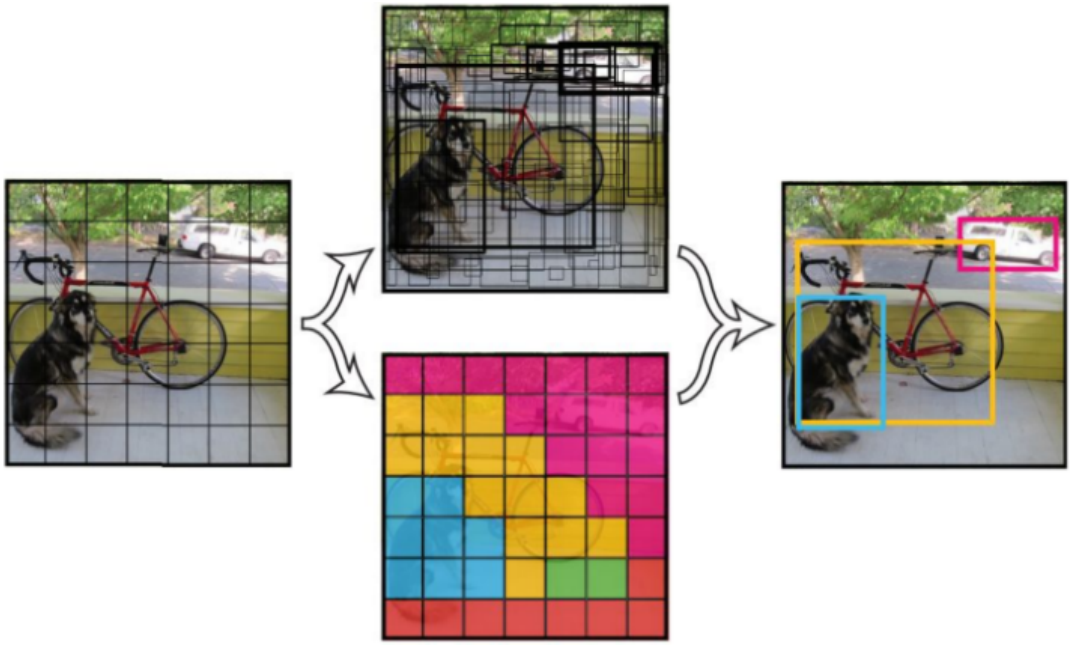


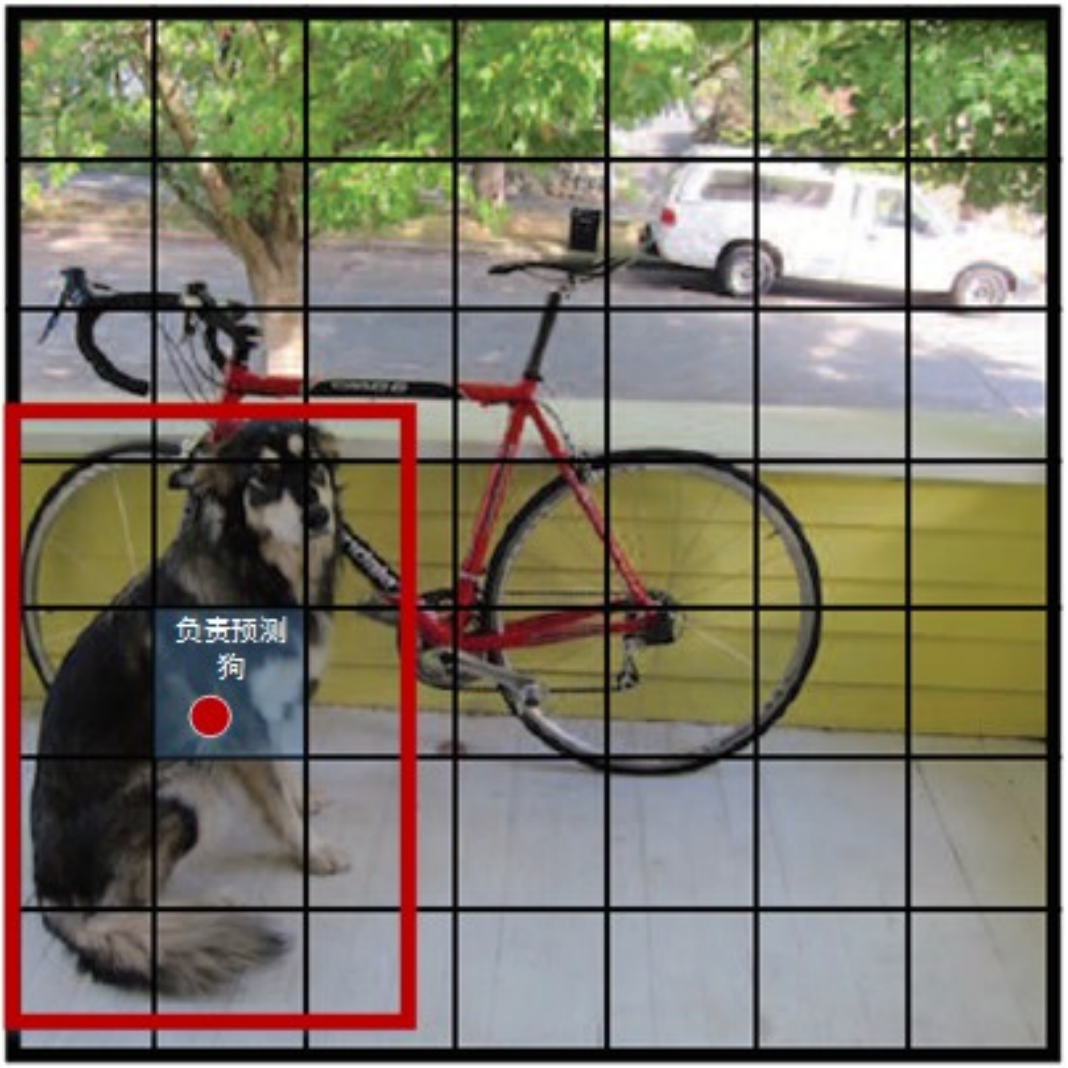
yolo-idea

本文逐步介绍YOLO v1~v3的设计历程。

YOLOv1基本思想

ppt：deepsystems.io 【绝对值得一看的ppt，一看就懂】

YOLO将输入图像分成SxS个格子，若某个物体 Ground truth 的中心位置的坐标落入到某个格子，那么这个格子就负责检测出这个物体。



yolo-grid-predict

每个格子预测B个bounding box及其置信度(confidence score)，以及C个类别概率。bbox信息(x,y,w,h)为物体的中心位置相对格子位置的偏移及宽度和高度,均被归一化.置信度反映是否包含物体以及包含物体情况下位置的准确性,定义为$Pr(Object)\times IOU_{truthpred}$,其中$Pr(Object)\in\{0,1\}$.

网络结构

YOLOv1网络借鉴了GoogLeNet分类网络结构。不同的是，YOLO未使用inception module，而是使用1x1卷积层（此处1x1卷积层的存在是为了跨通道信息整合）+3x3卷积层简单替代。

YOLOv1网络在最后使用全连接层进行类别输出，因此全连接层的输出维度是$S\times S\times (B\times 5+C)$。

YOLOv1网络比VGG16快(浮点数少于VGG的1/3),准确率稍差。

缺陷:

- 输入尺寸固定：由于输出层为全连接层，因此在检测时，YOLO训练模型只支持与训练图像相同的输入分辨率。其它分辨率需要缩放成改分辨率.
- 占比较小的目标检测效果不好.虽然每个格子可以预测B个bounding box，但是最终只选择只选择IOU最高的bounding box作为物体检测输出，即每个格子最多只预测出一个物体。当物体占画面比例较小，如图像中包含畜群或鸟群时，每个格子包含多个物体，但却只能检测出其中一个。

