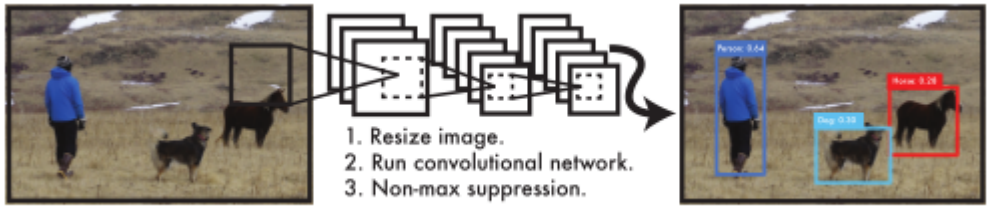
YOLOv1学习笔记

yolov1作为yolo系列的开山鼻祖，对于新手而言容易上手，并且整体的思想容易理解。本节里，我会向大家介绍我自己自学yolov1并穿插将其复现的过程。

一.yolov1的提出

yolov1在2016年的《Y ou Only Look Once:Unified, Real-Time Object Detection》(你只需要看一次：联合、实时的目标检测算法)论文中被提出，正式开创了one-stage目标检测算法的先河。



据该论文介绍，yolov1十分简单，相比于R-CNN的冗长的训练和预测过程，yolov1只需要像图像分类那样：resize图片，输入CNN卷积神经网络，然后通过一个**非极大值抑制**便可以得到最终的预测结果。

因此，yolov1相比于之前的目标检测的算法的一大优势就在于——速度非常快，yolo系列的后续网络都继承了该优点，所以yolo经常被部署到一些移动设备上。

再者，yolov1采用的是全局图片的训练，而R-CNN系列网络采用的是**先验框-region proposal bounding box**，属于局部图像，所以你可以理解为yolov1的网络前向传播过程实际上是一种对全局信息的**编码-encoding**，至于如何进行编码我们后续会讲到。

最后，yolov1的高速度性让其可以在生活的各个方面得以应用，所以它也有非常强大的通用性。

我们先讲他的优点和改进，最后我们复现完了，再来说一下yolov1的缺点。

二.yolov1的原理、思想

(1)端到端的训练思想

以往了R-CNN系列目标检测算法采用的是two-stage的训练思想，比如说：R-CNN是需要先使用SS(selective search)选择性搜索算法完成一个：单个输入图片 -> 多个先验框的过程，这个过程可以得到先验框。。

至于SS算法原理，你可以理解为：利用图像的一些特征，eg：清晰度、模糊度等来将图片中具有相同特征的部分归为一类，根据设定参数的不同，你可以通过一张图片获得多个不同区域。

得到了先验框后，再将先验框区域进行变形——目的是统一大小好输入CNN，然后输入CNN训练得到特征向量，特征向量一边输入SVM分类器进行分类，一边进行回归操作。因此，R-CNN是典型的two-stage目标检测算法。

yolov1刚刚已经讲到了，就是直接将一整张图片输入CNN，这个过程就是-encoder过程，得到的label不同于以往的目标检测，yolov1的label是一个7*7*30的信息张量，有了输入和label就可以进行一个典型的CNN训练了。

如果是预测，输入图片通过网络后，会输出7*7*30的信息张量，我们接下来要干的就是对这个信息张量进行解码-decoder，从中获取分类+回归的信息。

这种训练在原论文中叫做**unified detection**。

我们接下来就详细讲一下，encoder、decoder操作。

(2)encoder操作

encoder操作你可以理解为方便我们完成端到端的训练，**encoder过程就是把人类看得懂的数据转换为方便神经网络训练的数据**。

事实上，端到端的训练方式我们在之前的图像分类中就遇到了，图像分类就是一种——输入图片矩阵到CNN，然后设定神经网络输出一个(N,class_num)矩阵，训练过程“**一步到位**”，所以就叫做端到端。

"理论上来讲，只要设计得当，你可以让神经网络输出任何东西"。(神经网络本质就是计算图嘛，图不想链表、数那样，他是可以多对多的)

那么放在yolov1中，我们还是输入一个(batch_size,channels,height,width)的**训练图片张量**(代表这一个训练批次中，有batch_size张通道数为channels，高度为height，宽度为width)的图片。(后面简记为:(n,c,h,w))

