### YOLO系列

**You Only Look Once**，YOLO之前的目标检测方法（主要是R-CNN系列）都分成两步，先通过regional proposal产生若干可能包含物体的边界框，再通过分类器判断是具体类别，以及回归器来精确定位物体的边界。而正如其名所示，YOLO的特点是通过一次卷积过程完成了整个定位和分类的过程。

YOLO的准确率（定位误差和召回率）比R-CNN系列低，优点是速度快（高一个数量级），以及假阳性更低。YOLO也是一个系列，包括V1、V2和V3。

github：

原作者redmon版本：<https://github.com/pjreddie/darknet>

Alexy工程化版本：<https://github.com/AlexeyAB/darknet>

第三方团队Pytorch复现版：<https://github.com/ultralytics/yolov3>

#### YOLOv1（201506）

相较于R-CNN系列，YOLO V1的优势在于：

* 计算速度更快（45FPS，能达到Faster R-CNN的若干倍）
* 假阳性（False Positive）更低
* 由于其从整体着眼，泛化能力更强，在艺术作品中有更好的效果

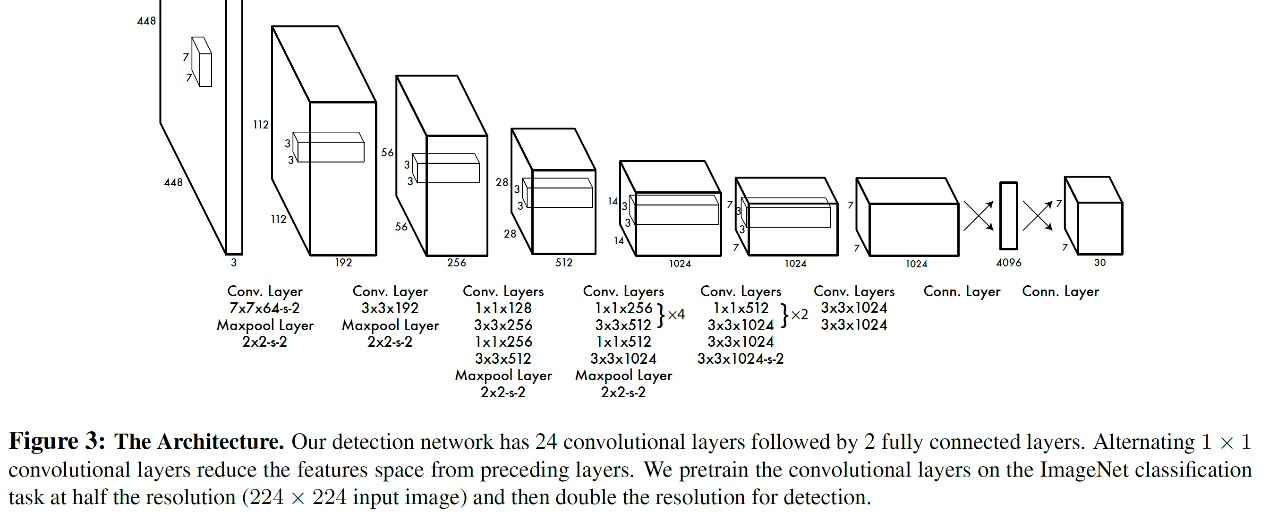
但其缺点也很明显：

* 准确率其实更低，且边界框更不准。
* 由于其原理限制，能预测的物体数量有限

YOLO将图像分成S\*S个格子，如果一个物体的中心落在某格子内，则该格子负责预测该物体，每个格子可以预测B个物体，每个物体用5个值用来描述：x,y,w,h,confidence，（x,y）是该物体的中心位置，w和h是物体的宽和高，confidence是该物体的置信度。此外每个格子有C个值来描述分类，每个值表示一个物体分类（注意是每个格子，不是每个物体）。

因此YOLO的输出张量为S\*S\*(B\*5+C)，通常S为7，B为2，C则为20（PASCAL VOC数据集），整个输出张量为7\*7\*30。

以下为YOLO的网络结构，24层卷积+2层全连接：



首先用前20层卷积层+1个平均池化层+1个全连接层，在ImageNet上进行分类训练，最后达到top-5有88%的正确率。

接着将后4层卷积层+2层全连接层接到之前训练好的20层卷积层之后，形成完整的网络，同时将输入分辨率从224\*224提升到448\*448。

因为在图片的大部分格子中没有物体，这部分权重占比太大，导致网络训练发散，为了避免这种情况，在训练中给有物体的格子和没有物体的格子的损失分配了不同的权重（分别为5和0.5）。