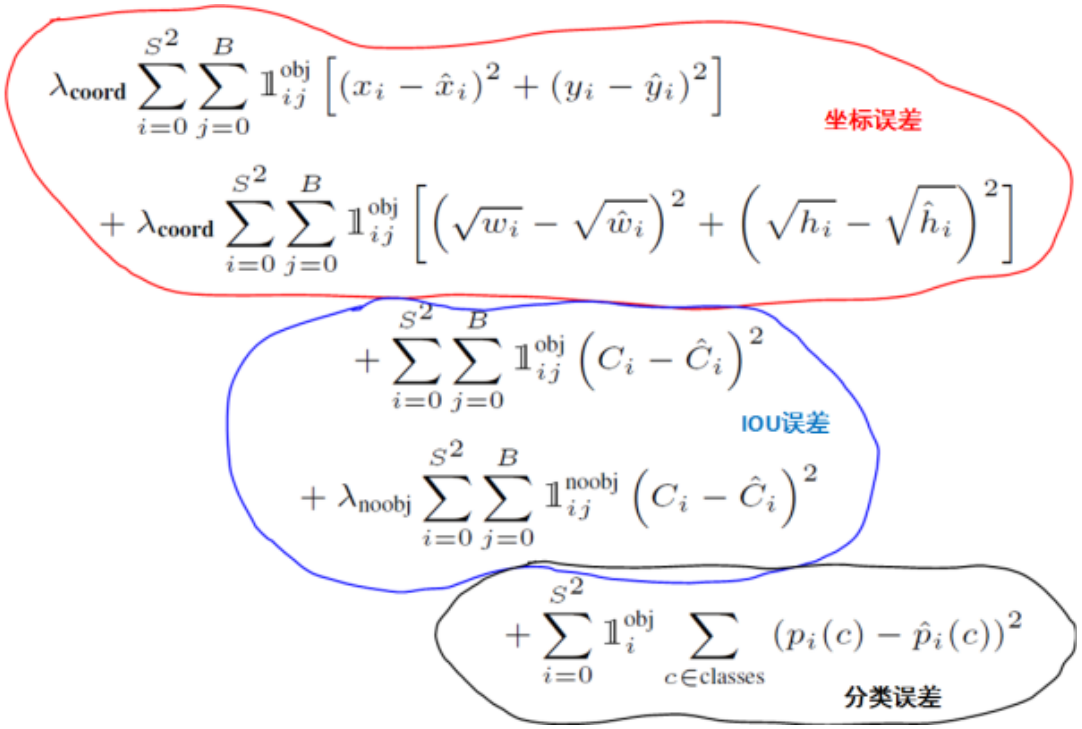
损失函数

YOLO全部使用了均方和误差作为loss函数.由三部分组成:坐标误差、IOU误差和分类误差。

$$loss=\sum_{i=0}^{s^2} coordErr+iouErr+clsErr$$

简单相加时还要考虑每种loss的贡献率,YOLO给coordErr设置权重$\lambda_{coord}=5$.在计算IOU误差时，包含物体的格子与不包含物体的格子，二者的IOU误差对网络loss的贡献值是不同的。若采用相同的权值，那么不包含物体的格子的confidence值近似为0，变相放大了包含物体的格子的confidence误差在计算网络参数梯度时的影响。为解决这个问题，YOLO 使用$\lambda_{noobj}=0.5$修正iouErr。（此处的‘包含’是指存在一个物体，它的中心坐标落入到格子内）。对于相等的误差值，大物体误差对检测的影响应小于小物体误差对检测的影响。这是因为，相同的位置偏差占大物体的比例远小于同等偏差占小物体的比例。YOLO将物体大小的信息项（w和h）进行求平方根来改进这个问题，但并不能完全解决这个问题。

综上，YOLO在训练过程中Loss计算如下式所示：



yolo-loss

其中有宝盖帽子符号(x^,y^,w^,h^,C^,p^)为预测值,无帽子的为训练标记值。1^{obj}_{ij}表示物体落入格子i的第j个bbox内.如果某个单元格中没有目标,则不对分类误差进行反向传播;B个bbox中与GT具有最高IoU的一个进行坐标误差的反向传播,其余不进行.

