目标检测|YOLOv2 原理与实现(附 YOLOv3)

前言

在前面的一篇文章中，我们详细介绍了 YOLOv1 的原理以及实现过程。这篇文章接着介绍 YOLOv2 的原理以及实现，YOLOv2 的论文全名为 YOLO9000: Better, Faster, Stronger，它斩获了 CVPR 2017 Best Paper Honorable Mention。在这篇文章中，作者首先在 YOLOv1 的基础上提出了改进的 YOLOv2，然后提出了一种检测与分类联合训练方法，使用这种联合训练方法在 COCO 检测数据集和 ImageNet 分类数据集上训练出了 YOLO9000 模型，其可以检测超过 9000 多类物体。所以，这篇文章其实包含两个模型：YOLOv2 和 YOLO9000，不过后者是在前者基础上提出的，两者模型主体结构是一致的。YOLOv2 相比 YOLOv1 做了很多方面的改进，这也使得 YOLOv2 的 mAP 有显著的提升，并且 YOLOv2 的速度依然很快，保持着自己作为 one-stage 方法的优势，YOLOv2 和 Faster R-CNN, SSD 等模型的对比如图 1 所示。这里将首先介绍 YOLOv2 的改进策略，并给出 YOLOv2 的 TensorFlow 实现过程，然后介绍 YOLO9000 的训练方法。近期，YOLOv3 也放出来了，YOLOv3 也在 YOLOv2 的基础上做了一部分改进，我们在最后也会简单谈谈 YOLOv3 所做的改进工作。

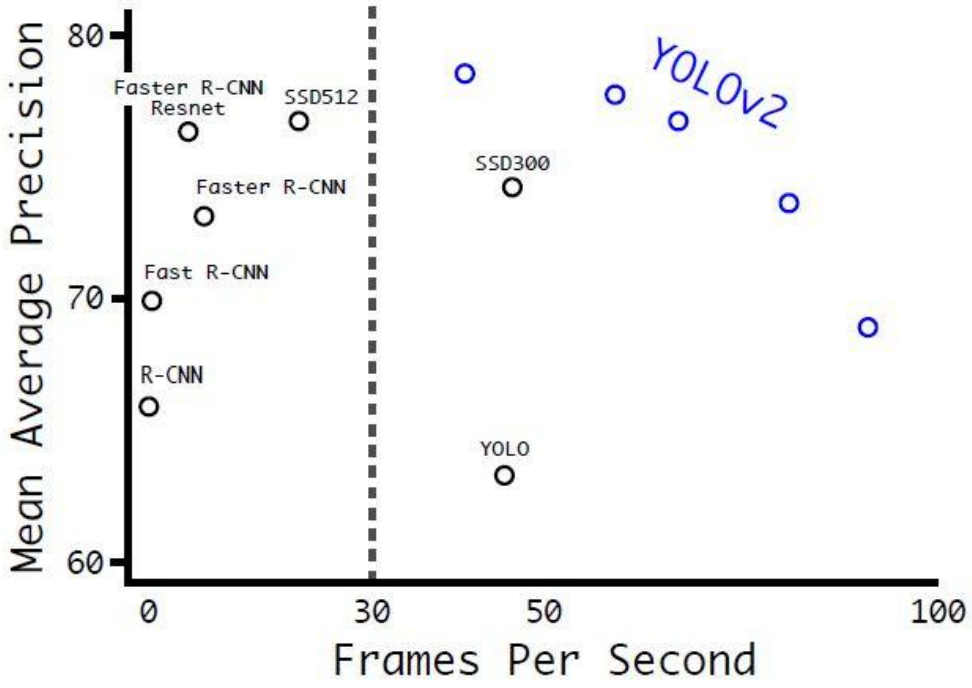

图 1：YOLOv2 与其它模型在 VOC 2007 数据集上的效果对比

YOLOv2 的改进策略

YOLOv1 虽然检测速度很快，但是在检测精度上却不如 R-CNN 系检测方法，YOLOv1 在物体定位方面（localization）不够准确，并且召回率（recall）较低。YOLOv2 共提出了几种改进策略来提升 YOLO 模型的定位准确度和召回率，从而提高 mAP，YOLOv2 在改进中遵循一个原则：保持检测速度，这也是 YOLO 模型的一大优势。YOLOv2 的改进策略如图 2 所示，可以看出，大部分的改进方法都可以比较显著提升模型的 mAP。下面详细介绍各个改进策略。

	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

图 2：YOLOv2 相比 YOLOv1 的改进策略

Batch Normalization

Batch Normalization 可以提升模型收敛速度，而且可以起到一定正则化效果，降低模型的过拟合。在 YOLOv2 中，每个卷积层后面都添加了 Batch Normalization 层，并且不再使用 droput。使用 Batch Normalization 后， YOLOv2 的 mAP 提升了 2.4%。

High Resolution Classifier

目前大部分的检测模型都会在先在 ImageNet 分类数据集上预训练模型的主体部分（CNN 特征提取器），由于历史原因，ImageNet 分类模型基本采用大小为 224 x 224 的图片作为输入，分辨率相对较低，不利于检测模型。所以 YOLOv1 在采用 224 x 224 分类模型预训练后，将分辨率增加至 448 x 448 ，并使用这个高分辨率在检测数据集上 finetune。但是直接切换分辨率，检测模型可能难以快速适应高分辨率。所以 YOLOv2 增加了在 ImageNet 数据集上使用 448 x 448 输入来 finetune 分类网络这一中间过程（10 epochs），这可以使得模型在检测数据集上 finetune 之前已经适用高分辨率输入。使用高分辨率分类器后，YOLOv2 的 mAP 提升了约 4%。

Convolutional With Anchor Boxes

在 YOLOv1 中，输入图片最终被划分为 7 x 7 网格，每个单元格预测 2 个边界框。YOLOv1 最后采用的是全连接层直接对边界框进行预测，其中边界框的宽与高是相对整张图片大小的，而由于各个图片中存在不同尺度和长宽比（scales and ratios）的物体，YOLOv1 在训练过程中学习适应不同物体的形状是比较困难的，这也导致 YOLOv1 在精确定位方面表现较差。YOLOv2 借鉴了 Faster R-CNN 中 RPN 网络的先验框（anchor boxes，prior boxes，SSD 也采用了先验框）策略。RPN 对 CNN 特征提取器得到的特征图（feature map）进行卷积来预测每个位置的边界框以及置信度（是否含有物体），并且各个位置设置不同尺度和比例的先验框，所以 RPN 预测的是边界框相对于先验框的 offsets 值（其实是 transform 值，详细见 Faster R_CNN 论文），采用先验框使得模型更容易学习。所以 YOLOv2 移除了 YOLOv1 中的全连接层而采用了卷积和 anchor boxes 来预测边界框。为了使检测所用的特征图分辨率更高，移除其中的一个 pool 层。在检测模型中，YOLOv2 不是采用 448 x 448 图片作为输入，而是采用 416 x 416 大小。因为 YOLOv2 模型下采样的总步长为 32 ，对于 416 x 416 大小的图片，最终得到的特征图大小为 13 x 13 ，维度是奇数，这样特征图恰好只有一个中心位置。对于一些大物体，它们中