对于不同大小的物体，同样的宽高误差显然有不同的意义，更小的物体对宽高误差更敏感。为了应对这种情况，在损失函数中使用w和h的平方根而非本身来衡量误差。

以下是损失函数，其中：

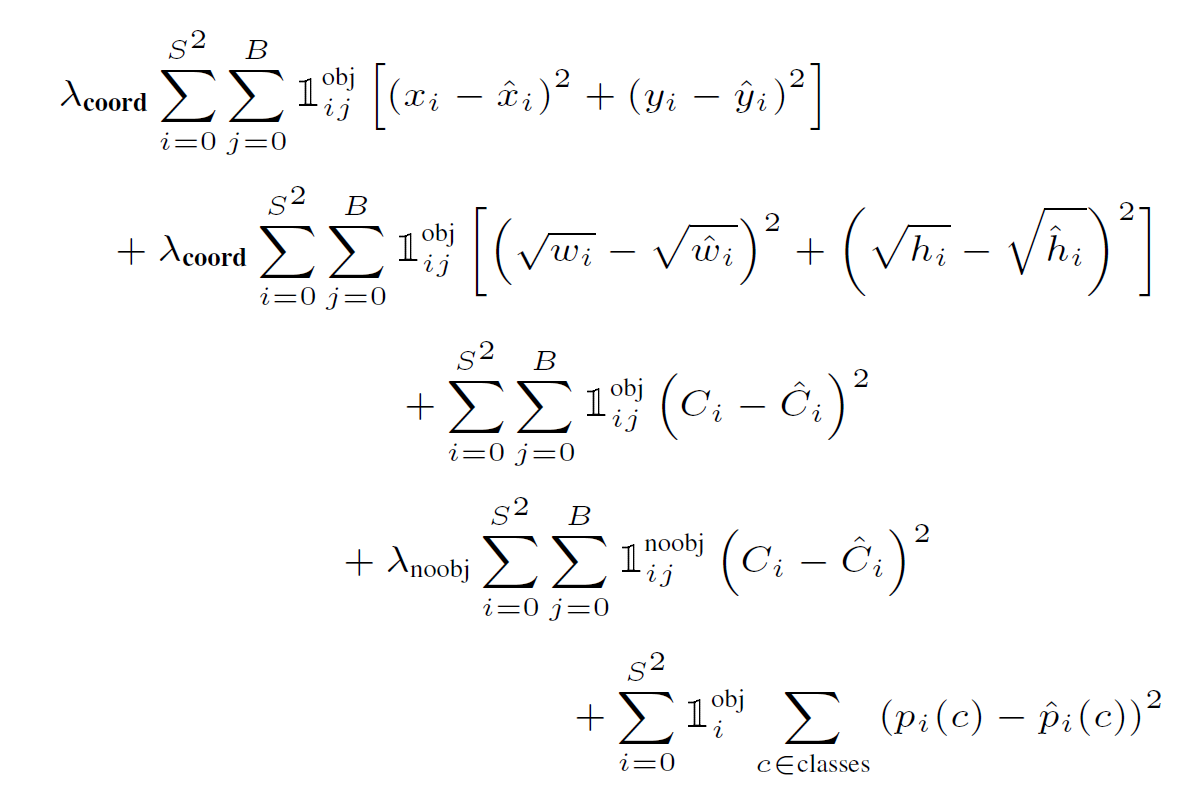
第一行是中心点坐标的误差（S\*S\*B\*2）

第二行是宽和高的误差（S\*S\*B\*2）

第三行是有物体的格子的置信度误差

第四行是没有物体的格子的置信度的误差

最后一行是类别的误差（S\*S\*B\*C）



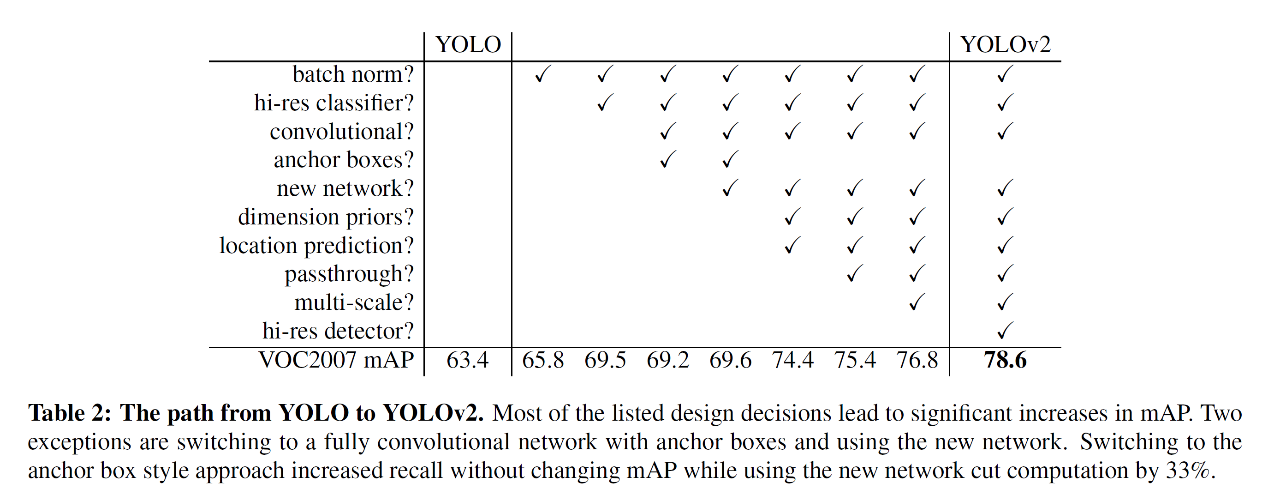
论文：

You Only Look Once: Unified, Real-Time Object Detection

<https://arxiv.org/pdf/1506.02640>

#### YOLOv2（201612）

YOLO V2的出现是为了解决YOLO V1的两个主要的问题：定位准确率较低，以及较低的识别率（TPR）。目的是在不增加网络复杂度的情况下提高这些值，为此文章中使用了一系列手段来实现，或者说，YOLO V2和YOLO V1的区别就是一系列细节，下表中显示了各种不同手段应用在YOLO上之后带来的mAP上的变化：



这些手段包括：

**Batch Normalization**

BN可以提高模型收敛速度，且可以降低模型的过拟合，YOLOv2中每个卷积层后面都增加了BN，且不再使用dropout。

