

# 目标检测|SSD 原理与实现

## 前言

目标检测近年来已经取得了很重要的进展，主流的算法主要分为两个类型（参考 RefineDet）：

（1）two-stage 方法，如 R-CNN 系算法，其主要思路是先通过启发式方法（selective search）或者 CNN 网络（RPN）产生一系列稀疏的候选框，然后对这些候选框进行分类与回归，two-stage 方法的优势是准确度高；（2）one-stage 方法，如 Yolo 和 SSD，其主要思路是均匀地在图片的不同位置进行密集抽样，抽样时可以采用不同尺度和长宽比，然后利用 CNN 提取特征后直接进行分类与回归，整个过程只需要一步，所以其优势是速度快，但是均匀的密集采样的一个重要缺点是训练比较困难，这主要是因为正样本与负样本（背景）极其不均衡（参见 Focal Loss），导致模型准确度稍低。不同算法的性能如图 1 所示，可以看到两类方法在准确度和速度上的差异。

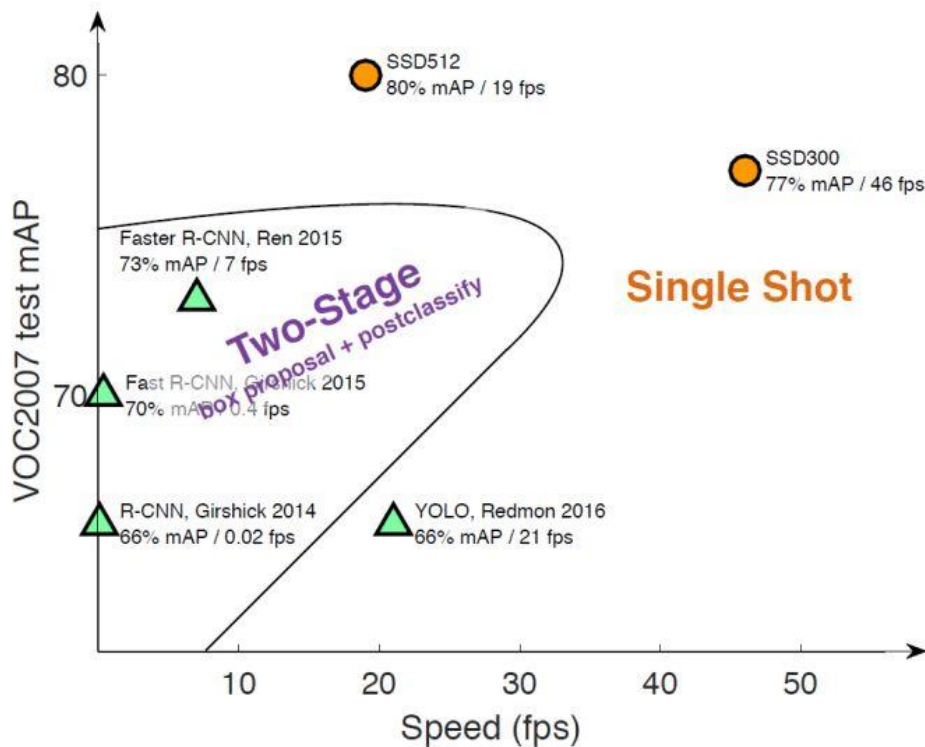


图 1 不同检测算法的性能对比

本文讲解的是 SSD 算法，其英文全名是 Single Shot MultiBox Detector，名字取得不错，Single shot 指明了 SSD 算法属于 one-stage 方法，MultiBox 指明了 SSD 是多框预测。在上一篇文章中我们已经讲了 Yolo 算法，从图 1 也可以看到，SSD 算法在准确度和速度（除了 SSD512）上都比 Yolo 要好很多。图 2 给出了不同算法的基本框架图，对于 Faster R-CNN，其先通过 CNN 得到候选框，然后再进行分类与回归，而 Yolo 与 SSD 可以一步到位完成检测。相比 Yolo，SSD 采用 CNN 来直接进行检测，而不是像 Yolo 那样在全连接层之后做检测。其实采用卷积直接做检测只是 SSD 相比 Yolo 的其中一个不同点，另外还有两个重要的改变，一是 SSD 提取了不同尺度的特征图来做检测，大尺度特征图（较靠前的特征图）可以用来检测小物体，

而小尺度特征图（较靠后的特征图）用来检测大物体；二是 SSD 采用了不同尺度和长宽比的先验框（Prior boxes, Default boxes, 在 Faster R-CNN 中叫做锚，Anchors）。Yolo 算法缺点是难以检测小目标，而且定位不准，但是这几点重要改进使得 SSD 在一定程度上克服这些缺点。下面我们详细讲解 SSD 算法的原理，并最后给出如何用 TensorFlow 实现 SSD 算法。

