目标检测（object detection）

通用目标检测旨在从自然图像中定位大量预定义类别的物体实例,是计算机视觉中最基本和最具挑战性的问题之一。近年来，深度学习技术已成为数据中学习特征表示的有力方法，并在通用目标检测领域取得了显著的突破。在深度学习快速发展的时期，本文提供对深度学习技术带来的这一领域最近成就的全面调研。本次调研包括 250 多项关键技术，涵盖了通用目标检测研究的许多方面：前沿的检测框架和基本子问题，包括目标特征表示，候选区域生成，上下文信息建模和训练策略等；评价问题，特别是 benchmark 数据集，评价指标和最先进的方法。（摘自 Deep Learning for Generic Object Detection: A Survey）

而文章中将目标检测分为了两类：特定实例的检测和特定类别的检测。第一种类型的目标是检测特定目标的实例，而第二种类型的目标是检测不同的预定义目标类别的实例（例如人类、汽车、自行车和狗）。从历史上看，在目标检测领域的大部分努力都集中在检测单个类别（如面孔和行人）或少数特定类别。与此相反，在过去的几年中，研究界已经开始朝着建立通用目标探测系统的挑战目标迈进，这些系统的目标探测能力的广度与人类不相上下。

目标检测的主要挑战

什么样的检测器、分类器是一个理想的分类器？文章给出了如下的定义：

理想分类器

- 高准确度
    - 高鲁棒性

        - 每个类别内的差异（不同的颜色、纹理、材质、形状等）
        - 目标实例差异（姿势、形变）
        - 成像条件和无约束环境（照明，视图点，刻度，遮挡，阴影，杂波，模糊，运动，天气状况）
        - 图像噪声（成像噪声，滤波失真，压缩噪声）
    - 高辨别性
        - 组内歧义
        - 过多的真实的目标分类（有结构与无结构的）
- 高效
    - 现实世界中成千上万的目标类别
    - 要求定位和识别目标
    - 大量可能的目标位置
    - 大规模图像/视频数据

高质量检测必须准确地定位和识别图像或视频帧中的物体，这样才能区分真实世界中各种各样的目标类别（例如高度的区别性），以及来自同一类别的目标实例，受限于类内外观的变化，可以被本地化和识别（例如、高鲁棒性)。高效率要求整个检测任务以足够高的帧速率运行，并使用可接受的内存和存储使用。

关于精度的挑战：

作者由呈像原理，陈列了影响精度的因素：内在因素和成像条件。

对于前者,每个目标类别可以有许多不同的目标实例,可能在一个或多个不同的颜色,质地,材料,形状,大小。

对于后者，这些变化是由成像条件的变化和不受约束的环境造成的，这可能会对物体的外观产生巨大的影响。特别地，不同的实例，甚至是相同的实例，都可以被捕获到不同的地方：不同的时间、地点、天气条件、摄像机、背景、光照、视点和观看距离。

目标检测的主要结构

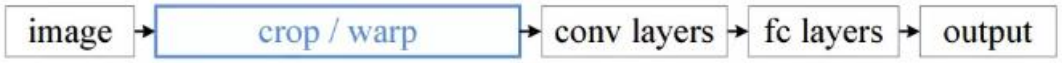
大体上可分为两大类：

1. 两级检测框架，其中包括对区域提案的预处理步骤，使整个检测过程分为两个阶段；
2. 单级检测框架，或区域建议自由框架，是一种不单独检测候选框的单一方法，使整个流水线处于单一阶段。

SPPNet

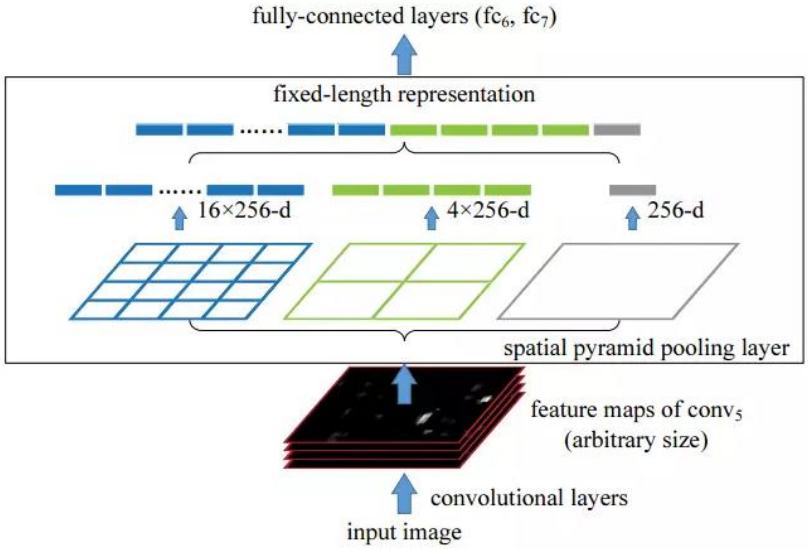
首先论文中说，在测试过程中，CNN 特征提取是 RCNN 检测的主要瓶颈，RCNN 检测管道需要从数千个扭曲区域中提取 CNN 特征来获得图像。在 CNN 网络中，由于全连接层的存在，与全连接层相连的最后一个卷积层的输出必须是固定的，也就是说，CNN 网络的输入图片也必须是固定的。

