Object Detection: Yolo Net

YWJ
HUST-IDC / No Address
ywj@hust.edu.cn

1 Introduction

YOLO(You Only Look Once)(Redmon et al., 2015)是CVPR2016目标检测领域的新秀，在VOC数据集上曾一度占据榜首,但很快被2017年初的SSD(Single Shot MultiBox Detector)(Liu et al., 2016)盖过了风头。但同SSD之于YOLO的进步相比，YOLO相对于FASTER-RCNN的进步是更有意义的。在YOLO之前，目标检测被当作分类任务，图片中的检测框(Bounding Box)是预先给定的，检测任务实际上是判断目标属于哪个BB，为了获得目标的类别，还必须训练一个额外的分类网络。但在YOLO之后，目标检测被当作回归任务，YOLO通过单个简单而深的卷积网络加上全连接层预测BB的中心位置、大小以及目标类别概率，之后的SSD与之大同小异，可以说YOLO开了这一类目标检测方法的先河，能理解YOLO的原理，就能很容易理解其它衍生网络了。

2 YOLO原理

2.1 特征提取

YOLO使用所谓的DARK-NET作为特征提取网络，这是一个24层卷积加4层最大池化的庞大网络，参考图1。原图输入(448，448)的图片经过两次步长为2的卷积以及4次大小为2的池化被缩小了64倍，最终产生(7,7,1024)的特征图，在其余卷积层不会改变输入大小。特征图经由两个全连接层产生最终输出形如(7,7,30)。

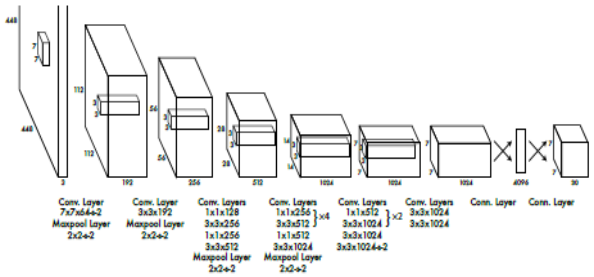

图 1: 网络结构

2.2 网络输出

前面已经提过，YOLO把目标检测当做了回归任务，因而不会预先确定BB的大小和位置，直观的想法是学习映射f:

$$f(image) = < coord, H, W, class_{pro} > \quad (1)$$

但由于图片中目标数量可能是多个，而且每张图片目标数各不相同，要求函数对同规格的输入给出数量不等的输出是困难的，因此YOLO隐式地指定了检测BB的数量。YOLO将输入图片划分为(7,7)的网格(grid)，每个grid负责预测指定数量的BB(这里设置为2)，为了判别目标数目，必须判断那些BB中有目标，YOLO的办法是对每个BB预测交并比(IOU)，IOU定义为预测BB和标记BB取交的面积比上取并的面积，如果标记BB的坐标落在当前grid内，预测可以通过调整BB的大小和位置使IOU接近1，反之无论如何调整，IOU都不可能是较大的值。

$$IOU = \frac{area(BB_{predict} \cap BB_{label})}{area(BB_{predict} \cup BB_{label})} \quad (2)$$

引入grid和IOU后，目标可以写成：

$$f(image)=\begin{bmatrix} <IOU,coord,H,W,class_{pro}>_1 \\ <IOU,coord,H,W,class_{pro}>_2 \end{bmatrix}_{7\times7} \tag{3}$$

$class_{pro}$是长为20经softmax归一化的预测概率向量，注意YOLO只对每个grid预测类别概率而非每个BB。每一个grid的预测向量形如$((IOU+coord_x+coord_y+H+W)\times2+class_{pro})$，对应VOC数据集上的20个目标类别，$class_{pro}$是长为20，每个grid需要产生预测长为$(5\times2+20)=30$,这与前面YOLO的输出(7,7,30)相对应。

2.3 损失计算

YOLO的损失分为三个部分：

1. 坐标损失。简单的计算预测与标记差的二阶泛数。δ_{ij}表示第i个grid中有目标且的第j个BB负责预测，在实现中，总是让IOU预测值最大的BB负责预测。

$$\sum_{i=0}^{S^2}\sum_{j}^{B}\delta_{ij}||BB_{predict}-BB_{label}||_2^2 \tag{4}$$