**高分辨率分类器**

High-resolution classifier，每个目标检测网络都会使用经ImageNet预训练的分类器，YOLOv1在训练的时候先在224\*224的图片上训练分类器，再在448\*448的图片上训练目标检测，这意味着网络需要同时应用低分辨率到高分辨率，以及分类到检测的两个转变。对于YOLOv2，在这两者之间增加了10个epoch的448\*448的图片分类训练做fine tune，然后在转到448\*448的目标检测做fine-tune。

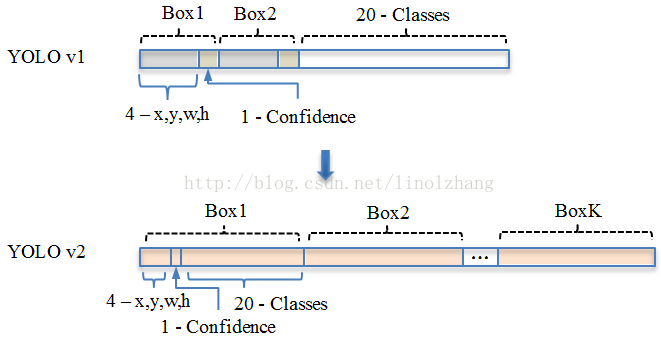
**带先验框的卷积**

锚点（anchor box）即先验框，这个概念是Faster R-CNN中提出的。YOLOv1直接预测每个边界框的坐标，而YOLOv2则借鉴了先验框的概念，预测相对于先验框的偏移值，这样可以更好的学习。并且通过设置不同长宽比的先验框，对于不同长宽比的物体可以有很高的预测结果（这也是YOLOv1 TPR低的一个原因）。

YOLOv2去除了v1中的全连接层，改用卷积层和先验框来预测边界框。为了使检测所用的特征图分辨率更高，移除了一个池化层。并且不使用448\*448而是416\*416作为输入，这使得全部下采样（总步长32）之后得到的特征图大小为13\*13，这样只有一个中心位置，便于预测中心点在中间的大物体。