训练过程

1）预训练。使用 ImageNet 1000 类数据训练YOLO网络的前20个卷积层+1个average池化层+1个全连接层。训练图像分辨率resize到224x224。

2）用步骤1）得到的前20个卷积层网络参数来初始化YOLO模型前20个卷积层的网络参数，然后用 VOC 20 类标注数据进行YOLO模型训练。检测通常需要有细密纹理的视觉信息,所以为提高图像精度，在训练检测模型时，将输入图像分辨率从224 × 224 resize到448x448。

训练时B个bbox的ground truth设置成一样的.

升级版 YOLO v2

为提高物体定位精准性和召回率，YOLO作者提出了 《YOLO9000: Better, Faster, Stronger》 (Joseph Redmon, Ali Farhadi, CVPR 2017, Best Paper Honorable Mention)，相比v1提高了训练图像的分辨率；引入了faster rcnn中anchor box的思想，对网络结构的设计进行了改进，输出层使用卷积层替代YOLO的全连接层，联合使用coco物体检测标注数据和imagenet物体分类标注数据训练物体检测模型。相比YOLO，YOLO9000在识别种类、精度、速度、和定位准确性等方面都有大大提升。

YOLOv2 改进之处

YOLO与Fast R-CNN相比有较大的定位误差，与基于region proposal的方法相比具有较低的召回率。因此YOLO v2主要改进是提高召回率和定位能力。下面是改进之处：

Batch Normalization： v1中也大量用了Batch Normalization，同时在定位层后边用了dropout，v2中取消了dropout，在卷积层全部使用Batch Normalization。

高分辨率分类器：v1中使用224 × 224训练分类器网络，扩大到448用于检测网络。v2将ImageNet以448×448 的分辨率微调最初的分类网络，迭代10 epochs。

Anchor Boxes：v1中直接在卷积层之后使用全连接层预测bbox的坐标。v2借鉴Faster R-CNN的思想预测bbox的偏移.移除了全连接层,并且删掉了一个pooling层使特征的分辨率更大一些.另外调整了网络的输入(448->416)以使得位置坐标是奇数只有一个中心点(yolo使用pooling来下采样,有5个size=2,stride=2的max pooling,而卷积层没有降低大小,因此最后的特征是416/(2^5)=13).v1中每张图片预测7x7x2=98个box,而v2加上Anchor Boxes能预测超过1000个.检测结果从69.5mAP,81% recall变为69.2 mAP,88% recall.

YOLO v2对Faster R-CNN的手选先验框方法做了改进,采样k-means在训练集bbox上进行聚类产生合适的先验框.由于使用欧氏距离会使较大的bbox比小的bbox产生更大的误差，而IOU与bbox尺寸无关,因此使用IOU参与距离计算,使得通过这些anchor boxes获得好的IOU分值。距离公式：

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

使用聚类进行选择的优势是达到相同的IOU结果时所需的anchor box数量更少,使得模型的表示能力更强,任务更容易学习.k-means算法代码实现参考:k means yolo.py.算法过程是:将每个bbox的宽和高相对整张图片的比例(wr,hr)进行聚类,得到k个anchor box,由于darknet代码需要配置文件中region层的anchors参数是绝对值大小,因此需要将这个比例值乘上卷积层的输出特征的大小.如输入是416x416,那么最后卷积层的特征是13x13.

