1 Yolov1

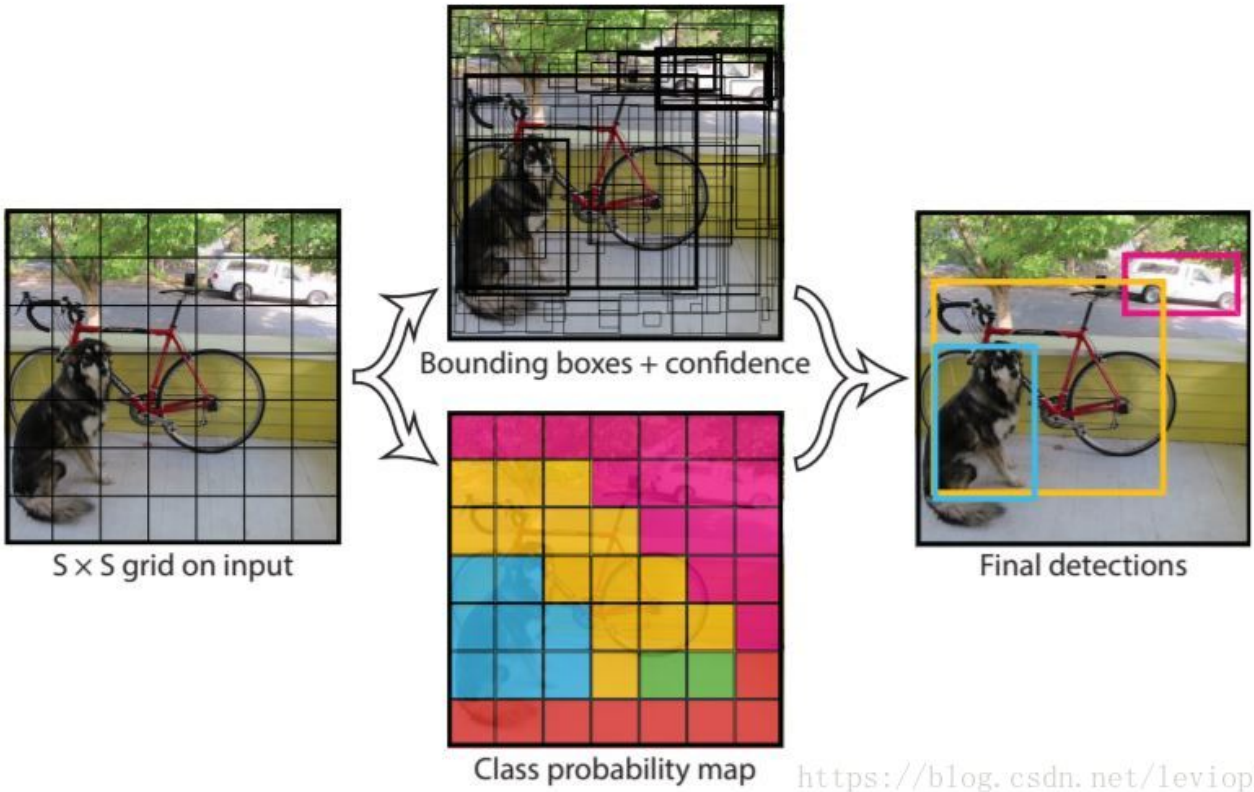
> 论文标题：《You Only Look Once: Unified, Real-Time Object Detection》
>
> 论文地址：https://arxiv.org/pdf/1506.02640.pdf
>
> 代码地址：https://github.com/SHU-FLYMAN/Yolov1-TensorFlow

1.1 Yolov1 **的核心**

Yolov1是Yolo系列的开山之作，以简洁的网络结构，详细的复现过程（作者给出详细教程）而受到CVer们的追捧。

> Yolov1**奠定了**Yolo**系列算法“分而治之”，粗暴回归的基调。**

在Yolov1上，输入图片被划分为 7×7 的网格，如下图所示，每个单元格独立作检测：



也就是划分为 $S \times S$ 个 cell，经过网络提取特征后输出：

- Bounding boxes x, y, w, h(后面会修正)
- Confidence 置信度
- Class probability 单元格的类别概率

综合每个cell的这些信息做了极大值抑制最终实现物体的目标检测，具体的细节会复杂一些。Yolo相比较其他目标检测算法，确实会难一点。但是后面很多One-stage都是在它基础上做的改进，如果要入门目标检测的话，我们不得不学习Yolo系列。

这里很容易被误导：

> 每个单元格的视野有限，而且很可能只有局部特征。这样就很难理解Yolo为何能检测到比 **网格单元** 大很多的物体。

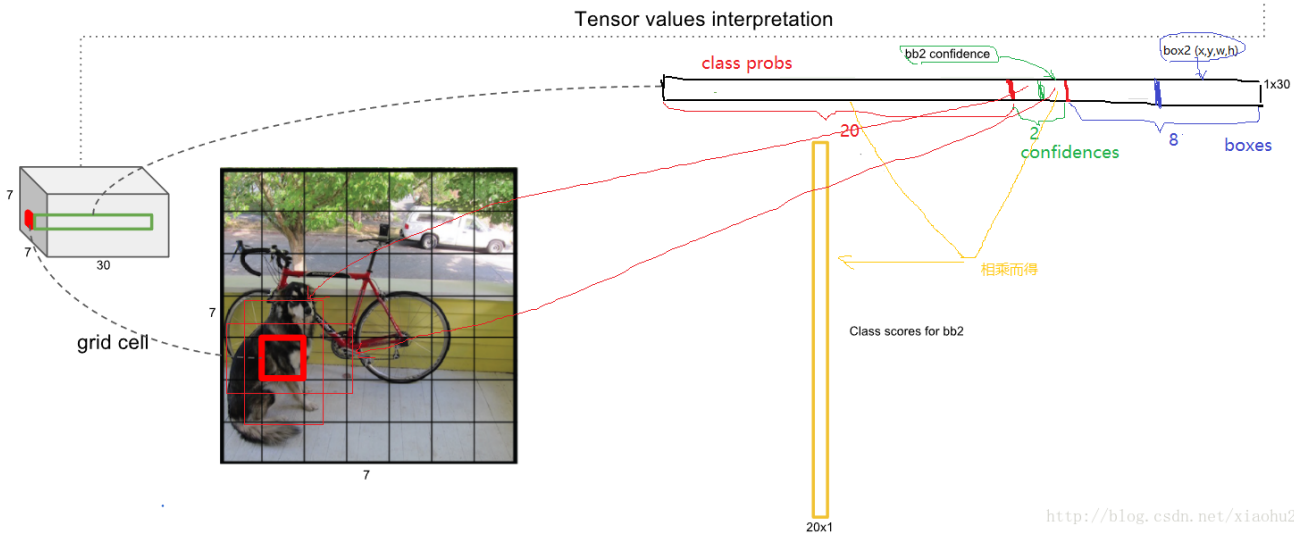
其实Yolo的做法并不是把每个单独的网格作为输入feed到模型中，在Inference的过程中，网格只是起到物体中心点位置的划分之用，并不是对图片进行切片，不会让网格脱离整体的关系。

请记住，上面这句话是整个yolo的核心，我们的网格只是起到物体中心点位置的划分之用，而不是把图片切成一块一块喂给神经网络。具体的如何操作我们后面会仔细说。

1.2 Yolo的输入输出

先说网络的的输入输出，训练过程的做法如下：

我们的输入是448*448**的图片，经过神经网络，每个网格** cell **预测** B(B=2) **个**boxes(X-center，Y-center，W，H，Confidence) **和 类别**class(20**个类别)，也就是网络对每个网格有**30**个输出。那么整个特征图将为**7 x 7 x 30 **维度的** Tensor：



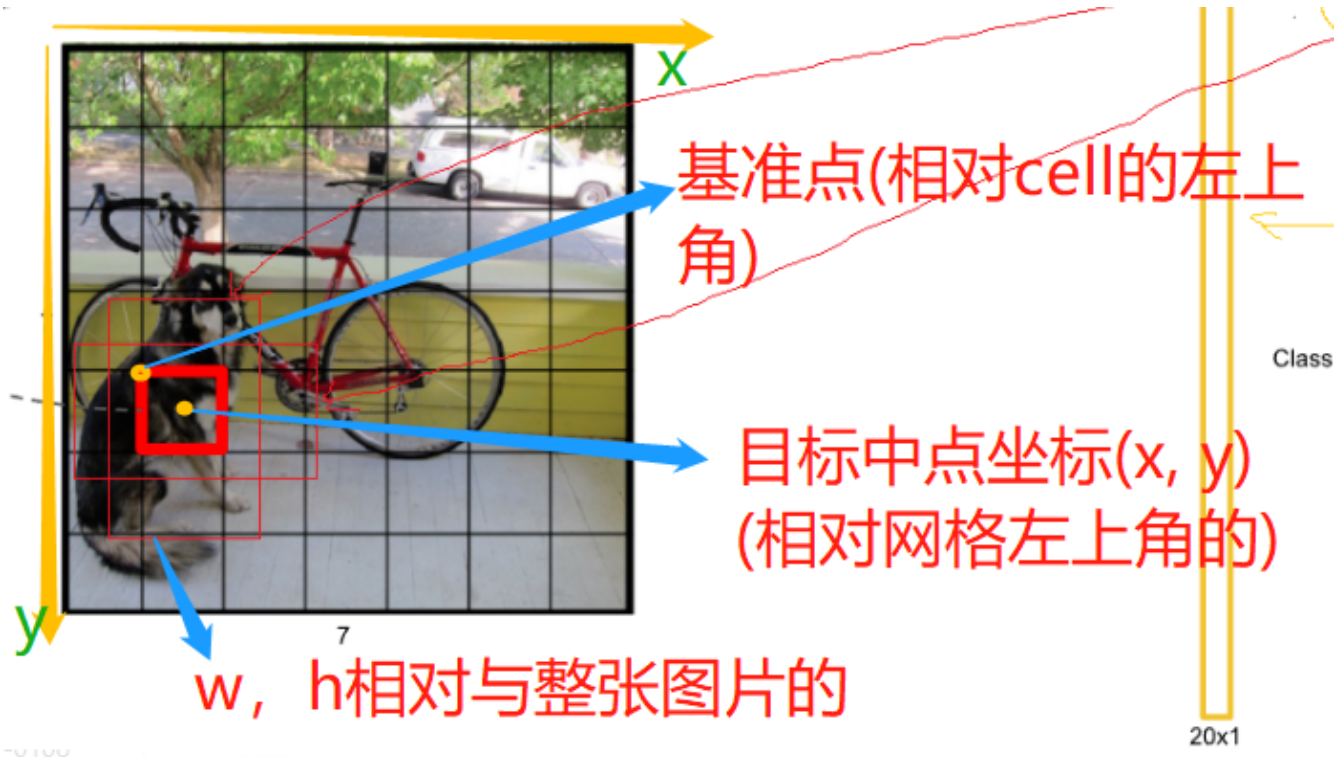
为了便于理解，我们**暂且这样认为训练过程中网络的每个**cell**的输出是下面这样的。**主要包括以下3个输出:

