使用 YOLO V3 神经网络算法
进行目标检测

课题报告

组员：孙剑波 1611640426（查找资料,上台演讲）
李书豪 1611640411（环境搭建,代码实现）
鲁海 1611640422（资料整理,书写报告）

班级： 计算机 S161
时间： 2019 年 1 月 11 日

目录

目标检测

概念：目标检测，也叫目标提取，是一种基于目标几何和统计特征的图像分割，它将目标的分割和识别合二为一，其准确性和实时性是整个系统的一项重要能力。尤其是在复杂场景中，需要对多个目标进行实时处理时，目标自动提取和识别就显得特别重要。

目标检测的任务是找出图像中所有感兴趣的目标（物体），确定它们的位置和大小，是机器视觉领域的核心问题之一。由于各类物体有不同的外观，形状，姿态，加上成像时光照，遮挡等因素的干扰，目标检测一直是机器视觉领域最具有挑战性的问题。

目标检测与图像分类有相似之处，但也有本质的区别：

图像分类：

- 一个图像的类别标签输出

目标检测：

- 输出一个边界框列表，或者一幅图像中每个对象的（x,y）坐标
- 与每个边界框关联的类别标签
- 与每个边界框和类别标签关联的概率或者置信度得分

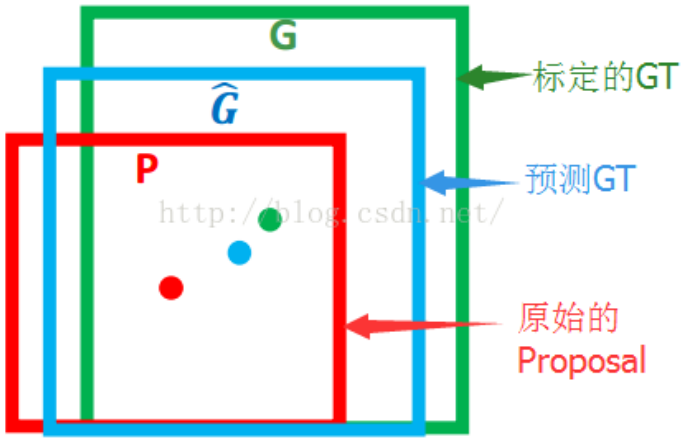
根本区别：

- 在进行图像分类时，我们输入一张图像，得到一个输出类别
- 在进行目标检测时，我们输入一张图像，得到多个边界框以及类别标签的输出

目标检测中使用的概念：

IoU：交并比（IoU），目标检测中使用的一个概念，是产生的候选框（candidate bound）与原标记框（ground truth bound）的交叠率，即它们的交集与并集的比值。最理想情况是完全重叠，即比值为 1。

边框回归(Bounding-box regression): 对于窗口一般使用四维向量 (x,y,w,h) 来表示， x,y 表示窗口的中心点坐标，w,h 表示窗口的宽与高。 如下图，红色的框 P 代表原始的 Proposal，绿色的框 G 代表目标的 Ground Truth， 边框回归目标是寻找一种关系使得输入原始的窗口 P 经过映射得到一个跟真实窗口 G 更接近的回归窗口 G^。



非极大值抑制(NMS): 非极大值抑制（Non-Maximum Suppression，NMS），顾名思义就是抑制不是极大值的元素，可以理解为局部最大搜索。这个局部代表的是一个邻域，邻域有两个参数可变，一是邻域的维数，二是邻域的大小。

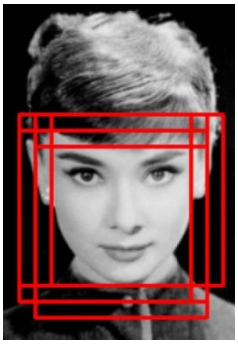
这里不讨论通用的 NMS 算法，而是用于目标检测中提取分数最高的窗口的。例如在行人检测中，滑动窗口经提取特征，经分类器分类识别后，每个窗口都会得到一个分数。但是滑动窗口会导致很多窗口与其他窗口存在包含或者大部分交叉的情况。这时就需要用到 NMS 来选取那些邻域里分数最高（是行人的概率最大），并且抑制那些分数低的窗口。