细粒度特征(fine grain features):在Faster R-CNN 和 SSD 均使用了不同的feature map以适应不同尺度大小的目标.YOLOv2使用了一种不同的方法，简单添加一个 pass through layer，把浅层特征图（26x26）连接到深层特征图(连接到新加入的三个卷积核尺寸为3 * 3的卷积层最后一层的输入)。 通过叠加浅层特征图相邻特征到不同通道（而非空间位置），类似于Resnet中的identity mapping。这个方法把26x26x512的特征图叠加成13x13x2048的特征图，与原生的深层特征图相连接，使模型有了细粒度特征。此方法使得模型的性能获得了1%的提升。

Multi-Scale Training: 和YOLOv1训练时网络输入的图像尺寸固定不变不同，YOLOv2（在cfg文件中random=1时）每隔几次迭代后就会微调网络的输入尺寸。训练时每迭代10次，就会随机选择新的输入图像尺寸。因为YOLOv2的网络使用的downsamples倍率为32，所以使用32的倍数调整输入图像尺寸{320,352，…，608}。训练使用的最小的图像尺寸为320 x 320，最大的图像尺寸为608 x 608。这使得网络可以适应多种不同尺度的输入.

YOLOv2网络结构

YOLOv2对v1的基础网络做了更改.

分类网络

YOLOv2提出了一种新的分类模型Darknet-19.借鉴了很多其它网络的设计概念.主要使用3x3卷积并在pooling之后channel数加倍(VGG);global average pooling替代全连接做预测分类,并在3x3卷积之间使用1x1卷积压缩特征表示(Network in Network);使用 batch normalization 来提高稳定性,加速收敛,对模型正则化.

Darknet-19的结构如下表:

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			

Darknet-19-arch

包含 19 conv + 5 maxpooling.

训练:使用Darknet框架在ImageNet 1000类上训练160 epochs,学习率初始为0.1,以4级多项式衰减.weight decay=0.0005 , momentum=0.9.使用标准的数据增广方法:random crops, rotations, (hue, saturation), exposure shifts.

之后将输入从224放大至448,学习率调整为0.001,迭代10 epochs.结果达到top-1 accuracy 76.5% , top-5 accuracy 93.3%.

检测网络

在分类网络中移除最后一个1x1的层,在最后添加3个3x3x1024的卷积层,再接上输出是类别个数的1x1卷积.

对于输入图像尺寸为Si x Si,最终3x3卷积层输出的feature map是Oi x Oi(Oi=Si/(2^5)),对应输入图像的Oi x Oi个栅格，每个栅格预测#anchors种boxes大小，每个box包含4个坐标值,1个置信度和#classes个条件类别概率，所以输出维度是Oi x Oi x #anchors x (5 + #classes)。

添加**跨层跳跃连接**（借鉴ResNet等思想）,融合粗细粒度的特征:将前面最后一个3x3x512卷积的特征图,对于416x416的输入,该层输出26x26x512,直接连接到最后新加的三个3x3卷积层的最后一个的前边.将26x26x512变形为13x13x1024与后边的13x13x1024特征按channel堆起来得到13x13x3072.从yolo-voc.cfg文件可以看到，第25层为route层，逆向9层拿到第16层26 * 26 * 512的输出，并由第26层的reorg层把26 * 26 * 512 变形为13 * 13 * 2048，再有第27层的route层连接24层和26层的输出，堆叠为13 * 13 * 3072，由最后一个卷积核为3 * 3的卷积层进行跨通道的信息融合并把通道降维为1024。

训练:作者在VOC07+12以及COCO2014数据集上迭代了160 epochs,初始学习率0.001,在60和90 epochs分别减小为0.1倍.