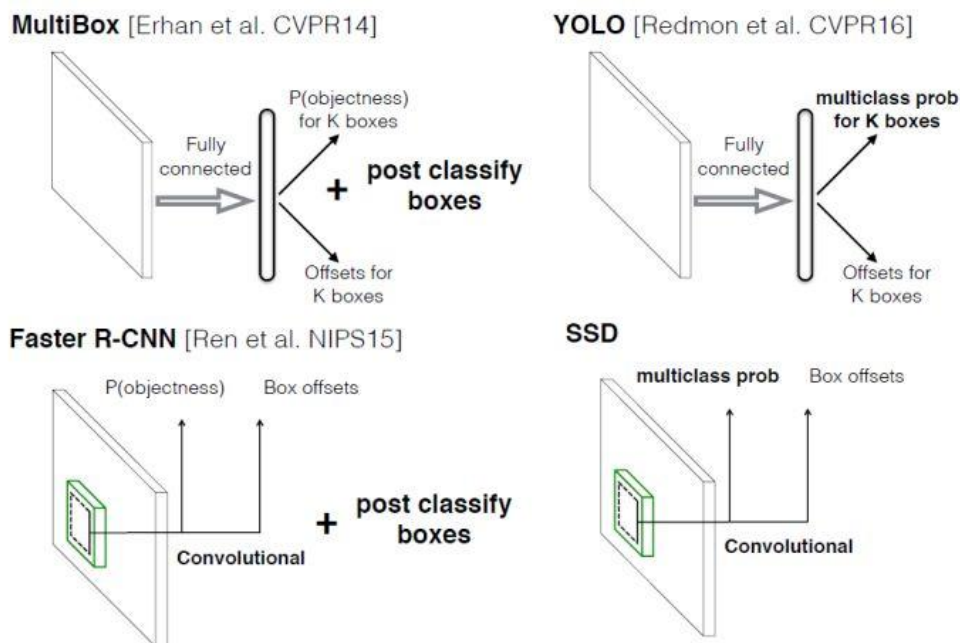


图 2 不同算法的基本框架图

## 设计理念

SSD 和 Yolo 一样都是采用一个 CNN 网络来进行检测，但是却采用了多尺度的特征图，其基本架构如图 3 所示。下面将 SSD 核心设计理念总结为以下三点：

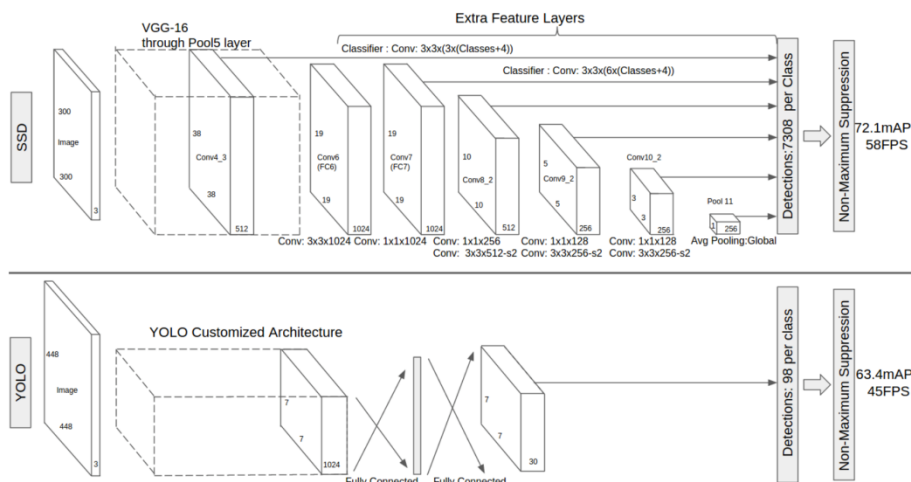


Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a  $300 \times 300$  input size significantly outperforms its  $448 \times 448$  YOLO counterpart in accuracy on VOC2007 *test* while also improving the run-time speed, albeit YOLO customized network is faster than VGG16.

图 3 SSD 基本框架

## (1) 采用多尺度特征图用于检测

所谓多尺度采用大小不同的特征图，CNN 网络一般前面的特征图比较大，后面会逐渐采用  $\text{stride}=2$  的卷积或者 pool 来降低特征图大小，这正如图 3 所示，一个比较大的特征图和一个比较小的特征图，它们都用来做检测。这样做的好处是比较大的特征图用来检测相对较小的目标，而小的特征图负责检测大目标，如图 4 所示， $8 \times 8$  的特征图可以划分更多的单元，但是其每个单元的先验框尺度比较小。

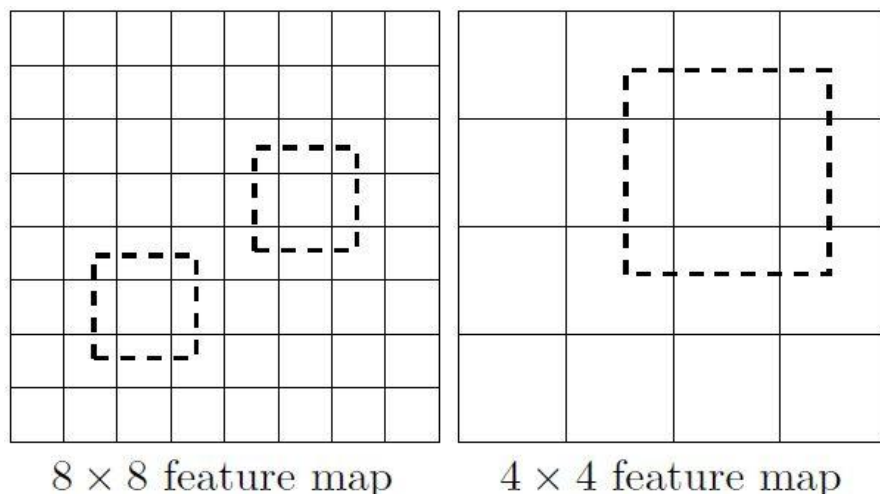


图 4 不同尺度的特征图

## (2) 采用卷积进行检测

与 Yolo 最后采用全连接层不同，SSD 直接采用卷积对不同的特征图来进行提取检测结果。对于形状为  $m \times n \times p$  的特征图，只需要采用  $3 \times 3 \times p$  这样比较小的卷积核得到检测值。