然后设计我们的神经网络——至于怎么设计待会儿再讲，让这个神经网络完成(n,c,h,w) -> (n,30,7,7)的映射。

ok，我们到这里得到了prediction，但是y_train必须和prediction的形状一样，所以我们设计一个函数，将传统的(object_num,5)标签矩阵转为(n,30,7,7)的标签张量。

yolov1中是这样转换的：

- 输入图像通过CNN输出的(n,30,7,7)，我们为了方便理解，取其一来看：(30,7,7)，然后这个张量实际上等于(7,7,30)(我们可以reshape嘛，一样的)。
- yolov1原论文规定:(7,7,30)的张量代表7*7个网格(一共49个网格)，每个网格有30个信息，这30个信息表示为:

  $$30 = num_{object} * (4 + 1) + num_{classes}$$

  也就是说：**yolov1把一张输入图片划分为7*7个网格，每个网格最多检测2个物体(num_object)**。

  4表示每个网格中每个物体的位置信息:(px,py,dw,dh)，注意并不是原始的位置信息哦，**px,py是物体中心点相对于它所处网格的相对坐标，dw,dh是物体宽、长相对于整张图片的长度**，这四个怎么算——马上会讲到。

  1表示置信度:该网格中有物体的概率。标签中为1。

  num_classes:表示物体种类，我们一般会给一个种类列表，网格中的标注物体是哪一类，记类别名称在类别列表中的下标为idx，那么这num_classes的第idx+1位置数值为1表示有，其余为0表示否，**也就是one-hot独热编码**。

---

tips:

1.如果没理解到：为什么神经网络输出的7,7,30张量就可以“指定为”:

$$30 = num_{object} * (4+1) + num_{classes}$$

其实，这就要回到神经网络原理，神经网络**最开始-epoch-1**你用的肯定是初始值去前向传播，那肯定你的预测值和标签是天差地别，loss很大，但是经过后向传播的微调权重后，前向传播预测得越来越精准，loss越来越小。

这就是为什么说：“只要神经网络结构和loss函数设计得当，神经网络可以完成非常多的事情”。

没理解到的小伙伴多理解一下，如果实在没理解到，建议好好复习一下**反向传播和神经网络**。

2.如果你没有理解到：为什么yolov1规定的7,7,30的输出代表7，7个网格，可以这样想，yolov1的思想就是这样的，他把输入的图片进行下采样，“浓缩”为了7，7尺寸的feature、map。这也验证了：“CNN前向传播过程中，图片尺寸减小，语义信息越丰富”，——你看是不是encode后的7，7，30张量代表了很多信息呀~

了解了如何编码，那我们来讲一下encoder是如何在训练和预测中起到作用的以及如何进行编码：

1.在训练过程中，我们需要对y_train进行转换：

- 我们拿到手的标签是这种原始标签:

$$(bbox_num, cls, x, y, w, h)$$

```
chair 263 211 324 339
chair 165 264 253 372
chair 5 244 67 374
chair 241 194 295 299
chair 277 186 312 220
```

- 然后，由于我们输入CNN前对x_train进行了reshape，所以对bbox进行reshap，详细的reshape讲解在之前的文章有讲到哦。
- reshape完成后，我们yolov1一般是reshape为448,448——刚好是7的倍数，然后进行**bbox归一化**——使用x,y,w,h分别去除以448,448(x,w除以reshape_width,y,h除以reshape_height)
- 其次，进行**标签数字化**，这个图像分类经常干，我就不讲咯。到这里我们的y_train记为:

$$(bbox_num, cls_num, dx, dy, dw, dh)$$

- 再者，到了最关键的一步：将(cls,dx,dy,dw,dh)转换为(7,7,30)的y_train：

  我们先创建一个7,7,30(7,7个网格，每个网格30个信息)的全0张量label，用于存储数据。

  然后我们遍历每一个bbox，由于现在得到dx,dy都是物体中心点相对于整张图片的坐标，那么如果我可以得到这个物体所处的网格的左上角相对于整张图片的坐标:

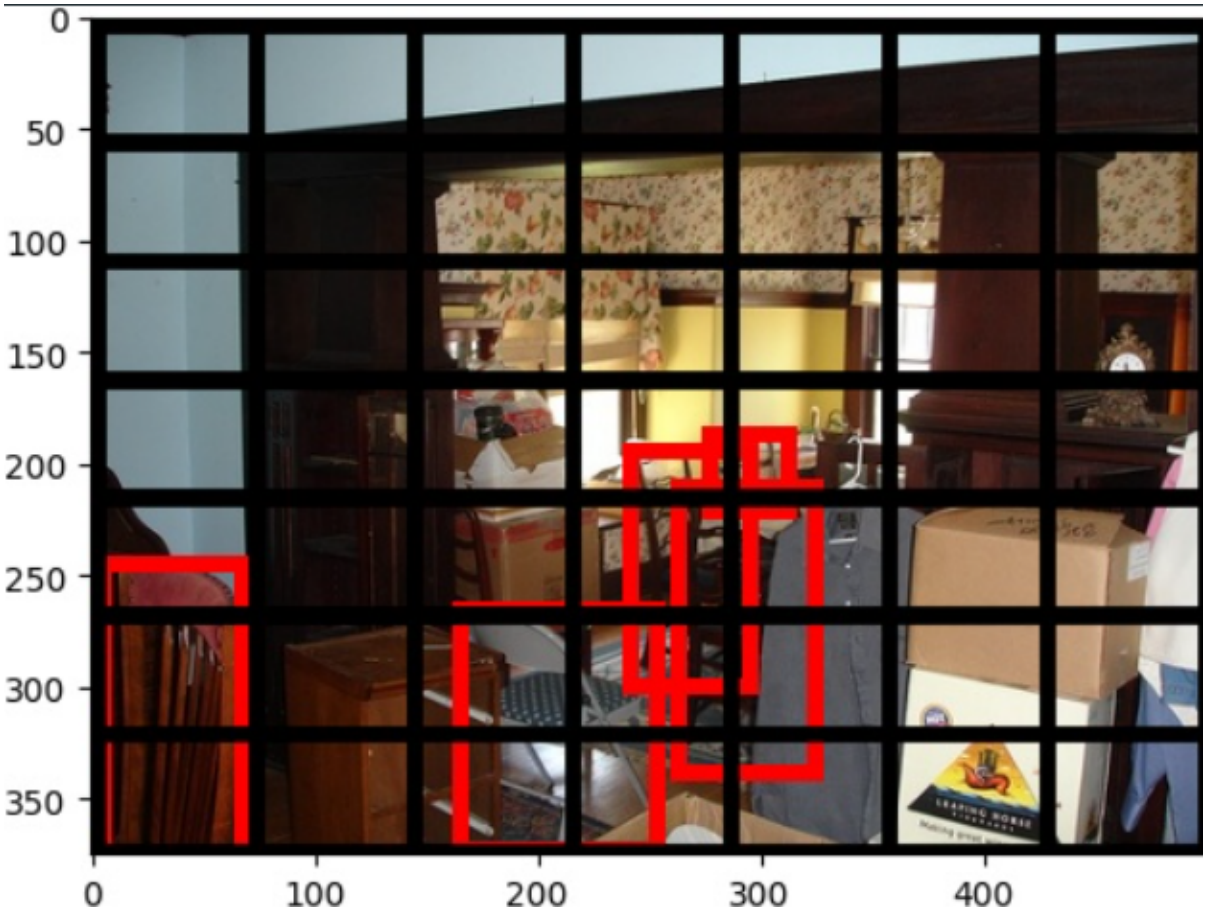
$$(dx_{min}, dy_{min})$$

  再把两者相减后除以网格大小是不是就是**物体中心点相对于网格的坐标-px,py**啦~

$$p_{x,y} = \frac{d_{x,y} - d_{xa,ya}}{gridsize_{x,y}} = \frac{d_{x,y}}{gridsize_{x,y}} - grid_{x,y}(grid_{x,y}\text{为网格在图片中的坐标})$$

  其实在计算(px,py,dw,dh)之前，我们就需要计算:

$$grid_{x,y}：\text{网格在图片中的坐标}(\text{从}0\text{开始哦 })$$



那么我们就在label[grid_x,grid_y](shape=(30,))这个网格中进行操作，将label[grid_x,grid_y,0:4]和label[grid_x,grid_y,5:9](也就是yolo每个网格需要预测/训练的两个object)存为(px,py,dw,dh)。