1.2.1 候选框坐标

1. x, y

模型预测的 x, y 为相对于到某个cell的左上角 的 0-1 比例（基准为cell的大小）。因为只有object落到这个cell中心，才会由这个cell负责，因此中心范围是在0-1之间，我们期望模型预测这个基于cell的偏移量。

这里要说一下坐标系，不同于一般的数学坐标系，我们用的坐标系其y轴方向向下，为h的方向，x的方向则为正常坐标系：



注意网络预测的x,y**只是偏移量坐标，例如第**2**行第二列的**grid**的一个**box**坐标预测是**0.2, 0.5**，实际上坐标是**(1+0.2, 1+0.5)/7**归一化到原图尺寸的坐标，之后我们与真正的**label**处理时会将其转换为全局坐标系，但是我们希望网络预测出的即是这个**object**相对于**cell**左上角的偏移量。**

2. w, h

我们期望预测出的w, h为相对于整个图像 0-1 范围的值。

而关于模型的坐标标签数据，我们后面再仔细说。

1.2.2 候选框置信度(Confidence)

Confidence作者的本意是想要预测出这个cell中是否存在object，正如做label时候，对应位置的标签就代表这个框内是否含落有某个object的中心，则标记为1。这个标签称为 Response 标签，主要负责标记这个cell中是否有Object。

但如果直接用该值作为预测出的Confidence的回归目标，其实也还OK，但是总觉得缺少一点东西，整个网络只要预测出这个cell里是否有东西就OK了，跟预测出来的框和原始Object的重合度没有任何关系。

> 举个例子，预测出来的Box离原来的object很远，但是仍旧在这个cell里面。在坐标损失中，预测Box离得很远会被惩罚，但在Confidence损失中，竟然没有受到任何惩罚。

因此作者引入IOU，即乘以label中的object跟预测出的object的交并比来替代作为object的Confidence标签。

> 我们希望网络拟合出的是预测结果的真实可能性。如果你预测离得很偏的话，这个IOU就会很小，因此，我们将response标签0-1值乘以实时计算的IOU来替代作为检测出的object的置信度标签。

因此最终，我们希望网络拟合出Confidence置信度包含两重信息，计算公式如下：

$$Confidence_{label} = Reponse_{(Object_{0-1})} \times IOU_{pred}^{truth} \tag{1}$$

综合起来，似乎是有点像我们预测出结果的置信度。因为它包含两重信息：

- 存在object的概率；
- 其次是，如果预测很偏，那么这个可信度确实是不怎么高。

但记住，我们做label时候Confidence对应位置的Response标签为0-1，只代表这个cell中是否有Object，最终我们希望网络拟合的并不是这个值，而是要乘以相应的IOU。

1.2.3 类别(classification)

如果这个cell的中心落有object，这个网格就会用one-hot标签标记出这个目标的类别，如果不含有目标，则为空。我们网络输出的也是用softmax归一化的概率值，代表整个cell属于某个类别的概率。

有一个需要注意的，真实情况下，每个cell中可能有多个类别的Object，但是Yolo认为整个cell里面只含有一个类别。因此每个cell只预测一组类别，也就是我们的预测值中，每个cell只有一个20维的class，代表整个cell属于哪个类别的概率，跟该cell有多少个boxes无关。

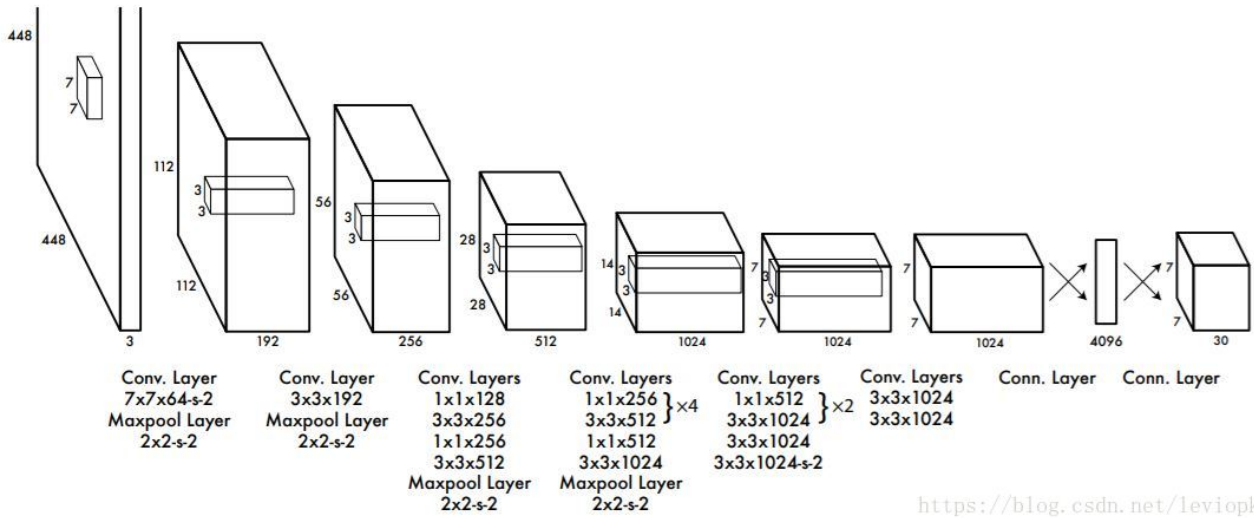
做标签的时候，也许一个cell中可能有2个类别，我们的处理是把第二个物体直接忽略掉，即不标记第二个类别。反正这种情况并不多见，在Yolov3对此做了改进，就让我们暂且视而不见吧，谁还没有疏漏呢？

1.2.4 网络结构

总结一下我们训练过程中最终的输出大小：

$$（7 \times 7）.cells \times （（4.coords + 1.confidence） \times 2.boxes + 20.classes） = 7 * 7 * 30 \tag{2}$$

具体的网络结构：




```python
def build_network(self,
                  images,
                  output_size,
                  scope='yolo_net'):
    with tf.variable_scope(scope):
        with slim.arg_scope(
                [slim.conv2d, slim.fully_connected],
                activation_fn=YOLONET.leaky_relu(self.leaky_alpha),
                weights_regularizer=slim.l2_regularizer(0.0005),
                weights_initializer=tf.truncated_normal_initializer(0.0, 0.01)):
            net = tf.pad(images,
                         np.array([[0, 0], [3, 3], [3, 3], [0, 0]]),
                         name='pad_1')
            net = slim.conv2d(net, 64, 7, 2, padding='VALID', scope='conv_2')
            net = slim.max_pool2d(net, 2, padding='SAME', scope='pool_3')
            net = slim.conv2d(net, 192, 3, scope='conv_4')
            net = slim.max_pool2d(net, 2, padding='SAME', scope='pool_5')
            net = slim.conv2d(net, 128, 1, scope='conv_6')
            net = slim.conv2d(net, 256, 3, scope='conv_7')
            net = slim.conv2d(net, 256, 1, scope='conv_8')
            net = slim.conv2d(net, 512, 3, scope='conv_9')
            net = slim.max_pool2d(net, 2, padding='SAME', scope='pool_10')
            net = slim.conv2d(net, 256, 1, scope='conv_11')
            net = slim.conv2d(net, 512, 3, scope='conv_12')
            net = slim.conv2d(net, 256, 1, scope='conv_13')
            net = slim.conv2d(net, 512, 3, scope='conv_14')
            net = slim.conv2d(net, 256, 1, scope='conv_15')
            net = slim.conv2d(net, 512, 3, scope='conv_16')
            net = slim.conv2d(net, 256, 1, scope='conv_17')
            net = slim.conv2d(net, 512, 3, scope='conv_18')
            net = slim.conv2d(net, 512, 1, scope='conv_19')
            net = slim.conv2d(net, 1024, 3, scope='conv_20')
            net = slim.max_pool2d(net, 2, padding='SAME', scope='pool_21')
            net = slim.conv2d(net, 512, 1, scope='conv_22')
            net = slim.conv2d(net, 1024, 3, scope='conv_23')
            net = slim.conv2d(net, 512, 1, scope='conv_24')
            net = slim.conv2d(net, 1024, 3, scope='conv_25')
            net = slim.conv2d(net, 1024, 3, scope='conv_26')
            net = tf.pad(net,
                         np.array([[0, 0], [1, 1], [1, 1], [0, 0]]), name='pad_27')
```

```
                net = slim.conv2d(net, 1024, 3, 2, padding='VALID', scope='conv_28')
                net = slim.conv2d(net, 1024, 3, scope='conv_29')
                net = slim.conv2d(net, 1024, 3, scope='conv_30')
                # NhWC变为NCHW,以矩阵最后个axis为方向展开
                net = tf.transpose(net, [0, 3, 1, 2], name='trans_31')
                net = slim.flatten(net, scope='flat_32')
                net = slim.fully_connected(net, 512, scope='fc_33')
                net = slim.fully_connected(net, 4096, scope='fc_34')
                net = slim.dropout(net, keep_prob=self.keep_prob,
                                   is_training=self.is_training,
                                scope='dropout_35')
                net = slim.fully_connected(net,
                                           output_size,
                                           activation_fn=None,
                                           scope='fc_36')
                # [Batch, 7, 7, 30] reshape 预测结果
                net = tf.reshape(net, name='predicts', shape=[None, self.cell_size, self.cell_size,
                                                              5 * self.boxes_per_cell +
self.class_num])
        return net
```

最后接入Loss层进行损失计算，由反向传播梯度计算来强行predict出这些位置、置信度等，这是训练的过程。有一点需要明确:

> 最后的$7 \times 7 \times 30$ 特征图上每个30维的特征和原来图像是有一一对应关系的，可以认为，其实一个cell的信息都包含在最后特征图的一个$7 \times 7 \times 30$的一个cell 的 Tensor中。

1.2.5 Summary

Predict的时候，过程会有些不一样，后面我们会说。记住Inference的重点(暂时这样认为，会不大一样，但为了方便大家理解):

> - 我们训练过程中网络输入为448 x 448的图片，输出为$[batch \times 7 \times 7 \times 30]$。
> - x, y, w, h ，我们预测出 x, y为相对于到 某个cell **的左上角** 的 0-1 比例，w, h是相对于整张图片的大小的比例。有2个bounding box，因此有2个 坐标。
> - Confidence对应位置的label为0-1值，但是我们希望网络拟合出的是$prob \times IOU$，因此我们需要喂给损失函数的时候做一个实时的转换。同理有2个预测的置信度。
> - 因为我们假设整个cell属于同一个类别，因此我们只有一个Classification，为20维度。