心点往往落入图片中心位置，此时使用特征图的一个中心点去预测这些物体的边界框相对容易些。所以在 YOLOv2 设计中要保证最终的特征图有奇数个位置。对于 YOLOv1，每个 cell 都预测 2 个 boxes，每个 boxes 包含 5 个值：(x,y,w,h,c)，前 4 个值是边界框位置与大小，最后一个值是置信度（confidence scores，包含两部分：含有物体的概率以及预测框与 ground truth 的 IOU）。但是每个 cell 只预测一套分类概率值（class predictions，其实是置信度下的条件概率值）,供 2 个 boxes 共享。YOLOv2 使用了 anchor boxes 之后，每个位置的各个 anchor box 都单独预测一套分类概率值，这和 SSD 比较类似（但 SSD 没有预测置信度，而是把 background 作为一个类别来处理）。

使用 anchor boxes 之后，YOLOv2 的 mAP 有稍微下降（这里下降的原因，我猜想是 YOLOv2 虽然使用了 anchor boxes，但是依然采用 YOLOv1 的训练方法）。YOLOv1 只能预测 98 个边界框（ 7 x 7 x 2 ），而 YOLOv2 使用 anchor boxes 之后可以预测上千个边界框（ 13 x 13 x num_anchors ）。所以使用 anchor boxes 之后，YOLOv2 的召回率大大提升，由原来的 81%升至 88%。

Dimension Clusters

在 Faster R-CNN 和 SSD 中，先验框的维度（长和宽）都是手动设定的，带有一定的主观性。如果选取的先验框维度比较合适，那么模型更容易学习，从而做出更好的预测。因此，YOLOv2 采用 k-means 聚类方法对训练集中的边界框做了聚类分析。因为设置先验框的主要目的是为了使得预测框与 ground truth 的 IOU 更好，所以聚类分析时选用 box 与聚类中心 box 之间的 IOU 值作为距离指标：

$$d(box, centroid) = 1 - IOU(box, centroid)$$

图 3 为在 VOC 和 COCO 数据集上的聚类分析结果，随着聚类中心数目的增加，平均 IOU 值（各个边界框与聚类中心的 IOU 的平均值）是增加的，但是综合考虑模型复杂度和召回率，作者最终选取 5 个聚类中心作为先验框，其相对于图片的大小如右边图所示。对于两个数据集，5 个先验框的 width 和 height 如下所示（来源：YOLO 源码的 cfg 文件）：

COCO: (0.57273, 0.677385), (1.87446, 2.06253), (3.33843, 5.47434), (7.88282, 3.52778), (9.77052, 9.16828)
VOC: (1.3221, 1.73145), (3.19275, 4.00944), (5.05587, 8.09892), (9.47112, 4.84053), (11.2364, 10.0071)

但是这里先验框的大小具体指什么作者并没有说明，但肯定不是像素点，从代码实现上看，应该是相对于预测的特征图大小（ 13 x 13 ）。对比两个数据集，也可以看到 COCO 数据集上的物体相对小点。这个策略作者并没有单独做实验，但是作者对比了采用聚类分析得到的先验框与手动设置的先验框在平均 IOU 上的差异，发现前者的平均 IOU 值更高，因此模型更容易训练学习。

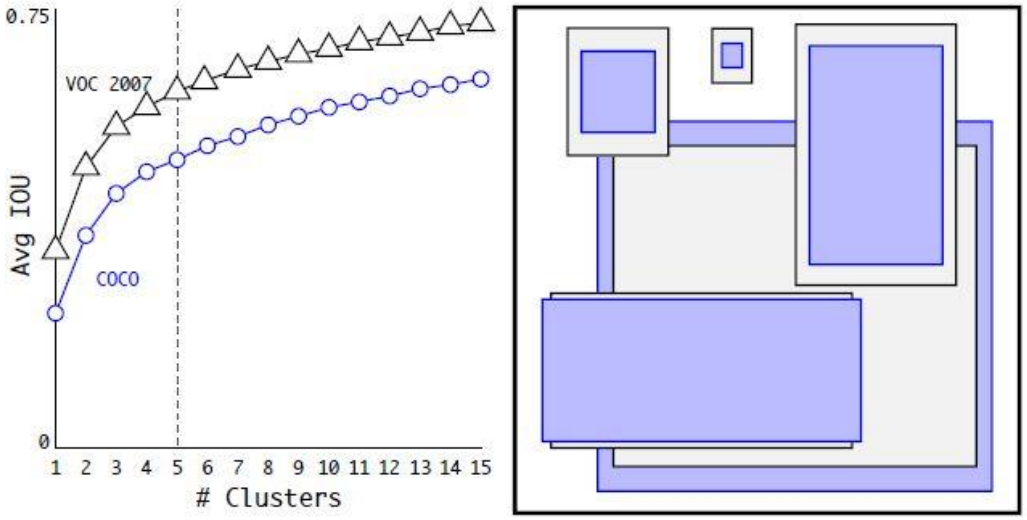

图 3：数据集 VOC 和 COCO 上的边界框聚类分析结果

New Network: Darknet-19

YOLOv2 采用了一个新的基础模型（特征提取器），称为 Darknet-19，包括 19 个卷积层和 5 个 maxpooling 层，如图 4 所示。Darknet-19 与 VGG16 模型设计原则是一致的，主要采用 3 x3 卷积，采用 2 x 2 的 maxpooling 层之后，特征图维度降低 2 倍，而同时将特征图的 channles 增加两倍。与 NIN(Network in Network)类似，Darknet-19 最终采用 global avgpooling 做预测，并且在 3 x 3 卷积之间使用 1 x 1 卷积来压缩特征图 channles 以降低模型计算量和参数。

Darknet-19 每个卷积层后面同样使用了 batch norm 层以加快收敛速度，降低模型过拟合。在 ImageNet 分类数据集上，Darknet-19 的 top-1 准确度为 72.9%，top-5 准确度为 91.2%，但是模型参数相对小一些。使用 Darknet-19 之后，YOLOv2 的 mAP 值没有显著提升，但是计算量却可以减少约 33%。

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

图 4：Darknet-19 模型结构

Direct location prediction

前面讲到，YOLOv2借鉴RPN网络使用anchor boxes来预测边界框相对先验框的offsets。边界框的实际中心位置 (x, y) ，需要根据预测的坐标偏移值 (t_x, t_y) ，先验框的尺度 (w_a, h_a) 以及中心坐标 (x_a, y_a) （特征图每个位置的中心点）来计算：

$$x = (t_x \times w_a) - x_a$$

$$y = (t_y \times h_a) - y_a$$

但是上面的公式是无约束的，预测的边界框很容易向任何方向偏移，如当 $t_x = 1$ 时边界框将向右偏移先验框的一个宽度大小，而当 $t_x = -1$ 时边界框将向左偏移先验框的一个宽度大小，因此每个位置预测的边界框可以落在图片任何位置，这导致模型的不稳定性，在训练时需要很长时间来预测出正确的offsets。所以，YOLOv2弃用了这种预测方式，而是沿用YOLOv1的方法，就是预测边界框中心点相对于对应cell左上角位置的相对偏移值，为了将边界框中心点约束在当前cell中，使用sigmoid函数处理偏移值，这样预测的偏移值在(0,1)范围内（每个cell的尺度看做1）。总结来看，根据边界框预测的4个offsets t_x, t_y, t_w, t_h ，可以按如下公式计算出边界框实际位置和大小：

