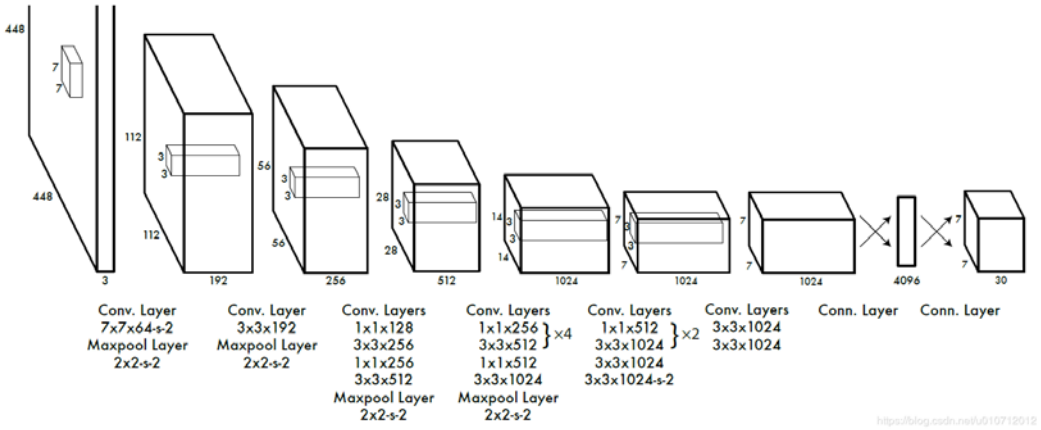
目标检测 YOLOv2 详细解释

本文链接：https://blog.csdn.net/u010712012/article/details/85274711



一、Yolo-v1 问题

1. 7 * 7 的卷积层，AlexNet 和 ZFNet 就是 7 * 7 的，大卷积核的好处是有更大的感受野，能将图像中更多的像素点作一个统计分析，但是当网络结构变深的时候，图像的感受野会在卷积过程中不断变大，所以早层用大的卷积核没有必要，VGG，GoogLeNet 用的 3 * 3 的 filter，7 * 7 比 3 * 3 有更大的计算量，同时 7 * 7 的时候用了 stride=2，说明经过 stride 之后的 feature map 的尺寸缩小为原来的一半，紧接着 7 * 7 的 max pooling 层，stride=2，说明经过一层后 feature map 变为原来的 1/4，在原图像中，物体在图像中的位置与这个物体的 feature 在 feature map 这个的位置是一一对应的，但是在图像中过早地下采样(只要是 stride 就会对图像下采样)，这种过分下采样会损坏物体在图像中的位置信息，所以设计网络的时候一般把下采样 maxpooling 往后延，在后面那几层做下采样，这样对整个物体的位置信息损害会少点。

2. 检测的问题，用的全连接 fc 层，这样完全把物体的定位信息给打破了，7*7 可以映射图像的像素区域，经过全连接层这样的映射关系完全就没有了，所以 Yolo-v1 是希望用 1 位 fc 层来分析出物体的位置信息，但实际上很难。

二、yolo-v2 改进

1. Batch Normalization

Yolov2—Backbone（网络骨架，如 VGG或自己定义的）,由原来的 24 层，减少为原来的 18 层，第一层conv-filter 为3*3，第一层的 stride 设置为1，不会下采样，在每一层后面都加上 batch-norm 层，就有了更快的收敛速度，初始化的时候相对 v1 更加的简单，BN 有一部分正则化的效果，分类的正确率会有所提升。

2. High Resolution Classifier

目前的目标检测方法中，基本上都会使用 ImageNet 预训练过的模型（classifier）来提取特征，如果用的是 AlexNet 网络，那么输入图片会被 resize 到不足 256 * 256，导致分辨率不够高，给检测带来困难。为此，新的 YOLO 网络把分辨率直接提升到了 448 * 448，这也意味之原有的网络模型必须进行某种调整以适应新的分辨率输入。