同时将label[grid_x,grid_y,4]和label[grid_x,grid_y,9]置为1——因为你在这儿标注物体，说明这里肯定有物体嘛，也就是存在物体概率=100%。

最后就是像上面说的，给一个种类列表，网格中的标注物体是哪一类，记类别名称在类别列表中的下标为idx，那么这num_classes的第idx+1位置数值为1表示有，其余为0表示否。也就是:label[grid_x,grid_y,10+idx]置为1.

- 最后，我们的标签就encode完成了。

2.在预测中，encoder过程和上面一样，但是值得注意的是：还是跟我在上面tips那里说的一样，你用初始化的权重去前向传播肯定刚开始loss非常大，所以各方面数值都比较“离谱”，这是正常的。

(3)decoder操作

decoder操作需要完成的就是在预测过程中，将网络输出的(n,30,7,7)先reshape为(n,7,7,30)——方便理解、处理，然后将转换为(98,25)的**信息矩阵**。

decoder过程其实就是encoder过程反向操作，要完成将每一个网格的30维信息转为25维度信息。

这个过程如果你搞清楚了encoder，那么就非常简单：

1. 首先，一样地初始化一个输出全0矩阵outputs(shape=(98,25))。
2. 然后，遍历每个网格(i,j)，利用encoder中(dx,dy,dw,dh)->(px,py,dw,dh)的推导公式反向推动将(px,py,dw,dh)转为(dx,dy,dw,dh)。**唯一数据要变的就是这个地方**。

3. 再者，我们来解决**如何存**的问题，这里就是**数据不变，存储方式变的地方**，是不是和encoder对应上啦？

   已知网格坐标(i,j)那么如果将这个7，7网格矩阵展平(7,7,30->98,25不就是展平吗？)，该网格下标为7*i+j。

   yolov1规定98个输出的bbox信息：偶数位置存储每个网格负责训练/预测的第一个物体信息，奇数位置存储每个网格负责训练/预测的第二个物体信息,也就是说:

   outputs[2(7i+j),0:4]=第一个物体信息,

   outputs[2(7i+j)+1,0:4]=第二个物体信息,

   outputs[2(7i+j),4]=第一个物体置信度,

   outputs[2(7i+j)+1,4]=第二个物体信息,

   outputs[2(7i+j),10+idx]=第二个分类信息,

   outputs[2(7i+j)+1,10+idx]=第二个分类信息,

4. 最后，就可以得到decoder结果啦。

(4)原理思想总结

可见，yolov1整体的训练策略就是输入一张图片，并非先验框，然后通过网络输出一个(7,7,30)的信息张量，最后拿预测值和label做回归，yolov1并非像以往的R-CNN一样，他最后预测出来的分类其实是分类概率，并不是跟传统图像分类一样直接是一个类别下标。

因此，物体分类和bbox回归在yolo训练中其实是**共享参数**。

三.网络设计和loss函数设计

有了思想原理，我们就得设计一个合理的网络结构和loss了。

(1)网络设计

刚刚我们已经提到了，我们要设计的以网络结构可以将输入图片矩阵转化为(n,30,7,7)的信息张量。

yolov1中，采用的是一个24个卷积层+2个分类linear层组成的CNN网络:

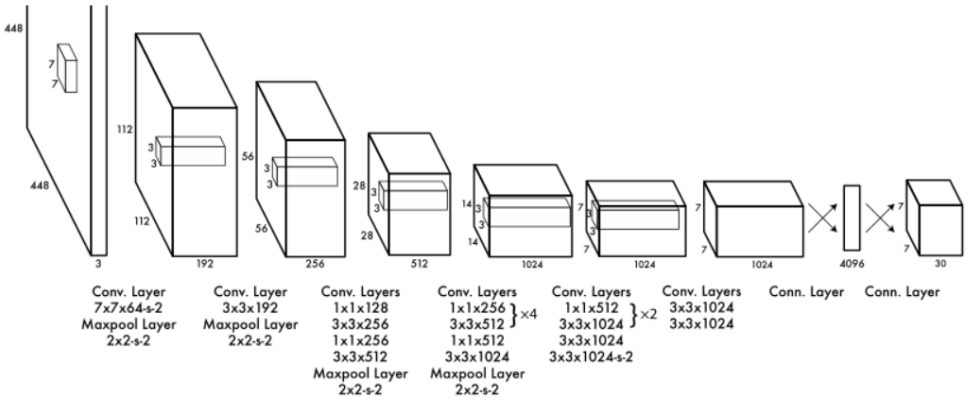


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

网络的每一个卷积层和第一个全连接层使用leakyrelu激活函数，最后一个全连接层使用sigmoid激活函数。

通过这个卷积神经网络可以完成5次下采样过程，输入224*224的图像，最后会变为尺寸为7*7的特征图。

事实上，yolov1网络结构采用的是resnet34(去除全连接层)+4个新增卷积层+2个新增全连接层。

(2)loss函数设计

yolov1中loss函数设计如下:

loss function:

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \qquad (3)
\end{aligned}
$$

这个公式由4部分组成:object_loss、classification_loss、confidence_loss、noobject_loss。

(1)object_loss

object_loss就是如果这个网格中有物体，那么计算其x,y的差平方相再相加、w,h的跟差平方再相加。

这里x，y使用的是典型的均方误差，而w、h使用的是根均分误差，我猜想的是，因为px、py要比dw、dh小一些，所以取平方根，让两者“重要性”都一样。

因为，如果一个训练参数值与其他的训练参数越大，可能在取梯度的时候，梯度值和其他的训练参数的梯度值相差较大，变化幅度可能也会比其他训练参数大，收敛时候不稳定。

(2)classification_loss

classification_loss是分类MSE均方误差，典型的图像分类误差，但是这里并没有用交叉熵误差函数，可能。。。是当时的发明人没有意识到？或者是为了保证端到端的完整性？

(3)confidence_loss

confidence_loss是置信度误差，也是使用的均方误差。。。

(4)noobject_loss

上面三个都是针对有问题的情况，那假如没有物体，就会将noobject记入。值得注意的是：yolov1论文还提出了loss权重的概念，就是上面图中的:

$$\lambda_{coord}、\lambda_{noobj}$$

前者为有物体的loss权重，后者为无物体的权重，设置不同权重有不同训练效果，论文中给定的最佳搭配是5,0.5。

值得一提的是，遍历每一个batch_size中的7,7,30和遍历每一个7,7,30的每个网格的时候，我们不是每个网格有两个预测框嘛，那么就选择与ground truth 的iou最大的进行loss计算，另外一个算入noobject_loss。