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

其中 (c_x, x_y) 为cell的左上角坐标，如图5所示，在计算时每个cell的尺度为1，所以当前cell的左上角坐标为 $(1, 1)$ 。由于sigmoid函数的处理，边界框的中心位置会约束在当前cell内部，防止偏移过多。而 p_w 和 p_h 是先验框的宽度与长度，前面说过它们的值也是相对于特征图大小的，在特征图中每个cell的长和宽均为1。这里记特征图的大小为 (W, H) （在文中是 $(13, 13)$），这样我们可以将边界框相对于整张图片的位置和大小计算出来（4个值均在0和1之间）：

$$b_x = (\sigma(t_x) + c_x)/W$$

$$b_y = (\sigma(t_y) + c_y)/H$$

$$b_w = p_w e^{t_w}/W$$

$$b_h = p_h e^{t_h}/H$$

如果再将上面的 4 个值分别乘以图片的宽度和长度（像素点值）就可以得到边界框的最终位置和大小了。这就是 YOLOv2 边界框的整个解码过程。约束了边界框的位置预测值使得模型更容易稳定训练，结合聚类分析得到先验框与这种预测方法，YOLOv2 的 mAP 值提升了约 5%。

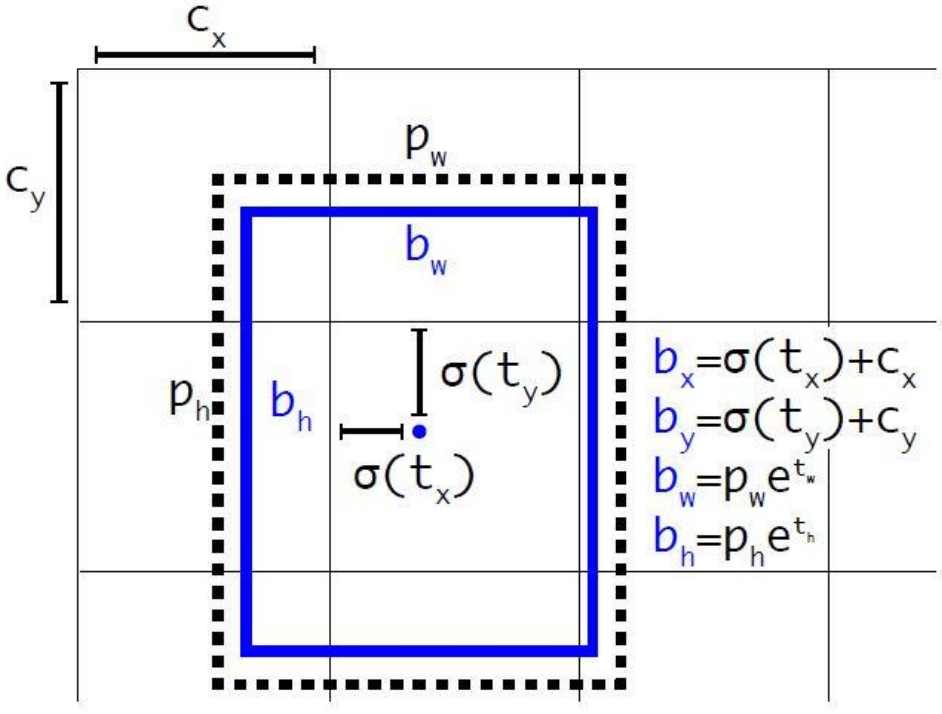

图 5：边界框位置与大小的计算示例图

Fine-Grained Features

YOLOv2的输入图片大小为 416×416 ，经过5次maxpooling之后得到 13×13 大小的特征图，并以此特征图采用卷积做预测。 13×13 大小的特征图对检测大物体是足够了，但是对于小物体还需要更精细的特征图（Fine-Grained Features）。因此SSD使用了多尺度的特征图来分别检测不同大小的物体，前面更精细的特征图可以用来预测小物体。YOLOv2提出了一种passthrough层来利用更精细的特征图。YOLOv2所利用的Fine-Grained Features是 26×26 大小的特征图（最后一个maxpooling层的输入），对于Darknet-19模型来说就是大小为 $26 \times 26 \times 512$ 的特征图。passthrough层与ResNet网络的shortcut类似，以前面更高分辨率的特征图为输入，然后将其连接到后面的低分辨率特征图上。前面的特征图维度是后面的特征图的2倍，passthrough层抽取前面层的每个 2×2 的局部区域，然后将其转化为channel维度，对于 $26 \times 26 \times 512$ 的特征图，经passthrough层处理之后就变成了 $13 \times 13 \times 2048$ 的新特征图（特征图大小降低4倍，而channles增加4倍，图6为一个实例），这样就可以与后面的 $13 \times 13 \times 1024$ 特征图连接在一起形成 $13 \times 13 \times 3072$ 大小的特征图，然后在此特征图基础上卷积做预测。在YOLO的C源码中，passthrough层称为reorg layer。在TensorFlow中，可以使用tf.extract_image_patches或者tf.space_to_depth来实现passthrough层：

```
out = tf.extract_image_patches(in, [1, stride, stride, 1], [1, stride, stride, 1], [1,1,1,1], padding="VALID")
// or use tf.space_to_depth
out = tf.space_to_depth(in, 2)
```

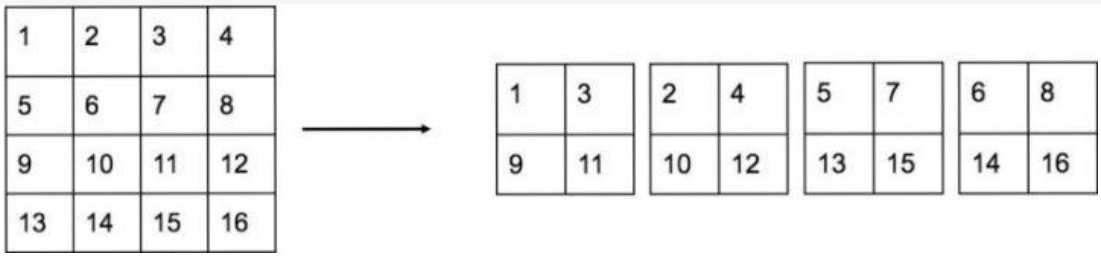

图 6：passthrough 层实例

另外，作者在后期的实现中借鉴了 ResNet 网络，不是直接对高分辨特征图处理，而是增加了一个中间卷积层，先采用 64 个 1 x 1 卷积核进行卷积，然后再进行 passthrough 处理，这样 26 x 26 x 512 的特征图得到 13 x 13 x 256 的特征图。这算是实现上的一个小细节。使用 Fine-Grained Features 之后 YOLOv2 的性能有 1%的提升。