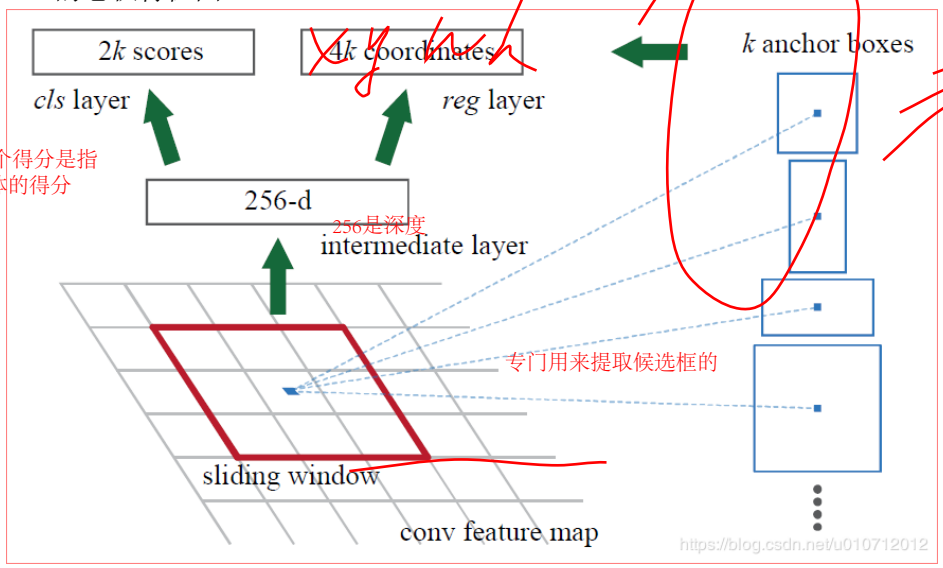
总网络：预训练网络+检测网络

对于 YOLO-v2，作者首先对分类网络（自定义的 darknet）进行了 fine-tune，分辨率改成 448 * 448，在 ImageNet 数据集上训练10epochs，训练后的网络就可以适应高分辨率的输入了。然后，作者对检测网络部分（也就是后半部分）也进行fine-tune。这样通过提升输入的分辨率， mAP 获得了 4%的提升。

3. Convolutional With Anchor Box

v2 的Convolutional With Anchor Boxes 继承了Faster-R-CNN 的RPN 层的Anchor。对于faster-R-CNN 来说每一次预测，RPN 都是基于一个Anchor Box 预测结果是在Anchor Box 基础上做一个差值，anchor box的宽高加上预测的差值，最终能得到检测结果，继承了 faster-r-cnn 的 9 个anchor，分 3 组，有相同的面积，内部 3 个anchor 有不同的宽高比。具体怎么操作呢？首先，作者去掉了后面的一个池化层以确保输出的卷积特征图有更高的分辨率。然后，通过缩减网络，让图片输入分辨率为416 * 416，这一步的目的是为了让后面产生的卷积特征图宽高都为奇数，这样就可以产生一个center cell。作者观察到，大物体通常占据了图像的中间位置， 就可以只用中心的一个cell来预测这些物体的位置，否则就要用中间的4个cell来进行预测，这个技巧可稍稍提升效率。最后，YOLOv2使用了卷积层降采样（factor为32），使得输入卷积网络的416 * 416图片最终得到13 * 13的卷积特征图（416/32=13）。

也是一种坐标转换



每个anchor的2个得分是指背景和前景物体的得分

256是深度

专门用来提取候选框的

为了引入anchor boxes来预测bounding boxes，v2 的检测网络中把所有全连接层都去掉了，后面只跟上卷积层，对图像的下采样也往后移了，使得前面的网络能充分利用定位信息。YOLO-v2 通过引入anchor boxes，预测的box数量超过了1千（以输出feature map大小为13*13为例，每个grid cell有9个anchor box的话，一共就是13*13*9=1521个，当然由后面第4点可知，最终每个grid cell选择5个anchor box）。顺便提一下，在Faster-R-CNN在输入大小为1000*600时的boxes数量大概是6000，在SSD300中boxes数量是8732。显然增加box数量是为了提高object的定位准确率。v2 使用了卷积层降采样(factor 为 32)，使得输入卷积网络 416*416 图片最终得到 13 * 13 的卷积特征图。加入 anchor box 后，可以预料是召回率上升，准确率下降。实际上准确率只有小幅度下降，但是召回率提升了 7%。