这里先介绍下传统解决方法，也就是 SPPNet 之前的做法，既然输入需要固定，那就在选取了感兴趣区域之后进行放缩，然后再传入卷积神经网络，流程如下图所示：



而图片裁剪或变形的一个问题是导致图片的信息缺失或变形，影响识别精度，如下图所示：

注意到这些明显的缺点，He 等人将传统的空间金字塔池(SPP)引入到 CNN 架构中。He 等人添加一个 SPP 层在最后一个卷积层和 FC 层之间。具体做法如下：



设上一层卷积层的特征图是 256 层，那么对于每一层都做一次 spatial pyramid pooling，也就是说对每一层用三个尺寸的网络去做 max pooling，这样形成了 16*256、4*256、1*256 维的特征，再将特征拼接成一个维度。同时，卷积层的特征图深度是不受输入图片的长宽影响，这样全连接层的输入就固定下来了。

同时，对卷积层可视化发现：输入图片的某个位置的特征反应在特征图上也是在相同位置。基于这一事实，对某个 ROI 区域的特征提取只需要在特征图上的相应位置提取就可以了。

这样，图片只需要经过一次 CNN，候选区域特征直接从整张图片特征图上提取。

总结

SPP-net 对 R-CNN 最大的改进就是特征提取步骤做了修改，其他模块仍然和 R-CNN 一样。特征提取不再需要每个候选区域都经过 CNN，只需要将整张图片输入到 CNN 就可以了，ROI 特征直接从特征图获取。和 R-CNN 相比，速度提高了百倍。

Fast RCNN

Girshick[64]提出了 Fast RCNN，解决了 RCNN 和 SPnet 的一些缺点，同时提

高了它们的检测速度和质量。开发一个流线化的训练过程来实现端到端检测器训练（当忽略区域建议生成过程时），该过程同时学习软最大分类器和使用多任务损失的特定于类的边界盒回归。

另外，之前 RCNN 的处理流程是先提 proposal，然后 CNN 提取特征，之后用 SVM 分类器，最后再做 bbox regression，而这三个阶段需要独立训练。这里用两个流程图，来表达 SPP-net 与 Fast RCNN 的区别