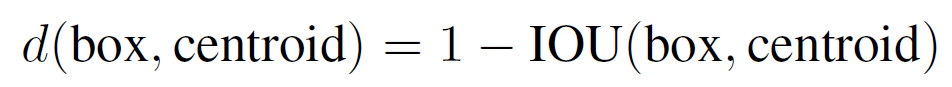
YOLOv1将图像分成7\*7的格子（也即其最终特征图的大小），每格可以有2个边界框（每个边界框有一个置信度来表示是否有物体，但这两个边界框共享一个分类预测），也就是说YOLOv1最多只能预测98个物体。

YOLOv2则大大提高了预测物体的数量，首先有13\*13个预测位置，并且每个位置有若干个先验框，每个先验框都有其对应的置信度（有无物体）和分类预测。与SSD不同，YOLO使用置信度来分辨有无物体，而SSD将背景作为一个单独的分类，因此SSD的分类预测数量是比实际的类别数量多1。

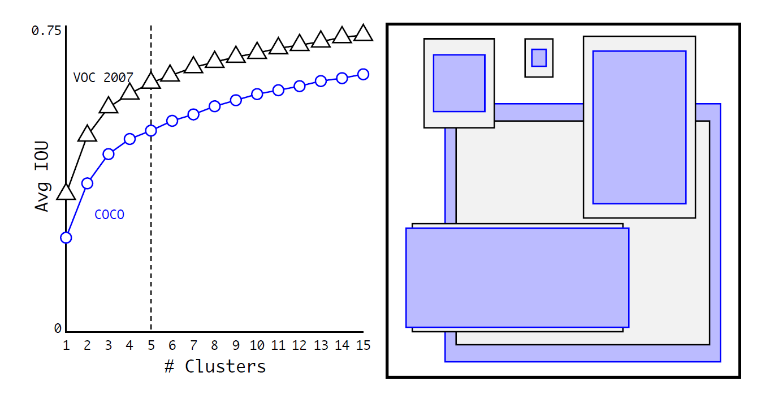


**维度聚类**

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：

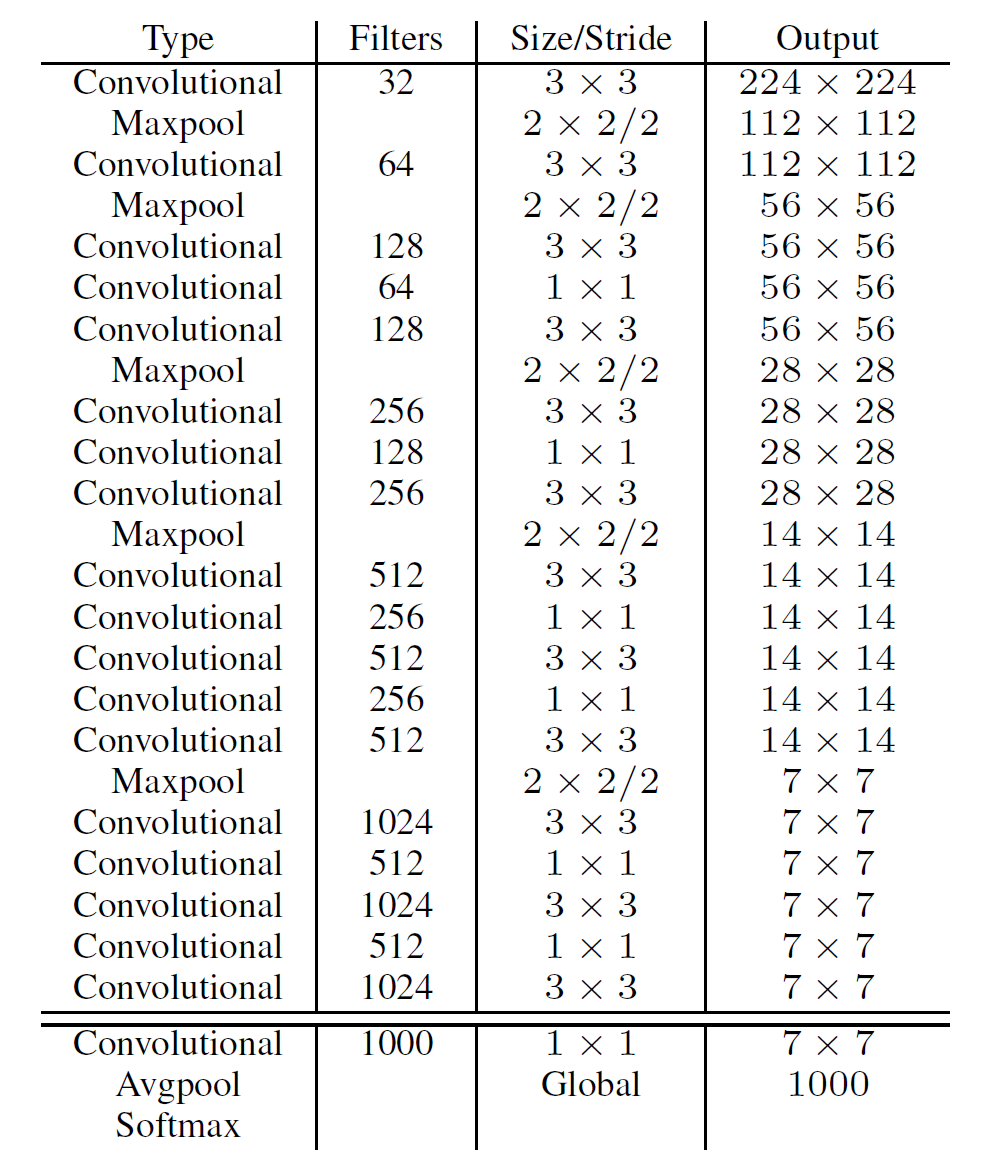


下图为在VOC和COCO数据集上的聚类分析结果，随着聚类中心数目的增加，平均IOU值（各个边界框与聚类中心的IOU的平均值）是增加的，但是综合考虑模型复杂度和召回率，作者最终选取5个聚类中心作为先验框，其相对于图片的大小如右侧图所示：



**新的网络**

YOLOv2采用了一个新的基础网络用作特征提取，被称为Darknet-19，包括19个卷积层和5个最大池化层：



Darknet-19与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\*3卷积之间使用1\*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%。

**直接位置预测**

Direct location prediction，直接Anchor Box回归导致模型不稳定，对应公式也可以参考 Faster-RCNN论文，该公式没有任何约束，中心点可能会出现在图像任何位置，这就有可能导致回归过程震荡，甚至无法收敛。

针对这个问题，作者在预测位置参数时采用了强约束方法：

       1）对应 Cell 距离左上角的边距为（Cx，Cy），σ定义为sigmoid激活函数，将函数值约束到［0，1］，用来预测相对于该Cell 中心的偏移（不会偏离cell）；

       2）预定Anchor（文中描述为bounding box prior）对应的宽高为（Pw，Ph），预测 Location 是相对于Anchor的宽高 乘以系数得到