4. Dimension Clusters（维度聚类）

作者在使用anchor的时候遇到了两个问题，第一个是anchor boxes的宽高维度往往是精选的先验框（hand-picked priors），虽说在训练过程中网络也会学习调整boxes的宽高维度，最终得到准确的bounding boxes。但是，如果一开始就选择了更好的、更有代表性的先验boxes维度，那么网络就更容易学到准确的预测位置。和以前的精选boxes维度不同，作者使用了K-means聚类方法类训练bounding boxes，可以自动找到更好的boxes宽高维度。可以看到，平衡复杂度和IOU之后，最终得到k值为5，意味着作者选择了5种大小的box维度来进行定位预测，这与手动精选的box维度不同。结果中扁长的框较少，而瘦高的框更多（这符合行人的特征），这种结论如不通过聚类实验恐怕是发现不了的。当然，作者也做了实验来对比两种策略的优劣，如下图，使用聚类方法，仅仅5种boxes的召回率就和Faster-R-CNN的9种相当。说明K-means方法的引入使得生成的boxes更具有代表性，为后面的检测任务提供了便利。

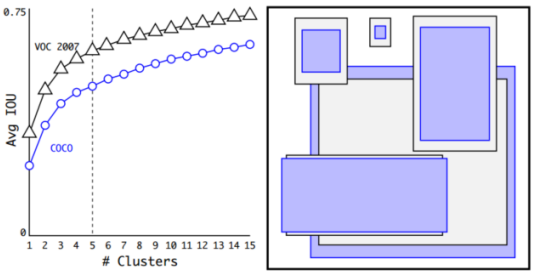


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. Both sets of priors favor thinner, taller boxes while COCO has greater variation in size than VOC.

Box Generation	#	Avg IOU
Cluster SSE	5	58.7
Cluster IOU	5	61.0
Anchor Boxes [15]	9	60.9
Cluster IOU	9	67.2

Table 1: Average IOU of boxes to closest priors on VOC 2007. The average IOU of objects on VOC 2007 to their closest, unmodified prior using different generation methods. Clustering gives much better results than using hand-picked priors.

通过聚类，使得一开始就选择了更好的，更具有代表性的先验 boxes 维度，那么网络就更容易学到准确的预测位置。

传统的 K-means 是用欧氏距离，也就意味着较大的 boxes 会比较小的 boxes 产生更多的 error。所以这里采用的评判标准是 IOU 得分（也就是 boxes 之间的交集除以并集），这样的话，error 就和 box 的尺度无关了，最终的距离函数为：

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

5. New Network: Darknet-19

v2 采用了一个新的特征提取器，包括19 个卷积层和5 个max-pooling 层，Darknet19 和VGG16模型设计原则是一致的，主要采用 3 * 3 卷积，采用 2 * 2 的 maxpooling 层之后，特征图维度降低2倍，而同时将特征图的 channels增加两倍。与 NIN 类似，Darknet19 最终采用 global avg pooling做预测，并且在3 * 3 卷积层之间使用1 * 1 卷积来压缩特征图channels 以降低模型计算量和参数。每个卷积层后面同样使用了BN层来加快收敛速度，降低模型过拟合。mAP没有显著提升，但是计算量少了。

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 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 × 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			

Table 6: Darknet-19.