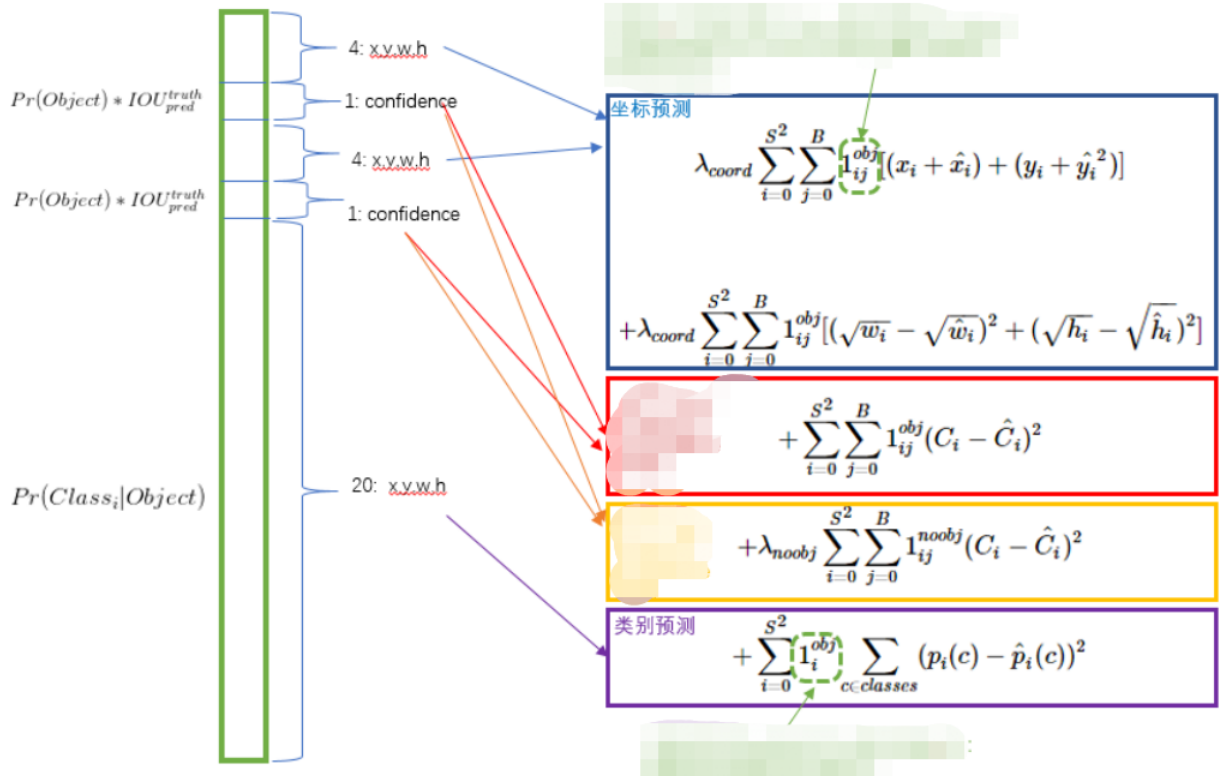
上面就是我们Inference的所有重点，下面我们讲损失函数，借助损失函数，我们说明修改后的网络输出我们希望它拟合什么。

1.3 Yolo的损失函数设计考量

下面我们讲我们的损失函数。请记住，Yolo的损失函数包括三部分，分别是:

- 位置坐标 损失 x, y, w, h
- 置信度损失 Confidence loss
- 类别损失 Classification loss

我们需要在这三部分损失函数之间找到一个平衡点。Yolo设计的损失函数公式如下(我将box分开了，方便大家看，实现的时候是取前Inference出的30维Tensor前8个为box坐标，后2个为Confidence，最后20个为整个cell的类别概率，现在看不懂没有关系，计算的时候我们将会说如何实现这个损失函数):



我们如何在三部分损失函数之间找到平衡点？YOLO主要从以下4个方面考虑:

1.3.1 坐标损失函数与类别损失函数平衡

> 每个小区域输出的8维位置坐标偏差的权重应该比20维类别预测偏差的权重要大。

1. 因为从体量上考虑，20维的影响很容易超过8维的影响，导致分类准确但位置偏差过大。
2. 再者，最后会在整体的分类预测结果上综合判断物体的类别（极大值抑制等）。因此单个小区域的分类误差稍微大一些，不至于影响最后的结果。

因此最终设置：

> **位置坐标损失** 和 **类别损失函数** 的权重比重为 5 : 1。

1.3.2 置信度损失函数内部权重

> 我们需要减弱不包含object的网格对网络的贡献。

在不包含有目标物体的网格中，cell的标签置信度为0。但是图片上大部分区域都是不含目标物体的，而我们网络有object的cell对应位置的标签为1，注意我们喂给损失函数作为label的Confidence值是实时计算的，也就是$1 \times IOU$。

这些置信度为0的标签对梯度的贡献会远远大于含有目标物体对应位置标签为1的网格对梯度的贡献，这就容易导致网络不稳定或者发散。换句话说，网络会倾向于预测每个cell都不含有物体。因为大部分情况下，这种情况是对的。所以需要减弱不包含object的网格的贡献。

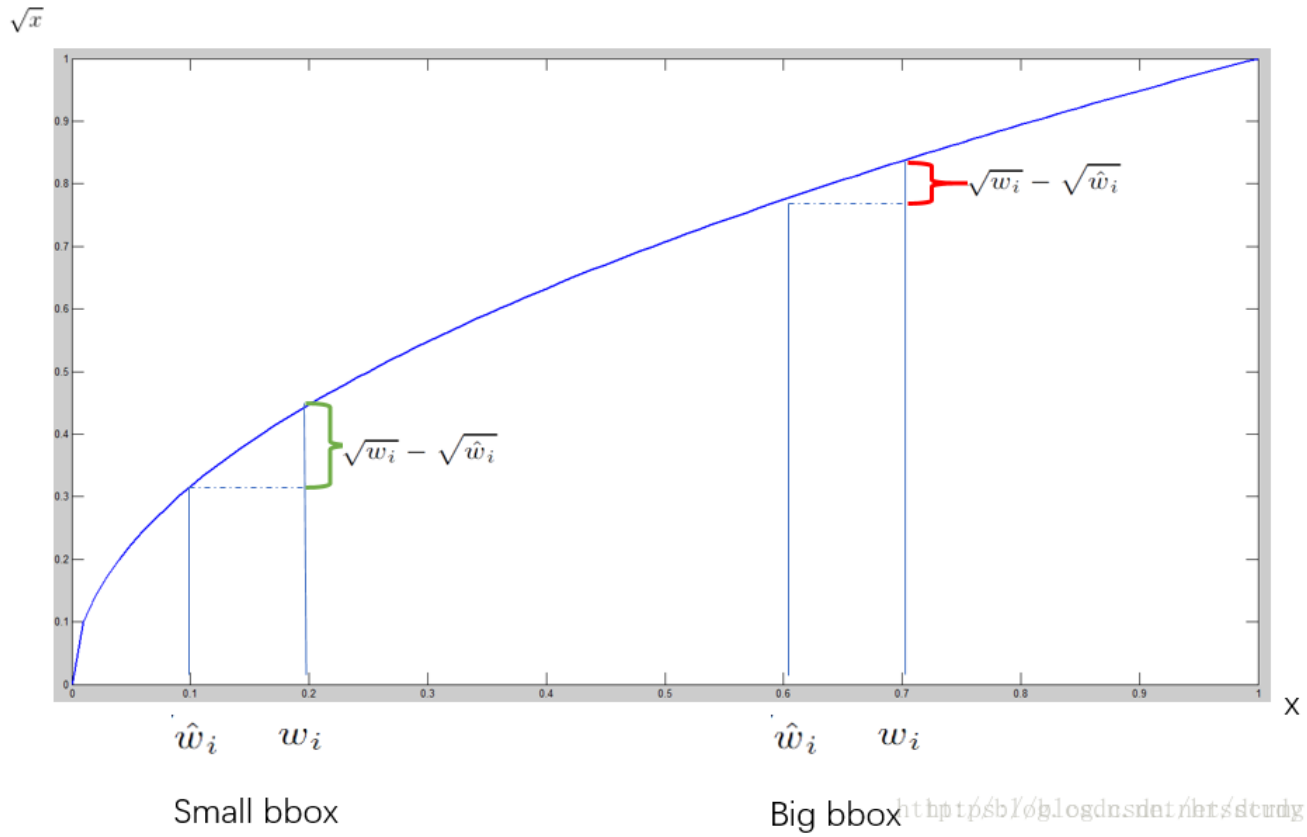
因此最终设置：

> 在**不包含** object 的cell计算损失时，**置信度损失**的**权重为**0.5，在**包含** object**时，权重取**1。

1.3.3 物体目标大小有区别

考虑到目标物体有大有小，对于大的物体，坐标预测存在一些偏差无伤大雅；但对于小的目标物体，偏差一点可能就是另外一样东西了。

下图可见，对于水平方向上同等尺度的增量，其x越小，其取根号之后产生的偏差越大。如图绿色段明显大于红色段。 换句话说，在小目标检测上，如果错了一点，损失函数会很大。而在大目标上，就算错了同样的量，损失函数也不会很大。



因此最终设置：

> 将位置坐标损失中 w 和 h 分别取 平方根 来 代替 原本的 w 和 h。

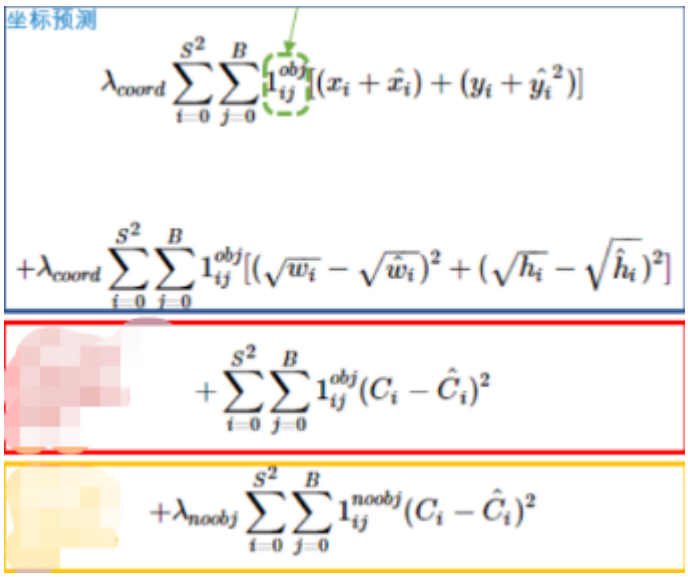
也就是说，我们网络Inference推断出原来为w, h的位置，我们希望网络拟合出的是w, h的根号值。那我们怎么做？其实很简单，计算损失函数的时候，我们将网络Inference出的Tensor取个平方再和真实标签的w, h做计算。这样拟合的即是object的根号值。