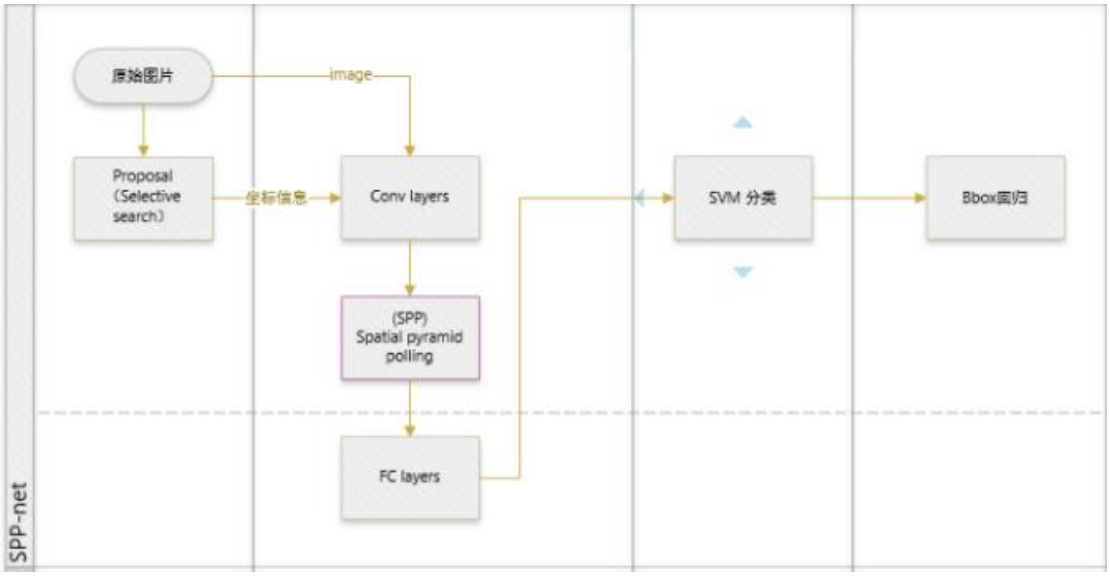


图 SPP-Net 结构

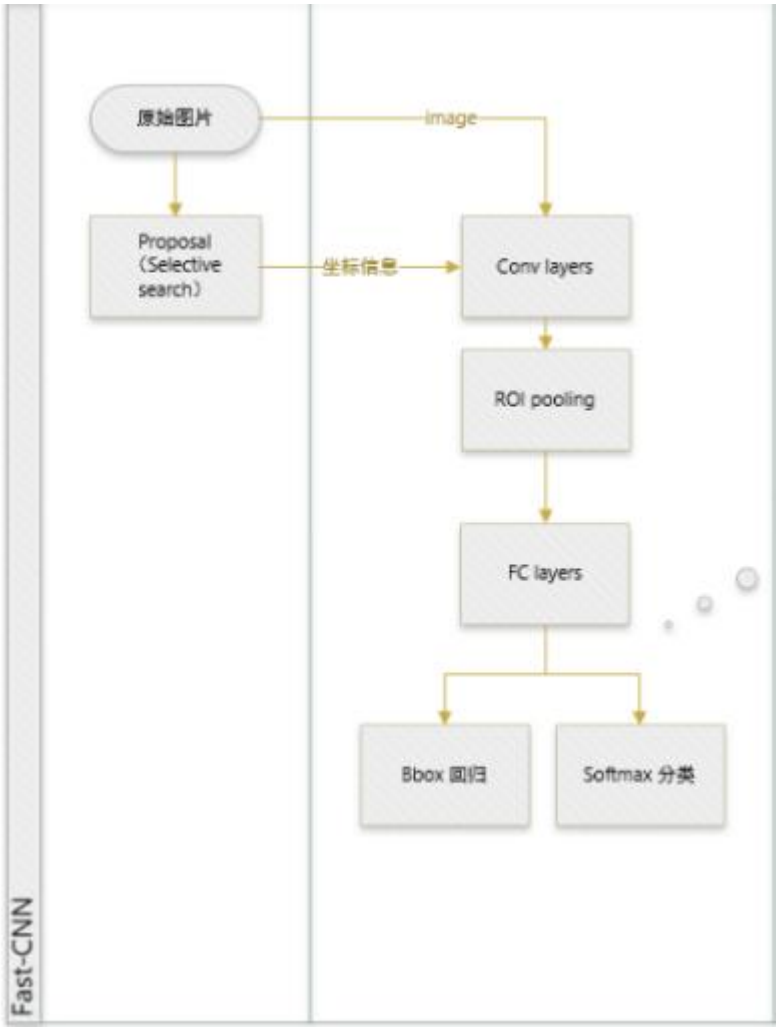


图 Fast-CNN 结构

可以看到，SPPNet 在训练时，是隔离的：提取 proposal，CNN 提取特征，SVM 分类，bbox regression。

这里可以得出 Fast-RCNN 的两大主要贡献点：

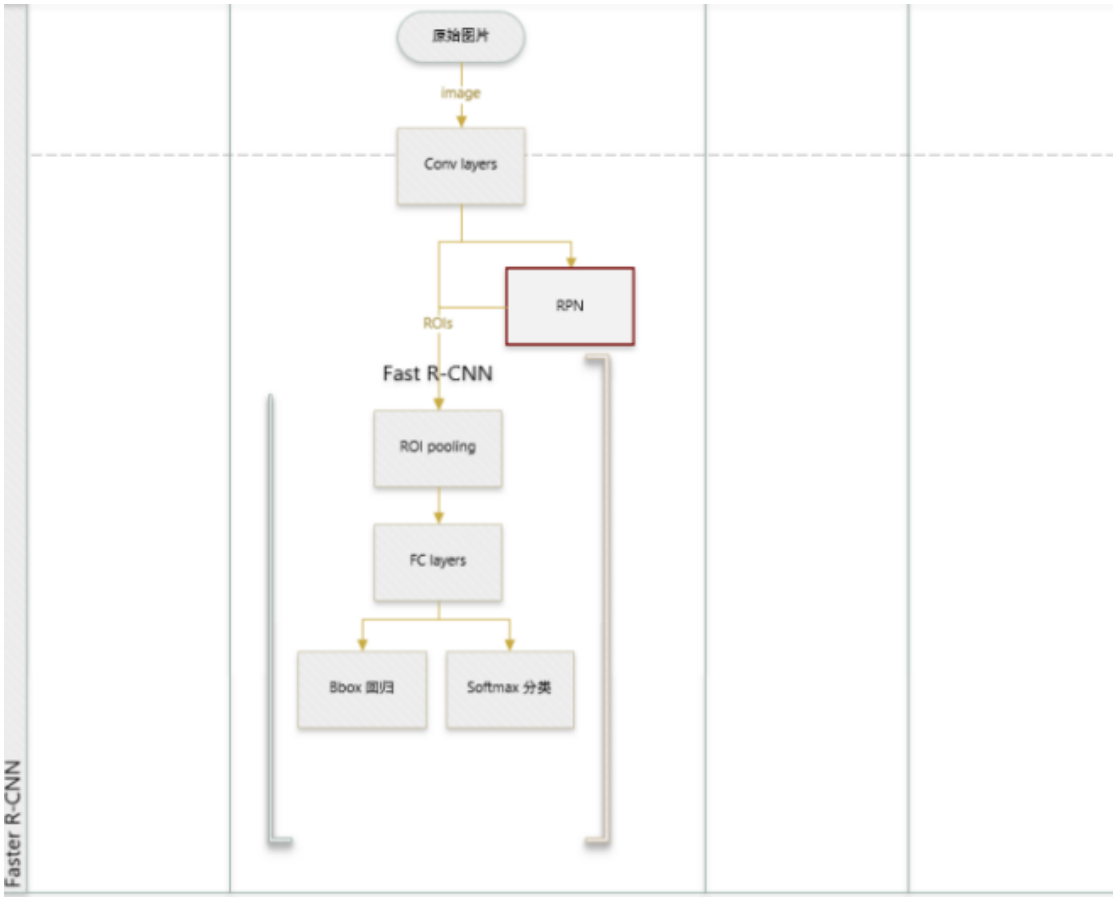
1. 实现大部分 end-to-end（端到端）的训练，把 Bbox 回归与分类放在了一起（SVM 分类，bbox 回归 联合起来在 CNN 阶段训练）。那么这种网络怎么训练呢？作者将多个任务的损失函数写在了一起，这样解决 SPPNet 的各个训练部分的隔离。

2. ROI Pooling

ROI 就是特殊的 SPP，只不过它没有考虑多个空间尺度，只用单个尺度。论文通过实验得到的结论是多尺度学习性能能提高，不过计算量成倍的增加，故单尺度训练的效果更好。