6. Direct location prediction

前面讲到，YOLO-v2 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 (tx,ty)，先验框的尺度 (wa,ha)以及中心坐标 (xa,ya)（特征图每个位置的中心点）来计算：

实际就是一个-1~1的比例值

$$x=(tx\times wa)-xa \qquad y=(ty\times ha)-ya$$

可能偏移值较大，超过图像范围

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 tx=1（tx，ty 都是相对偏移值，属于 0-1 之间）时边界框将向右偏移先验框的一个宽度大小，而当 tx=−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，YOLO-v2 弃用了这种预测方式，而是沿用 YOLO-v1 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 - tx,ty,tw,th，可以按如下公式计算出边界框实际位置和大小：

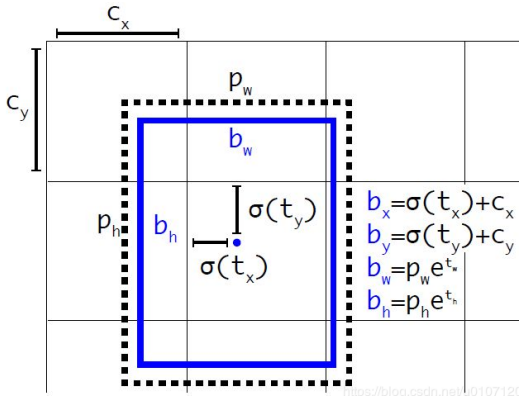
$$bx=\sigma(tx)+cx \qquad by=\sigma(ty)+cy \qquad bw=p_w e^{tw} \qquad bh=p_h e^{th}$$

其中 (cx,xy) 为 cell 的左上角坐标，如图 5 所示。

p_w和p_h在代码中是以

ANCHOR = [0.57273, 1.87446, 3.33843, 7.88282, 9.77052, 0.677385, 2.06253, 5.47434, 3.52778, 9.16828]

的形式给出



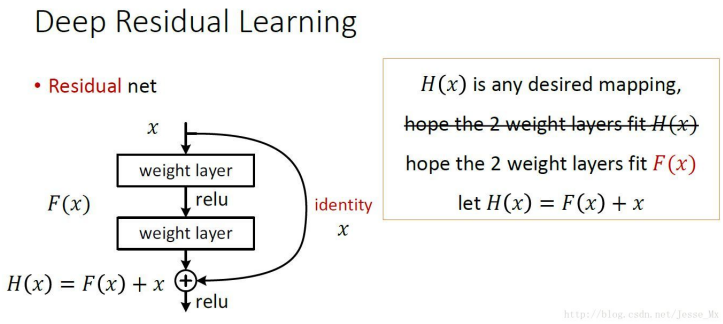
在计算时每个 cell 的尺度为 1，所以当前 cell 的左上角坐标为 (1,1) 。由于 sigmoid 函数的处理，边界框的中心位置会约束在当前 cell 内部，防止偏移过多。而 *pw* 和 *ph* 是先验框的宽度与长度，前面说过它们的值也是相对于特征图大小的，在特征图中每个 cell 的长和宽均为 1。这里记特征图的大小为 (W, H) （在文中是 (13, 13) ），这样我们可以将边界框相对于整张图片的位置和大小计算出来（4 个值均在 0 和 1 之间）：

$$bx=(\sigma(tx)+cx)/W \qquad by=(\sigma(ty)+cy)/H \qquad bw=p_w e^{tw}/W \qquad bh=p_h e^{th}/H$$

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 YOLO-v2 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，YOLO-v2 的 mAP 值提升了约 5%。