NMS 基本框架：

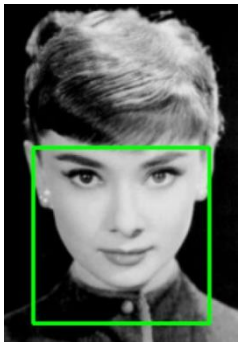
1. 将所有框的得分排序，选中最高分及其对应的框；

2. 遍历其余的框，如果和当前最高分框的重叠面积(IOU)大于一定阈值，我们就将框删除。

3. 从未处理的框中继续选一个得分最高的，重复上述过程。



Original



After NMS

Darknet

Darknet 由 Joseph Redmon 提出的是一个较为轻型的完全基于 C 与 CUDA 的开源深度学习框架，其主要特点就是容易安装，没有任何依赖项（OpenCV 都可以不用），移植性非常好，支持 CPU 与 GPU 两种计算方式。

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。

yolo 层: 指定 anchors 等信息, 计算 loss 等。YOLOv3 使用三个 yolo 层作为输出。

upsample 层: 上采样层, 进行 2 倍上采样。

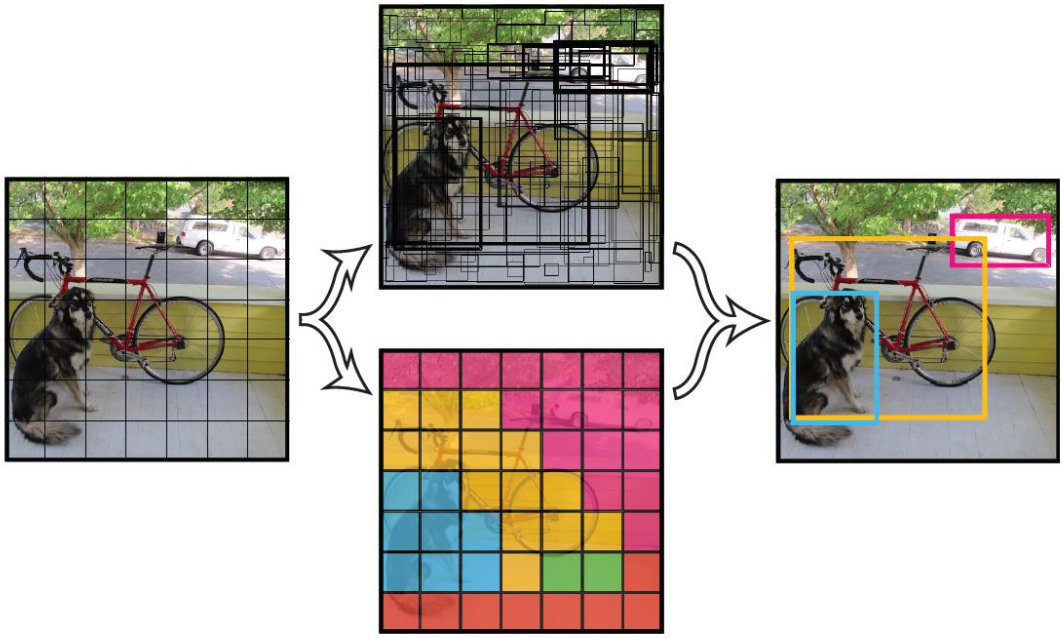
region 层和 Detection 层均是 YOLOv2 模型所使用的层, upsample 层和 yolo 层在 YOLOv3 中使用.

YOLO v1

YOLO 是 You Only Look Once 的缩写。它是 Joseph Redmon 针对 Darknet 框架提出的一种使用深度卷积神经网络学得的特征来检测对象的目标检测器。

YOLO 算法的基本思想：

首先通过特征提取网络对输入图像提取特征，得到一定 size 的 feature map，比如 13*13，然后将输入图像分成 13*13 个 grid cell，接着如果 ground truth 中某个 object 的中心坐标落在哪个 grid cell 中，那么就由该 grid cell 来预测该 object。