Faster R-CNN

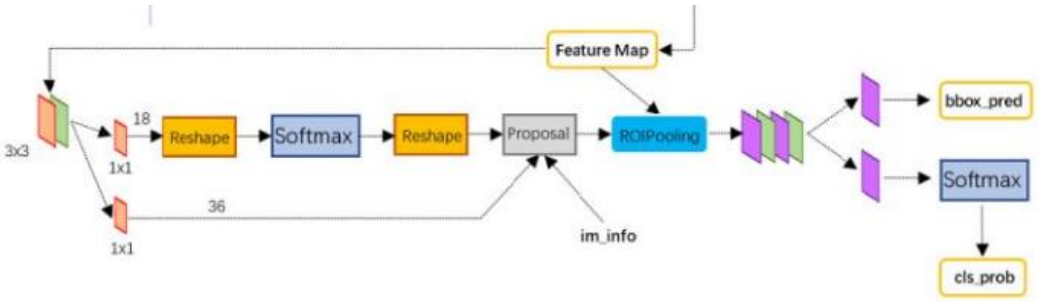
前面提到 Fast R-CNN 实现了大部分端到端的训练，唯一隔离的是 proposal 阶段即提议阶段，往前常规方法在这阶段是执行诸如 selective search 之类的算法。Faster 的作者使用区域生成网络 RPN 层,来把 proposal 阶段也用 CNN 实现。所以，可以简单的认为 Faster R-CNN 是“区域生成网络+fast RCNN”



区域生成网络 （ Region Proposal Networks ）

首先思考一个问题：怎样的区域是一个好的区域？ 我们需要的区域是能

较好包裹住物体，即区域又不很大又不很小。所以 Faster 的做法是：在每一个对应的原图设计不同的固定尺度窗口（bbox），根据该窗口与 ground truth 的 IOU 给它正负标签，让它学习里面是否有 object，这样就训练一个网络（Region Proposal Network）。