这样一部分解决了目标大小有偏差的问题，不过也没有从根本上解决问题，之后我们会看后面Yolov3怎么做改进。

1.3.4 是否每个bounding box 信息都需要

我们来思考一下，这里每个cell会预测出2个bounding box，也就是有两个Confidence和两个坐标的信息。但仔细思考一下，我们是否两个bounding box都是需要的？我们思考一下。

我们先来看坐标损失函数和置信度损失函数：



首先，对于坐标损失来说，其实我们预测的时候经过非极大值抑制只会保留一个Bounding box，那么对于第二个bounding box来说，如果我们将其加入到损失函数的训练，反而不能让网络专心与优化最好的那个Bounding box。显然易见，对于坐标损失，我们只需要一个bounding box，而且我们不需要优化cell中没有object的预测Bounding box：

> 我们只计算cell中有**目标**，并且IOU**最高**的预测出的bounding box的坐标损失，对于其他的bounding box，我们不考虑。

但是对于置信度来说，情况就有些复杂了。初初考虑，似乎回归两个置信度对于网络来说都有助于网络判断出整个cell是否有object，因为最后预测的时候，我们需要用 **置信度** 乘以 **类别概率**来计算类别置信度，经过极大值抑制等方法选出最终的Bounding box。

但是我们想一下，问题在于，这个置信度是怎么拟合出来的？它需要用到IOU，而IOU需要用到坐标。**第二个Bounding box的坐标我们没有回归！**也就是说，我们如果要让它去拟合认为第二个Bounding box是有object的是有问题的，关键在于这个置信度计算用到的IOU不可信！

那我们能像坐标损失一样，这个不可信的第二个置信度就不去回归吗？显然不行。我们最终需要挑出一个置信度最高的 Bounding box，**因此我们要让第二个不可信预测的置信度的值尽可能降低。**

也就是让网络认为，第二个不好的 Bounding box**是不存在**object，**这样有利于我们筛选出最好的**bounding box。

还不能理解吗？相比较一些Two-stage目标检测算法，Yolo确实有点难以理解，但记住Yolo的做法:

> 我们有很多的备胎(Bounding box)，但是我们只保留最好的那个Bounding box优化。

这样而来，其实置信度损失的处理和坐标损失是相类似的，总结一下：

> - 对于有object，确切说是保留下来的bounding box，我们正常计算坐标损失和置信度损失。
> - 对于没有保留下来的bounding box，对于坐标损失来说，我们不进行回归，也就是乘以0，而置信度损失，我们需要它认为整个cell中是否有object。

我们再强调一下：

> 尽管有object的cell中第二个bounding box事实上是有object的，讲道理是应该让它去拟合认为整个cell是有object的，但是由于这个bounding box的坐标没有进行回归，也就是作为计算出来作为置信度标签的IOU不可信，因此我们应该避免网络选到这个不可信的bounding box(通过置信度），也就是我们需要它去拟合认为自己没有object。

我们通过一个称为 **目标掩模** (类似于光刻掩模)的 变量去筛选这个信息。后面我们详细讲这个 **目标掩模如何计算。**

最终总结：

1. 对于坐标bounding box，我们只回归IOU最高的Box中。也就是认为只有这个Box中有Object，其余的没有。
2. 对于置信度，我们同样认为只有IOU最高的Box中有Object，因为权重不同，因此我们需要掩模来区别运算有目标的object以及没有目标的object。

1.3.4 Summary

主要有一下几点：

> 1. **位置坐标损失** 和 **类别损失函数** 的权重比重为 5 : 1。
> 2. 在**不包含** object 的cell计算损失时，**置信度损失**的**权重为**0.5，在**包含** object**时，权重取**1。
> 3. 将位置坐标损失中预测出的值 **取平方**，这样预测出的值就取到了目标 w, h 的 平方根。
> 4. 虽然有2个Bounding box，但我们只认为**最高**IOU的 box 是有object的。通过 **目标掩模** 来筛选这个信息。

1.4 编写Yolo的准备工作

现在我们需要制作Label，虽然还不知道损失函数具体怎么计算，但是我们也可以先做Label，主要有以下几点：

> 1. 因为我们假设一个cell中只有一个object，所以我们只有一个Bounding box，算上类别标签，所以维度是25。
> 2. 其次我们制作标签的时候，直接按Yolo格式进行标注比较麻烦，因为大多数数据增强包只支持传统的`x1, y1, x2, y2`格式的对应转换。

因此我们在最后计算损失函数的时候才会做一个转换，将其转换为所需要的格式。

具体原因，主要是为了方便，总而言之，我们的标签仅仅是VOC格式的图片数据集，下面我们提供了脚本`download.sh`,在Linux环境下运行下面脚本即可：

```
sh download.sh
```

我们下载的数据即是VOC2007，后面我们代码实现会说怎么做这个转换。总而言之，最后喂给我们的loss层的label是这样的，为一个`[Batch, 7, 7, 25]`的标签格式，主要有以下几点：

> 1. `7, 7`，为了保证跟图像存储的格式是一致的，`axis=1`为`h`的方向，`axis=2`为`w`的方向，这在后面计算损失的时候会用到。
> 2. x, y是基于全图坐标系0-1范围的值，w, h同样也是，另外我们并没有将其做根号处理。因此计算损失函数的时候，我们需要做个转换。
> 3. confidence对应位置Response标签为0-1值，代表是否有object，喂给损失函数的Confidence需要临时计算。其余20维度的为类别one-hot标签。

具体的转换我将代码贴出来，我们认为你能够熟练使用Tensorflow，这里我们用了`tf.data`进行数据读取，数据增强用了`imgaug`库，虽然会损失一点性能，但是它的Bounding box可以跟着一起变化，我们步骤如下:

1. 解析VOC，将信息未做任何变换写为`tfrecords`二进制公式。
2. 读取`tfrecords`，利用`imgaug`进行数据增强及`reshape`为448。
3. 将标签转换为Yolo格式
4. 检查数据增强是否发生错误

但我们没有给图像除以255.0使其范围在0-1之间，TensorFlow比较麻烦一点在于最后构建完整个计算图才能够进行Debug，比如查找Nan值等，但其实很多时候，我们写TensorFlow程序，能不能跑通整个流程还两说，实在吐血地难用。两个类的关键在于`transform`函数。

1.4.1 解析voc数据，制作Tfrecords

```
import os
import numpy as np
import imgaug as ia
import tensorflow as tf
import matplotlib.pyplot as plt
import xml.etree.ElementTree as ET

from tqdm import tqdm
from imgaug import augmenters as iaa

from configs import CLASS, Class_to_index, Colors_to_map  # 一些不发生改变的全局信息


class To_tfrecords(object):
    def __init__(self,
                 load_folder='data/pascal_voc/VOCdevkit/VOC2007',
                 txt_file='trainval.txt',
                 save_folder='data/tfr_voc'):
        self.load_folder = load_folder
        self.save_folder = save_folder
        self.txt_file = txt_file
        self.usage = self.txt_file.split('.')[0]
        if not os.path.exists(self.save_folder):
            os.makedirs(self.save_folder)
        self.classes = CLASS
        self.class_to_index = Class_to_index

    def transform(self):
        # 1. 获取作为训练集/验证集的图片编号
        txt_file = os.path.join(self.load_folder, 'ImageSets', 'Main', self.txt_file)
        with open(txt_file) as f:
            image_index = [_index.strip() for _index in f.readlines()]

        # 2. 开始循环写入每一张图片以及标签到tfrecord文件
        with tf.python_io.TFRecordWriter(os.path.join(
                self.save_folder, self.usage + '.tfrecords')) as writer:
            for _index in tqdm(image_index, desc='开始写入tfrecords数据'):
                filename = os.path.join(self.load_folder, 'JPEGImages', _index) + '.jpg'
                xml_file = os.path.join(self.load_folder, 'Annotations', _index) + '.xml'
                assert os.path.exists(filename)
                assert os.path.exists(xml_file)

                img = tf.gfile.FastGFile(filename, 'rb').read()
                # 解析label文件
```

```
                label = self._parser_xml(xml_file)

                filename = filename.encode()
                # 需要将其转换一下用str >>> bytes encode()
                label = [float(_) for _ in label]
                # Example协议
                example = tf.train.Example(
                    features=tf.train.Features(feature={
                    'filename': tf.train.Feature(bytes_list=tf.train.BytesList(value=[filename])),
                    'img': tf.train.Feature(bytes_list=tf.train.BytesList(value=[img])),
                    'label': tf.train.Feature(float_list=tf.train.FloatList(value=label))
                    }))
                writer.write(example.SerializeToString())

    def _parser_xml(self, xml_file):
        tree = ET.parse(xml_file)
        # 得到某个xml_file文件中所有的object
        objs = tree.findall('object')
        label = []
        for obj in objs:
            """
            <object>
                <name>chair</name>
                <pose>Rear</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                    <xmin>263</xmin>
                    <ymin>211</ymin>
                    <xmax>324</xmax>
                    <ymax>339</ymax>
                </bndbox>
            </object>
            """
            category = obj.find('name').text.lower().strip()
            class_id = self.class_to_index[category]

            bndbox = obj.find('bndbox')
            """
            <bndbox>
                <xmin>263</xmin>
                <ymin>211</ymin>
                <xmax>324</xmax>
                <ymax>339</ymax>
            </bndbox>
            """
            x1 = bndbox.find('xmin').text
            y1 = bndbox.find('ymin').text
            x2 = bndbox.find('xmax').text
            y2 = bndbox.find('ymax').text
            label.extend([x1, y1, x2, y2, class_id])
        return label
```