## (3) 设置先验框

在 Yolo 中，每个单元预测多个边界框，但是其都是相对这个单元本身（正方块），但是真实目标的形状是多变的，Yolo 需要在训练过程中自适应目标的形状。而 SSD 借鉴了 Faster R-CNN 中 anchor 的理念，每个单元设置尺度或者长宽比不同的先验框，预测的边界框（bounding boxes）是以这些先验框为基准的，在一定程度上减少训练难度。一般情况下，每个单元会设置多个先验框，其尺度和长宽比存在差异，如图 5 所示，可以看到每个单元使用了 4 个不同的先验框，图片中猫和狗分别采用最适合它们形状的先验框来进行训练，后面会详细讲解训练过程中的先验框匹配原则。

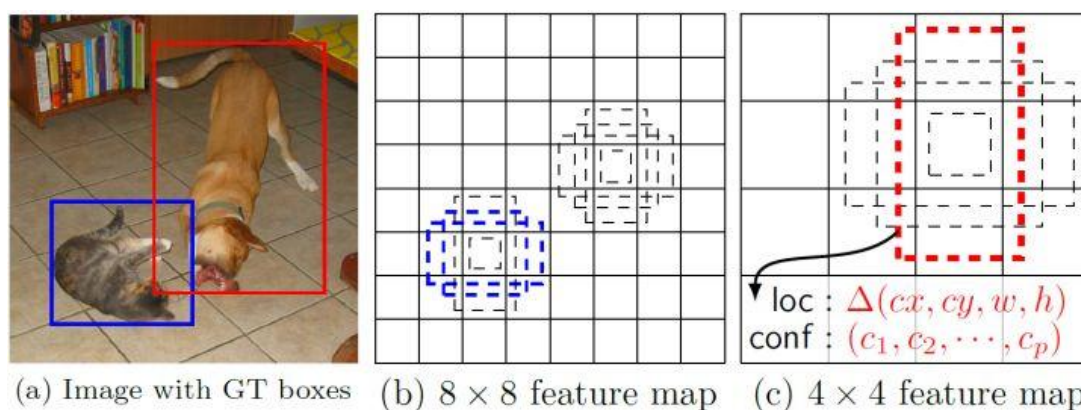


图 5 SSD 的先验框

SSD 的检测值也与 Yolo 不太一样。对于每个单元的第一个先验框，其都输出一套独立的检测值，对应一个边界框，主要分为两个部分。第一部分是各个类别的置信度或者评分，值得注意的是 SSD 将背景也当做了一个特殊的类别，如果检测目标共有  $c$  个类别，SSD 其实需要预测  $c+1$  个置信度值，其中第一个置信度指的是不含目标或者属于背景的分类。后面当我们说  $c$  个类别置信度时，请记住里面包含背景那个特殊的类别，即真实的检测类别只有  $c-1$  个。在预测过程中，置信度最高的那个类别就是边界框所属的类别，特别地，当第一个置信度值最高时，表示边界框中并不包含目标。第二部分就是边界框的 location，包含 4 个值  $(cx, cy, w, h)$ ，分别表示边界框的中心坐标以及宽高。但是真实预测值其实只是边界框相对于先验框的转换值 (paper 里面说是 offset，但是觉得 transformation 更合适，参见 R-CNN)。先验框位置用  $d = (d^{cx}, d^{cy}, d^w, d^h)$  表示，其对应边界框用  $b = (b^{cx}, b^{cy}, b^w, b^h)$  表示，那么边界框的预测值  $l$  其实是  $b$  相对于  $d$  的转换值：

$$l^{cx} = (b^{cx} - d^{cx}) / d^w, \quad l^{cy} = (b^{cy} - d^{cy}) / d^h$$

$$l^w = \log(b^w / d^w), \quad l^h = \log(b^h / d^h)$$

习惯上，我们称上面这个过程为边界框的编码 (encode)，预测时，你需要反向这个过程，即进行解码 (decode)，从预测值  $l$  中得到边界框的真实位置  $b$ ：

$$b^{cx} = d^w l^{cx} + d^{cx}, \quad b^{cy} = d^h l^{cy} + d^{cy}$$

$$b^w = d^w \exp(l^w), \quad b^h = d^h \exp(l^h)$$

然而，在 SSD 的 Caffe 源码实现中还有 trick，那就是设置 variance 超参数来调整检测值，通过 bool 参数 variance\_encoded\_in\_target 来控制两种模式，当其为 True 时，表示 variance 被包含在预测值中，就是上面那种情况。但是如果是 False (大部分采用这种方式，训练更容易?)，就需要手动设置超参数 variance，用来对  $l$  的 4 个值进行放缩，此时边界框需要这样解码：

$$b^{cx} = d^w (\text{variance}[0] * l^{cx}) + d^{cx}, \quad b^{cy} = d^h (\text{variance}[1] * l^{cy}) + d^{cy}$$

$$b^w = d^w \exp(\text{variance}[2] * l^w), \quad b^h = d^h \exp(\text{variance}[3] * l^h)$$

综上所述，对于一个大小  $m \times n$  的特征图，共有  $mn$  个单元，每个单元设置的先验框数目记为  $k$ ，那么每个单元共需要  $(c+4)k$  个预测值，所有的单元共需要  $(c+4)kmn$  个预测值，由于 SSD 采用卷积做检测，所以就需要  $(c+4)k$  个卷积核完成这个特征图的检测过程。