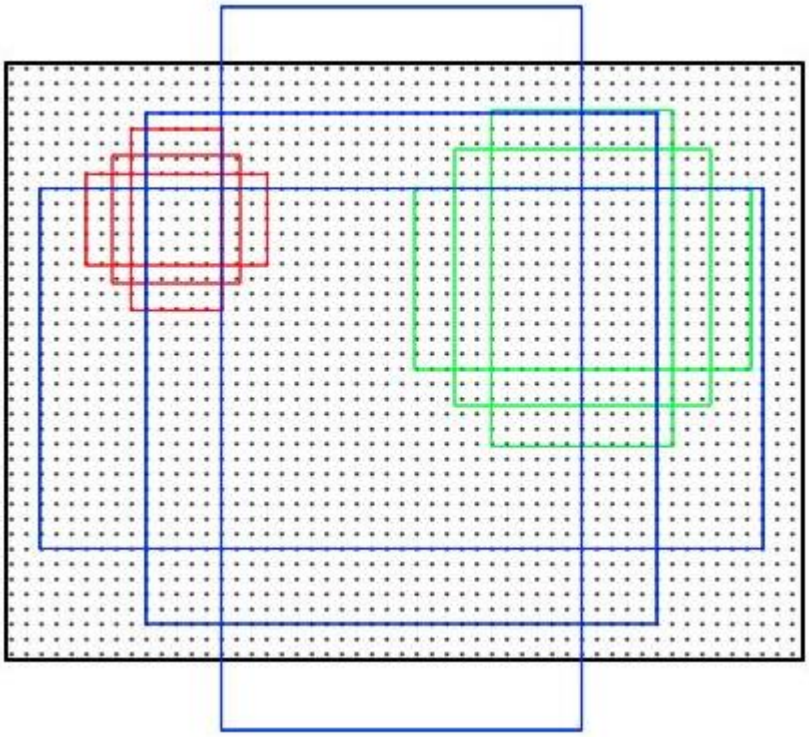


RPN 网络结构

可以看到 RPN 网络实际分为 2 条线，上面一条通过 softmax 分类 anchors 获得 positive 和 negative 分类，下面一条用于计算对于 anchors 的 bounding box regression 偏移量，以获得精确的 proposal。而最后的 Proposal 层则负责综合 positive anchors 和对应 bounding box regression 偏移量在原始 Feature Map 上获取 proposals，同时剔除太小和超出边界的 proposals。其实整个网络到了 Proposal Layer 这里，就完成了相当于目标定位的功能。下面开始具体讲解这个过程中的一些细节。

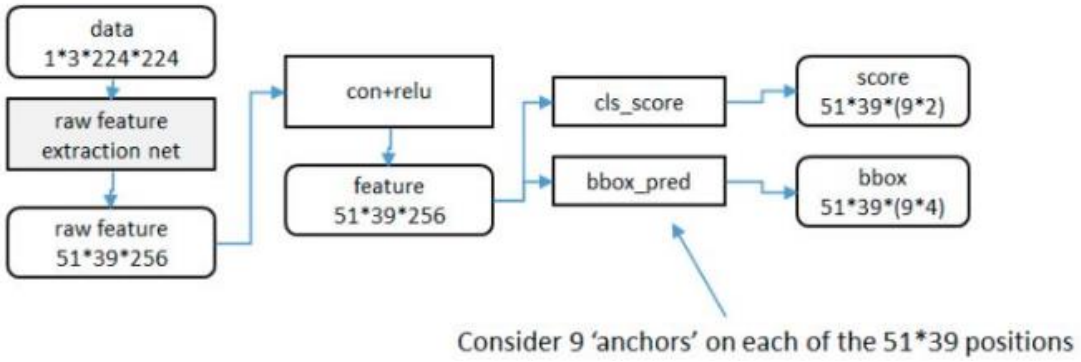
对于 RPN 有个很重要的概念：anchor。anchor 本质是 SPP(spatial pyramid pooling)思想的逆向。而 SPP 本身是做什么的呢，就是将不同尺寸的输入 resize 成为相同尺寸的输出。所以 SPP 的逆向就是，将相同尺寸的输出，倒推得到不同尺寸的输入。

比如如下图所示：我们如果使用 3 个面积尺寸、3 个长宽尺寸，通过变换可以得到 9 个图。



为什么要使用 anchor 呢?

下面是整个 faster RCNN 结构的示意图:



作者在第三列开始，使用了 anchors，此处传来的 feature map 是 51*39*256（256 是层数），在这个特征参数的基础上，通过一个 3x3 的滑动窗口进行滑动，stride=1，padding=2，这样一来，滑动得到的就是 51x39 个 3x3 的窗口。我认为这样做的根本目的是因为，逆向 SPP 过程，一个中心点可以对应很多个在原图上的 feature map，anchor 的思想就是尽可能找全可能符合要求的 proposals.