1.4.2 数据增强，制作Dataset

```
class Dataset(object):
    def __init__(self,
                 filenames,
                 batch_size=32,
```

```
                enhance=False,
                image_size=448,
                cell_size=7):
        self.filenames = filenames
        self.batch_size = batch_size
        self.enhance = enhance
        self.image_size = image_size
        self.cell_size = cell_size
        if self.enhance:
            self.seq = Dataset._seq()

    def transform(self):
        dataset = tf.data.TFRecordDataset(self.filenames)
        dataset = dataset.map(Dataset._parser)
        # 数据对图片以及标签进行处理 应用Python逻辑
        dataset = dataset.map(map_func=lambda image, label: tf.py_func(func=self._process, inp=[image,
label], Tout=[tf.uint8, tf.float32]), num_parallel_calls=8)
        dataset = dataset.shuffle(buffer_size=100)
        dataset = dataset.batch(self.batch_size).repeat()
        return dataset

    # 对图像进行处理
    def _process(self, image, label):
        label = np.reshape(label, (-1, 5))
        label = [list(label[row, :]) for row in range(label.shape[0])]
        bbs = ia.BoundingBoxesOnImage([ia.BoundingBox(x1=x1, y1=y1, x2=x2, y2=y2, label=class_id) for
                                       x1, y1, x2, y2, class_id in label], shape=image.shape)
        # 1. 数据增强
        if self.enhance:
            image, bbs = self._aug_images(image, bbs)
        # 2. 图像resize
        image, bbs = self._resize(image, bbs)
        # 3. 制作yolo标签
        label = self._to_yolo(bbs)
        return image, label

    def _to_yolo(self, bbs):
        """

        Args:
            bbs:#标记类别，pascal_voc数据集一共有20个类，哪个类是哪个，则在响应的位置上的index是1

        Returns: [7, 7, 25]

        """
        label = np.zeros(shape=(self.cell_size, self.cell_size, 25), dtype=np.float32)

        for bounding_box in bbs.bounding_boxes:
            x_center = bounding_box.center_x
            y_center = bounding_box.center_y
            h = bounding_box.height
            w = bounding_box.width
            class_id = bounding_box.label
            x_ind = int((x_center / self.image_size) * self.cell_size)
            y_ind = int((y_center / self.image_size) * self.cell_size)
            # 对每个object,如果这个cell中有object了,则跳过标记
            if label[y_ind, x_ind, 0] == 1:
                continue
            # 1. confidence标签(对每个object在对应位置标记为1)
            label[y_ind, x_ind, 0] = 1
```

```
            # 2. 设置标记的框，框的形式为(x_center, y_center, width, height)
            label[y_ind, x_ind, 1:5] = [coord / self.image_size for coord in [x_center, y_center, w,
h]]
            # 3. 标记类别，pascal_voc数据集一共有20个类，哪个类是哪个，则在响应的位置上的index是1
            label[y_ind, x_ind, int(5 + class_id)] = 1
        return label

    def _resize(self, image, bbs):
        image_rescaled = ia.imresize_single_image(image, sizes=(self.image_size, self.image_size))
        bbs_rescaled = bbs.on(image_rescaled)
        return image_rescaled, bbs_rescaled.remove_out_of_image().clip_out_of_image()

    def _aug_images(self, image, bbs):
        """如果需要数据增强,调用这个程序即可"""
        # 每次批次调用一次，否则您将始终获得与每批次完全相同的扩充!
        seq_det = self.seq.to_deterministic()
        image_aug = seq_det.augment_image(image)
        bbs_aug = seq_det.augment_bounding_boxes([bbs])[0]
        return image_aug, bbs_aug.remove_out_of_image().clip_out_of_image()

    @staticmethod
    def _seq():
        """数据增强模块,定制发生什么变化"""
        seq = iaa.Sequential([
            iaa.Flipud(0.5),
            iaa.Fliplr(0.5),
            iaa.Crop(percent=(0, 0.1)),
            iaa.Sometimes(0.5, iaa.GaussianBlur(sigma=(0, 0.5))),
            iaa.ContrastNormalization((0.75, 1.5))])
        return seq

    @staticmethod
    def _parser(record):
        features = {"img": tf.FixedLenFeature((), tf.string),
                    "label": tf.VarLenFeature(tf.float32)}
        features = tf.parse_single_example(record, features)
        img = tf.image.decode_jpeg(features["img"])
        label = features["label"].values
        return img, label
```

1.4.3 迭代数据集，检查错误

```
class ShowImageLabel(object):
    def __init__(self,
                 image_size,
                 cell_size,
                 batch_size):
        self.image_size = image_size
        self.cell_size = cell_size
        self.batch_size = batch_size

    def parser_label(self, image, yolo_label):
        label = []
        for h_index in range(self.cell_size):
            for w_index in range(self.cell_size):
                if yolo_label[h_index, w_index, 0] == 0:
                    continue
                x_center, y_center, w, h = yolo_label[h_index, w_index, 1:5]
                class_id = np.argmax(yolo_label[h_index, w_index, 5:])
```

```
                x_1 = int((x_center - 0.5 * w) * self.image_size)
                y_1 = int((y_center - 0.5 * h) * self.image_size)
                x_2 = int((x_center + 0.5 * w) * self.image_size)
                y_2 = int((y_center + 0.5 * h) * self.image_size)
                label.append(ia.BoundingBox(x1=x_1, y1=y_1, x2=x_2, y2=y_2, label=class_id))
        return image, ia.BoundingBoxesOnImage(label, shape=image.shape)

    @staticmethod
    def draw_box(image, bbs):
        """ 绘制图片以及对应的bounding box

        Args:
            img: numpy array
            boxes: BoundingBoxesOnImage对象
        """
        for bound_box in bbs.bounding_boxes:
            x_center = bound_box.center_x
            y_center = bound_box.center_y
            _class = CLASS[bound_box.label]
            image = bound_box.draw_on_image(image,
                                            color=Colors_to_map[_class],
                                            alpha=0.7,
                                            thickness=2,
                                            raise_if_out_of_image=True)
            image = ia.draw_text(image,
                                 y=y_center,
                                 x=x_center-20,
                                 color=Colors_to_map[_class],
                                 text=_class)
        plt.imshow(image)
        plt.title("Iamge size >>> {}".format(image.shape))
        plt.axis('off')
        plt.xticks([])
        plt.yticks([])
        plt.show()


if __name__ == '__main__':
    check = 15
    to_tfrecord = To_tfrecords(txt_file='trainval.txt')
    to_tfrecord.transform()
    train_generator = Dataset(filenames='data/tfr_voc/trainval.tfrecords',
                              enhance=True)
    train_dataset = train_generator.transform()
    iterator = train_dataset.make_one_shot_iterator()
    next_element = iterator.get_next()
    # 检查生成的图像及 bounding box
    show_images = ShowImageLabel(448, 7, 32)
    count = 0
    with tf.Session() as sess:
        for i in range(10):
            images, labels = sess.run(next_element)
            while count < check:
                image, label = images[count, ...], labels[count, ...]
                image, label = show_images.parser_label(image, label)
                show_images.draw_box(image, label)
                count += 1
```

虽然比较复杂，但是记住我们最终实现的功能，就是把VOC格式的数据读取进来，做了数据增强以及reshape为 448 x 448，做了个标签转换，将其转换为yolo格式。

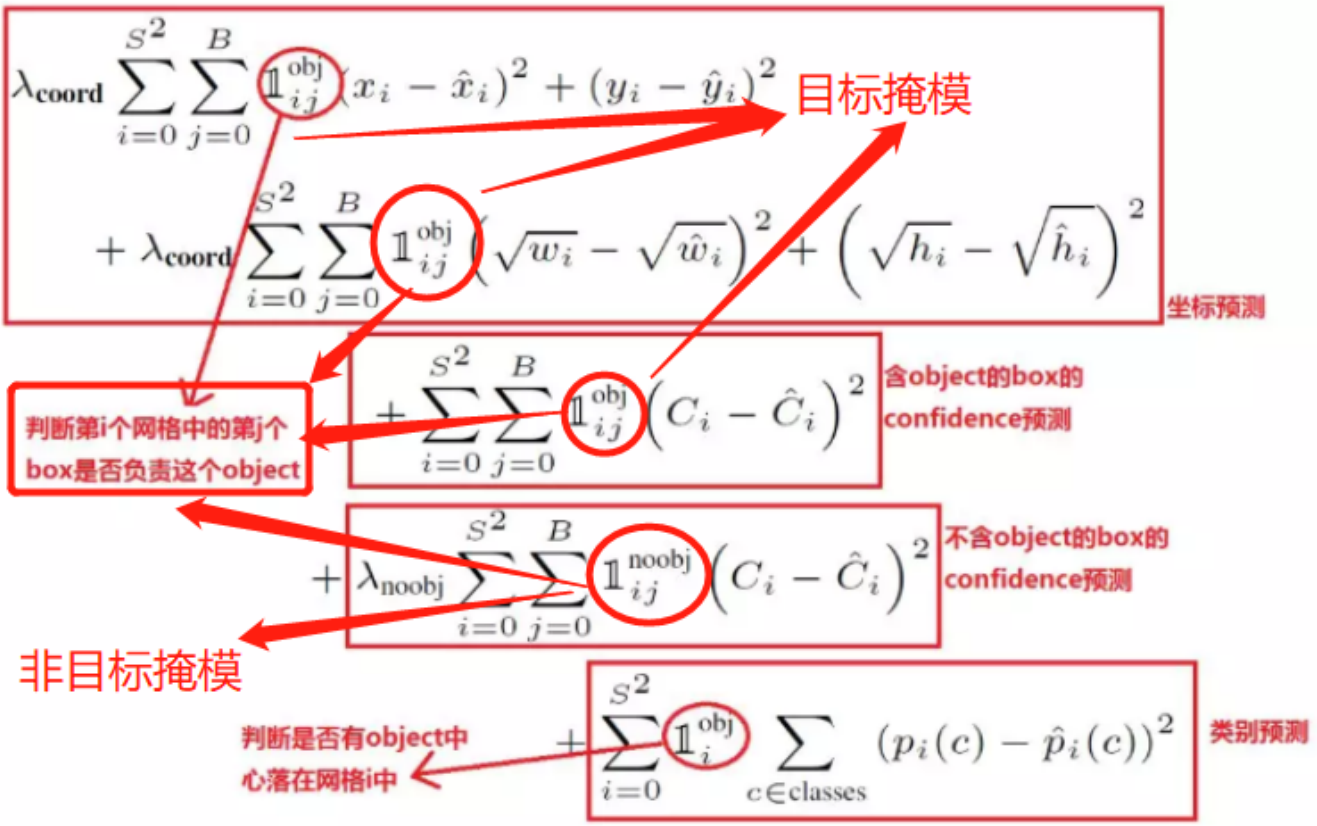
1.5 Yolov1 损失函数实现

现在我们回顾下所有标签、损失函数的输入输出以及我们希望网络拟合的目标：

	Inference	Label
`shape`	`[batch, 7, 7, 30]`	`[batch, 7, 7, 25]`
bounding box数量	2	1
x, y	希望拟合的 x, y 是相对于某个cell左上角的 0-1 比例，基准为cell.	1. x, y 是相对于图片左上角的 0-1 比例，基准为整张图片。 2. 我们需要根据掩模筛选出需要的
w, h	目标大小有区别，我们希望拟合$\sqrt{w}, \sqrt{h}$,但w, h对于整张图片的0-1值	1. w, h是相对于整张图片的0-1范围的值 2. 我们需要根据掩模筛选出需要的
confidence	拟合的Confidence	IOU * 目标/非目标掩模
classification	1 x 20	1 x 20

我们回顾一下损失函数构成，并且补充一些细节：

1. 坐标损失
2. 置信度损失
3. 类别损失



有以下几个问题：

1. x, y, w, h不统一怎么计算?
2. 如何让object只由它的中心所在的cell负责?
3. 目标掩模怎么计算?
4. 这些损失如何综合?