2. 预测类别损失。类别标记是对应类的one-hot向量。δ_i表示第i个grid中有目标。

$$\sum_{i=0}^{S^2}\delta_i||class_{pro}-class_{label}||_2^2 \tag{5}$$

2. IOU损失，$IOU_{predict}$是预测向量中的一位，IOU_{label}根据预测向量中的BB和样本标记中的BB计算而来，为了防止网络对每个grid都给出很大的预测BB，这一项对没有目标中心落在其中的grid给出的任何IOU做了惩罚。

$$\sum_{i=0}^{S^2}\sum_{j}^{B}[\delta_{ij}||IOU_{predict}-IOU_{label}||_2^2 + (1-\delta_{ij})||IOU_{predict}||_2^2] \tag{6}$$

3 YOLO的实现

3.1 VOC数据集的处理

整个工程的控制参数都记录在config模块中的parameter类中。在其它用到这些控制参数的模块中都声明了para对象，可以把para当作记录了参数信息的全局对象。

在voc.py文件中给出了pascal_voc类用来处理voc数据集。在第一次声明对象时传入para参数，该类会自行处理para中目的路径下的voc数据集，并将结果缓存在para中的cache路径下，再次创建pascal_voc对象时，该类会直接读取缓存而非重复处理voc数据。调用该类成员generate_batch会返回一批图片张量以及对应的标记张量，大小由para中的参数控制。

整个voc数据集被写成按图片索引顺序随机的list缓存在pkl文件中，每一项是[图片名，标记]的形式，generate_batch每次根据图片名读取para.batch_size张的图片和数目相同的标记，分别装入numpy矩阵，generate同时维护cursor指针，指示下次开始读取的位置，cursor指针越界后会被重置，同时缓存list会被随机重排。

每张图片的标记是形如(7,7,25)的张量，对应一张图片每个grid的预测向量，前5位是(是否存在目标，中心相对坐标，高，长)。后20位为目标类别的one-hot向量，对不存在目标的gird，对应的向量是全0向量。

3.2 损失计算的实现

这是整个工程最麻烦的部分，为了叙述方便，在这一段使用默认参数来描述张量形状，batch_size=48，grid数量(7×7)，每个grid预测两个BB。任务是根据网络输出的(48,7,7,30)张量和标记包含的(48,7,7,25)张量计算三项损失。

1. 首先分割网络输出和标记。网络的预测值由三部分组成：(IOU,BB,class)，对应的将(48,7,7,30)按最后一维分割为三个

张量：IOU=(48,7,7,2)，BB=(48,7,7,8)，class=(48,7,7,20)，三个张量分别表示三个部分的预测。同样地标记被分割为obj=(48,7,7,1)，BB'=(48,7,7,4)，class'=(48,7,7,20)，obj是二值张量，记录了哪些grid中有目标。

2. 重排前两项预测。由于前两项预测最后一维包含了两个BB的信息，因此增加一个维度来区分这两个BB,IOU和BB张量被reshape成：IOU=(48,7,7,2,1)，BB=(48,7,7,2,4)。对应地BB'=(48,7,7,1,4)。

3. 计算IOU。根据BB=(48,7,7,2,4)和标记BB'=(48,7,7,1,4)计算形如IOU'=(48,7,7,2,1)的标记值。YOLO_net类中提供了IOU方法用于完成这部分计算。

4. 生成mask。前面已经提到过，两个BB中只有IOU较大的哪个负责预测。因次比较IOU'=(48,7,7,2,1)第3维上的两个值，较大的置1，较小的置0，得到同形的mask=(48,7,7,2,1)，他记录了哪些BB负责预测。

5. 计算损失。经由上面的处理，损失可以表示为：

$$
\begin{aligned}
box\_loss &= ||obj \odot mask \odot (BB - BB')||_2^2 \\
IOU\_loss &= ||obj \odot mask \odot (IOU - IOU') \\
&\quad + (1 - obj \odot mask) \odot IOU||_2^2 \\
class\_loss &= ||obj \odot mask \odot (class - class')||_2^2
\end{aligned} \tag{7}
$$