## 网络结构

SSD 采用 VGG16 作为基础模型，然后在 VGG16 的基础上新增了卷积层来获得更多的特征图以用于检测。SSD 的网络结构如图 5 所示。上面是 SSD 模型，下面是 Yolo 模型，可以明显看到 SSD 利用多尺度的特征图做检测。模型的输入图片大小是  $300 \times 300$  (还可以是  $512 \times 512$ ，其与前者网络结构没有差别，只是最后新增一个卷积层，本文不再讨论)。

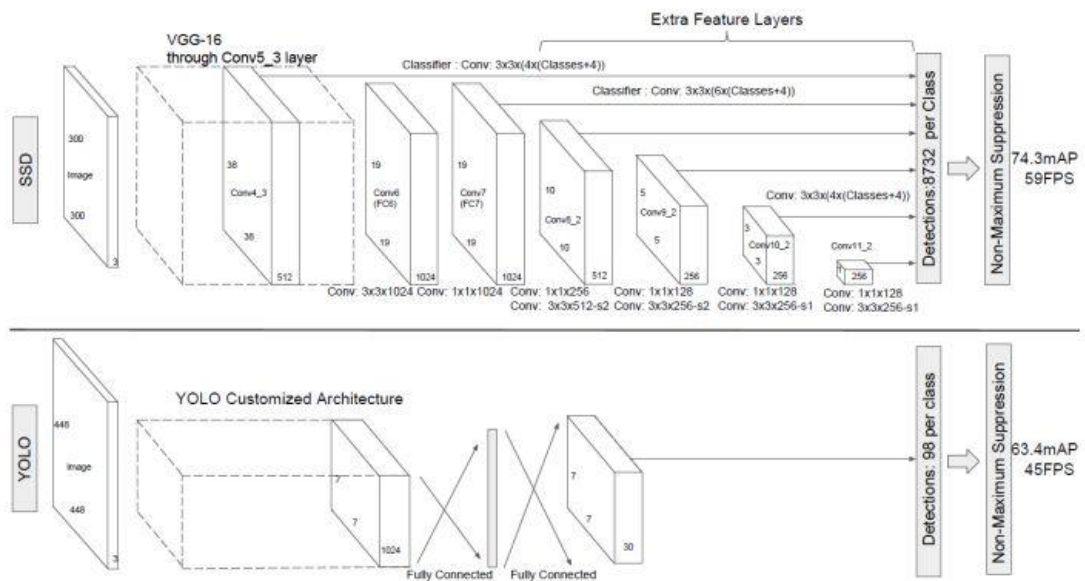


图 5 SSD 网络结构

采用 VGG16 做基础模型，首先 VGG16 是在 ILSVRC CLS-LOC 数据集预训练。然后借鉴了 DeepLab-LargeFOV，分别将 VGG16 的全连接层 fc6 和 fc7 转换成  $3 \times 3$  卷积层 conv6 和  $1 \times 1$  卷积层 conv7，同时将池化层 pool5 由原来的 stride=2 的  $2 \times 2$  变成 stride=1 的  $3 \times 3$ （猜想是不想 reduce 特征图大小），为了配合这种变化，采用了一种 Atrous Algorithm，其实就是 conv6 采用扩展卷积或带孔卷积（Dilation Conv），其在不增加参数与模型复杂度的条件下指数级扩大卷积的视野，其使用扩张率(dilation rate)参数，来表示扩张的大小，如下图 6 所示，(a)是普通的  $3 \times 3$  卷积，其视野就是  $3 \times 3$ ，(b)是扩张率为 1，此时视野变成  $7 \times 7$ ，(c)扩张率为 3 时，视野扩大为  $15 \times 15$ ，但是视野的特征更稀疏了。Conv6 采用  $3 \times 3$  大小但 dilation rate=6 的扩展卷积。

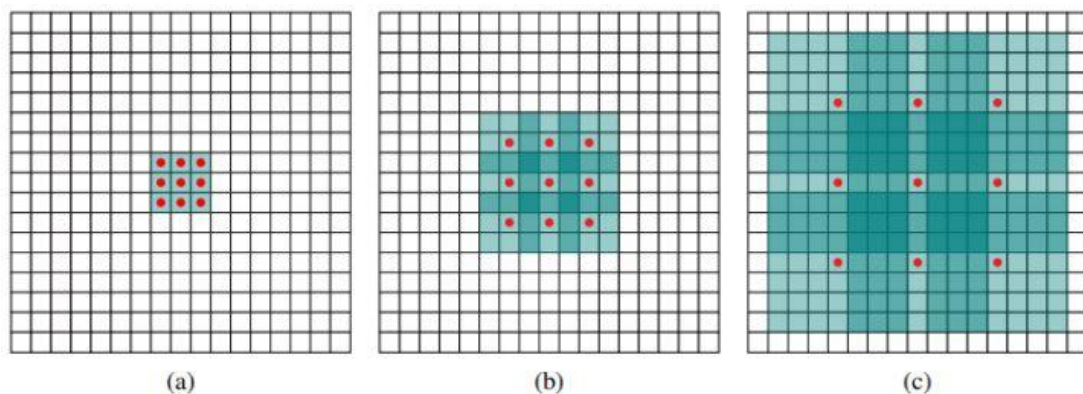


图 6 扩展卷积

然后移除 dropout 层和 fc8 层，并新增一系列卷积层，在检测数据集上做 finetuing。

其中 VGG16 中的 Conv4\_3 层将作为用于检测的第一个特征图。conv4\_3 层特征图大小是  $38 \times 38$ ，但是该层比较靠前，其 norm 较大，所以在其后面增加了一个 L2 Normalization 层（参见 ParseNet），以保证和后面的检测层差异不是很大，这个和 Batch Normalization 层不太一样，其仅仅是对每个像素点在 channle 维度做归一化，而 Batch Normalization 层是在 [batch\_size,