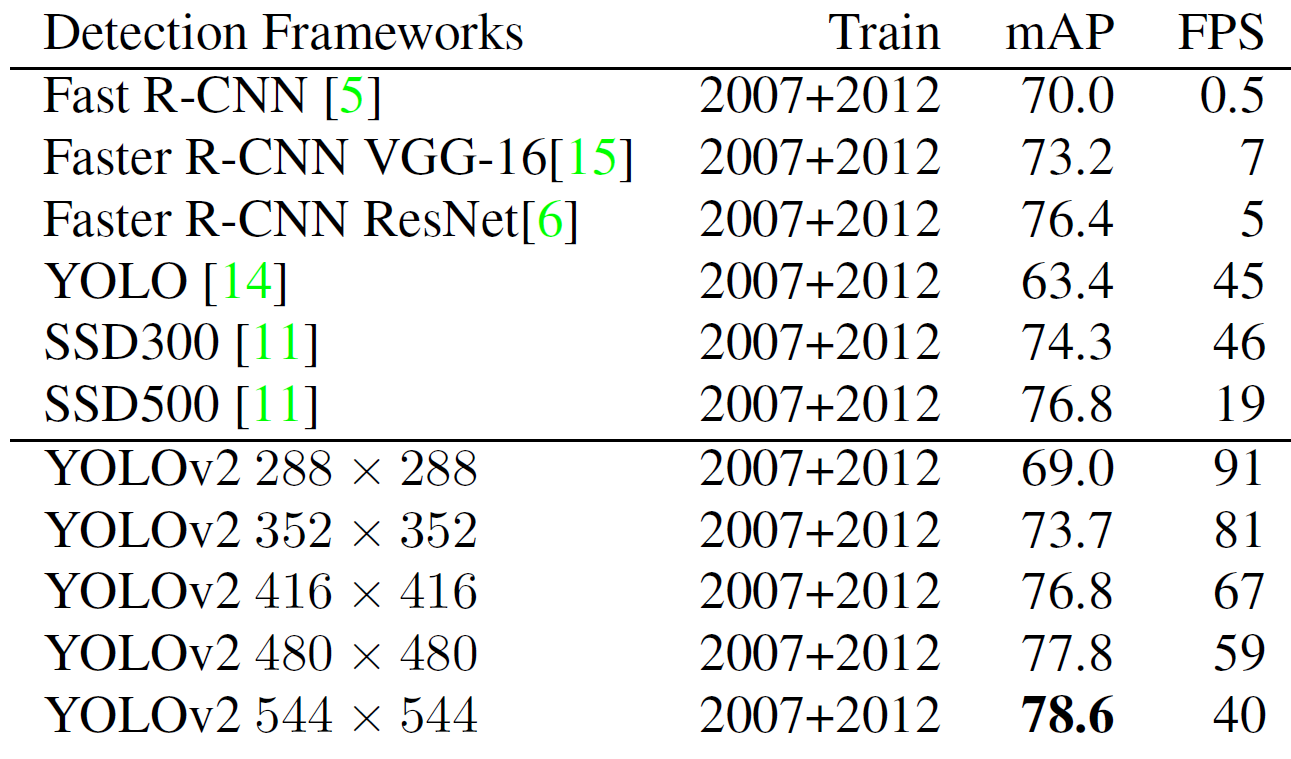
**细粒度特征**

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\*26\*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\*26\*512的特征图，经passthrough层处理之后就变成了13\*13\*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\*13\*1024特征图连接在一起形成13\*13\*3072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

**多尺度训练**

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倍数的值输入图片。