3.3 网络预训练

由于使用斜率为1的relu作为激活，且图片被归一化到(-1,1)的范围，可以想象网络中存在大量的绝对值小于1的层间梯度，而跨层梯度可以看作层间梯度的累乘，因此从网络输入到第26个卷积层的输出间的梯度大概率是接近0的值，也就是说，最终损失经过层层绝对值小于1的梯度反向传播到第一层的参数时，损失已经是个接近0的值了(Erhan et al., 2010)，

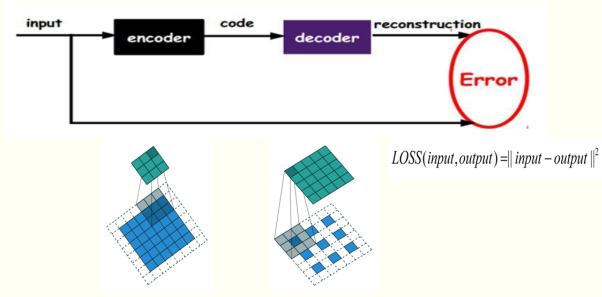

图 2: 预训练

这导致在训练过程中越是排在前面的卷积层参数越难更新。增大训练量确实可以解决这个问题，实际上YOLO的原作者就是这么干的，YOLO原文中的预训练方式是在mnist分类任务上训练了一个星期之久。但这是不必要的，卷积层只负责将原图映射成固定规格的张量，而根据这些张量完成不同的学习任务是全连接层的作用，因此在mnist分类任务上训练的卷积层可以被直接搬到目标检测任务中。基于这样的认识，预训练只需要确保卷积层在映射过程中能保留原图片的信息即可。可以想象，对随机初始化的卷积层，输入任何图片得到的输出都是噪声图，不可能根据这些噪声恢复出原图，因为原图片的信息在经过卷积后全部损失掉了。但经过训练的卷积层给出的输出是包含有原图信息的，所以网络才能依据输入不同的原图给出不同的输出。为了让卷积层输出原图的表示张量的同时能尽可能保留原图信息，可以用端对端的模型单独训练某一卷积层。在这里，对26层卷积层逐层预训练，编码器使用卷积，解码器使用规格相同的转置卷积,见图2。为了防止学出大到导致后级溢出的权重，每次更新参数时将所有参数修剪到(-1,1)。

正式在voc上的训练按照原文给定的训练方法，唯一不同的是，由于显存限制batch_size从64调整为45。

4 对YOLO的看法

YOLO使用单个网络完成了全套目标检测任务，这与之前的各种rcnn相比实属开天辟地，这也使得YOLO的检测速度极快，在

一块1050Ti上YOLO检测帧率就能达到基准的30fps以上。将目标检测当作回归任务也使得YOLO不在受到预设BB的限制，理论上YOLO给出任意位置大小的BB。

但YOLO的限制也很明显，使用26层之深的网络只能检测(448,448)分辨率的图片，第一层网络卷积核设为(7,7)，这意小于图片长宽的1/64的目标在第一层卷积后信息几乎全部丢失了，直接导致YOLO很难检出小目标。当然，在voc数据集中几乎没有这样的小目标。

grid设置为(7,7)，尽管每个grid预测2个BB，但不论在损失计算和最后的检测任务中都只有IOU较大的BB被认为是当前grid的预测值，这导致原图中两个或以上数量的目标中心落在同一个BB中时，YOLO一定只能检出一个，且对每张图最多只能检出49个目标。这并不影响YOLO在voc排行榜上的名次，因为voc里目标重合的情况也很少，且没有哪张图目标数量能多达49。

YOLO的作者给出了YOLO的升级版名为YOLO V2或者YOLO 9000(Redmon and Farhadi, 2016)，具体不再赘述。

References

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. 2010. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research* 11(Feb):625–660. http://www.jmlr.org/papers/v11/erhan10a.html.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. *arXiv:1512.02325 [cs]* 9905:21–37. ArXiv: 1512.02325. https://doi.org/10.1007/978-3-319-46448-0˙2.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2015. You Only Look Once: Unified, Real-Time Object Detection. *arXiv:1506.02640 [cs]* ArXiv: 1506.02640. http://arxiv.org/abs/1506.02640.

Joseph Redmon and Ali Farhadi. 2016. YOLO9000: Better, Faster, Stronger. *arXiv:1612.08242 [cs]* ArXiv: 1612.08242. http://arxiv.org/abs/1612.08242.