7. Fine-Grained Features（细粒度特征）

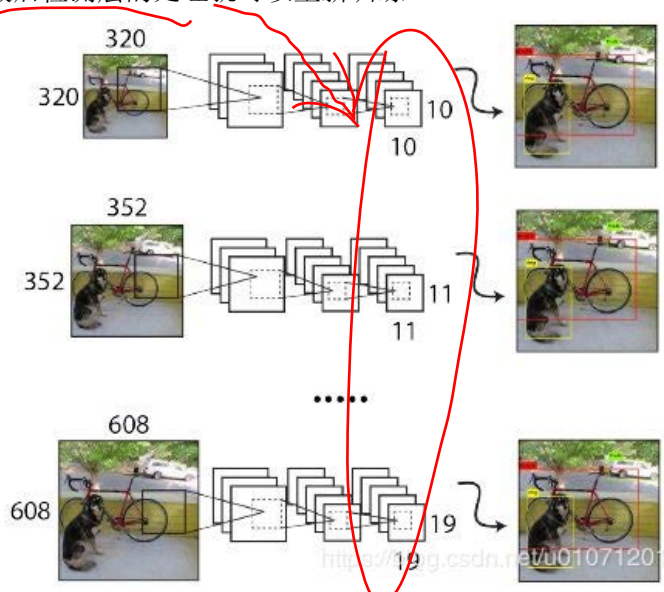
同时 Yolo-v2也参考了SSD 的思路，输出定位信息是利用了前面好几层的定位信息的，v2 也是如此，backbone 输入320*320，最后输出是 10 * 10，相当于 32 倍下采样，若按照 v1 的思想 10 * 10 会直接用来产生定位信息，而 v2 不止用了最后一层的 feature map 来产生定位信息，仿照SSD，v2用前面好几层卷积层产生的定位信息得到最后的结果，在 backbone 之后，加了两层卷积层，把第9层的输出拿出来(Route(-9))，(Route就是拿前面的层 Route(-9)往前推9层)，也就是第13 层卷积层，输出大小是20*20，对于这个需要和第 18 层的输出 10*10 的结果和在一起，但是他们分辨率不一样，所以这里有个 Reorg 层就是做这个事情，它会把 20*20 拆分成4个10*10的，所以来自13层的20*20会重新组合成10*10的feature map，和来自18层卷积层输出的feature map 拼在一起，最终形成一个10*10的feature map，contact 之后再用卷积层去卷积，最后第 23 层的卷积结果用来输出定位结果。像这种 Route 的操作我们称为 pass-through 层，好处是：13 层的20*20,14层的10*10，每做一次下采样定位信息都会被损害一次，若最后输出的结果能利用不同层之间的 feature map 的话，会对物体的定位结果有增强效果。



8. Multi-Scale Training

由于 YOLO-v2 模型中只有卷积层和池化层，所以 YOLO-v2 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，YOLO-v2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,...,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

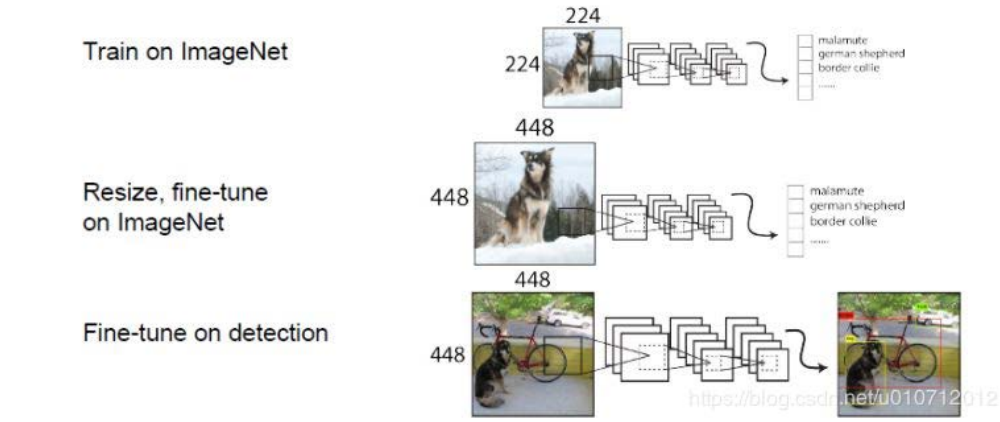
，对应的特征图大小为19×19 。在训练过程，每隔 10 个 iterations 随机选择一种输入图片大小，然后只需要修改对最后检测层的处理就可以重新训练。