对于每个 3x3 的窗口，作者就计算这个滑动窗口的中心点所对应的原始图片的中心点。然后作者假定，这个 3x3 窗口，是从原始图片上通过 SPP 池化得到的，而这个池化的区域的面积以及比例，就是一个个的 anchor。换句话说，**对于每个 3x3 窗口，作者假定它来自 9 种不同原始区域的池化，但是这些池化在原始图片中的中心点，都完全一样。这个中心点，就是 3x3 窗口中心点所对应的原始图片中的中心点。**如此一来，在每个窗口位置，我们都可以根据 9 个不同长宽比例、不同面积的 anchor，逆向推导出它所对应的原始图片中的一个区域，这个区域的尺寸以及坐标，都是已知的。而这个区域，就是我们想要的 proposal。所以我们通过滑动窗口和 anchor，成功得到了 51x39x9 个原始图片的 proposal。接下来，每个 proposal 我们只输出 6 个参数：每个 proposal 和 ground truth 进行比较得到的前景概率和背景概率(2 个参数）（对应图上的 cls_score）；由于每个 proposal 和 ground truth 位置及尺寸上的差异，从 proposal 通过平移放缩得到 ground truth 需要的 4 个平移放缩参数（对应图上的 bbox_pred）

就此得到了 51*39*9 = 18k 个 anchor box

当然并不是所有的 anchor box 都符合要求：首先应该保证每个人工标定的 ground true 至少对应着一个正样本 anchor，然后可以使用非极大值抑制，通过阈值来继续增加正样本 anchor，而如果其与任意一个标定的重叠比例都小于 0.3，记为负样本。

此外，在训练过程中，还有值得注意的一个点，我们经过上述步骤筛选正负样本后，一般情况，负样本比例会远多于正样本。文中提到，如果对每幅图的所有 anchor 都去优化 loss function，那么最终会因为负样本过多导致最终得到的

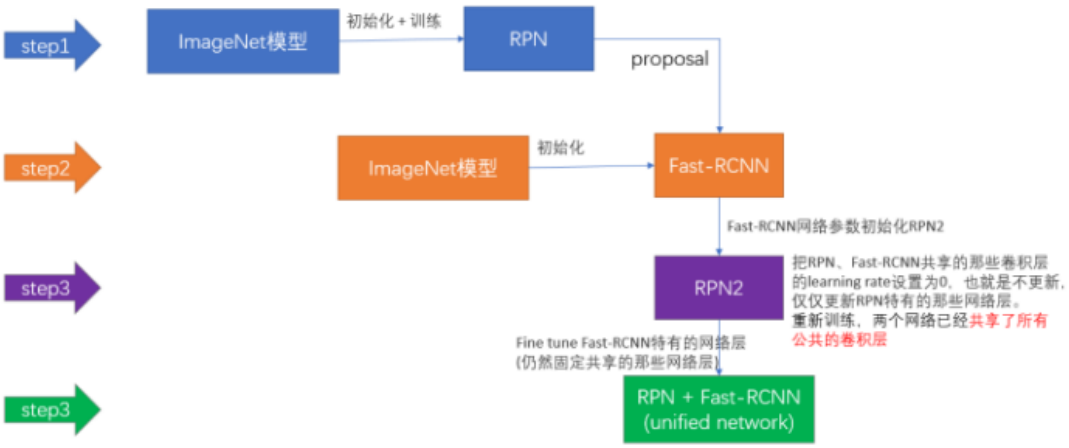
模型对正样本预测准确率很低。

因此 在每幅图像中随机采样 256 个 anchors 去参与计算一次 mini-batch 的损失。正负比例 1:1 (如果正样本少于 128 则补充采样负样本)

关于 Faster-RCNN 的训练：

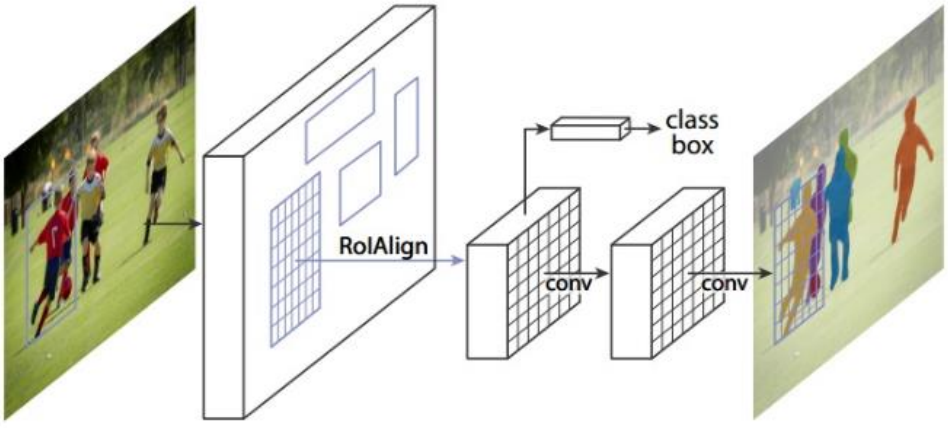
作者发布的源代码里却用了另外一种叫做 4-Step Alternating Training 的方法，这里参考知乎：https://zhuanlan.zhihu.com/p/24916624?refer=xiaoleimlnote 对该方法的描述：