最小为320\*320，此时对应的特征图大小为10\*10，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。

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\*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）。



论文：

YOLO9000: Better, Faster, Stronger

<https://arxiv.org/pdf/1612.08242>

参考文章:

<https://blog.csdn.net/shanlepu6038/article/details/84778770>

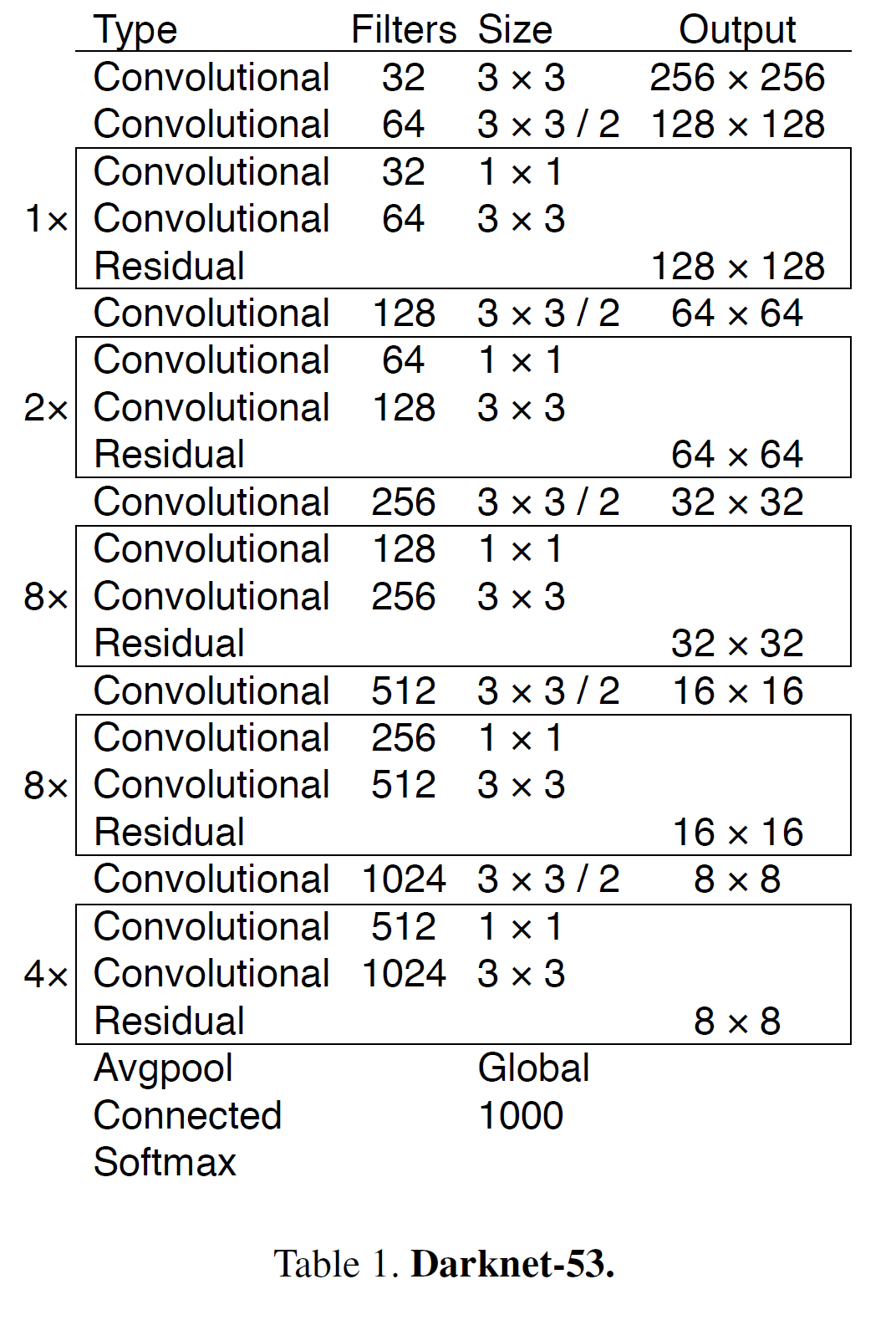
<https://blog.csdn.net/anqian123321/article/details/82627332>