下面我们开始实现这个代码，我们先取出对应的分量：

```
def loss_layer(self, predicts, labels, scope='loss'):
        """

        Args:
            predicts: Net的输出 [Batch, 7, 7, 30]
            labels: label[Batch, 7, 7, 25], [Batch, (h方向), (w方向), 25]
                    为与计算机存储图片格式相同,我们存储标签的时候先按h方向,再w方向.

        """

        ###############预测###############
        # 1.0 [batch, 7, 7, 2, 4] 2 坐标预测
        pre_boxes = tf.reshape(predicts[..., : 4 * self.boxes_per_cell],
                               shape=[None, self.cell_size, self.cell_size, self.boxes_per_cell, 4])
        # 1.1 [batch, 7, 7, 2] 2 置信度预测
        pre_confidence = predicts[..., 4 * self.boxes_per_cell: 5 * self.boxes_per_cell]
        # 1.2 [batch, 7, 7, 20] 1 分类预测
        pre_class = predicts[..., 5 * self.boxes_per_cell:]

        ###############标签###############
        # 2.0 [batch, 7, 7, 4] 1 坐标标签
        lab_boxes = labels[..., :4]  # 全局坐标系下 x, y, w, h [0-1]范围
```

```
        # 2.1 [batch, 7, 7, 1] 1 对应Response标签 # 0 or 1, 代表该cell中是否有object
        lab_response = labels[..., 4]
        # 2.2 [batch, 7, 7, 20] 2 1 类别标签 20维 one-hot 标签
        lab_class = labels[..., 5:]
```

1.5.1 类别损失

回到我们提到的类别损失，我们知道只有在有object的cell上，标签上才有类别标签，为一个20维的one-hot标签，其余的本应该为None。但是这样的结果是，整个数据需要存储为稀疏矩阵，而且计算机需要一一判断这里是否有值。为了容易计算，我们将其值存储为0。

但是这又带来一个问题，当卷积网络在提取特征做预测的时候，这个Inference出的值并不是0，这样两个标签相减计算均方差就会给网络带来损失。事实上我们是不需要计算没有object网格的坐标损失的。

因此我们将其乘以Response标签(也就是标记7×7 cell 中哪个cell有object，shape=`[batch, 7, 7, 1]`)，在没有目标的位置，这些位置的Cell是0值，这样处理后，类别损失计算后不需要的会被乘以0，我们就过滤掉了这部分多余的计算。虽然这样效率会变低，但是这却是计算机矩阵化处理的技巧之一。

让我们记住：

> 我们只需要计算有object的cell中的类别损失，对于多余出来的计算，我们可以在矩阵相乘的时候乘以0来处理掉。

关于Tensorflow的一些矩阵操作，不明白的可以查看我的笔记，另外如果你对一些操作不太明确的话，建议你和我一样新建一个`tensor_experiment.py`文件，并且不将其加入版本控制，主要用来试验一些不确定的操作。

下面我们看代码：

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

类别预测

类别损失相对较为容易，每个cell中只有一个classification label，我们只有当这个cell中有label目标的时候，才会去计算这个损失。

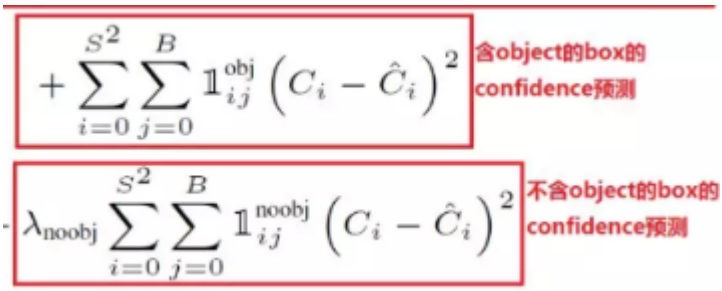
```
def class_loss(self, pre_class, lab_class, lab_response):
        """

        Args:
            pre_class: 预测类别 [batch, 7, 7, 20]
            lab_class: 标签类别 [batch, 7, 7, 20] 20维 one-hot标签
            lab_response: 标记某个Cell中是否有object [batch, 7, 7, 1]
            weight: 权重系数之前说过,类别损失和坐标损失的比重是1 : 5

        Returns:
            类别损失
        """
        # 乘以 label_response 来去除掉没有Object位置多计算出的类别损失
        # 注意一个cell中是否有物体是通过置信度损失来回归的
        # 这种分开的思想让不同位置的参数回归不同属性,而不是把它们融合在一起
        with tf.name_scope('class loss'):
            delta = lab_response * (pre_class - lab_class)
```

```
            class_loss = self.class_scale * tf.reduce_mean(tf.reduce_sum(tf.square(delta), axis=[1, 2, 3]), name='class_loss')
        return class_loss
```

我们计算损失只需要调用函数即可:

```
class_loss = self.class_loss(pre_class, lab_class, lab_response)
```

1.5.2 置信度损失



对于置信度损失，我们知道两点：

1. 在Label中Response标签为0-1值，代表每个cell中是否有object, `shape=[batch, 7, 7, 1]`. 我们的置信度标签: Response**标签 * 掩模** 计算得来。
2. 我们希望没有Object的cell对网络的贡献小一点，而有目标的cell对网络的贡献大一点。否则网络只要全部预测出没有Object损失就很小了。我们定的比例是1:2。

但有以下三点需要考虑的:

1. 其次如果计算IOU的话，这个坐标怎么进行转换，怎么计算IOU?
2. 目标掩模怎么计算?
3. 没有Object的cell和有Object的cell贡献的损失比例是1:2，我们希望没有object的cell对网络贡献小一点。那我们怎么做?

IOU主要通过标签label中的object坐标和预测出的object坐标计算得来。可问题是：Label中的x, y, w, h基于全图坐标系的0-1值，而：

- Inference出的x, y 基于cell坐标系，因此需要做一个转换。
- Inference出的是$\sqrt{w}, \sqrt{h}$，虽然是基于全图坐标系，但是我们需要将其平方之后才能跟用来计算。

那么我们计算IOU的时候需要做个转换，因为后面计算坐标损失的时候也需要用到，所以我们提前做一个转换。

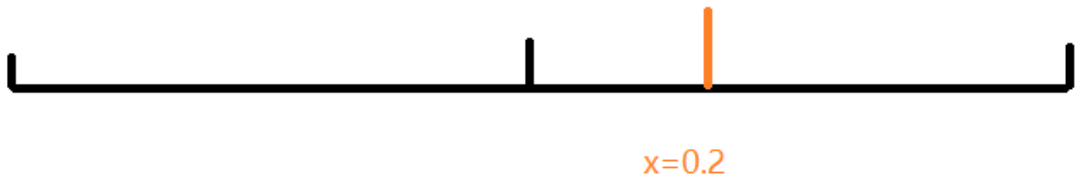
1.5.2.1 坐标转换

先大致确认下函数输入输出，

```python
def pre_to_label_coord(self, pre_boxes):
        """

        Args:
            pre_boxes: Net Inference 基于cell左上角
                       x, y 偏移量, sqrt(w), sqrt(h)  [Batch, 7, 7, 2, 4]

        Returns:
            转换为全图坐标系下的boxes信息, x, y, w, h [Batch, 7, 7, 2, 4]
        """
        # 我们希望将基于cell的坐标变换为全局坐标系
        pass
```

$\sqrt{w}, \sqrt{h}$是比较容易的，只需要平方一下即可，而x, y则不好变换。

人类思考问题都是从最简单的情况考虑起的，我们从最简单的一维的角度思考，我们将一条线段均分为2段，坐标假设为x_{cell}，注意这个x_{cell}是以所在cell的左边为基准的，我们如何将其转换为全局坐标系？



对于一维问题，我们只需要$(0.2+1)/2=0.6$，其实就是$(x_{cell}+x_{ind})/cell$这么一个简单的公式，那么对于对于每个cell中的x_{cell}我们怎么做转换，其实加上`np.range(cell)`即可。

现在确认一下我们的公式：

```python
x_global = (x_cell + np.range(cell)) / cell
```

那么这样我们就很容易做这个处理，现在我们转换到二维空间，其实也很好理解。但是现在我们的`pre_boxes`是`shape=[batch, 7(h), 7(w), 2, 4]`，难度在于这是个高维操作，其实很简单，操作高维数组的时候，请抛弃掉那些具体的想象，其实整个原理关键在于`np.range(cell)`，其实不就是对于某个`axis`，当我的索引上升时，对应值也会上升，比如说：对于`axis=1`,其`[:, index=0, :, :, :] = 0`，而`[:,index=1, :, :, :] = 1`。那么最终：

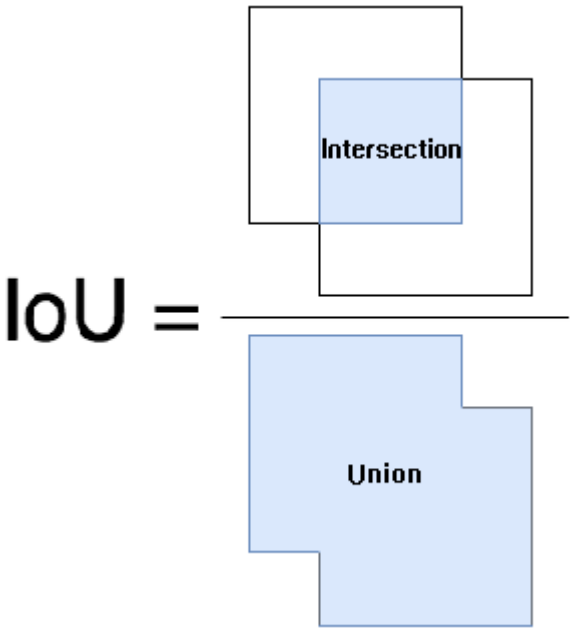
```python
def pre_to_label_coord(self, pre_boxes):
        """坐标转换基于cell的x, y, sqrt(w), sqrt(h) >> 基于全图的x, y, w, h"""

        # 1. 沿着axis=2的方向逐渐增大,我们希望shape=[1, 7, 7, 2(bounding box), 1]1
        offset_axis_2 = tf.tile(tf.expand_dims(tf.range(7), axis=0),
                                multiples=[7, 1])
        offset_axis_2 = tf.tile(tf.reshape(offset_axis_2, shape=[1, 7, 7, 1, 1]),
                                multiples=[1, 1, 1, 2, 1])
        # 3. 沿着axis=1的方向变大
        offset_axis_1 = tf.transpose(offset_axis_2, (0, 2, 1, 3, 4))

        w = tf.square(pre_boxes[..., 2])
        h = tf.square(pre_boxes[..., 3])

        # 4. 计算x的时候.因为图像是以h, w格式存储的,也就是 x变化 在axis=2上递增
        global_x = (pre_boxes[..., 0] + offset_axis_2) / self.cell_size
```

```
        global_y = (pre_boxes[..., 1] + offset_axis_1) / self.cell_size
        global_pre_boxes = tf.stack([global_x, global_y, w, h], axis=-1)
        return global_pre_boxes
```

