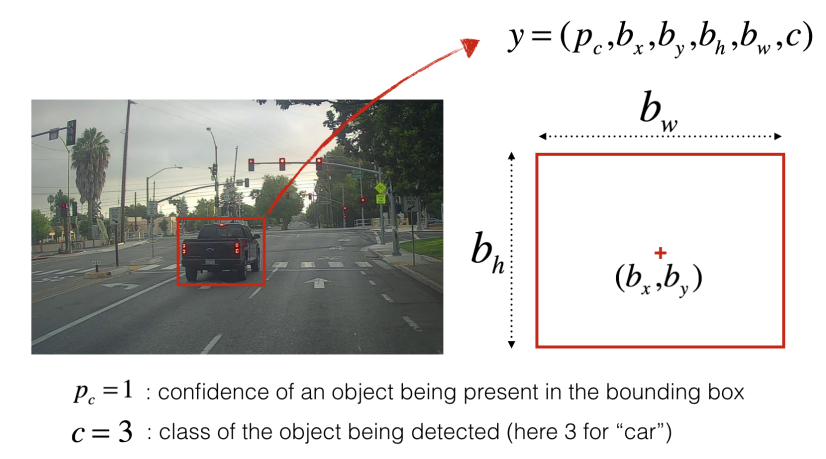
**自动驾驶 -- 汽车识别**

使用YOLO算法进行对象识别（使用Keras框架）

1. **Problem Statement**

假设你现在在做自动驾驶的汽车，你想着首先应该做一个汽车检测系统，为了搜集数据，你已经在你的汽车前引擎盖上安装了一个照相机，在你开车的时候它会每隔几秒拍摄一次前方的道路

您已经将所有这些图像收集到一个文件夹中，并通过在您找到的每辆车周围画边界框来标记它们。下面是一个关于边框的例子：



假如你想让YOLO识别80个类别的物体（见coco\_classes.txt文件），你可以把分类标c从1-80进行标记，**或者把它变成80维的向量（80个数字），在对应的位置上写0或者1**，因为YOLO的模型训练起来比较久，我这里使用预训练好的模型进行使用。

1. **YOLO算法**

YOLO(“你只看一次”)是一种流行的算法，因为它既能实现高精度，又能实时运行。这种算法“只看一次”图像，从某种意义上来说只需要一个正向传播通过网络进行预测。然后经过非最大值抑制(non-max-suppression)后，输出识别的对象和边框。

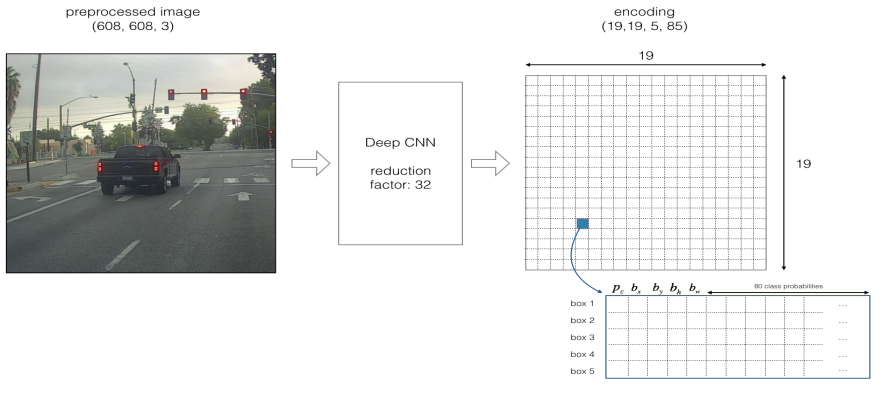
**2.1 Model detail**

首先要知道的是：

* 1. 输入的是批量图片，shape is (m,608,608,3)
  2. 输出是一个包含识别类的bounding box列表，每个bounding box有个数字（Pc，bx，by，bh，bw，c），如果c扩展为80-dimensional vector，每个bounding box将有85数字

我将使用5个anchor boxes，所以算法的大致流程是这样的：IMAGE(m,608,608,3)—>DEEP CNN—>ENCODING(m,19,19,5,85)