Multi-Scale Training

由于YOLOv2模型中只有卷积层和池化层，所以YOLOv2的输入可以不限于 416×416 大小的图片。为了增强模型的鲁棒性，YOLOv2采用了多尺度输入训练策略，具体来说就是在训练过程中每间隔一定的iterations之后改变模型的输入图片大小。由于YOLOv2的下采样总步长为32，输入图片大小选择一系列为32倍数的值： $\{320, 352, \ldots, 608\}$ ，输入图片最小为 320×320 ，此时对应的特征图大小为 10×10 （不是奇数了，确实有点尴尬），而输入图片最大为 608×608 ，对应的特征图大小为 19×19 。在训练过程，每隔10个iterations随机选择一种输入图片大小，然后只需要修改对最后检测层的处理就可以重新训练。

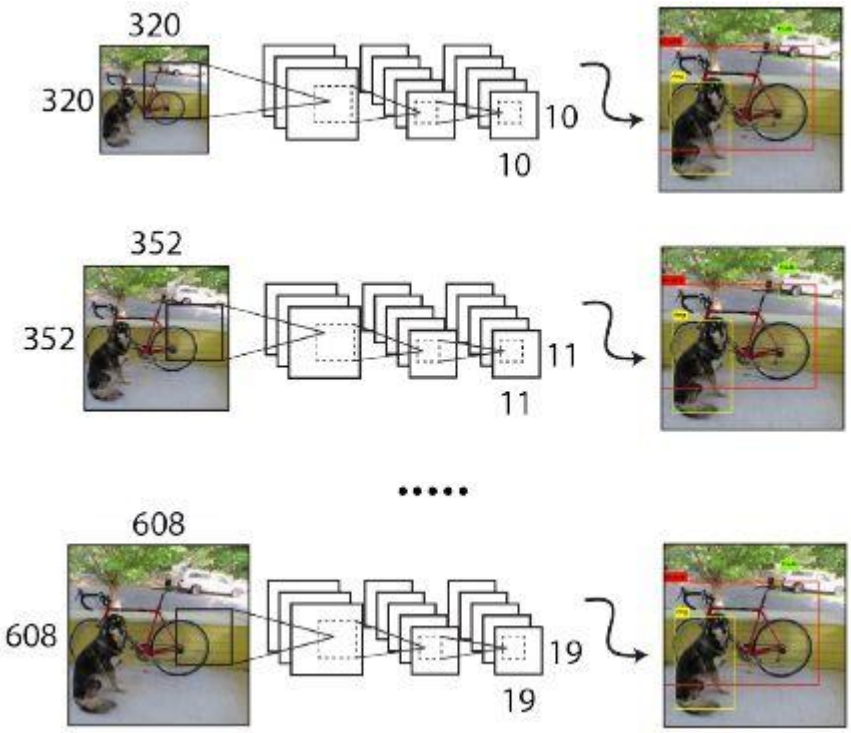

图 7：Multi-Scale Training

采用 Multi-Scale Training 策略，YOLOv2 可以适应不同大小的图片，并且预测出很好的结果。在测试时，YOLOv2 可以采用不同大小的图片作为输入，在 VOC 2007 数据集上的效果如下图所示。可以看到采用较小分辨率时，YOLOv2 的 mAP 值略低，但是速度更快，而采用高分辨输入时，mAP 值更高，但是速度略有下降，对于 544 x 544 ，mAP 高达 78.6%。注意，这只是测试时输入图片大小不同，而实际上用的是同一个模型（采用 Multi-Scale Training 训练）。

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

图 8：YOLOv2 在 VOC 2007 数据集上的性能对比

总结来看，虽然 YOLOv2 做了很多改进，但是大部分都是借鉴其它论文的一些技巧，如 Faster R-CNN 的 anchor boxes，YOLOv2 采用 anchor boxes 和卷积做预测，这基本上与 SSD 模型（单尺度特征图的 SSD）非常类似了，而且 SSD 也是借鉴了 Faster R-CNN 的 RPN 网络。从某种意

义上来说，YOLOv2 和 SSD 这两个 one-stage 模型与 RPN 网络本质上无异，只不过 RPN 不做类别的预测，只是简单地区分物体与背景。在 two-stage 方法中，RPN 起到的作用是给出 region proposals，其实就是作出粗糙的检测，所以另外增加了一个 stage，即采用 R-CNN 网络来进一步提升检测的准确度（包括给出类别预测）。而对于 one-stage 方法，它们想要一步到位，直接采用“RPN”网络作出精确的预测，要因此要在网络设计上做很多的 tricks。YOLOv2 的一大创新是采用 Multi-Scale Training 策略，这样同一个模型其实就可以适应多种大小的图片了。

YOLOv2 的训练

YOLOv2的训练主要包括三个阶段。第一阶段就是先在ImageNet分类数据集上预训练Darknet-19，此时模型输入为 224×224 ，共训练160个epochs。然后第二阶段将网络的输入调整为 448×448 ，继续在ImageNet数据集上finetune分类模型，训练10个epochs，此时分类模型的top-1准确度为76.5%，而top-5准确度为93.3%。第三个阶段就是修改Darknet-19分类模型为检测模型，并在检测数据集上继续finetune网络。网络修改包括（网路结构可视化）：移除最后一个卷积层、global avgpooling层以及softmax层，并且新增了三个 $3 \times 3 \times 2014$ 卷积层，同时增加了一个passthrough层，最后使用 1×1 卷积层输出预测结果，输出的channels数为： $\text{num\_anchors} \times (5 + \text{num\_classes})$ ，和训练采用的数据集有关系。由于anchors数为5，对于VOC数据集输出的channels数就是125，而对于COCO数据集则为425。这里以VOC数据集为例，最终的预测矩阵为 T （shape为 $(\text{batch\_size}, 13, 13, 125)$ ），可以先将其reshape为 $(\text{batch\_size}, 13, 13, 5, 25)$ ，其中 $T[:, :, :, :, 0:4]$ 为边界框的位置和大小 (t_x, t_y, t_w, t_h) ， $T[:, :, :, :, 4]$ 为边界框的置信度，而 $T[:, :, :, :, 5:]$ 为类别预测值。