现在我们就可以用两个全局坐标计算IOU了。

1.5.2.2 IOU计算

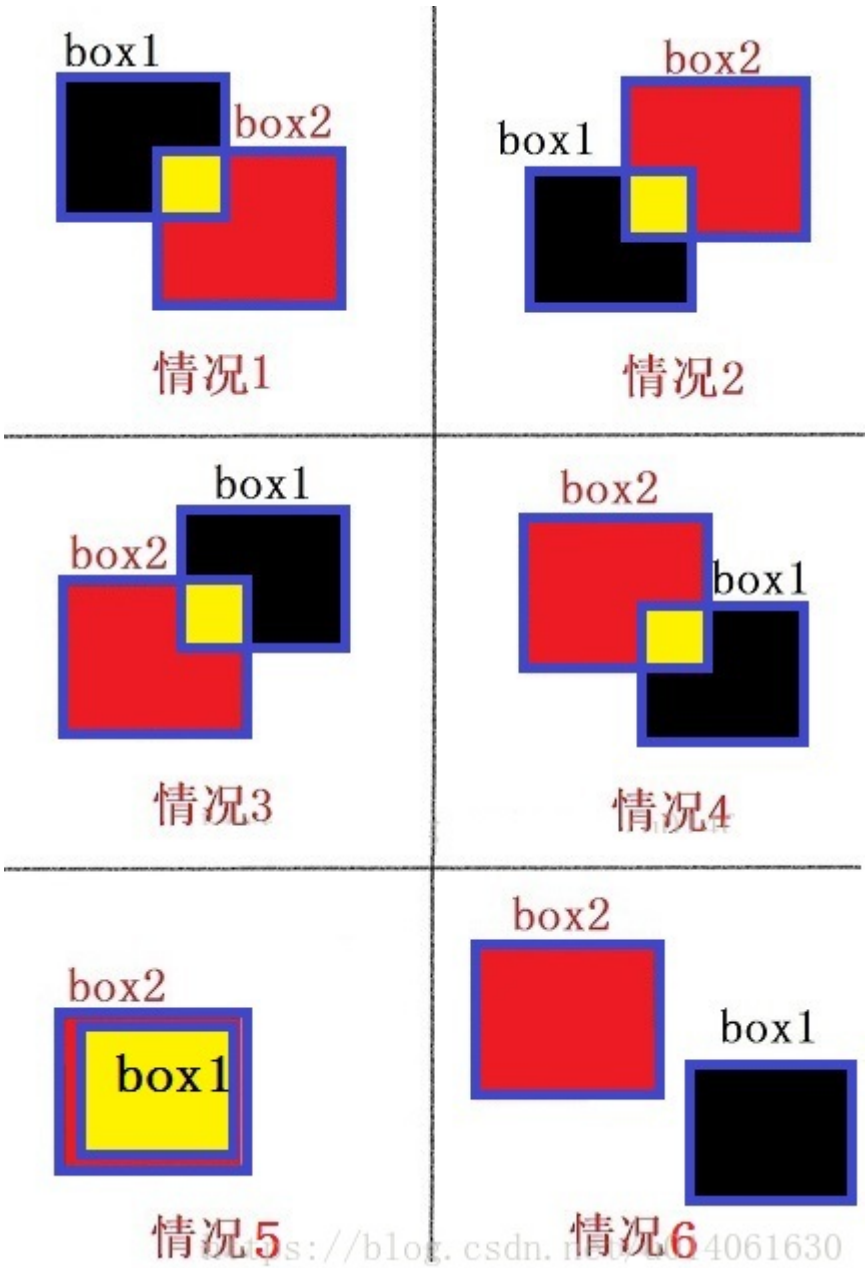
IOU称为交并比（Intersection over Union），计算的是 “预测的边框” 和 “真实的边框” 的交集和并集的比值:



人类解决问题的方式都是从简单的入手的，之后再逐渐复杂，输入肯定是两个x, y, w, h，我们肯定需要将坐标系转换一下，变为标记左上角的x_1, y_1以及右下角的x_2, y_2，回忆下我们的坐标系，不同于一般坐标系，我们的**坐标系y轴方向向下。**

首先的想法是:

> 我们考虑两个边框的相对位置，然后按照相对位置分情况讨论来计算交集。

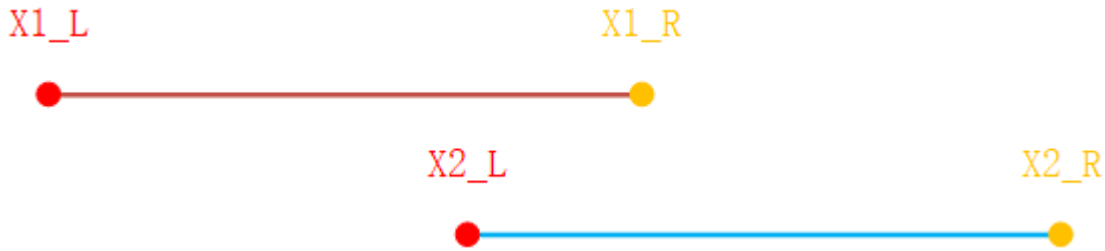


但是如果这样写程序的话，你确定你的导师不会打死你吗？我们来实现一版不让导师打死的IOU计算代码，我们将其矩阵化处理：

很多问题复杂，我们先从最简单的情况考虑，我们从X轴方向看，其实只存在3种情况：

1. 两个box没有交集

2. 部分重叠



3. 完全重叠

我们思考以下几个问题：

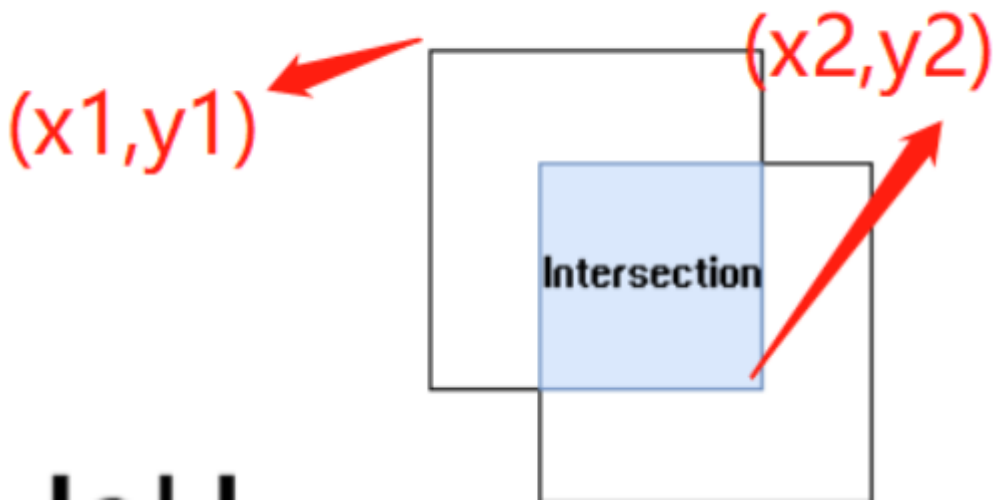
- 我们如何计算交集？

  > min(X_R)-max(X_L)即为交集的值，我们将其与0值比较，取最大值。

- 我们如何计算并集的面积？

  > 其实非常简单，我们只需要求两个 bounding box 各自的面积，然后减掉交集即可。

知道了这个思路，那么我们就可以着手开始实现了，在此之前，我们需要对坐标进行转换，从x, y, w, h转换为所需要的x_1, y_1, x_2, y_2：




```
def calc_iou(self, boxes1, boxes2, scope='IOU'):
        """ 计算训练时候 bounding box 和 label 的IOU

        Args:
            boxes1: 预测 Boxes
                    [Batch, CELL_SIZE, CELL_SIZE, BOXES_PER_CELL, 4] / [x, y, w, h]
            boxes2: label Boxes
                    [Batch, CELL_SIZE, CELL_SIZE, BOXES_PER_CELL, 4] /[x, y, w, h]
```

```
            scope: 我们

        Returns:
             IOU: 4-D tensor [BATCH_SIZE, CELL_SIZE, CELL_SIZE, BOXES_PER_CELL]
        """
        with tf.variable_scope(scope):
            """transform (x_center, y_center, w, h) to (x1, y1, x2, y2)
            1. 涉及矩阵操作的时候,Tesnorflow一般只是将前面的维度当做Batch
            2. 不涉及矩阵的操作,我们可以拿出单个元素考虑,之后将其矩阵化
            """
            boxes1_voc = tf.stack([boxes1[..., 0] - boxes1[..., 2] / 2.0,
                                   boxes1[..., 1] - boxes1[..., 3] / 2.0,
                                   boxes1[..., 0] + boxes1[..., 2] / 2.0,
                                   boxes1[..., 1] + boxes1[..., 3] / 2.0],
                                   axis=-1)
            boxes2_voc = tf.stack([boxes2[..., 0] - boxes2[..., 2] / 2.0,
                                   boxes2[..., 1] - boxes2[..., 3] / 2.0,
                                   boxes2[..., 0] + boxes2[..., 2] / 2.0,
                                   boxes2[..., 1] + boxes2[..., 3] / 2.0],
                                   axis=-1)

            # calculate the left up point & right down point
            # [batch, 7, 7, 2, 2] 对应位置最大值, maximum 支持广播,但不能指定轴
            lu = tf.maximum(boxes1_voc[..., :2], boxes2_voc[..., :2])  # x1, y1 max(X_L)
            rd = tf.minimum(boxes1_voc[..., 2:], boxes2_voc[..., 2:])  # x2, y2 min(X_R)

            # [batch, 7, 7, 2, 2]   min(X_R)-max(X_L)
            intersection = tf.maximum(0.0, rd - lu)
            # [batch, 7, 7, 2] 2个bounding box跟label的IOU,因此这里有2个
            inter_square = intersection[..., 0] * intersection[..., 1]

            # calculate the boxs1 square and boxs2 square by w*h
            # [batch, 7, 7, 2] * [batch, 7, 7, 2]
            square1 = boxes1[..., 2] * boxes1[..., 3]
            square2 = boxes2[..., 2] * boxes2[..., 3]
            union_square = tf.maximum(square1 + square2 - inter_square, 1e-10)

        return tf.clip_by_value(inter_square / union_square, 0.0, 1.0)
```

1.5.2.3 掩模计算

没有Object的cell和有Object的cell贡献的损失比例是1:2，我们希望没有object的cell对网络贡献小一点。但怎么认为这个cell中是否有object呢?