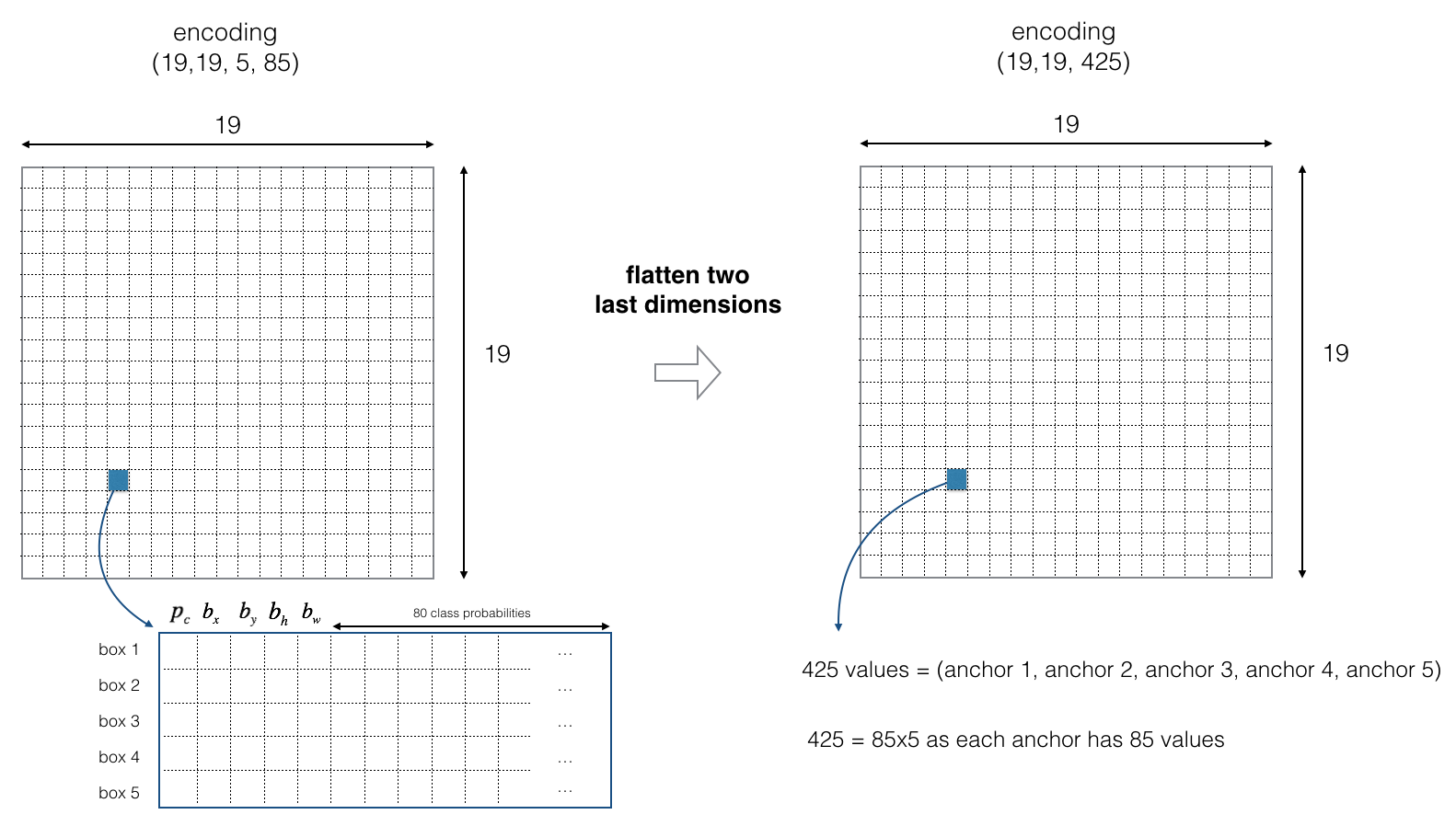
下面是ENCODING编码的情况：



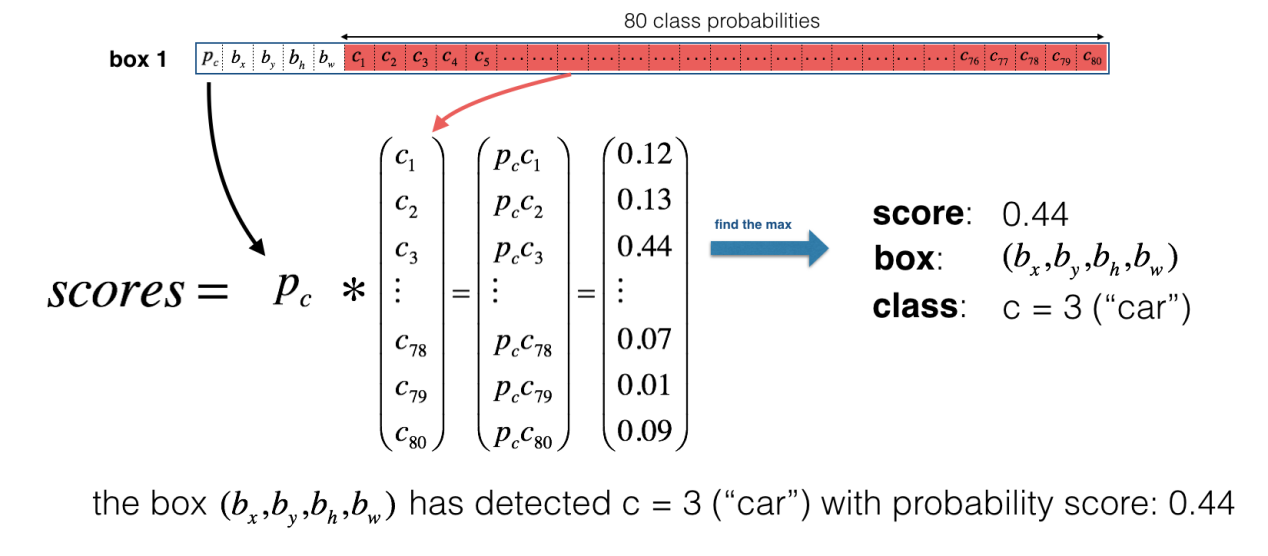
如果对象的中心/中点在单元格内，那么单元格就负责识别该对象。

因为我们使用了5个anchor boxes，每个cell（19×19 cells）对5个anchor boxes进行encode。Anchor boxes由高度和宽度定义。

为了简单起见，我们flatten(19,19,5,85）编码的后面两个维度，所以DEEP CNN的输出变成了(19,19,425)，如下图：



现在，对于每个框(每个单元格)，我们将计算以下elementwise乘积，并提取该框包含某个类的概率，如下：



我们来看一下可视化的预测图片：



每个单元格会输出5个anchor boxes，总的来说，观察一次图像（一次前向传播），该模型需要预测19×19×5=1805个anchor boxes，不同颜色代表不同的分类，在上图中只绘制了模型所猜测的高概率的anchor boxes，但是anchor boxes依旧是太多了，我们希望算法的输出为更少的anchor boxes，所以这就要用到non\_max\_supppression，具体步骤如下：

1. 舍弃掉低概率的anchor boxes(meaning，anchor boxes没那么大的信心确定为该类)
2. 当几个anchor boxes相互重叠并检测同一个物体时，只选择一个anchor box

**2.2 Filtering with a threshold on class scores（过滤类分数的阈值）**

应用第一个阈值过滤器，你将会并且掉任何anchor box的class “score”低于阈值的anchor boxes

模型一共有19×19×5×85个数字。每个anchor由85个数字组成（Pc，bx，by，bh，bw，80-dimensions）

，将维度（19,19,5,85）或者（19,19,425）换成下面的维度有利于下一步的操作：

*box\_confidence: --tensor of shape (19, 19, 5,1)*包含19x19单元格中每个单元格预测的5个锚框中的所有的锚框的P.

*boxes: -- tensor of shape (19, 19, 5, 4)*包含了所有的锚框的（*px,py,ph,pw*） *box\_class\_probs: -- tensor of shape (19, 19, 5, 80)*containing the detection probabilities (*c*1,*c*2,...*c*80)for each of the 80 classes for each of the 5 anchor boxes per cell.