如上图所示，输入图片被划分为 7x7 个单元格，每个单元格独立作检测。

在这里很容易被误导：每个网格单元的视野有限而且很可能只有局部特征，这样就很难理解 yolo 为何能检测比 grid_cell 大很多的物体。其实，yolo 的做法并不是把每个单独的网格作为输入 feed 到模型，在 inference 的过程中，网格只是物体中心点位置的划分之用，并不是对图片进行切片，不会让网格脱离整体的关系。

因为每个 grid cell 都会预测固定数量的 bounding box（YOLO v1 中是 2 个，YOLO v2 中是 5 个，YOLO v3 中是 3 个，这几个 bounding box 的初始

size 是不一样的)，那么这几个 bounding box 中最终是由哪一个来预测该 object? 答案是：这几个 bounding box 中只有和 ground truth 的 IOU 最大的 bounding box 才是用来预测该 object 的。

也就是说每个网格要预测 B 个 bounding box，每个 bounding box 除了要回归自身的位置之外，还要附带预测一个 confidence 值。这个 confidence 代表了所预测的 box 中含有 object 的置信度和这个 box 预测的有多准两重信息，其值是这样计算的：

$$\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

其中如果有 object 落在一个 grid cell 里，第一项取 1，否则取 0。 第二项是预测的 bounding box 和实际的 groundtruth 之间的 IoU 值。

每个 bounding box 要预测(x, y, w, h)和 confidence 共 5 个值，每个网格还要预测一个类别信息，记为 C 类。则 SxS 个网格，每个网格要预测 B 个 bounding box 还要预测 C 个 categories。输出就是 S x S x (5*B+C)的一个 tensor。注意：class 信息是针对每个网格的，confidence 信息是针对每个 bounding box 的。

每个小格会对应 C 个概率值，找出最大概率对应的类别 P(Class|object)，并认为小格中包含该物体或者该物体的一部分。

由于 yolo 是端到端训练，对于预测框的位置、size、种类、置信度(score)等信息的预测都通过一个损失函数来训练。

Loss = λcoordλcoord * 坐标预测误差(1) + 含 object 的 box confidence 预测误差

(2)+ λnoobjλnoobj* 不含 object 的 box confidence 预测误差(3) + 每个格子中类别预测误差(4)

(l_{ij}^{obj} 取值为 0 和 1，即单元格内是否有目标。λ_{coord} (坐标误差)=5，λ_{noobj} (置信度误差)=0.5)

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \quad (1)$$

坐标预测

判断第i个网格中的第j个box是否负责这个object

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2 \quad (2)$$

含object的box的confidence预测

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2 \quad (3)$$

不含object的box的confidence预测

判断是否有object中心落在网格i中

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2 \quad (4)$$

类别预测

缺陷:

- YOLO 对相互靠的很近的物体，还有很小的群体 检测效果不好，这是因为一个网格中只预测了两个框，并且只属于一类。
- 同一类物体出现的新的不常见的长宽比和其他情况时，泛化能力偏弱。
- 由于损失函数的问题，定位误差是影响检测效果的主要原因。尤其是大小物体的处理上，还有待加强。

YOLO v2

YOLO v2 主要改进是提高召回率和定位能力。

- **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。调整了网络的输入(448->416)，以使得位置坐标为奇数，这样就只有一个中心点。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.

without anchor	69.5 mAP	81% recall
with anchor	69.2 mAP	88% recall

- YOLO v2 对 Faster R-CNN 的首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，因此因此，对于距离判断，作者用了： $D(\text{box},\text{centroid}) = 1 - IOU(\text{box},\text{centroid})$
  作者对 k-means 算法取了各种 k 值，并且画了一个曲线图：