> 事实上Yolo基于假设一个cell中只有一类物体，那么就是Bounding box IOU最高的那个框中有object，其余的均没有Object。

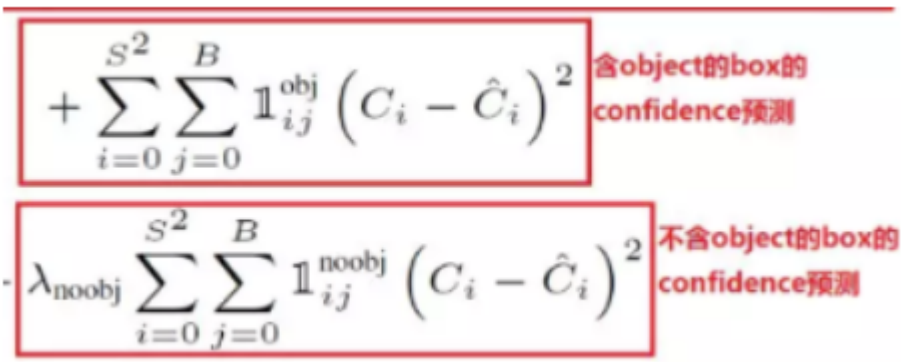
这个记录是否有目标的值称为掩模，因为它很像我们的光刻，我们只对其中一部分进行操作，其余的被挡住了。

```
def mask(self, iou_pre_label, label_response):
        """

        Args:
            iou_pre_label: 两个 Bounding box 的 IOU [batch, 7, 7, 2]
            label_response: [batch, 7, 7, 1]  0-1

        Returns:
            object_mask: 有目标的掩模,有object并且IOU最高的bounding box
```

```
            no_object_mask: 其余情况

        """
        # [BATCH_SIZE, CELL_SIZE, CELL_SIZE, BOXES_PER_CELL]
        # tf.reduce_max会在某个轴上进行计算,比较的是自己的值
        # tf.maxmium比较的是两个值,可以广播
        # 用之前说过的乘以0的技巧来避免逻辑判断
        object_mask = tf.reduce_max(iou_pre_label, axis=-1, keep_dims=True)
        object_mask = tf.cast((iou_pre_label>= object_mask), tf.float32)
        # 还需要乘以 response, 这样子被过滤出来的只有IOU最大的Box
        object_mask = object_mask * label_response

        no_object_mask = tf.ones_like(object_mask, dtype=tf.float32) - object_mask
        return object_mask, no_object_mask
```

1.5.2.4 置信度计算

我们看公式:

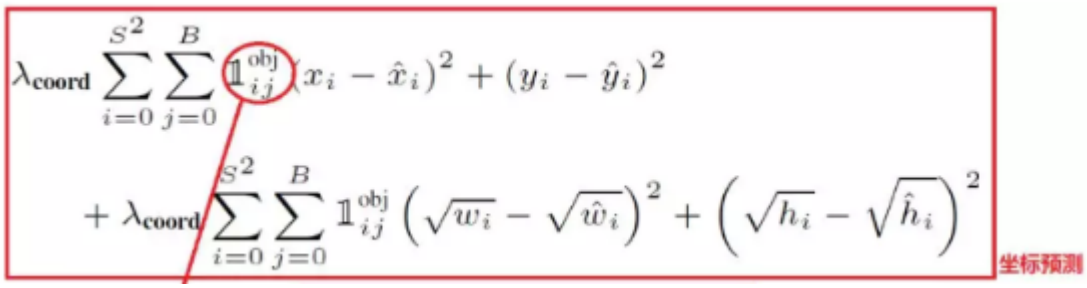



```
def confidence_loss(self,
                        pre_confidence,
                        iou_pre_label,
                        object_mask,
                        no_object_mask,
                        ):
        """

        Args:
            pre_confidence: 预测置信度 shape=[batch, 7, 7, 2]
            iou_pre_label: IOU shape=[batch, 7, 7, 2]
            object_mask: 目标掩模 [batch, 7, 7, 2] 有目标的位置是1,其余为0
            no_object_mask: 非目标掩模 [batch, 7, 7, 2] 没有目标的位置是1,其余为0
        """
        with tf.name_scope('Confidence loss'):
            with tf.name_scope("Object Confidence loss"):
                # 用目标掩模进行判断是否有目标
                # 实际是 object_mask * pre_confidence - object_mask * iou_pre_label
                # 我们将式子合并之后变为下面的样子
                object_confidence_delta = object_mask * (pre_confidence - iou_pre_label)
                object_confidence_loss = self.object_confidence_scale * tf.reduce_mean(
                    tf.reduce_sum(tf.square(object_confidence_delta), axis=[1, 2, 3]))
            with tf.name_scope('No Object Confidence loss'):
                # 只要预测出置信度就是错的,我们用掩模抑制
                # 实际是 no_object_mask * pre_confidence - no_object_mask * 0 因为这些位置没有Object,因
此iou标签即为0
```

```
                no_object_confidence_delta = no_object_mask * pre_confidence
                no_object_confidence_loss = self.no_object_confidence_scale * tf.reduce_mean(
                    tf.reduce_sum(tf.square(no_object_confidence_delta), axis=[1, 2, 3]))
        return object_confidence_loss, no_object_confidence_loss
```

1.5.3 中心坐标损失

坐标损失很好计算，主要涉及一些坐标转换的问题：



我们知道：

	Inference	Label
x, y	基于cell	基于全图
对应w, h位置	$\sqrt{w}, \sqrt{h}$	w, h
数量	2	1

损失函数是基于Inference的，因此我们要将Label的转换回来：

```
def label_to_pre_cood(self, lab_boxes):
        offset_axis_2 = tf.tile(tf.expand_dims(tf.range(7), axis=0),
                                multiples=[7, 1])
        offset_axis_2 = tf.tile(tf.reshape(offset_axis_2, shape=[1, 7, 7, 1, 1]),
                                multiples=[1, 1, 1, 2, 1])
        offset_axis_1 = tf.transpose(offset_axis_2, (0, 2, 1, 3, 4))
        sqrt_w = tf.sqrt(lab_boxes[..., 2])
        sqrt_h = tf.sqrt(lab_boxes[..., 3])
        cell_x = lab_boxes[..., 0] * self.cell_size - offset_axis_2
        cell_y = lab_boxes[..., 1] * self.cell_size - offset_axis_1
        cell_lab_boxes = tf.stack([cell_x, cell_y, sqrt_w, sqrt_h], axis=-1)
        return cell_lab_boxes
```

我们在代码之前执行过lab_boxes的复制，因此写的函数：

```
def coord_loss(self,
                   pre_boxes,
                   lab_boxes,
                   object_mask):
        """

        Args:
            pre_boxes: [batch, 7, 7, 2, 4] 基于cell的x, y以及全图 sqrt(w), sqrt(h)
            lab_boxes: [batch, 7, 7, 2, 4] 基于全图的x, y, w, h
```

```
            object_mask: [batch, 7, 7, 2]

        Returns:

        """
        with tf.name_scope('Coord loss'):
            coord_mask = tf.expand_dims(object_mask, axis=-1)
            cell_lab_boxes = self.label_to_pre_cood(lab_boxes)
            coord_delta = coord_mask * (pre_boxes - cell_lab_boxes)
            coord_loss = self.coord_scale * tf.reduce_mean(
                tf.reduce_sum(tf.square(coord_delta), axis=[1, 2, 3, 4]))
        return coord_loss
```

1.5.4 Summary

我尽力将功能分开，使得大家能够看明白每个步骤到底做什么，最终汇总损失如下：

```
def loss_layer(self, predicts, labels, scope='loss'):
        """


        Args:
            predicts: 网络的输出 [Batch, 7, 7, 30]
            labels: label[Batch, 7, 7, 25], [Batch, (h方向), (w方向), 25]
                    为与计算机存储图片格式相同,我们存储标签的时候先按 h / w 顺序存储.
        """
        with tf.name_scope('Predict Tensor'):
            ############### 预测 ###############
            """
            1. 预测坐标: x, y 基于cell, sqrt(w), sqrt(h) 基于全图 (0-1)范围内
            2. 2个bounding box,拥有2个坐标以及置信度
            """
            # 1. Bounding box 坐标预测 [batch, 7, 7, :8] >> shape=[batch, 7, 7, 2, 4]
            pre_boxes = tf.reshape(predicts[..., : 4 * self.boxes_per_cell],
                                   shape=[None, self.cell_size, self.cell_size, self.boxes_per_cell, 4])
            # 2. Bounding box 置信度预测 [batch, 7, 7, 8:10] >> shape=[batch, 7, 7, 2]
            pre_confidence = predicts[..., 4 * self.boxes_per_cell: 5 * self.boxes_per_cell]
            # 3. class 类别预测 [batch, 7, 7, 10:] >> shape=[batch, 7, 7, 20]
            pre_class = predicts[..., 5 * self.boxes_per_cell:]
        with tf.name_scope('Label Tensor'):
            ############### 标签 ###############
            """
            1. 标签坐标: x,y,w,h 均基于全图(0-1)
            """
            # 1. box response_label [batch, 7, 7, 0]  >> shape=[batch,7, 7, 1]
            # lab_response 只负责标记cell中是否有object,置信度标签需要跟IOU实时计算
            lab_response = labels[..., 0]
            # 2. box 坐标label [batch, 7, 7, 1:5] >> shape=[batch, 7, 7, 2, 4]
            lab_boxes = tf.reshape(labels[..., 1:5],
                                   shape=[None, self.cell_size, self.cell_size, 1, 4])
            lab_boxes = tf.tile(lab_boxes, [1, 1, 1, self.boxes_per_cell, 1])
            # 3. class 类别标签 [batch, 7, 7, 5:] >> shape=[batch, 7, 7, 20]
            lab_class = labels[..., 5:]

        ############### 损失函数 ###############
        with tf.variable_scope(scope):
            # 1. 类别损失
            class_loss = self.class_loss(pre_class, lab_class, lab_response)
            # 2. 坐标转换基于cell的x, y, sqrt(w), sqrt(h) >> 基于全图的x, y, w, h
```

```
            global_pre_boxes = self.pre_to_label_coord(pre_boxes)  # [batch, 7, 7, 2, 4]
            # 3. 计算iou shape=[batch, 7, 7, 2]
            iou_pre_label = self.calc_iou(global_pre_boxes, lab_boxes)
            # 4. 目标掩模 / 非目标掩模
            object_mask, no_object_mask = self.mask(iou_pre_label, lab_response)
            # 5. 置信度损失
            object_confidence_loss, no_object_confidence_loss = self.confidence_loss(
                pre_confidence, iou_pre_label, object_mask, no_object_mask)
            # 6. 坐标损失
            coord_loss = self.coord_loss(pre_boxes, lab_boxes, object_mask)

            tf.losses.add_loss(class_loss)
            tf.losses.add_loss(object_confidence_loss)
            tf.losses.add_loss(no_object_confidence_loss)
            tf.losses.add_loss(coord_loss)

            tf.summary.scalar('class_loss', class_loss)
            tf.summary.scalar('object_confidence_loss', object_confidence_loss)
            tf.summary.scalar('no_object_confidence_loss', no_object_confidence_loss)
            tf.summary.scalar('coord_loss', coord_loss)
```