width, height]三个维度上做归一化。归一化后一般设置一个可训练的放缩变量 gamma，使用 TF 可以这样简单实现：

```
# l2norm (not batch norm, spatial normalization)
def l2norm(x, scale, trainable=True, scope="L2Normalization"):
    n_channels = x.get_shape().as_list()[-1]
    l2_norm = tf.nn.l2_normalize(x, [3], epsilon=1e-12)
    with tf.variable_scope(scope):
        gamma = tf.get_variable("gamma", shape=[n_channels, ],
                                dtype=tf.float32,
                                initializer=tf.constant_initializer(scale),
                                trainable=trainable)
    return l2_norm * gamma
```

从后面新增的卷积层中提取 Conv7，Conv8\_2，Conv9\_2，Conv10\_2，Conv11\_2 作为检测所用的特征图，加上 Conv4\_3 层，共提取了 6 个特征图，其大小分别是 (38,38),(19,19),(10,10),(5,5),(3,3),(1,1)，但是不同特征图设置的先验框数目不同（同一个特征图上每个单元设置的先验框是相同的，这里的数目指的是一个单元的先验框数目）。先验框的设置，包括尺度（或者说大小）和长宽比两个方面。对于先验框的尺度，其遵守一个线性递增规则：随着特征图大小降低，先验框尺度线性增加：

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1}(k - 1), k \in [1, m]$$

其中 m 指的特征图个数，但却是 5，因为第一层（Conv4\_3 层）是单独设置的， $s_k$  表示先验框大小相对于图片的比例，而  $s_{min}$  和  $s_{max}$  表示比例的最小值与最大值，paper 里面取 0.2

和 0.9。对于第一个特征图，其先验框的尺度比例一般设置为  $s_{min}/2 = 0.1$ ，那么尺度为  $300 \times 0.1 = 30$ 。对于后面的特征图，先验框尺度按照上面公式线性增加，但是先将尺度比例先扩

大 100 倍，此时增长步长为  $\left\lfloor \frac{\lfloor s_{max} \times 100 \rfloor - \lfloor s_{min} \times 100 \rfloor}{m - 1} \right\rfloor = 17$ ，这样各个特征

图的  $s_k$  为 20,37,54,71,88，将这些比例除以 100，然后再乘以图片大小，可以得到各个特征图的尺度为 60,111,162,213,264，这种计算方式是参考 SSD 的 Caffe 源码。综上，可以得到各个特征图的先验框尺度 30,60,111,162,213,264。对于长宽比，一般选

取  $a_r \in \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$ ，对于特定的长宽比，按如下公式计算先验框的宽度与高度（后面的  $s_k$  均指的是先验框实际尺度，而不是尺度比例）：

$$w_k^a = s_k \sqrt{a_r}, h_k^a = s_k / \sqrt{a_r}$$

默认情况下，每个特征图会有一个  $a_r = 1$  且尺度为  $s_k$  的先验框，除此之外，还会设置一个尺度为  $s'_k = \sqrt{s_k s_{k+1}}$  且  $a_r = 1$  的先验框，这样每个特征图都设置了两个长宽比为1但大小不同的正方形先验框。注意最后一个特征图需要参考一个虚拟  $s_{m+1} = 300 \times 105/100 = 315$  来计算  $s'_m$ 。因此，每个特征图一共有 6 个先验框  $\{1, 2, 3, \frac{1}{2}, \frac{1}{3}, 1'\}$ ，但是在实现时，Conv4\_3, Conv10\_2和Conv11\_2层仅使用4个先验框，它们不使用长宽比为  $3, \frac{1}{3}$  的先验框。

每个单元的先验框的中心点分布在各个单元的中心，即  $(\frac{i+0.5}{|f_k|}, \frac{j+0.5}{|f_k|})$ ,  $i, j \in [0, |f_k|)$ ，其中  $|f_k|$  为特征图的大小。

得到了特征图之后，需要对特征图进行卷积得到检测结果，图7给出了一个  $5 \times 5$  大小的特征图的检测过程。其中Priorbox是得到先验框，前面已经介绍了生成规则。检测值包含两个部分：类别置信度和边界框位置，各采用一次  $3 \times 3$  卷积来进行完成。令  $n_k$  为该特征图所采用的先验框数目，那么类别置信度需要的卷积核数量为  $n_k \times c$ ，而边界框位置需要的卷积核数量为  $n_k \times 4$ 。由于每个先验框都会预测一个边界框，所以SSD300一共可以预测

$$38 \times 38 \times 4 + 19 \times 19 \times 6 + 10 \times 10 \times 6 + 5 \times 5 \times 6 + 3 \times 3 \times 4 + 1 \times 1 \times 4 = 8732$$