Darknet训练VOC的参数如下:

```
learning_rate 0.0001batch 64max_batches   45000                policy
steps 100 25000 35000                                            scales 10  1  1
```

网络结构如下(输入416,5个类别,5个anchor box; 此结构信息由Darknet框架启动时输出):

```
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  32
    1 max          2 x 2 / 2   416 x 416 x  32   ->   208 x 208 x  32
    2 conv     64  3 x 3 / 1   208 x 208 x  32   ->   208 x 208 x  64
    3 max          2 x 2 / 2   208 x 208 x  64   ->   104 x 104 x  64
    4 conv    128  3 x 3 / 1   104 x 104 x  64   ->   104 x 104 x 128
    5 conv     64  1 x 1 / 1   104 x 104 x 128   ->   104 x 104 x  64
    6 conv    128  3 x 3 / 1   104 x 104 x  64   ->   104 x 104 x 128
    7 max          2 x 2 / 2   104 x 104 x 128   ->    52 x  52 x 128
    8 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
    9 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128
   10 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256
   11 max          2 x 2 / 2    52 x  52 x 256   ->    26 x  26 x 256
   12 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
   13 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
   14 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
   15 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256
   16 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512
   17 max          2 x 2 / 2    26 x  26 x 512   ->    13 x  13 x 512
   18 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
   19 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512
   20 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
   21 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512
   22 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024
   23 conv   1024  3 x 3 / 1    13 x  13 x1024   ->    13 x  13 x1024
   24 conv   1024  3 x 3 / 1    13 x  13 x1024   ->    13 x  13 x1024
   25 route  16
   26 reorg              / 2    26 x  26 x 512   ->    13 x  13 x2048
   27 route  26 24
   28 conv   1024  3 x 3 / 1    13 x  13 x3072   ->    13 x  13 x1024
   29 conv     50  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x  50
   30 detection
```

YOLO v2-network

YOLO9000

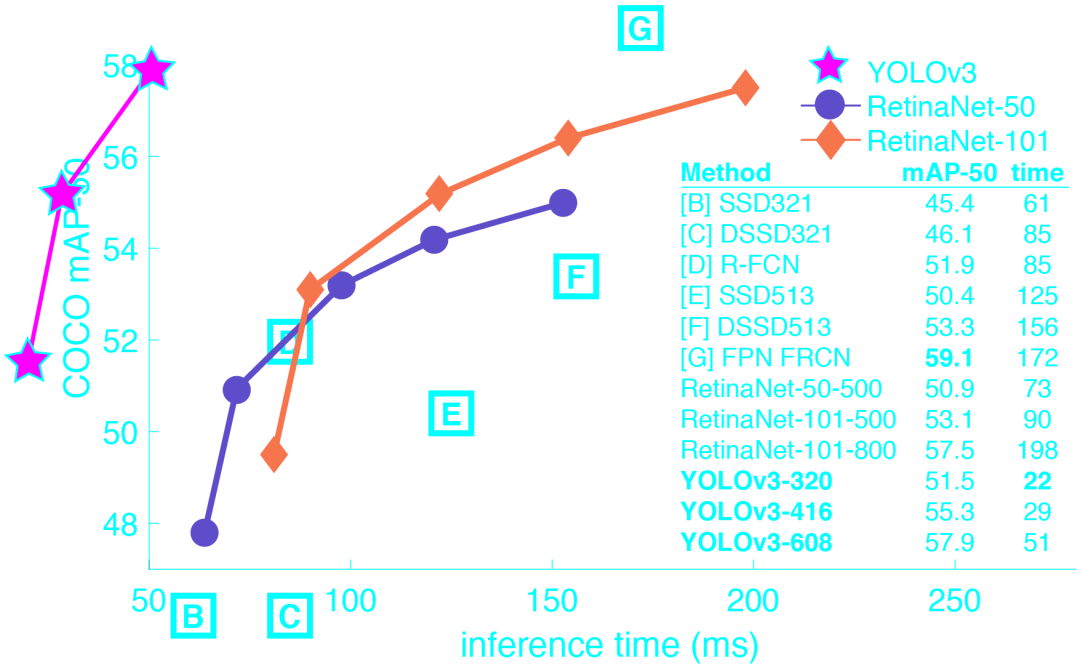
提出了一种联合训练方法，能够容许同时使用目标检测数据集和分类数据集。使用有标记的检测数据集精确定位，使用分类数据增加类别和鲁棒性。

YOLOv3

YOLOv3在Pascal Titan X上处理608x608图像速度达到20FPS，在 COCO test-dev 上 mAP@0.5 达到 57.9%，与RetinaNet（FocalLoss论文所提出的单阶段网络）的结果相近，并且速度快4倍.