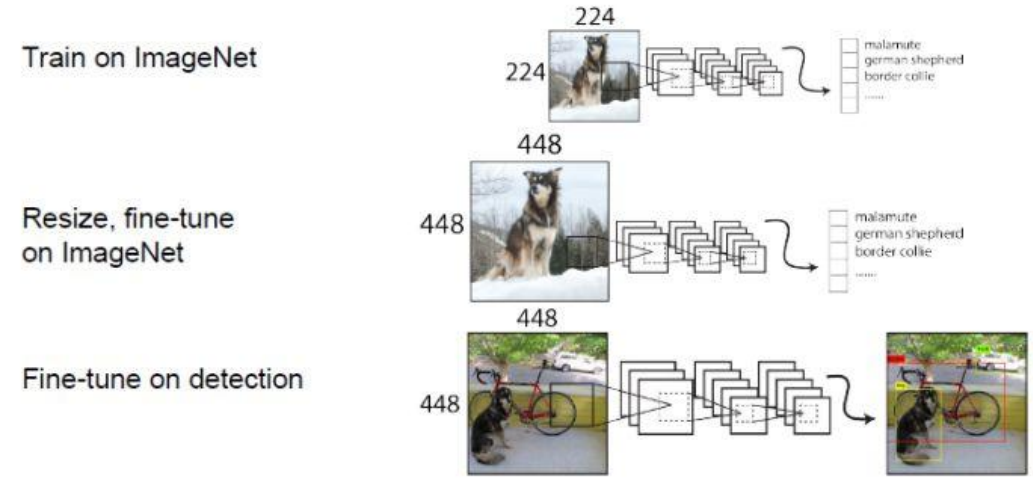

图 9：YOLOv2 训练的三个阶段

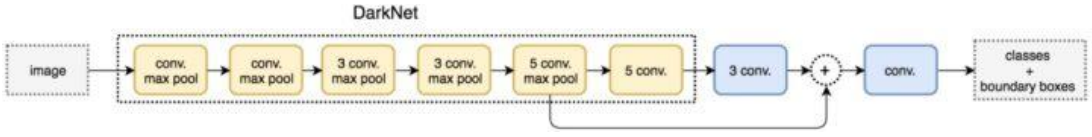

图 10：YOLOv2 结构示意图

YOLOv2 的网络结构以及训练参数我们都知道了，但是貌似少了点东西。仔细一想，原来作者并没有给出 YOLOv2 的训练过程的两个最重要方面，即先验框匹配（样本选择）以及训练的损失函数，难怪 Ng 说 YOLO 论文很难懂，没有这两方面的说明我们确实不知道 YOLOv2 到

底是怎么训练起来的。不过默认按照 YOLOv1 的处理方式也是可以处理，我看了 YOLO 在 TensorFlow 上的实现 darkflow（见 yolov2/train.py），发现它就是如此处理的：和 YOLOv1 一样，对于训练图片中的 ground truth，若其中心点落在某个 cell 内，那么该 cell 内的 5 个先验框所对应的边界框负责预测它，具体是哪个边界框预测它，需要在训练中确定，即由那个与 ground truth 的 IOU 最大的边界框预测它，而剩余的 4 个边界框不与该 ground truth 匹配。YOLOv2 同样需要假定每个 cell 至多含有一个 grounth truth，而在实际上基本不会出现多于 1 个的情况。与 ground truth 匹配的先验框计算坐标误差、置信度误差（此时 target 为 1）以及分类误差，而其它的边界框只计算置信度误差（此时 target 为 0）。YOLOv2 和 YOLOv1 的损失函数一样，为均方差函数。

但是我看了 YOLOv2 的源码（训练样本处理与 loss 计算都包含在文件 region_layer.c 中，YOLO 源码没有任何注释，反正我看了是直摇头），并且参考国外的 blog 以及 allanzelener/YAD2K（Ng 深度学习教程所参考的那个 Keras 实现）上的实现，发现 YOLOv2 的处理比原来的 v1 版本更加复杂。先给出 loss 计算公式：

$$
\begin{aligned}
loss_t = \sum_{i=0}^{W}\sum_{j=0}^{H}\sum_{k=0}^{A} \quad & 1_{Max\,IOU < Thresh}\,\lambda_{noobj} * \left(-b_{ijk}^{o}\right)^2 \\
& + 1_{\,t<12800}\,\lambda_{prior} * \sum_{r\,\epsilon\,(x,y,w,h)} \left(prior_k^r - b_{ijk}^r\right)^2 \\
& + 1_k^{truth}\Big(\,\lambda_{coord} * \sum_{r\,\epsilon\,(x,y,w,h)} \left(truth^r - b_{ijk}^r\right)^2 \\
& \qquad + \lambda_{obj} * \left(IOU_{truth}^k - b_{ijk}^o\right)^2 \\
& \qquad + \lambda_{class} * \Big(\sum_{c=1}^{C} \left(truth^c - b_{ijk}^c\right)^2\Big)\Big)
\end{aligned}
$$

我们来一点点解释，首先 W,H 分别指的是特征图（ 13 X 13 ）的宽与高，而 A 指的是先验框数目（这里是 5），各个 λ 值是各个 loss 部分的权重系数。第一项 loss 是计算 background 的置信度误差，但是哪些预测框来预测背景呢，需要先计算各个预测框和所有 ground truth 的 IOU 值，并且取最大值 Max_IOU，如果该值小于一定的阈值（YOLOv2 使用的是 0.6），那么这个预测框就标记为 background，需要计算 noobj 的置信度误差。第二项是计算先验框与预测宽的坐标误差，但是只在前 12800 个 iterations 间计算，我觉得这项应该是在训练前期使预测框快速学习到先验框的形状。第三大项计算与某个 ground truth 匹配的预测框各部分 loss 值，包括坐标误差、置信度误差以及分类误差。先说一下匹配原则，对于某个 ground truth，首先要确定其中心点要落在哪个 cell 上，然后计算这个 cell 的 5 个先验框与 ground truth 的 IOU 值（YOLOv2 中 bias_match=1），计算 IOU 值时不考虑坐标，只考虑形状，所以先将先验框与 ground truth 的中心点都偏移到同一位置（原点），然后计算出对应的 IOU 值，IOU 值最大的那个先验框与 ground truth 匹配，对应的预测框用来预测这个 ground truth。在计算 obj 置信度时，target=1，但与 YOLOv1 一样而增加了一个控制参数 rescore，当其为 1 时，target 取预测框与 ground truth 的真实 IOU 值（cfg 文件中默认采用这种方式）。对于那些没有与 ground truth 匹配的先验框（与预测框对应），除去那些 Max_IOU 低于阈值的，其它的就全部忽略，不计算任何误差。这点在 YOLOv3 论文中也有相关说明：YOLO 中一个 ground truth 只会与一个先验框匹配（IOU 值最好的），对于那些 IOU 值超过一定阈值的先验框，其预测结果就忽略了。这和 SSD 与 RPN 网络的处理方式有很大不同，因为它们可以将一个 ground

truth 分配给多个先验框。尽管 YOLOv2 和 YOLOv1 计算 loss 处理上有不同，但都是采用均方差来计算 loss。另外需要注意的一点是，在计算 boxes 的 w 和 h 误差时，YOLOv1 中采用的是平方根以降低 boxes 的大小对误差的影响，而 YOLOv2 是直接计算，但是根据 ground truth 的大小对权重系数进行修正：l.coord_scale * (2 - truth.w*truth.h)（这里 w 和 h 都归一化到(0,1)），这样对于尺度较小的 boxes 其权重系数会更大一些，可以放大误差，起到和 YOLOv1 计算平方根相似的效果（参考 YOLO v2 损失函数源码分析）。

```
// box 误差函数，计算梯度
float delta_region_box(box truth, float *x, float *biases, int n,
int index, int i, int j, int w, int h, float *delta, float scale,
int stride)
{
    box pred = get_region_box(x, biases, n, index, i, j, w, h,
stride);
    float iou = box_iou(pred, truth);

    // 计算ground truth 的offsets 值
    float tx = (truth.x*w - i);
    float ty = (truth.y*h - j);
    float tw = log(truth.w*w / biases[2*n]);
    float th = log(truth.h*h / biases[2*n + 1]);

    delta[index + 0*stride] = scale * (tx - x[index + 0*stride]);
    delta[index + 1*stride] = scale * (ty - x[index + 1*stride]);
    delta[index + 2*stride] = scale * (tw - x[index + 2*stride]);
    delta[index + 3*stride] = scale * (th - x[index + 3*stride]);
    return iou;
}
```