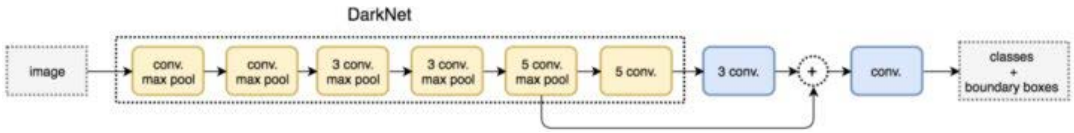


总结来看，虽然YOLO-v2做了很多改进，但是大部分都是借鉴其它论文的一些技巧，如 Faster R-CNN 的 anchor boxes，YOLO-v2 采用 anchor boxes 和卷积做预测，这基本上与 SSD 模型（单尺度特征图的 SSD）非常类似了，而且 SSD 也是借鉴了 Faster R-CNN 的 RPN 网络。从某种意义上来说，YOLO-v2 和 SSD 这两个 one-stage 模型与 RPN 网络本质上无异，只不过 RPN 不做类别的预测，只是简单地区分物体与背景。在two-stage方法中，RPN起到的作用是给出 region proposals，其实就是作出粗糙的检测，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。YOLO-v2 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。

三、YOLO-v2 的训练

YOLO-v2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fine-tune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 fine-tune 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 avg pooling 层以及 softmax 层，并且新增了三个 3×3×2014 卷积层，同时增加了一个 pass-through层，最后使用1×1 卷积层输出预测结果，输出的channels 数为num anchors×(5+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 (batchsize, 13, 13, 125)），可以先将其 reshape 为 (batchsize, 13, 13, 5, 25)，其中 T[:, :, :, :, 0:4] 为边界框的位置和大小 (tx,ty,tw,th)， T[:, :, :, :, 4] 为边界框的置信度，而 T[:, :, :, :, 5:] 为类别预测值。





YOLO-v2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YOLO-v2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YOLO-v2 到底是怎么训练起来的。不过默认按照YOLO-v1的处理方式也是可以处理，我看了YOLO在TensorFlow上的实现darkflow，发现它就是如此处理的：和 YOLO-v1一样，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。YOLO-v2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）。YOLO-v2 和YOLO-v1 的损失函数一样，为均方差函数。

但是我看了YOLO-v2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YOLO-v2的处理比原来的v1版本更加复杂。先给出 loss 计算公式：

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

我们来一点点解释，首先 W, H 分别指的是特征图（ 13×13 ）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*MaxIOU*，如果该值小于一定的阈值（YOLO-v2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（YOLO-v2 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 YOLO-v1一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YOLO-v3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 YOLO-v2 和 YOLO-v1 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，YOLO-v1 中采用的是平方根以降低boxes的大小对误差的影响，而YOLO-v2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 YOLO-v1 计算平方根相似的效果。

四、YOLO-v2 实验对比

最终的 YOLO-v2 模型在速度上比 YOLO-v1 还快（采用了计算量更少的 Darknet-19 模型），而且模型的准确度比 YOLO-v1 有显著提升，详情见 paper。

在小尺寸图片检测中，YOLO-v2成绩很好，输入为228 * 228的时候，帧率达到90FPS，mAP几乎和Faster R-CNN的水准相同。使得其在低性能GPU、高帧率视频、多路视频场景中更加适用。在大尺寸图片检测中，YOLO-v2达到了先进水平，VOC2007 上mAP为78.6%，仍然高于平均水准，下图是YOLO-v2和其他网络的成绩对比：

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

Table 3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

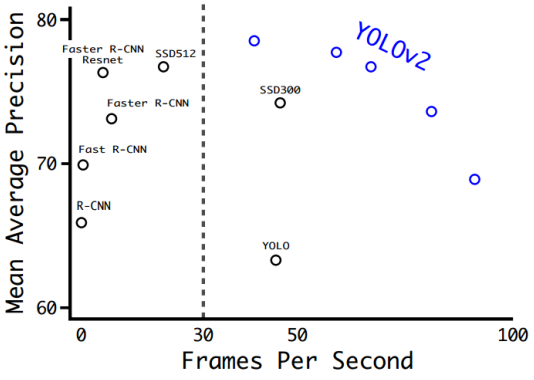


Figure 4: Accuracy and speed on VOC 2007.