四.训练结果

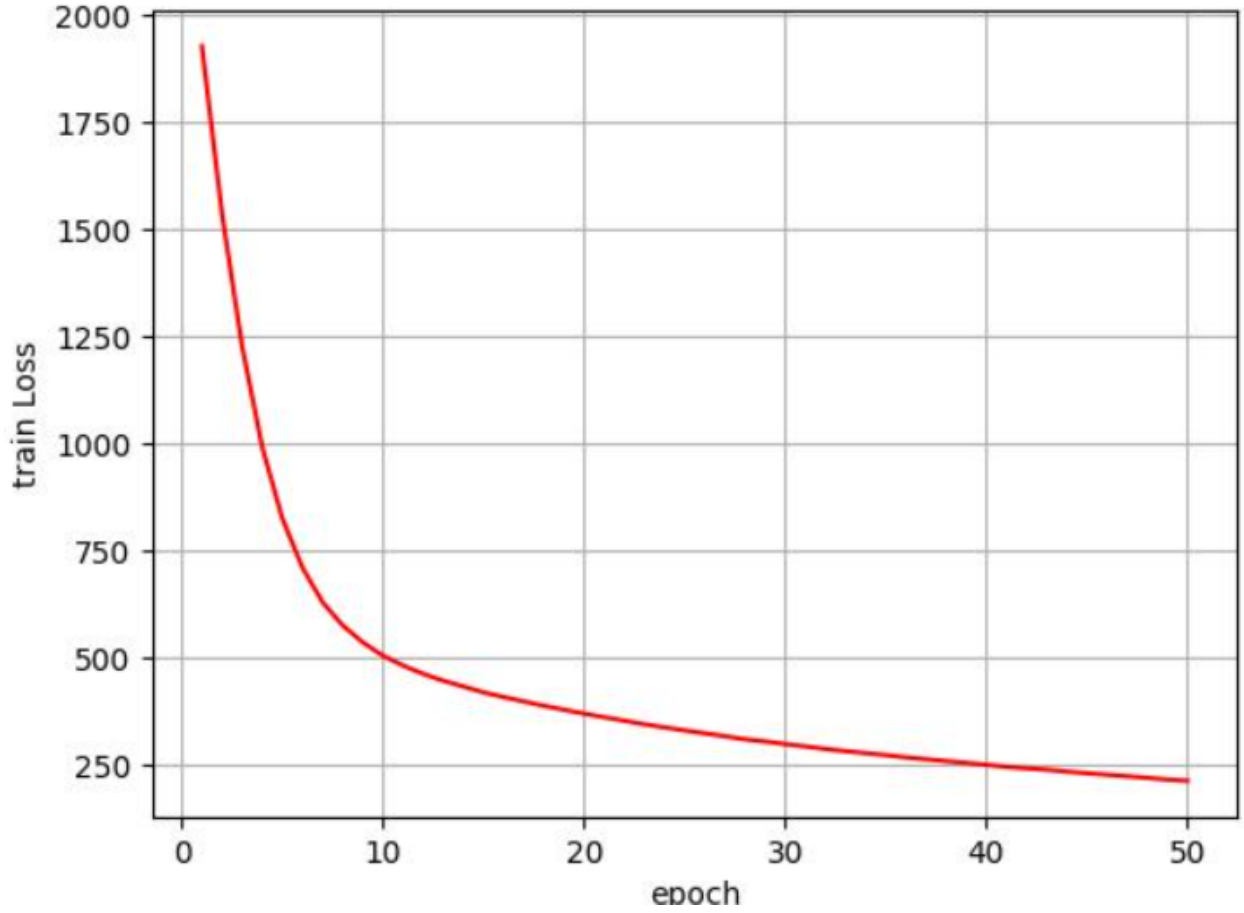
自己写了个log_generator记录了训练日志，根据训练日志分析情况。

第一阶段，额。。。我不小心把第一阶段的日志给删除了，反正当时琢磨了一会儿，主要是因为发现**loss中存在nan的问题并且loss降低很慢**，网上查找了原因有人都说是sqrt遇到负数的问题。但是，我后面想了一下，既然有leakyrelu函数在也不至于全是负数吧——因为当时我debug，**发现是object_loss矩阵在第三个step就全为nan**。