最终的 YOLOv2 模型在速度上比 YOLOv1 还快（采用了计算量更少的 Darknet-19 模型），而且模型的准确度比 YOLOv1 有显著提升，详情见 paper。

YOLOv2 在 TensorFlow 上实现

这里参考 YOLOv2 在 Keras 上的复现（见 yhcc/yolo2）,使用 TensorFlow 实现 YOLOv2 在 COCO 数据集上的 test 过程。首先是定义 YOLOv2 的主体网络结构 Darknet-19：

```
def darknet(images, n_last_channels=425):
    """Darknet19 for YOLOv2"""
    net = conv2d(images, 32, 3, 1, name="conv1")
    net = maxpool(net, name="pool1")
    net = conv2d(net, 64, 3, 1, name="conv2")
    net = maxpool(net, name="pool2")
    net = conv2d(net, 128, 3, 1, name="conv3_1")
```

```
    net = conv2d(net, 64, 1, name="conv3_2")
    net = conv2d(net, 128, 3, 1, name="conv3_3")
    net = maxpool(net, name="pool3")
    net = conv2d(net, 256, 3, 1, name="conv4_1")
    net = conv2d(net, 128, 1, name="conv4_2")
    net = conv2d(net, 256, 3, 1, name="conv4_3")
    net = maxpool(net, name="pool4")
    net = conv2d(net, 512, 3, 1, name="conv5_1")
    net = conv2d(net, 256, 1, name="conv5_2")
    net = conv2d(net, 512, 3, 1, name="conv5_3")
    net = conv2d(net, 256, 1, name="conv5_4")
    net = conv2d(net, 512, 3, 1, name="conv5_5")
    shortcut = net
    net = maxpool(net, name="pool5")
    net = conv2d(net, 1024, 3, 1, name="conv6_1")
    net = conv2d(net, 512, 1, name="conv6_2")
    net = conv2d(net, 1024, 3, 1, name="conv6_3")
    net = conv2d(net, 512, 1, name="conv6_4")
    net = conv2d(net, 1024, 3, 1, name="conv6_5")
    # ---------
    net = conv2d(net, 1024, 3, 1, name="conv7_1")
    net = conv2d(net, 1024, 3, 1, name="conv7_2")
    # shortcut
    shortcut = conv2d(shortcut, 64, 1, name="conv_shortcut")
    shortcut = reorg(shortcut, 2)
    net = tf.concat([shortcut, net], axis=-1)
    net = conv2d(net, 1024, 3, 1, name="conv8")
    # detection layer
    net = conv2d(net, n_last_channels, 1, batch_normalize=0,
                 activation=None, use_bias=True, name="conv_dec")
    return net
```

然后实现对 Darknet-19 模型输出的解码：

```
def decode(detection_feat, feat_sizes=(13, 13), num_classes=80,
           anchors=None):
    """decode from the detection feature"""
    H, W = feat_sizes
    num_anchors = len(anchors)
    detetion_results = tf.reshape(detection_feat, [-1, H * W,
num_anchors,
                                       num_classes + 5])

    bbox_xy = tf.nn.sigmoid(detetion_results[:, :, :, 0:2])
    bbox_wh = tf.exp(detetion_results[:, :, :, 2:4])
    obj_probs = tf.nn.sigmoid(detetion_results[:, :, :, 4])
```

```
    class_probs = tf.nn.softmax(detetion_results[:, :, :, 5:])

    anchors = tf.constant(anchors, dtype=tf.float32)

    height_ind = tf.range(H, dtype=tf.float32)
    width_ind = tf.range(W, dtype=tf.float32)
    x_offset, y_offset = tf.meshgrid(height_ind, width_ind)
    x_offset = tf.reshape(x_offset, [1, -1, 1])
    y_offset = tf.reshape(y_offset, [1, -1, 1])

    # decode
    bbox_x = (bbox_xy[:, :, :, 0] + x_offset) / W
    bbox_y = (bbox_xy[:, :, :, 1] + y_offset) / H
    bbox_w = bbox_wh[:, :, :, 0] * anchors[:, 0] / W * 0.5
    bbox_h = bbox_wh[:, :, :, 1] * anchors[:, 1] / H * 0.5

    bboxes = tf.stack([bbox_x - bbox_w, bbox_y - bbox_h,
                       bbox_x + bbox_w, bbox_y + bbox_h], axis=3)

    return bboxes, obj_probs, class_probs
```

我将 YOLOv2 的官方训练权重文件转换了 TensorFlow 的 checkpoint 文件（下载链接），具体的测试 demo 都放在我的 GitHub 上了，感兴趣的可以去下载测试一下，至于 train 的实现就自己折腾吧，相对会棘手点。

图 11：YOLOv2 在自然图片上的测试

YOLO9000

YOLO9000 是在 YOLOv2 的基础上提出的一种可以检测超过 9000 个类别的模型，其主要贡献点在于提出了一种分类和检测的联合训练策略。众多周知，检测数据集的标注要比分类数据集打标签繁琐的多，所以 ImageNet 分类数据集比 VOC 等检测数据集高出几个数量级。在 YOLO 中，边界框的预测其实并不依赖于物体的标签，所以 YOLO 可以实现在分类和检测数据集上的联合训练。对于检测数据集，可以用来学习预测物体的边界框、置信度以及为物体分类，而对于分类数据集可以仅用来学习分类，但是其可以大大扩充模型所能检测的物体种类。作者选择在 COCO 和 ImageNet 数据集上进行联合训练，但是遇到的第一问题是两者的类别并不是完全互斥的，比如"Norfolk terrier"明显属于"dog"，所以作者提出了一种层级分类方法（Hierarchical classification），主要思路是根据各个类别之间的从属关系（根据 WordNet）建立一种树结构 WordTree，结合 COCO 和 ImageNet 建立的 WordTree 如下图所示：