YOLO v3的模型比之前的模型复杂了不少，可以通过改变模型结构的大小来权衡速度与精度。

速度对比如下：



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

YOLOv3 compare

改进之处：

- 多尺度预测 （类FPN）
- 更好的基础分类网络（类ResNet）和分类器

分类器-类别预测：

YOLOv3不使用Softmax对每个框进行分类，主要考虑因素有两个：

1. Softmax使得每个框分配一个类别（score最大的一个），而对于Open Images这种数据集，目标可能有重叠的类别标签，因此Softmax不适用于多标签分类。
2. Softmax可被独立的多个logistic分类器替代，且准确率不会下降。

分类损失采用binary cross-entropy loss.

多尺度预测

每种尺度预测3个box， anchor的设计方式仍然使用聚类,得到9个聚类中心,将其按照大小均分给3中尺度.

- 尺度1: 在基础网络之后添加一些卷积层再输出box信息.
- 尺度2: 从尺度1中的倒数第二层的卷积层上采样(x2)再与最后一个16x16大小的特征图相加,再次通过多个卷积后输出box信息.相比尺度1变大两倍.
- 尺度3: 与尺度2类似,使用了32x32大小的特征图.

参见网络结构定义文件yolov3.cfg

基础网络 Darknet-53

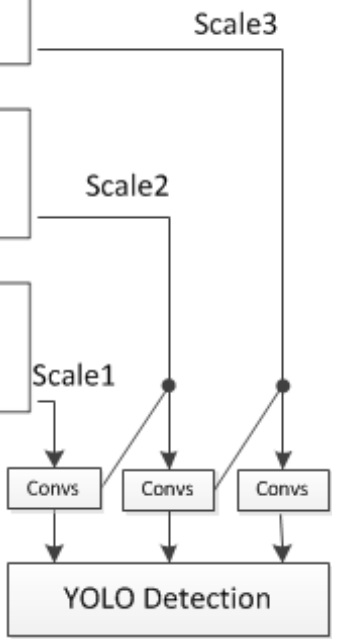
仿ResNet， 与ResNet-101或ResNet-152准确率接近,但速度更快.对比如下:

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [13]	74.1	91.8	7.29	1246	**171**
ResNet-101[3]	77.1	93.7	19.7	1039	53
ResNet-152 [3]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

darknet-53 compare

网络结构如下：

	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			



YOLOv3在mAP@0.5及小目标APs上具有不错的结果,但随着IOU的增大,性能下降,说明YOLOv3不能很好地与ground truth切合.

边框预测

作者尝试了常规的预测方式(Faster R-CNN),然而并不奏效: x,y的偏移作为box的长宽的线性变换.

$$\begin{cases} \end{cases} G_x\hat{} = P_w t_x(P) + P_x \quad G_y\hat{} = P_h t_y(P) + P_y \quad G_w\hat{} = P_w e^{t_w(P)} \quad G_h\hat{} = P_h e^{t_h(P)}$$

仍采用之前的logistic方式:

$$b_x b_y b_w b_h = \sigma(t_x) + c_x = \sigma(t_y) + c_y = p_w e^{t_w} = p_h e^{t_h} \quad (1)\ (2)\ (3)\ (4)$$

其中c_x, c_y是网格的坐标偏移量, p_w, p_h是预设的anchor box的边长.最终得到的边框坐标值是b_*,而网络学习目标是t_*.

优缺点

优点

- 快速,pipline简单.
- 背景误检率低。
- 通用性强。YOLO对于艺术类作品中的物体检测同样适用。它对非自然图像物体的检测率远远高于DPM和RCNN系列检测方法。

但相比RCNN系列物体检测方法，YOLO具有以下缺点：

- 识别物体位置精准性差。
- 召回率低。在每个网格中预测两个bbox这种约束方式减少了对同一目标的多次检测(R-CNN使用的region proposal方式重叠较多),相比R-CNN使用Selective Search产生2000个proposal（RCNN测试时每张超过40秒）,yolo仅使用7x7x2个.