1. 第一步：用 ImageNet 模型初始化，独立训练一个 RPN 网络；
2. 第二步：仍然用 ImageNet 模型初始化，但是使用上一步 RPN 网络产生的 proposal 作为输入，训练一个 Fast-RCNN 网络，至此，两个网络每一层的参数完全不共享；
3. 第三步：使用第二步的 Fast-RCNN 网络参数初始化一个新的 RPN 网络，但是把 RPN、Fast-RCNN 共享的那些卷积层的 learning rate 设置为 0，也就是不更新，仅仅更新 RPN 特有的那些网络层，重新训练，此时，两个网络已经共享了所有公共的卷积层；
4. 第四步：仍然固定共享的那些网络层，把 Fast-RCNN 特有的网络层也加入进来，形成一个 unified network，继续训练，fine tune Fast-RCNN 特有的网络层，此时，该网络已经实现我们设想的目标，即网络内部预测 proposal 并实现检测的功能。



Mask RCNN

Mask RCNN 是基于 Faster RCNN 改进而来，其核心改进有两大点：一是用 ROI Align 代替了 ROI Pooling，二是在 ROI Align 之后添加卷积层，进行 mask 预测的任务。所以下面着重对这两个点进行说明。



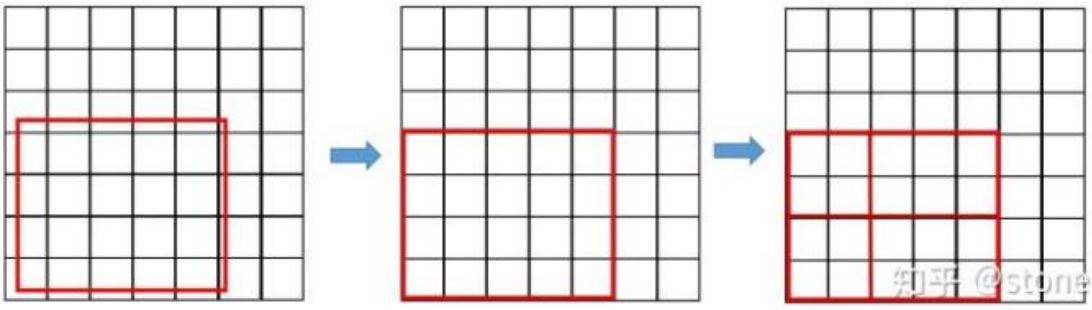
ROI Align

Faster R-CNN 存在的问题是：特征图与原始图像是不对准的（mis-alignment），所以会影响检测精度。而 Mask R-CNN 提出了 RoIAlign 的方法来取代 ROI pooling，RoIAlign 可以保留大致的空间位置。

什么叫特征图与原始图像不对准呢？在 Faster RCNN 中，有两次整数化的过程：

1. region proposal 的 xywh 通常是小数，但是为了方便操作会把它整数化。
2. 将整数化后的边界区域平均分割成 k x k 个单元，对每一个单元的边界进行整数化（比如对于 7*8 的边界区域，要划分为 4 份，那么 7/2 显然是不能整除的，所以需要整数化）。

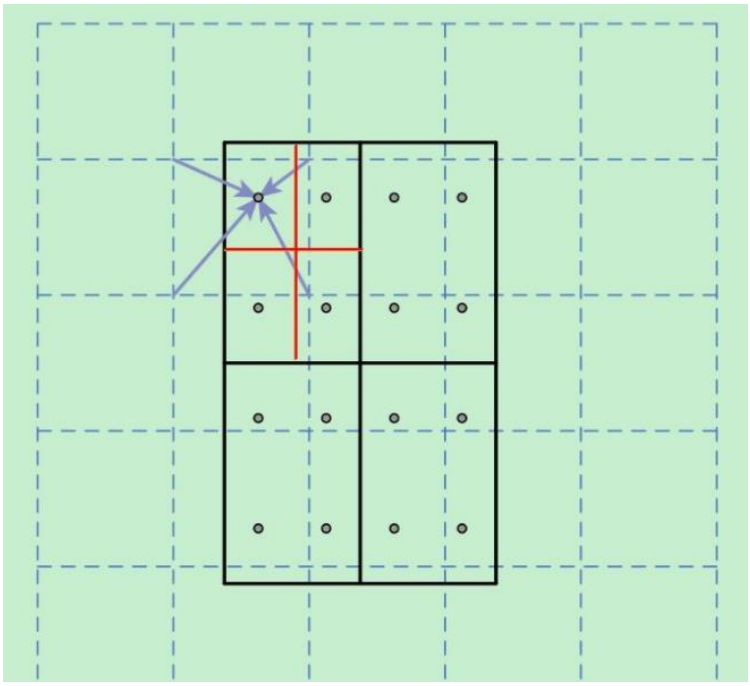
两次整数化的过程如下图所示：



事实上，经过上述两次整数化，此时的候选框已经和最开始回归出来的位置有一定的偏差，这个偏差会影响检测或者分割的准确度。在论文里，作者把它总结为“不匹配问题”（misalignment）。为了解决这个问题，ROI Align 方法取消整数化操作，保留了小数，在实际操作中，ROI Align 并不是简单地补充出候选区域边界上的坐标点，然后进行池化，而是重新进行设计。