个边界框，这是一个相当庞大的数字，所以说SSD本质上是密集采样。

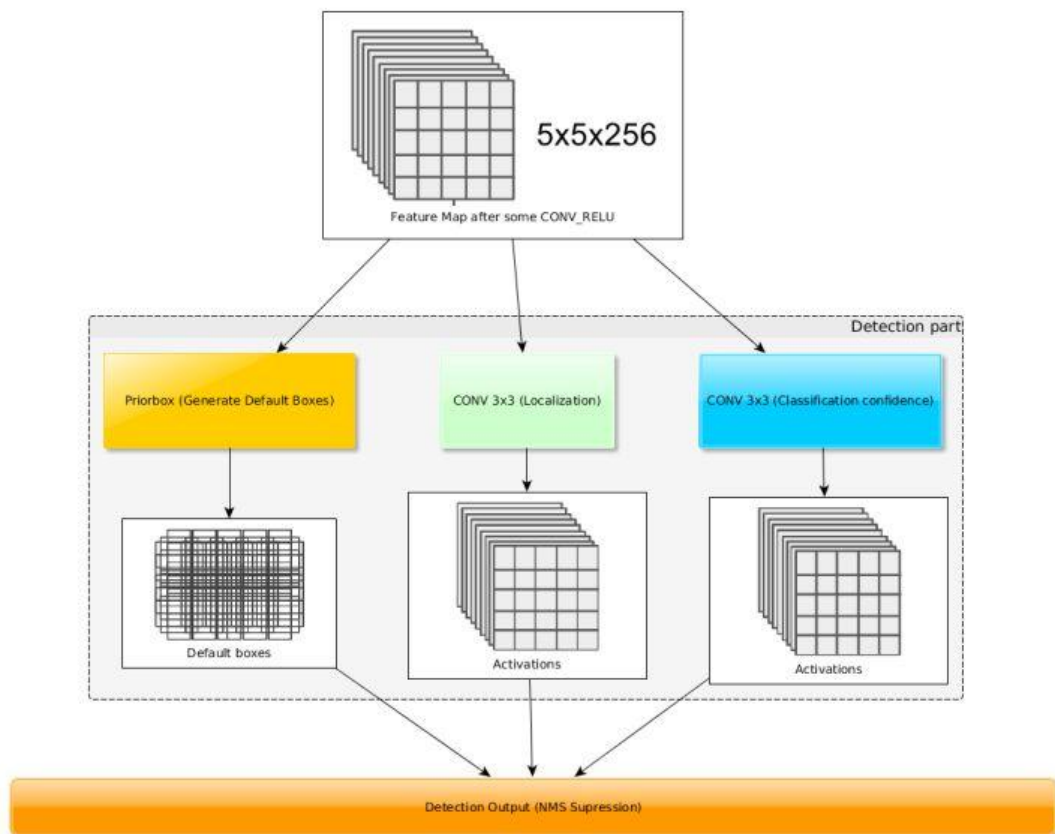


图 7 基于卷积得到检测结果

## 训练过程

### (1) 先验框匹配

在训练过程中，首先要确定训练图片中的 **ground truth**（真实目标）与哪个先验框来进行匹配，与之匹配的先验框所对应的边界框将负责预测它。在 Yolo 中，**ground truth** 的中心落在哪个单元格，该单元格中与其 **IOU** 最大的边界框负责预测它。但是在 SSD 中却完全不一样，SSD 的先验框与 **ground truth** 的匹配原则主要有两点。首先，对于图片中每个 **ground truth**，找到与其 **IOU** 最大的先验框，该先验框与其匹配，这样，可以保证每个 **ground truth** 一定与某个先验框匹配。通常称与 **ground truth** 匹配的先验框为正样本（其实应该是先验框对应的预测 box，不过由于是一一对应的就这样称呼了），反之，若一个先验框没有与任何 **ground truth** 进行匹配，那么该先验框只能与背景匹配，就是负样本。一个图片中 **ground truth** 是非常少的，而先验框却很多，如果仅按第一个原则匹配，很多先验框会是负样本，正负样本极其不平衡，所以需要第二个原则。第二个原则是：对于剩余的未匹配先验框，若某个 **ground truth** 的 **IOU** 大于某个阈值（一般是 0.5），那么该先验框也与这个 **ground truth** 进行匹配。这意味着某个 **ground truth** 可能与多个先验框匹配，这是可以的。但是反过来却不可以，因为一个先验框只能匹配一个 **ground truth**，如果多个 **ground truth** 与某个先验框 **IOU** 大于阈值，那么先验框只与 **IOU** 最大的那个先验框进行匹配。第二个原则一定在第一个原则之后进行，仔细考虑一下这种情况，如果某个 **ground truth** 所对应最大 **IOU** 小于阈值，并且所匹配的先验框却与另外一个 **ground truth** 的 **IOU** 大于阈值，那么该先验框应该匹配谁，答案应该是前者，首先要确保某个 **ground truth** 一定有一个先验框与之匹配。但是，这种情况我觉得基本上是不存在的。由于先验框很多，某个 **ground truth** 的最大 **IOU** 肯定大于阈值，所以可能只实施第二个原则就可以了，这里的 TensorFlow 版本就是只实施了第二个原则，但是这里的 Pytorch 两个原则都实施了。图 8 为一个匹配示意图，其中绿色的 GT 是 **ground truth**，红色为先验框，FP 表示负样本，TP 表示正样本。