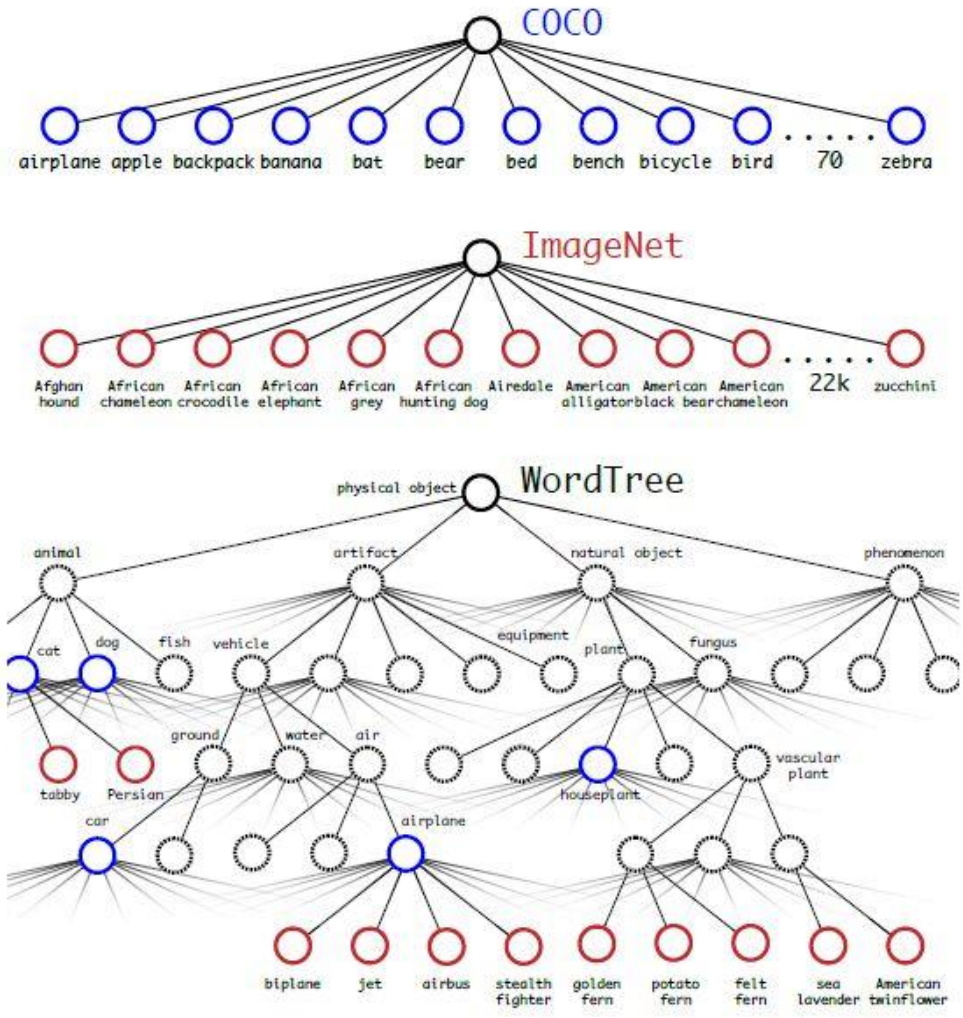

图 12：基于 COCO 和 ImageNet 数据集建立的 WordTree

WordTree 中的根节点为"physical object"，每个节点的子节点都属于同一子类，可以对它们进行 softmax 处理。在给出某个类别的预测概率时，需要找到其所在的位置，遍历这个 path，然后计算 path 上各个节点的概率之积。

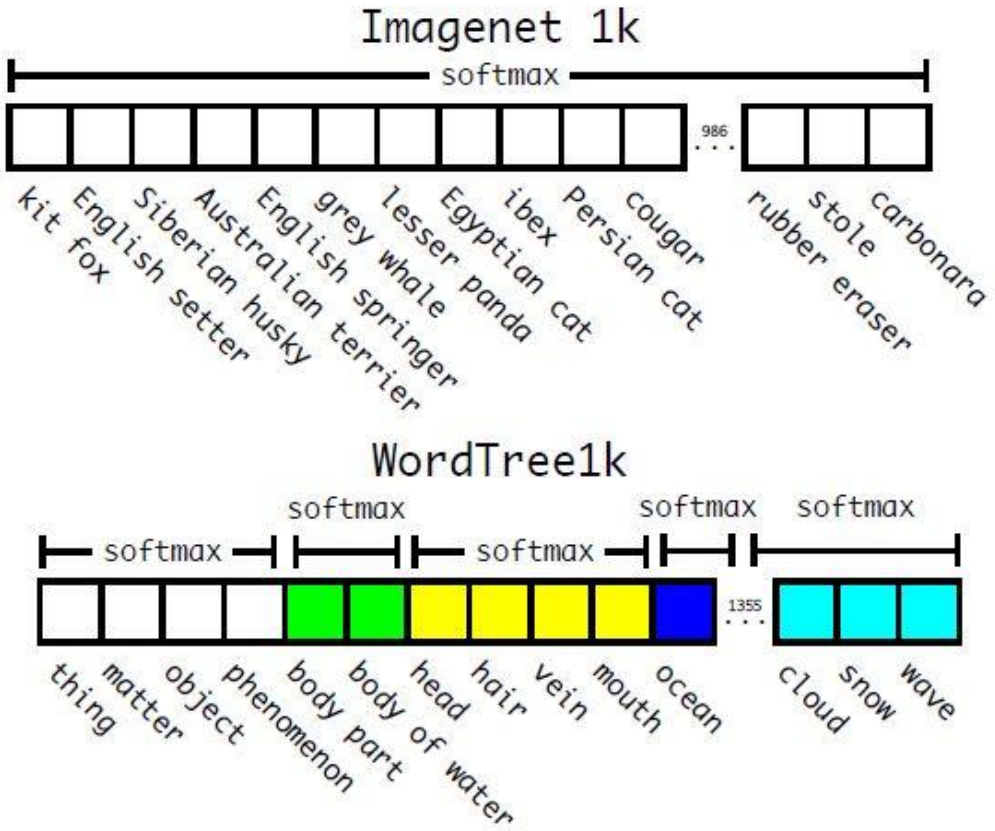

图 13：ImageNet 与 WordTree 预测的对比

在训练时，如果是检测样本，按照YOLOv2的loss计算误差，而对于分类样本，只计算分类误差。在预测时，YOLOv2给出的置信度就是 $Pr(physical\ object)$ ，同时会给出边界框位置以及一个树状概率图。在这个概率图中找到概率最高的路径，当达到某一个阈值时停止，就用当前节点表示预测的类别。

通过联合训练策略，YOLO9000可以快速检测出超过9000个类别的物体，总体mAP值为19.7%。我觉得这是作者在这篇论文作出的最大的贡献，因为YOLOv2的改进策略亮点并不是很突出，但是YOLO9000算是开创之举。

YOLOv3

近期，YOLOv3 发布了，但是正如作者所说，这仅仅是他们近一年的一个工作报告（TECH REPORT），不算是一个完整的 paper，因为他们实际上是把其它论文的一些工作在 YOLO 上尝试了一下。相比 YOLOv2，我觉得 YOLOv3 最大的变化包括两点：使用残差模型和采用 FPN 架构。YOLOv3 的特征提取器是一个残差模型，因为包含 53 个卷积层，所以称为 Darknet-53，从网络结构上看，相比 Darknet-19 网络使用了残差单元，所以可以构建得更深。另外一个点是采用 FPN 架构（Feature Pyramid Networks for Object Detection）来实现多尺度检测。YOLOv3 采用了 3 个尺度的特征图（当输入为 416 x 416 时）：(13 x 13)，(26 x 26)，(52 x 52)，VOC 数据集上的 YOLOv3 网络结构如图 15 所示，其中红色部分为各个尺度特征图的检测结果。YOLOv3 每个位置使用 3 个先验框，所以使用 k-means 得到 9 个先验框，并将其划分到 3 个尺度特征图上，尺度更大的特征图使用更小的先验框，和 SSD 类似。