如下图所示，虚线部分表示 feature map，实线表示 ROI，这里将 ROI 切分成

2x2 的单元格。如果采样点数是 4，那我们首先将每个单元格子均分成四个小方格（如红色线所示），每个小方格中心就是采样点。这些采样点的坐标通常是浮点数，所以需要对采样点像素进行双线性插值（如四个箭头所示），就可以得到该像素点的值了。然后对每个单元格内的四个采样点进行 maxpooling，就可以得到最终的 ROIAlign 的结果。



对于 mask 层的理解现在还没很好，网上相关博客也很少，准备后面进行论文阅读的方式学习。

Mask R-CNN 实验

根据官方文档，基本安装步骤如下：

一、安装

Mask R-CNN 是基于 Python 3, Keras, and TensorFlow 的

1. 使用 git 将代码 clone 在本地

git clone https://github.com/matterport/Mask_RCNN.git

2. 进入到 Mask R-CNN 的目录下安装依赖库

pip3 install -r requirements.txt

3. 下载预训练的 COCO 权重（mask_rcnn_coco.h5）文件：

https://github.com/matterport/Mask_RCNN/releases

完成上面的步骤就可以测试 Mask R-CNN 的测试代码了。

二、测试

最简单的测试程序

因为还没有涉及到对源代码的里面，这里仅仅对测试用例代码进行展示。

1. 首先是对 config 参数的初始化：

```
class InferenceConfig(coco.CocoConfig):
    # Set batch size to 1 since we'll be running inference on
    # one image at a time. Batch size = GPU_COUNT * IMAGES_PER_GPU
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1
config = InferenceConfig()
config.display()
```

2. 生成模型并加载训练权重

```
# Create model object in inference mode.
model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR, config=config)

# Load weights trained on MS-COCO
model.load_weights(COCO_MODEL_PATH, by_name=True)
```

3. 导入种类名称

```
# COCO Class names
# Index of the class in the list is its ID. For example, to get ID of
# the teddy bear class, use: class_names.index('teddy bear')
class_names = ['BG', 'person', 'bicycle', 'car', 'motorcycle', 'airplane',
               'bus', 'train', 'truck', 'boat', 'traffic light',
               'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird',
               'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear',
               'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie',
               'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
               'kite', 'baseball bat', 'baseball glove', 'skateboard',
               'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup',
               'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
               'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza',
               'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed',
               'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote',
               'keyboard', 'cell phone', 'microwave', 'oven', 'toaster',
               'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors',
               'teddy bear', 'hair drier', 'toothbrush']
```

4. 随机选取图片测试

```
# Load a random image from the images folder
file_names = next(os.walk(IMAGE_DIR))[2]
image = skimage.io.imread(os.path.join(IMAGE_DIR, random.choice(file_names)))

# Run detection
results = model.detect([image], verbose=1)

# Visualize results
r = results[0]
visualize.display_instances(image, r['rois'], r['masks'], r['class_ids'],
                            class_names, r['scores'])

Processing 1 images
image                    shape: (476, 640, 3)         min:    0.00000  max:  255.00000
molded_images            shape: (1, 1024, 1024, 3)    min: -123.70000  max:  120.30000
image_metas              shape: (1, 89)               min:    0.00000  max: 1024.00000
```

最终结果如下所示:

三、 训练自己的模型

1. 使用标注软件 VIA（VGG Image Annotator）标注需要训练的数据
2. 然后仿照 Mask_RCNN/samples/balloon/balloon.py 修改代码

（1）修改 Config 类中的

```
NAME="custom"
```

（2）修改 Dataset 类中的 load_balloon 函数，修改成自己的函数名，然后加入自己的实例标签

```
self.add_class("custom", 1, "custom")
```

修改

```
self.add_image(
                "custom",
                image_id=a['filename'],  # use file name as a unique image id
                path=image_path,
                width=width, height=height,
                polygons=polygons)
```

（3）修改 Dataset 类中的 load_mask 函数中

```
if image_info["source"] != "custom":
```

（4）修改 Dataset 类中的 image_reference 函数中

```
if info["source"] == "balloon":
```