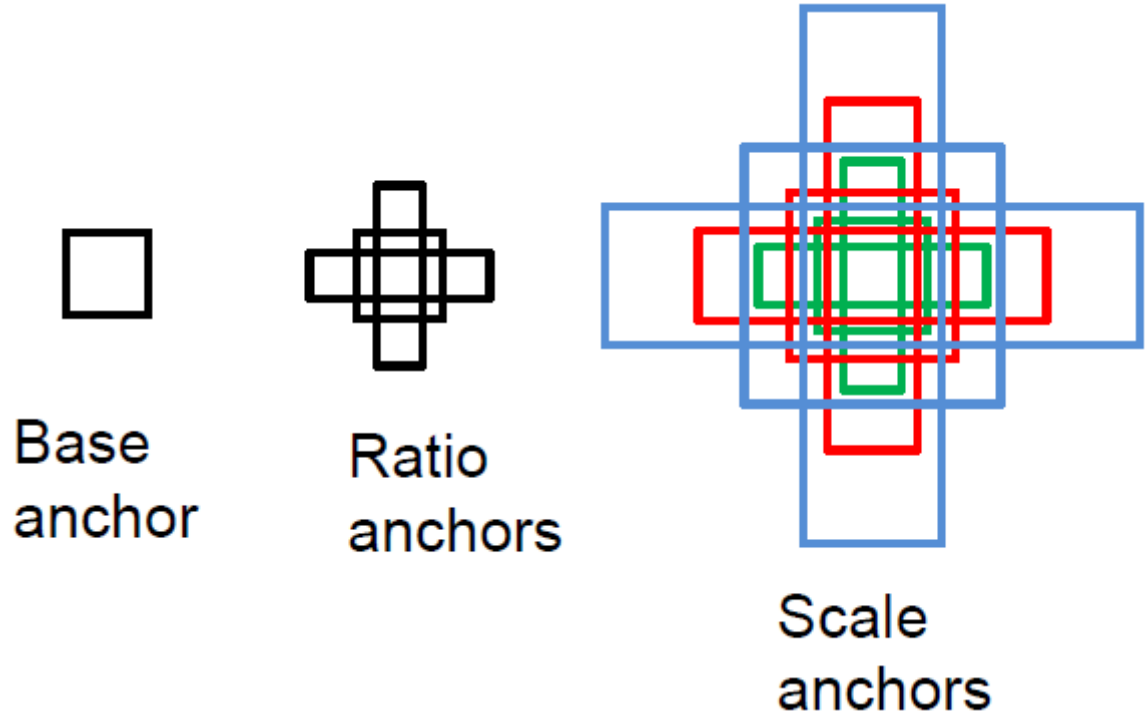
– ratios:[0.5,1, 2]

- 矩形框面积不变
- $Wa=W*ratio, Ha=H/ratio$

– scales: [8, 16, 32]

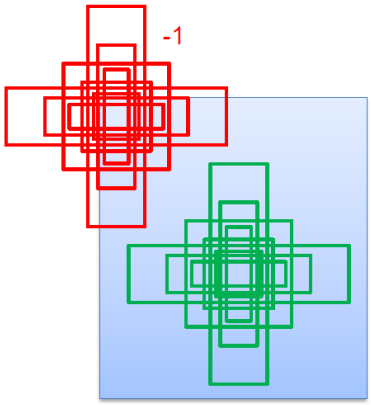
- $Ws=Wa*scale, Hs=Ha*scale$

– feat_stride :16



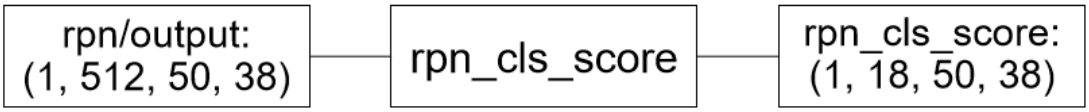
生成256个正例和负例的anchor

- rpn-labels
  - RPN_NEGATIVE_OVERLAP: 0.3
    - 负例(0)：anchor与ground truth的 IoU<0.3
  - RPN_POSITIVE_OVERLAP: 0.7
    - 正例(1)：anchor与ground truth的IoU>0.7
    - 正例(1) : 与ground truth重叠度最高的anchors
  - RPN_BATCHSIZE: 256
    - 正例 + 负例
  - 其他anchors
    - -1: 不考虑



Rpn_cls_score

- num_output: 18
  - 2(bg/fg) * 9(anchors)
- kernel_size: 1 pad: 0 stride: 1



区分背景和前景

Rpn_loss_cls

$$\mathrm{CE}(\mathrm{label}, \mathrm{output}) = -\sum_i \mathrm{label}_i \log(\mathrm{output}_i)$$

Rpn_loss_bbox

$$e = \sum w_{out} * SmoothL1(w_{in} * (b_{pred} - b_{targets}))$$

NMS非极大值抑制：

抑制不是极大值的元素，可以理解为局部最大搜索，除冗余的检测框,保留最好的一个.

ROI_pooling

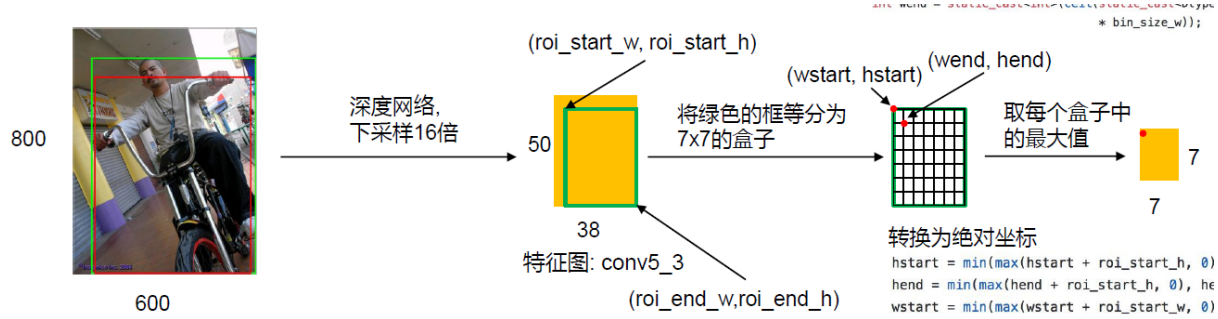
确定上一阶段的每个region proposal是否属于目标一类或者背景。

`ROI pooling`提出的根本原因：
ROI pooling层能实现**training和testing的显著加速**，并提高检测accuracy。该层有两个输入：

- 从具有多个卷积核池化的深度网络中获得的固定大小的`feature maps`；
- 一个表示所有`ROI`的N*5的矩阵，其中N表示`ROI`的数目。第一列表示图像index，其余四列表示其余的左上角和右下角坐标；

ROI pooling具体操作如下：
（1）根据输入image，将**ROI映射到feature map对应位置**；
（2）将映射后的区域划分为**相同大小的sections**（sections数量与输出的维度相同）；
（3）对每个sections进行`max pooling`操作；

这样我们就可以从不同大小的方框得到固定大小的相应 的`feature maps`。值得一提的是，输出的`feature maps`的大小**不取决于ROI和卷积feature maps大小**。ROI pooling 最大的好处就在于极大地提高了处理速度。



Faster-rcnn应用

后续补上

反馈与建议

- 微博：@柏林designer
- 邮箱：wwj123@zju.edu.cn