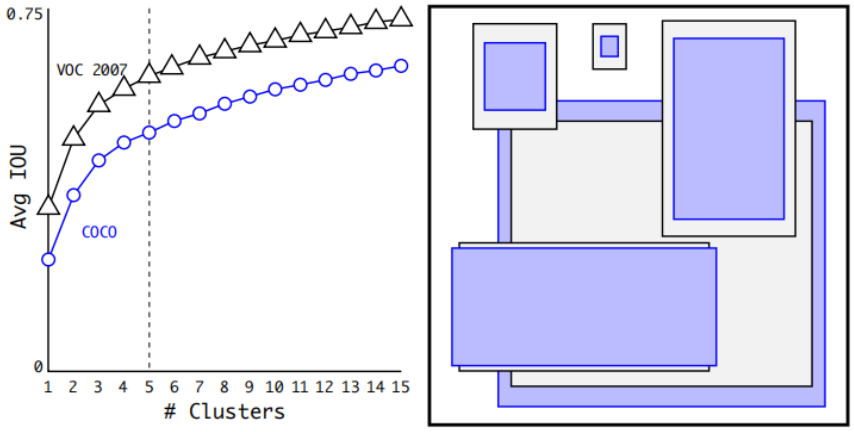
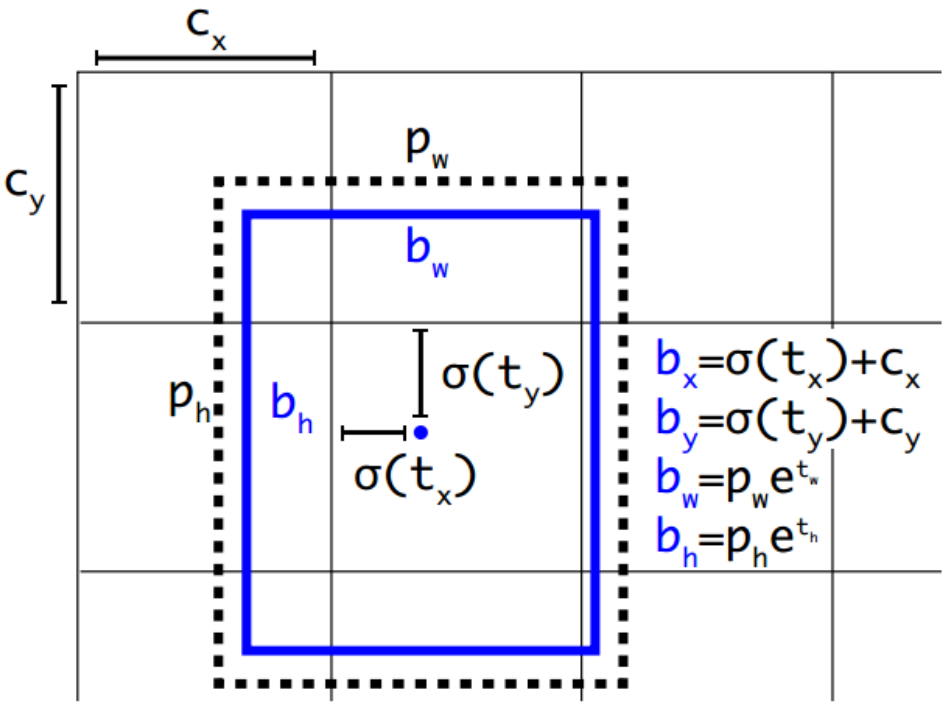


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. 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.

作者最终选择了 k=5，这是在模型复杂度和高召回率之间取了一个折中。聚类得到的框和之前手动挑选的框大不一样。有稍微短宽的和高瘦一些的(框)。

bounding box 的坐标预测方式：



tx、ty、tw、th 就是模型的预测输出。cx 和 cy 表示 grid cell 的坐标，比如某层的 feature map 大小是 13*13，那么 grid cell 就有 13*13 个，第 0 行第 1 列的 grid cell 的坐标 cx 就是 0，cy 就是 1。pw 和 ph 表示预测前 bounding box 的 size。bx、by。bw 和 bh 就是预测得到的 bounding box 的中心的坐标和 size。坐标的损失采用的是平方误差损失。

- **细粒度特征(fine grain features)**：借鉴了 Faster R-CNN 和 SSD 使用的不同尺寸的 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，把浅层特征图连接到深层特征图。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。

V2 对 V1 的基础网络也做了修改:

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.