YOLO v.s. Faster R-CNN

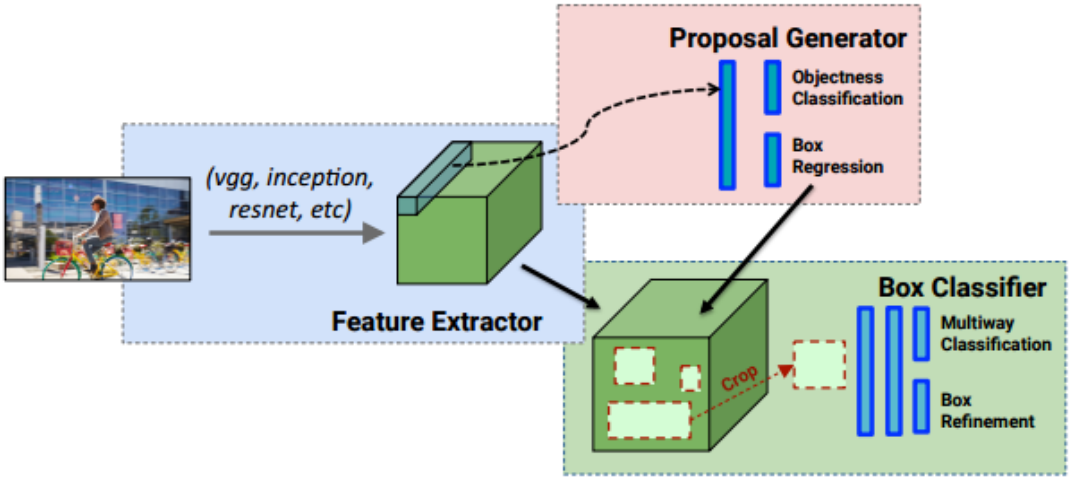
1. 统一网络:

YOLO没有显示求取region proposal的过程。Faster R-CNN中尽管RPN与fast rcnn共享卷积层，但是在模型训练过程中，需要反复训练RPN网络和fast rcnn网络.

相对于R-CNN系列的"看两眼"(候选框提取与分类，图示如下),YOLO只需要Look Once.

2. YOLO统一为一个回归问题

而R-CNN将检测结果分为两部分求解：物体类别（分类问题），物体位置即bounding box（回归问题）。



R-CNN pipline

Darknet 框架

Darknet 由 C 语言和 CUDA 实现，对GPU显存利用效率较高(CPU速度差一些，通过与SSD的Caffe程序对比发现存在CPU较慢,GPU较快的情况). Darknet 对第三方库的依赖较少,且仅使用了少量GNU linux平台C接口,因此很容易移植到其它平台,如Windows或嵌入式设备.

参考Windows 版 Darknet (YOLOv2) 移植, 代码在此.

region层:参数anchors指定kmeans计算出来的anchor box的长宽的绝对值(与网络输入大小相关),num参数为anchor box的数量,

另外还有bias_match,classes,coords等参数.在parser.c代码中的parse_region函数中解析这些参数,并保存在region_layer.num参数保存在l.n变量中;anchors保存在l.biases数组中.region_layer的前向传播中使用for(n = 0; n < l.n; ++n)这样的语句,因此,如果在配置文件中anchors的数量大于num时,仅使用前num个,小于时内存越界.

region层的输入和输出大小与前一层(1x1 conv)的输出大小和网络的输入大小相关.

Detection层: 坐标及类别结果输出层.

参考

- YOLO主页 https://pjreddie.com/darknet/yolo/
- YOLOv3: An Incremental Improvement
- YOLO9000: Better, Faster, Stronger
- You Only Look Once: Unified, Real-Time Object Detection