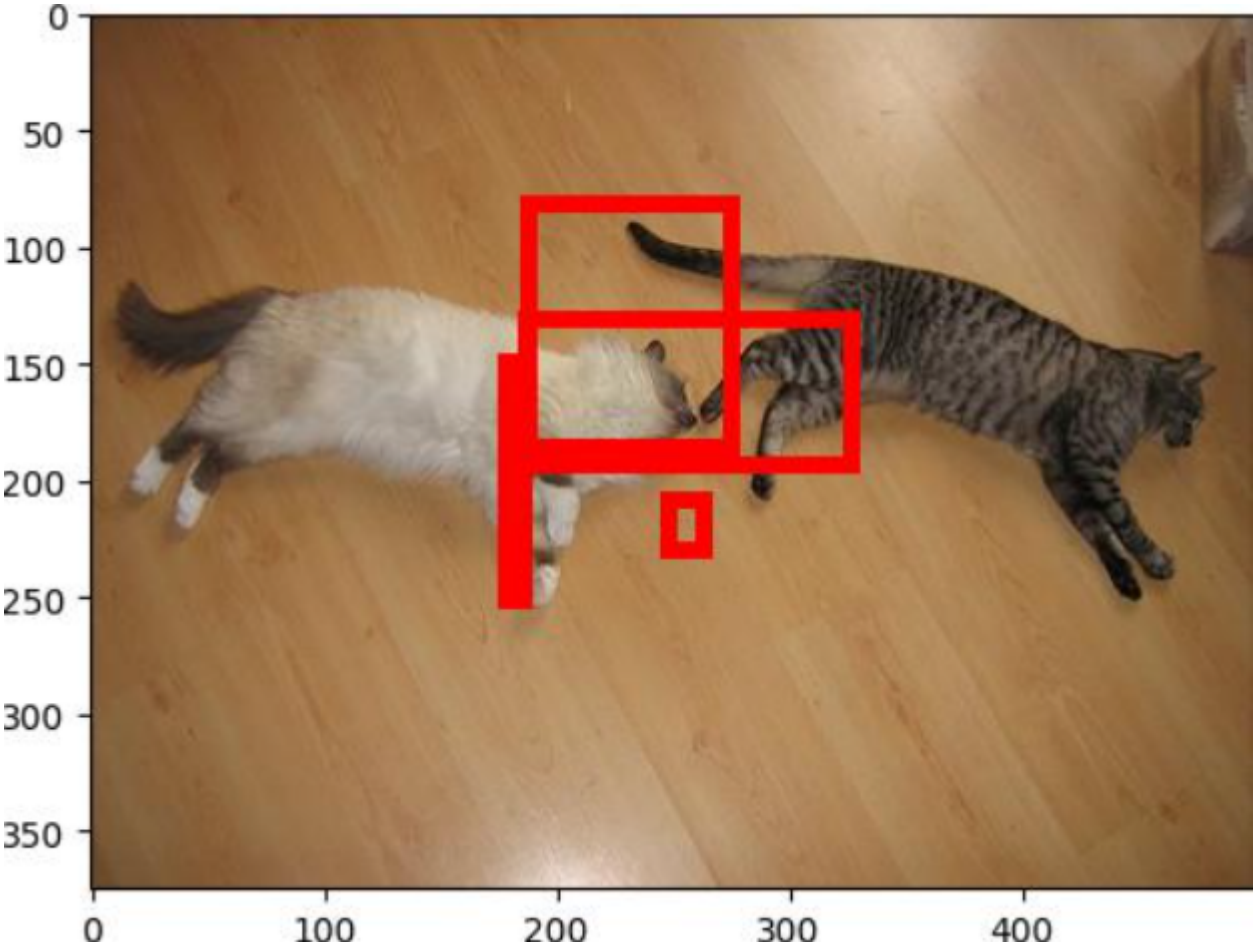
思考一下前向传播和后向传播？刚开始1、2step都没有nan，反而是预测值越来越小，那说明跟权值——特别是全连接层的权值脱不了干系，后面发现，确实全连接层的权值也是越来越小，说明可能是在反向传播微调参数的时候，**步子太大了**，再结合到网上说明的“目标检测的lr一般要比图像分类小一些”，所以将lr从0.001改为0.00001，nan不再出现。

第二阶段，这一阶段我尝试使用了纯resnet34和yolov1-resnet，反向yolov1-resnet的效果确实要好得多——主要是因为自己的电脑跑不了yolov1-resnet，后面去服务器上跑的。

第三阶段，调整训练参数并调整了数据处理策略。数据处理策略第一、二阶段我是使用**原始的、没有resize的图片的大小去归一化 bbox**，但是后面发现这种做法有点问题，于是改为了本文刚开始讲的encoder中的策略。



最后，训练的yolov1-loss稳定在了:25-30之前，但是预测效果并不尽人意：



五.总结

yolov1确实在训练思路上沿用图像分类端到端的训练策略并大幅度简化了训练难度、步骤，同时大幅度提高了预测速度，推荐新手入门目标检测在学习了R-CNN系列网络原理后，学习yolov1并进行复现。

yolov1的缺点在我看来：

- 首先就是对于重叠物体难以处理，yolov1的encoder和decoder原理就决定了yolov1处理重叠物体非常难受。
- yolov1本质还是使用了**overfeat的滑动窗口检测模型**的思维，但是yolov1速度非常快，随之而来的还有一个**遗传问题**——对于不同大小形状的物体，yolov1就难以同时解决，所以yolov3中提出了FPN(特征金字塔)。
- loss函数设计尚有缺陷。

但是在当时two-stage目标检测算法流行，特别是faster-RCNN流行的情况下，yolov1能够回归本质、开创新路子，已经是十分了不起了。