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

图 14：YOLOv3 所用的 Darknet-53 模型

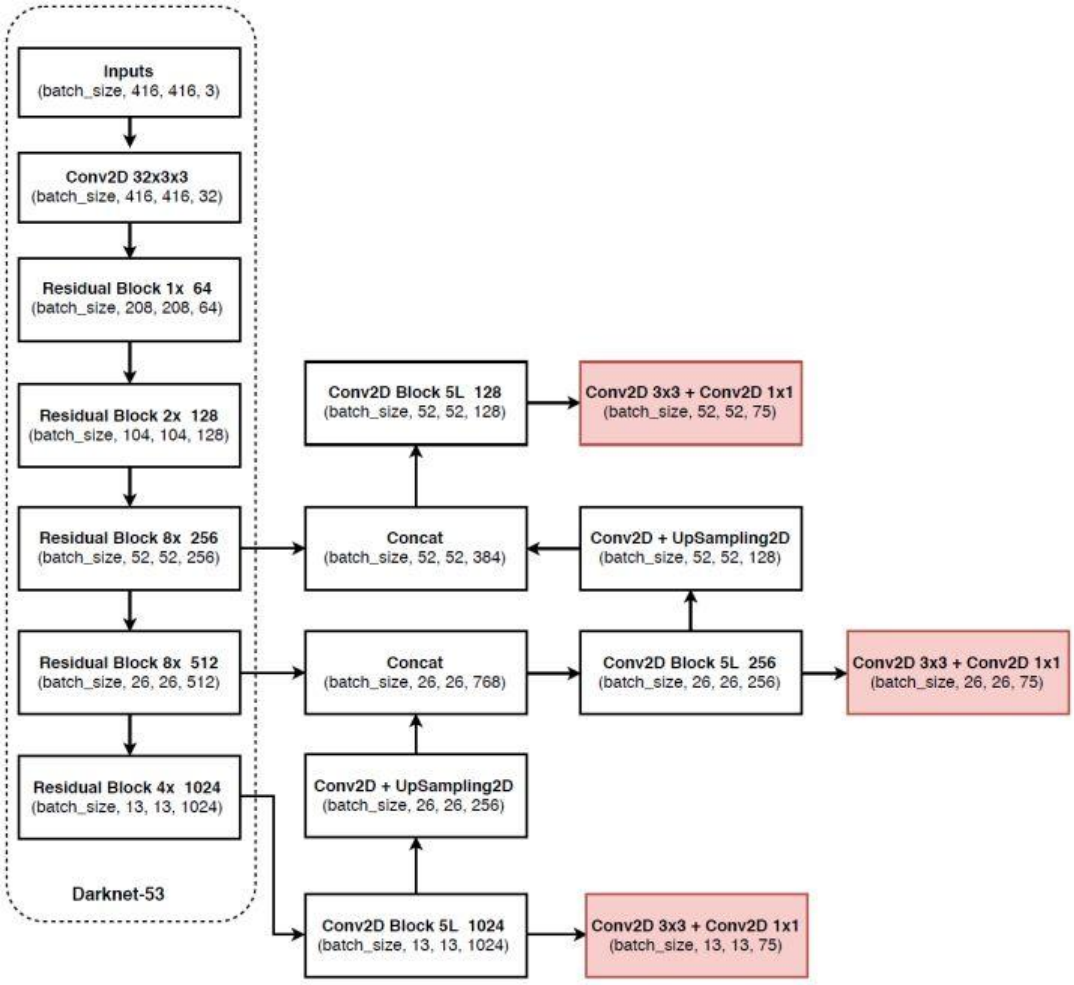

图 15 YOLOv3 网络结构示意图（VOC 数据集）

YOLOv3 与其它检测模型的对比如下图所示，可以看到在速度上 YOLOv3 完胜其它方法，虽然 AP 值并不是最好的（如果比较 AP-0.5，YOLOv3 优势更明显）。

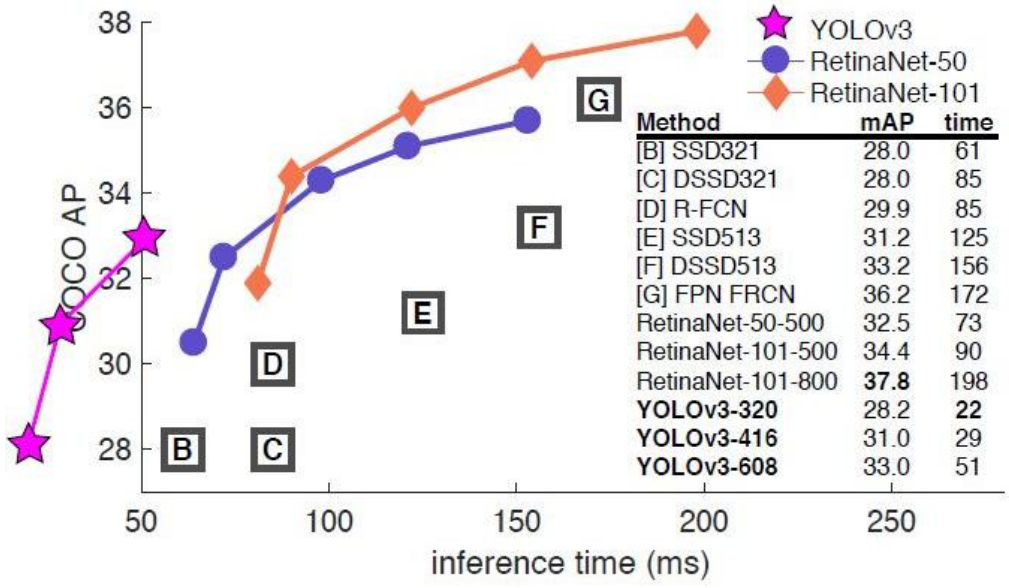

图 16：YOLOv3 在 COCO 测试集与其它检测算法对比图

小结

从 YOLO 的三代变革中可以看到，在目标检测领域比较好的策略包含：设置先验框，采用全卷积做预测，采用残差网络，采用多尺度特征图做预测。期待未来有更好的策略出现。本人水平有限，文中难免出现错误，欢迎指正！

参考

1. Darknet 官网.
2. thtrieu/darkflow.
3. You Only Look Once: Unified, Real-Time Object Detection.
4. YOLO9000: Better, Faster, Stronger.
5. YOLOv3: An Incremental Improvement.
6. yhcc/yolo2.
7. pytorch-yolo2.
8. Training Object Detection (YOLOv2) from scratch using Cyclic Learning Rates.
9. allanzelener/YAD2K.
10. marvis/pytorch-yolo3.