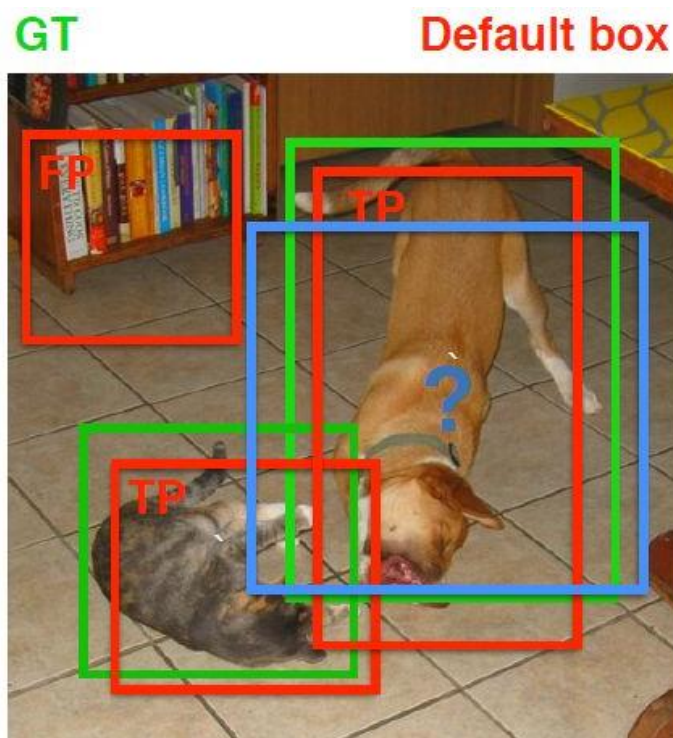


图 8 先验框匹配示意图



尽管一个 ground truth 可以与多个先验框匹配，但是 ground truth 相对先验框还是太少了，所以负样本相对正样本会很多。为了保证正负样本尽量平衡，SSD 采用了 hard negative mining，就是对负样本进行抽样，抽样时按照置信度误差（预测背景的置信度越小，误差越大）进行降序排列，选取误差的较大的 top-k 作为训练的负样本，以保证正负样本比例接近 1:3。

## （2）损失函数

训练样本确定了，然后就是损失函数了。损失函数定义为位置误差（locatization loss, loc）与置信度误差（confidence loss, conf）的加权和：

$$L(x, c, l, g) = \frac{1}{N} (L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

其中  $N$  是先验框的正样本数量。这里  $x_{ij}^p \in \{1, 0\}$  为一个指示参数，当  $x_{ij}^p = 1$  时表示第  $i$  个先验框与第  $j$  个 ground truth 匹配，并且 ground truth 的类别为  $p$ 。  $c$  为类别置信度预测值。  $l$  为先验框的所对应边界框的位置预测值，而  $g$  是 ground truth 的位置参数。对于位置误差，其采用 Smooth L1 loss，定义如下：

$$L_{loc}(x, l, g) = \sum_{i \in Pos} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \quad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \quad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

$$\text{smooth}_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

由于  $x_{ij}^p$  的存在，所以位置误差仅针对正样本进行计算。值得注意的是，要先对 ground truth 的  $g$  进行编码得到  $\hat{g}$ ，因为预测值  $l$  也是编码值，若设置 variance\_encoded\_in\_target=True，编码时要加上 variance：

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w / \text{variance}[0], \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h / \text{variance}[1]$$

$$\hat{g}_j^w = \log(g_j^w/d_i^w) / \text{variance}[2], \hat{g}_j^h = \log(g_j^h/d_i^h) / \text{variance}[3]$$

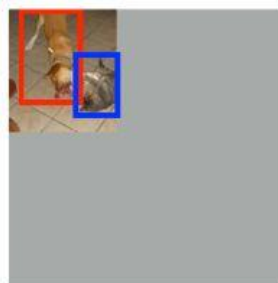
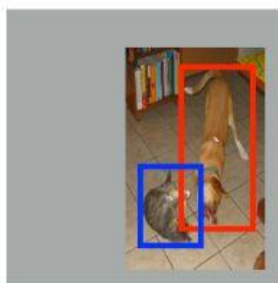
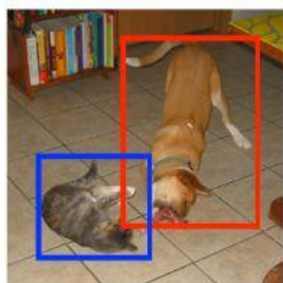
对于置信度误差，其采用 softmax loss：

$$L_{conf}(x, c) = - \sum_{i \in Pos} x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

权重系数  $\alpha$  通过交叉验证设置为 1。

### (3) 数据扩增

采用数据扩增 (Data Augmentation) 可以提升 SSD 的性能, 主要采用的技术有水平翻转 (horizontal flip), 随机裁剪加颜色扭曲 (random crop & color distortion), 随机采集块域 (Randomly sample a patch) (获取小目标训练样本), 如下图所示:



Random expansion creates more **small** training examples

图 9 数据扩增方案

其它的训练细节如学习速率的选择详见论文, 这里不再赘述。

### 预测过程

预测过程比较简单, 对于每个预测框, 首先根据类别置信度确定其类别 (置信度最大者) 与置信度值, 并过滤掉属于背景的预测框。然后根据置信度阈值 (如 0.5) 过滤掉阈值较低的预测框。对于留下的预测框进行解码, 根据先验框得到其真实的位置参数 (解码后一般还需要做 clip, 防止预测框位置超出图片)。解码之后, 一般需要根据置信度进行降序排列, 然后仅保留 top-k (如 400) 个预测框。最后就是进行 NMS 算法, 过滤掉那些重叠度较大的预测框。最后剩余的预测框就是检测结果了。

### 性能评估

首先整体看一下 SSD 在 VOC2007, VOC2012 及 COCO 数据集上的性能, 如表 1 所示。相比之下, SSD512 的性能会更好一些。加\*的表示使用了 image expansion data augmentation (通过 zoom out 来创造小的训练样本) 技巧来提升 SSD 在小目标上的检测效果, 所以性能会有所提升。