#### YOLOv3（201804）

YOLOv3在YOLOv2的基础上做了一些改动，包括如下：

**新的基础网络Darknet-53**

相较于上一版本中的Darknet19，Darknet53通过加入残差结构，大大增加了网络深度。此外YOLOv3中是完全没有**池化层**和**全连接层**的，张量的尺寸变化是通过改变卷积核的步长来是实现的。



**多尺度检测**

Predictions across scales，YOLOv2中使用passthrough结构来进行检测细粒度特征，在v3中得到了进一步发扬光大，吸取了FPN的特点，YOLOv3输出3个不同尺度的特征图，分别是13\*13，26\*26，52\*52，

论文：

YOLOv3: An Incremental Improvement

<https://arxiv.org/pdf/1804.02767>

github：第三方团队在Pytorch上复现的YOLOv3

<https://github.com/ultralytics/yolov3>

#### YOLOv4（202004）

YOLOv3之后其作者Joseph Redmon宣布退出CV研究（因为他觉得自己的开源算法被用在军事上），于是YOLOv4的作者Alexey就“接手”了之后的开发。Alexey跟Redmon认识，而且之前就做了很多YOLO的工作，主要是一些工程化方面的东西，比如移植到Windows、fix bug、增加模型等等，Alexey的github库也是除官方之外最权威的。

YOLOv4中没有之前几个版本的YOLO的创新性的改进，更多的贡献是在工程化的方面的，在YOLOv3之上测试并应用了很多新出来的技术（数据增强、激活函数、Loss函数、dropout等等）。

论文：

<https://arxiv.org/pdf/2004.10934.pdf>

github：Alexey版本的各个YOLO

<https://github.com/AlexeyAB/darknet>

参考：

<https://www.zhihu.com/question/390194081>

<https://zhuanlan.zhihu.com/p/135899403>

#### YOLOv5（Ultralytics）

YOLOv5并不是一篇论文。

如果说YOLOv3是官方（Joseph Redmon），YOLOv4是半官方（AlexeyAB），那YOLOv5就是民间的了，其作者是Ultralytics，这家公司之前就一直在做YOLO上的各种工作，包括移植到Pytorch上，之前对YOLOv3的移植在github上也获得了各种移植版本中最高的star：<https://github.com/ultralytics/yolov3>

事实上v5这个称号也是它自封的，和YOLOv4一样，YOLOv5也是在YOLOv3之上加了各种tricks形成的，但是并没有发表论文，具体的tricks目前尚未看到介绍。