def yolo\_filter\_boxes(box\_confidence, boxes, box\_class\_probs, thresthod = .6):  
 *"""  
 Filters YOLO boxes by thresholding on object and class confidence.* ***:param*** *box\_confidence: --tensor of shape (19, 19, 5,1)* ***:param*** *boxes: -- tensor of shape (19, 19, 5, 4)* ***:param*** *box\_class\_probs: -- tensor of shape (19, 19, 5, 80)* ***:param*** *thresthod: -- real value, if [highest class probability score < threshold],then get rid of the corresponding box]* ***:return****:  
 scores -- tensor of shape(None, ),containing the class probability score for selected boxes  
 boxes -- tensor of shape(None, 4),containing(b\_x, b\_y, b\_h, b\_w) coordinates of selected boxes  
 classes -- tensor of shape(None, ),containing the index of the class detected by the selected boxes  
 """* ## First step：计算锚框的得分  
 box\_scores = box\_confidence \* box\_class\_probs #(19,19,5,80)  
 ## Second step：找到最大值的锚框索引以及对应的最大值的锚框  
 box\_classes = K.argmax(box\_scores,axis=-1) #axis = -1表示对最后一维操作  
 box\_class\_scores = K.max(box\_scores,axis=-1)  
 ## Third step：根据阈值创建掩码  
 filtering\_mask = (box\_class\_scores>=thresthod)  
 ## 对scores， boxes 以及classes使用掩码  
 scores = tf.boolean\_mask(box\_class\_scores,filtering\_mask)  
 boxes = tf.boolean\_mask(boxes, filtering\_mask)  
 classes = tf.boolean\_mask(box\_classes, filtering\_mask)  
  
 return scores, boxes, classes

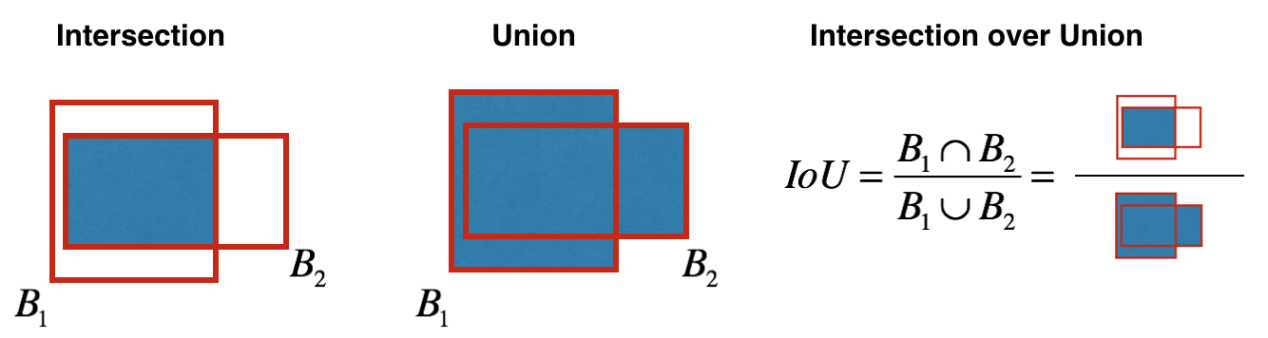
2.3 non\_max\_suppression

即使是通过score阈值过滤了一些score较低的分类，，但是依旧还是有很大anchor被保留下来，这里我们就要进行第二次过滤，如下图所示，将左边的图片变成右边的图片，这就叫做non\_maximum suppression(非最大值抑制)--NMS



上图例子中，模型预测了3辆车，但是实际上这3辆车都是同一辆车，我们使用NMS将会去选择3个anchor中最高概率的1个anchor。

那么如何实现NMS呢？我们需要运用Intersection over Union（I0U）交并比，如下



Implement iou(). Some hints:

* In this exercise only, we define a box using its two corners (upper left and lower right): (x1, y1, x2, y2) rather than the midpoint and height/width.
* To calculate the area of a rectangle you need to multiply its height (y2 - y1) by its width (x2 - x1)
* You'll also need to find the coordinates (xi1, yi1, xi2, yi2) of the intersection of two boxes. Remember that:
  + xi1 = maximum of the x1 coordinates of the two boxes
  + yi1 = maximum of the y1 coordinates of the two boxes
  + xi2 = minimum of the x2 coordinates of the two boxes
  + yi2 = minimum of the y2 coordinates of the two boxes

def iou(box1, box2):  
 *"""  
 实现两个锚框的交并比的计算* ***:param*** *box1: 第一个锚框，shape(x1,y1,x2,y2)* ***:param*** *box2: 第二个锚框，shape(x1,y1,x2,y2)* ***:return****:  
 iou:实数，交并比  
 """* # 计算相交的区域的面积  
 xi1 = np.maximum(box1[0], box2[0])  
 yi1 = np.maximum(box1[1], box2[1])  
 xi2 = np.minimum(box1[2], box2[2])  
 yi2 = np.minimum(box1[3], box2[3])  
 inter\_area = (xi1 - xi2) \* (yi1 - yi2)  
 # 计算并集 Union(A,B) = A + B - Inter(A, B)  
 box1\_area = (box1[2] - box1[0]) \* (box1[3] - box1[1])  
 box2\_area = (box2[2] - box2[0]) \* (box2[3] - box2[1])  
 union\_area = box1\_area + box2\_area - inter\_area  
 # 计算交并比  
 iou = inter\_area / union\_area  
 return iou