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

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%。

整体改进效果:

序号	描述		YOLO							YOLOv2
1	batch norm代替dropout防overfit	batch norm?		√	√	√	√	√	√	√
2	448x448 finetune ImageNet	hi-res classifier?			√	√	√	√	√	√
3	grid: 13 x 13代替7 x7，416 / 32 = 13为奇数	convolutional?				√	√	√	√	√
4	除去FC，加入anchors，提高recall	anchor boxes?				√	√			
5	设计新分类网络darknet-19作为基础网络	new network?					√	√	√	√
6	Kmeans离线选取anchor个数k=5	dimension priors?						√	√	√
7	新的位置预测方法	location prediction?						√	√	√
8	增加跳级路特征 => 结合高低分辨率	passthrough?							√	√
9	FCN能适用多尺度(32倍数)，尺度=> 分格数	multi-scale?							√	√
10	-	hi-res detector?								√
		VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

YOLO v3

YOLO v3 的改进：

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

分类器：

- YOLOv3 不使用 Softmax 对每个框进行分类，而使用多个 logistic 分类器，因为 Softmax 不适用于多标签分类，用独立的多个 logistic 分类器准确率也不会下降。
- 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 种尺度。
    - 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.

输出处理

对于大小为 416 x 416 的图像，YOLO 可以预测 ((52 x 52) + (26 x 26) + 13 x 13)) x 3 = 10647 个边界框。但是，示例中只有一个对象，所以需要将检测次数从 10647 减少到 1 。

目标置信度阈值：根据它们的 objectness 分数过滤边界框。分数低于阈值的边界框会被忽略。

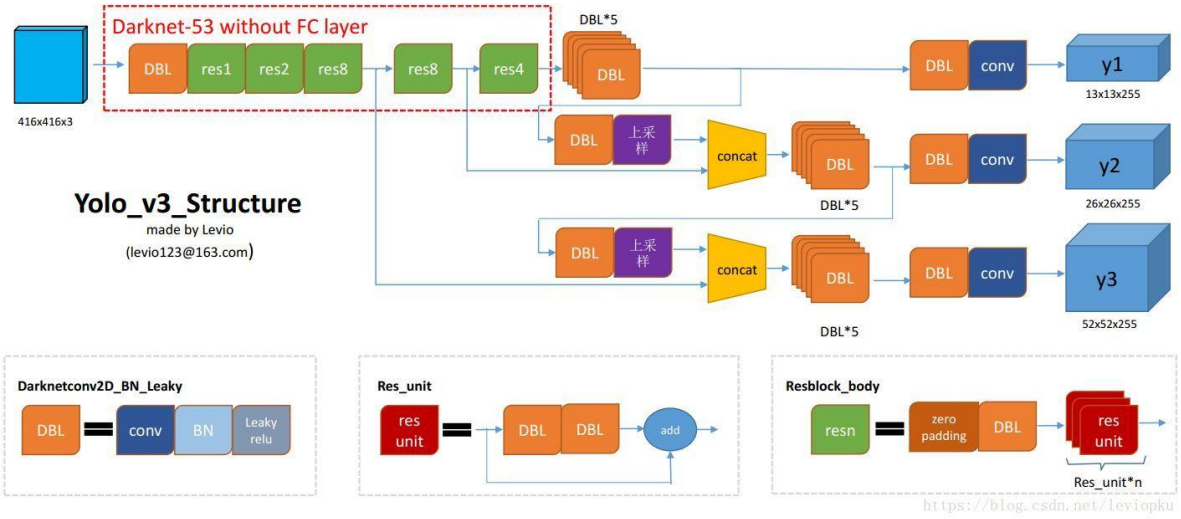
非极大值抑制：解决对同一个图像的多次检测的问题。

基础网络 Darknet-53：

	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			

Table 1. **Darknet-53.**

YOLO v3 结构图：



DBL: 如图 1 左下角所示，也就是代码中的 Darknetconv2d_BN_Leaky，是 yolo_v3 的基本组件。就是卷积+BN+Leaky relu。对于 v3 来说，BN 和 leaky relu 已经是和卷积层不可分离的部分了(最后一层卷积除外)，共同构成了最小组件。

resn： n 代表数字，有 res1，res2，… ,res8 等等，表示这个 res_block 里含有多少个 res_unit。这是 yolo_v3 的大组件，yolo_v3 开始借鉴了 ResNet 的残差