和YOLOv4最大的区别在于它是Pytorch实现的。

github：

<https://github.com/ultralytics/yolov5>

### Darknet

**Darknet**是**YOLO**系列原作者Redmon开发的一套框架，C语言的实现和接口。Darknet虽然通用性不够，但是工程化相当不错。

官网：<https://pjreddie.com/darknet/yolo/>

github：<https://github.com/pjreddie/darknet>

Darknet是C语言的，下载（clone）后通过make生成一个名为darknet的可执行文件，该文件是darknet框架的主文件，可以用于训练、执行等等一系列操作。

**注意**：Darknet是有歧义的，可以指代这个深度学习框架，也可以指代使用这个框架开发的一个网络模型，这个网络模型可以用于图像分类，同时也是YOLO的主干网络。

#### 使用说明

以下为Darknet主程序（用于训练、执行darknet格式的网络配置和权重）使用说明

* 下载及编译

git clone https://github.com/pjreddie/darknet

cd darknet

可以选择修改Makefile（GPU=1, CUDNN=1, OPENCV=1)

make

* 获取权重

cd cfg

wget https://pjreddie.com/media/files/yolov3.weights

wget https://pjreddie.com/media/files/yolov3-tiny.weights

* 图像分类

./darknet classifier predict cfg/imagenet1k.data cfg/darknet19.cfg cfg/darknet19.weights data/dog.jpg

or input image path on prompt:

./darknet classifier predict cfg/imagenet1k.data cfg/darknet19.cfg cfg/darknet19.weights

* 目标检测

./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg

相当于：

./darknet detector test cfg/coco.data cfg/yolov3.cfg yolov3.weights data/dog.jpg

手动输入图片路径：

./darknet detect cfg/yolov3.cfg yolov3.weights

指定阈值 (缺省为0.25):

./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg -thresh 0.1

使用yolo-tiny:

./darknet detect cfg/yolov3-tiny.cfg yolov3-tiny.weights data/dog.jpg

摄像头实时检测：

./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights

检测视频文件：

./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights <video file>

* 训练

在VOC上训练（需要设置好其他配置）：

./darknet detector train cfg/voc.data cfg/yolov3-voc.cfg darknet53.conv.74

在COCO上训练（需要设置好其他配置）：

./darknet detector train cfg/coco.data cfg/yolov3.cfg darknet53.conv.74

* 冻结部分网络（前N层，下例为15）

./darknet partial cfg/yolov3-tiny.cfg yolov3-tiny.weights yolov3-tiny.conv.15 15

* OpenCV测试

./darknet imtest data/eagle.jpg

#### 文件说明

* darknet文件

darknet的主体，编译生成的ELF执行文件

* .cfg文件

Darknet的.cfg文件用于描述**网络结构**及**超参数**，通常位于cfg目录下，其参数定义：

[net] 字段

[https://github.com/AlexeyAB/darknet/wiki/CFG-Parameters-in-the-%5Bnet%5D-section](https://github.com/AlexeyAB/darknet/wiki/CFG-Parameters-in-the-%5bnet%5d-section)

层字段

<https://github.com/AlexeyAB/darknet/wiki/CFG-Parameters-in-the-different-layers>

* .weights文件

.weights文件是权重数据，须与cfg文件描述的网络结构对应

* .data文件

.data文件用于配置目标检测器（Object Detection）或者图像分类器（Image classification），通常位于cfg目录下。

目标检测器的配置cfg/coco.data，定义如下：

classes= 20 #种类

train = /home/pjreddie/data/voc/train.txt #训练集图片列表

valid = /home/pjreddie/data/voc/2007\_test.txt #验证集图片列表

names = data/voc.names #各个分类的名字

backup = backup #存放训练中备份的权重文件的目录

* 数据格式

YOLO有自己的数据格式，训练集或者验证集的图片列表被放在某个txt文件里，这些文件被.data文件中的train、valid字段指定。训练集图片的label文件有如下格式：

1. label文件与图片文件同名，扩展名改为txt
2. label文件与图片文件同目录，或者图片文件在images目录,label文件在同级labels目录，形如aaa/bbb/images/pic1.jpg、aaa/bbb/labels/pic1.txt
3. label文件中每一行代表一个object，有5个值，分别为类别、中心x、中心y、宽度、高度，这5个值以空格隔开，后面四个值取值范围从0到1，为点或者长度占全宽/全高的比例。

* .names文件

.names文件通常位于data目录中，按行存放了每个分类的名字