Method	VOC2007 test		VOC2012 test		COCO test-dev2015		
	07+12	07+12+COCO	07++12	07++12+COCO	trainval35k		
	0.5	0.5	0.5	0.5	0.5:0.95	0.5	0.75
SSD300	74.3	79.6	72.4	77.5	23.2	41.2	23.4
SSD512	76.8	81.6	74.9	80.0	26.8	46.5	27.8
SSD300*	77.2	81.2	75.8	79.3	25.1	43.1	25.8
SSD512*	<b>79.8</b>	<b>83.2</b>	<b>78.5</b>	<b>82.2</b>	<b>28.8</b>	<b>48.5</b>	<b>30.3</b>

表 1 SSD 在不同数据集上的性能

SSD 与其它检测算法的对比结果（在 VOC2007 数据集）如表 2 所示，基本可以看到，SSD 与 Faster R-CNN 有同样的准确度，并且与 Yolo 具有同样较快地检测速度。

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

表 2 SSD 与其它检测算法的对比结果（在 VOC2007 数据集）

文章还对 SSD 的各个 trick 做了更为细致的分析，表 3 为不同的 trick 组合对 SSD 的性能影响，从表中可以得出如下结论：

- 数据扩增技术很重要，对于 mAP 的提升很大；
- 使用不同长宽比的先验框可以得到更好的结果；

	SSD300				
more data augmentation?	✓	✓	✓	✓	✓
include $\{\frac{1}{2}, 2\}$ box?	✓		✓	✓	✓
include $\{\frac{1}{3}, 3\}$ box?	✓			✓	✓
use atrous?	✓	✓	✓		✓
VOC2007 test mAP	65.5	71.6	73.7	74.2	<b>74.3</b>

表 3 不同的 trick 组合对 SSD 的性能影响

同样的，采用多尺度的特征图用于检测也是至关重要的，这可以从表 4 中看出：

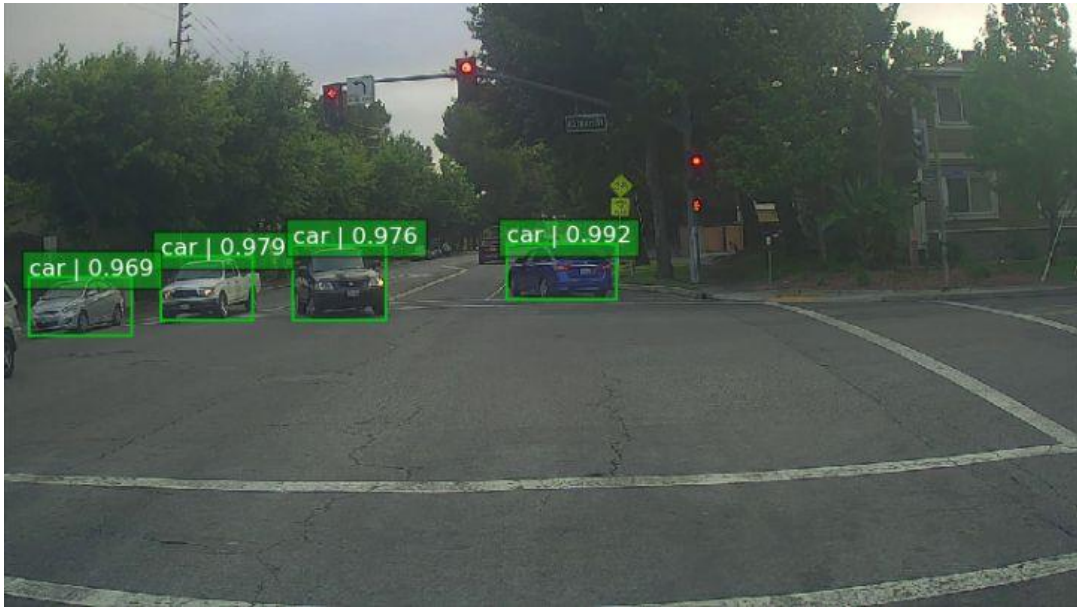
Prediction source layers from:						mAP		# Boxes
conv4_3	conv7	conv8_2	conv9_2	conv10_2	conv11_2	use boundary boxes?		
						Yes	No	
✓	✓	✓	✓	✓	✓	74.3	63.4	8732
✓	✓	✓	✓	✓		<b>74.6</b>	63.1	8764
✓	✓	✓	✓			73.8	68.4	8942
✓	✓	✓				70.7	69.2	9864
✓	✓					64.2	64.4	9025
	✓					62.4	64.0	8664

表

#### 4 多尺度特征图对 SSD 的影响

## TensorFlow 上的实现(详见 Github)

详细的代码放在 [Github](#) 上了，然后看下一个自然图片的检测效果：



如果你想实现 SSD 的 train 过程,你可以参考附录里面的 Caffe, TensorFlow 以及 Pytorch 实现。

## 小结

SSD 在 Yolo 的基础上主要改进了三点: 多尺度特征图, 利用卷积进行检测, 设置先验框。这使得 SSD 在准确度上比 Yolo 更好, 而且对于小目标检测效果也相对好一点。由于很多实现细节都包含在源码里面, 文中有描述不准或者错误的地方在所难免, 欢迎交流指正。

## 参考文献

1. [SSD: Single Shot MultiBox Detector](#)
2. [SSD Slide](#)
3. [SSD Caffe](#)
4. [SSD TensorFlow](#)
5. [SSD Pytorch](#)
6. [leonardoaraujosantos Artificial Intelligence online book](#)