结构，使用这种结构可以让网络结构更深(从 v2 的 darknet-19 上升到 v3 的 darknet-53，前者没有残差结构)。对于 res_block 的解释，可以在图 1 的右下角直观看到，其基本组件也是 DBL。

concat：张量拼接。将 darknet 中间层和后面的某一层的上采样进行拼接。拼接的操作和残差层 add 的操作是不一样的，拼接会扩充张量的维度，而 add 只是直接相加不会导致张量维度的改变。

我们可以借鉴 netron 来分析网络层，整个 yolo_v3_body 包含 252 层，组成如下：

可以得出，对于代码层面的 layers 数量一共有 252 层，包括 add 层 23 层(主要用于 res_block 的构成，每个 res_unit 需要一个 add 层，一共有 1+2+8+8+4=23 层)。除此之外，BN 层和 LeakyReLU 层数量完全一样(72 层)，在网络结构中的表现为：每一层 BN 后面都会接一层 LeakyReLU。卷积层一共有 75 层，其中有 72 层后面都会接 BN+LeakyReLU 的组合构成基本组件 DBL。看结构图，可以发现上采样和 concat 都有 2 次，和表格分析中对应上。每个 res_block 都会用上一个零填充，一共有 5 个 res_block。

整个 v3 结构里面，是没有池化层和全连接层的。前向传播过程中，张量的尺寸变换是通过改变卷积核的步长来实现的，比如 stride=(2, 2)，这就等于将图像边长缩小了一半(即面积缩小到原来的 1/4)。在 yolo_v2 中，要经历 5 次缩小，会将特征图缩小到原输入尺寸的 1/2^5，即 1/32。输入为 416x416，则输出为 13x13(416/32=13)。

检测结果

YOLO 只能检测出属于训练所用数据集中类别的对象。这里受限于硬件设备以及为了更好的检测效果，检测器使用官方权重文件，这些权重通过在 COCO 数据集上训练网络而获得，可以检测 80 个对象类别。