作者在VOC2012上对YOLO-v2进行训练，下图是和其他方法的对比。YOLO-v2精度达到了73.4%，并且速度更快。同时YOLOV2也在COCO上做了测试（IOU=0.5），也和Faster R-CNN、SSD作了成绩对比。总的来说，比上不足，比下有余。

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [5]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster R-CNN [15]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [14]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300 [11]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [11]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [6]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

Table 4: PASCAL VOC2012 `test` detection results. YOLOv2 performs on par with state-of-the-art detectors like Faster R-CNN with ResNet and SSD512 and is 2 − 10× faster.

		0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Fast R-CNN [5]	train	19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast R-CNN[1]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Faster R-CNN[15]	trainval	21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [1]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster R-CNN[10]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
SSD300 [11]	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512 [11]	trainval35k	**26.8**	**46.5**	**27.8**	**9.0**	**28.9**	**41.9**	**24.8**	**37.5**	**39.8**	**14.0**	**43.5**	**59.0**
YOLOv2 [11]	trainval35k	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4

Table 5: Results on COCO `test-dev2015`. Table adapted from [11]

五、总结

YOLOv2速度的改进（Faster）

YOLO一向是速度和精度并重，作者为了改善检测速度，也作了一些相关工作。大多数检测网络有赖于VGG-16作为特征提取部分，VGG-16的确是一个强大而准确的分类网络，但是复杂度有些冗余。224 * 224的图片进行一次前向传播，其卷积层就需要多达306.9亿次浮点数运算。

YOLOv2使用的是基于Google-net的定制网络，比VGG-16更快，一次前向传播仅需85.2亿次运算。可是它的精度要略低于VGG-16，单张224 * 224取前五个预测概率的对比成绩为88%和90%（低一点点也是可以接受的）。

Darknet-19

YOLOv2使用了一个新的分类网络作为特征提取部分，参考了前人的先进经验，比如类似于VGG，作者使用了较多的3 * 3卷积核，在每一次池化操作后把通道数翻倍。借鉴了network in network的思想，网络使用了全局平均池化（global average pooling），把1 * 1的卷积核置于3 * 3的卷积核之间，用来压缩特征。也用了batch normalization（前面介绍过）稳定模型训练。最终得出的基础模型就是Darknet-19，其包含19个卷积层、5个最大值池化层（maxpooling layers），下图展示网络具体结构。Darknet-19运算次数为55.8亿次，imagenet图片分类top-1准确率72.9%，top-5准确率91.2%。

Training for classification

作者使用Darknet-19在标准1000类的ImageNet上训练了160次，用的随机梯度下降法，starting learning rate 为0.1，polynomial rate decay 为4，weight decay为0.0005 ，momentum 为0.9。训练的时候仍然使用了很多常见的数据扩充方法（data augmentation），包括random crops, rotations, and hue, saturation, and exposure shifts。（这些训练参数是基于darknet框架，和caffe不尽相同）。初始的224 * 224训练后，作者把分辨率上调到了448 * 448，然后又训练了10次，学习率调整到了0.001。高分辨率下训练的分类网络在top-1准确率76.5%，top-5准确率93.3%。

Training for detection

分类网络训练完后，就该训练检测网络了，作者去掉了原网络最后一个卷积层，转而增加了三个3 * 3 * 1024的卷积层（可参考darknet中cfg文件），并且在每一个上述卷积层后面跟一个1 * 1的卷积层，输出维度是检测所需的数量。对于VOC数据集，预测5种boxes大小，每个box包含5个坐标值和20个类别，所以总共是5 * （5+20）= 125个输出维度。同时也添加了转移层（passthrough layer），从最后那个3 * 3 * 512的卷积层连到倒数第二层，使模型有了细粒度特征。作者的检测模型以0.001的初始学习率训练了160次，在60次和90次的时候，学习率减为原来的十分之一。其他的方面，weight decay为0.0005，momentum为0.9，依然使用了类似于Faster-RCNN和SSD的数据扩充（data augmentation）策略。

YOLOv2分类的改进（Stronger）

这一部分，作者使用联合训练方法，结合wordtree等方法，使YOLOv2的检测种类扩充到了上千种。