检测环境：

系统：Windows 10 64 位

处理器：Intel Xeon E5-2680 v2

显卡：GTX 1060 5G

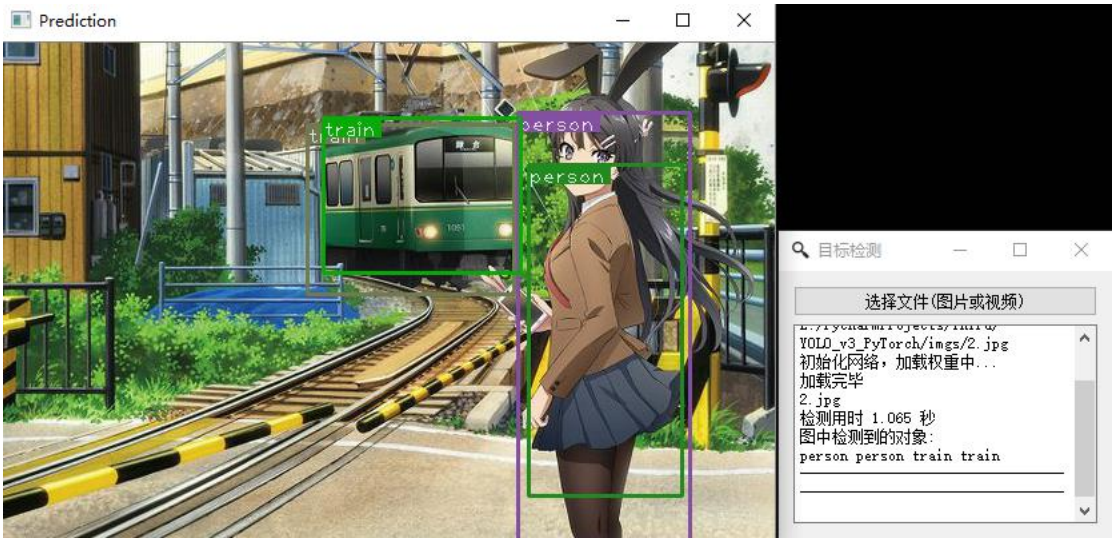
检测结果：

1．分辨率：416x416，**置信度：**0.5，**NMS：**0.4

2．分辨率：416x416，**置信度：**0.3，**NMS：**0.4

可以看出，降低置信度阈值可以显示更多的检测结果。

3. 分辨率：416x416，**置信度：**0.5，**NMS：**0.8

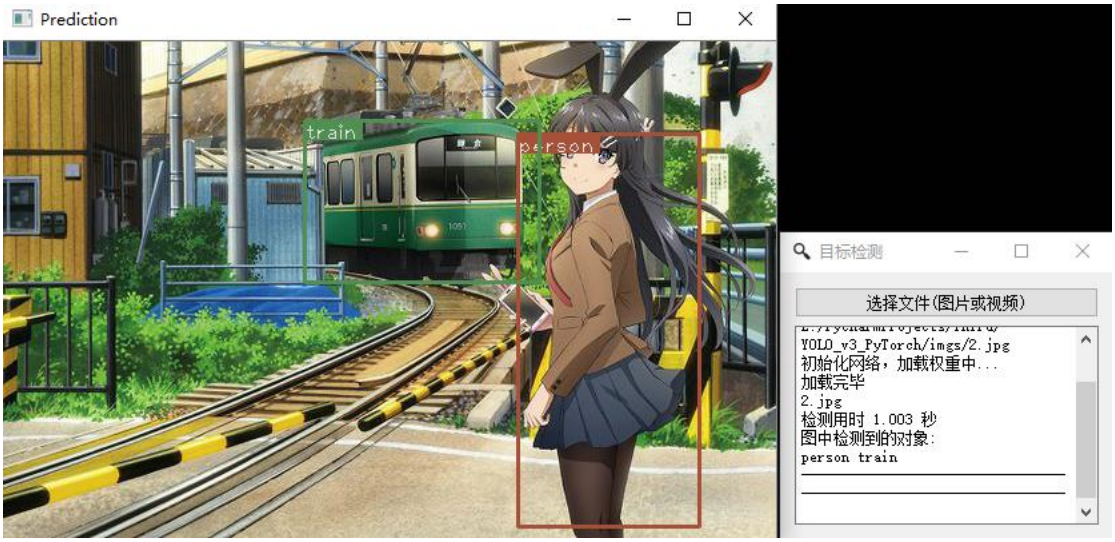


可以看出，增大 NMS，这时只有 IOU 大于 0.8 的 b-box 才会被删除

4. 分辨率：832x832，**置信度：**0.5，**NMS：**0.4

可以看出，增大输入图像的分辨率，检测用时增加，但检测的准确度增加。

5．分辨率：128x128，**置信度：**0.5，**NMS：**0.4



可以看出，减小输入图像的分辨率，检测用时变短，此时并没有影响到检测结果。

6．分辨率：64x64，**置信度：**0.5，**NMS：**0.4

可以看出，继续降低图像输入的分辨率，检测速度得到提升，但准确度下降。

（由于以上所有检测用时都是第一次检测，所以时间较长，正常情况下在**分辨率：**

416x416，**置信度**：0.5，**NMS**：0.4 的情况下每张图片只需要不到 0.1 秒。）

视频检测(分辨率：416x416，**置信度**：0.5，**NMS**：0.4)：

实验总结

YOLO v3 比其他检测系统在各种检测数据集中更快。此外，它可以以各种图像大小运行，以提供速度和精度之间的平滑权衡。所以 YOLO v3 无疑是目前目标检测的首选算法之一。

参考资料

1. You Only Look Once: Unified, Real-Time Object Detection——https://arxiv.org/pdf/1506.02640.pdf
2. YOLO9000: Better, Faster, Stronger——https://arxiv.org/pdf/1612.08242v1.pdf
3. YOLOv3: An Incremental Improvement——https://pjreddie.com/media/files/papers/YOLOv3.pdf
4. https://blog.csdn.net/leviopku
5. https://www.cnblogs.com/fariver/p/7446921.html
6. https://www.cnblogs.com/jins-note/p/9804389.html
7. https://www.cnblogs.com/makefile/p/YOLOv3.html
8. https://www.cnblogs.com/xbit/p/10036981.html
9. https://blog.csdn.net/KKKSQJ/article/details/83587138
10. https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/
11. https://pjreddie.com/darknet/