现在可以实现non-max suppression了，关键步骤如下：

1. 选择最高的分值的anchor box
2. 计算其他anchor box与选择出来的anchor box重叠的部分，剔除与该anchor box重叠较大的anchor，保证每个cell只有一个anchor
3. 返回步骤1

这将删除与所选框有大量重叠的所有框。剩下的只有“best”anchor box。

下面我们使用TensorFlow来实现yolo\_non\_max\_suppression()，其实在TensorFlow中有两个内置的方法去实现non-max-suppression（所以实际上不需要用到我们之前实现的iou()方法）

* [tf.image.non\_max\_suppression()](https://www.tensorflow.org/api_docs/python/tf/image/non_max_suppression" \t "http://localhost:8888/notebooks/_blank)
* [K.gather()](https://www.tensorflow.org/api_docs/python/tf/gather" \t "http://localhost:8888/notebooks/_blank)

def yolo\_non\_max\_suppression(scores, boxes, classes, max\_boxes = 10, iou\_threshold = 0.5):  
 *"""  
 Applies Non-max suppression (NMS) to set of boxes  
 Implement yolo\_non\_max\_suppression using Tensorflow* ***:param*** *scores: --tensor类型，( ,None ),yolo\_filter\_boxes()的输出* ***:param*** *boxes:-- tensor类型，(4,None),yolo\_filter\_boxes()的输出* ***:param*** *classes: --tensor类型，( ,None ),yolo\_filter\_boxes()的输出* ***:param*** *max\_boxes:-- Integer,预测锚框数量的最大值* ***:param*** *iou\_threshold: --real value，交并比阈值* ***:return****:  
 scores: --tensor,( ,None),predicted score for each box  
 boxes: --tensor,(4,None),predicted box coordinates  
 classes: --tensor,( ,None),predicted class for each box  
 Note:The "None" dimension of the output tensors has obviously to be less than max\_boxes.  
 """* # tensor类型，将被用于tf.image.non\_max\_suppression()方法中  
 max\_boxes\_tensor = K.variable(max\_boxes, dtype="int32")  
 # 初始化变量max\_boxes\_tensor  
 K.get\_session().run(tf.variables\_initializer([max\_boxes\_tensor]))  
 # 使用tf.image.non\_max\_suppression()来获取我们保留框对应的索引列表  
 nms\_indices = tf.image.non\_max\_suppression(boxes, scores, max\_boxes,iou\_threshold)  
  
 # 使用K.gather()来选择保留的锚框  
 scores = K.gather(scores, nms\_indices)  
 boxes = K.gather(boxes, nms\_indices)  
 classes = K.gather(classes, nms\_indices)  
  
 return scores, boxes, classes

**2.4 整合两个过滤器实现对所有框进行过滤**

下面实现一个函数了，该函数的输入是DEEP CNN (19x19x5x85 dimension encoding)的输出，并使用刚刚实现的函数过滤所有框

我们实现一个yolo\_eval()函数：输入是DEEP CNN的输出，并使用score和NMS过滤器对anchor进行过滤，（这里有个细节说一下，我们表示一个anchor box有方式，①通过midpoint and height/width，②通过corner coordinate表示）YOLO使用以下功能（我们提供）在不同时间在几种这样的格式之间进行转换：

boxes = yolo\_boxes\_to\_corners(box\_xy, box\_wh)

它将yolo锚框坐标（x，y，w，h）转换为角的坐标（x1，y1，x2，y2）以适应yolo\_filter\_boxes()的输入

boxes = yolo\_utils.scale\_boxes(boxes, image\_shape)

YOLO网络是在608 × 608图像上训练的，如果你想在不同尺寸的的图像上进行测试，比如，汽车检测的数据是720 ×1280的图片，这一步重新调整了anchor box的大小，这样它们就可以绘制在原始的720x1280图像上（上面的两个方法直接调用就行）。

def yolo\_eval(yolo\_outputs, image\_shape=(720.,1280.), max\_boxes=10, score\_threshold=0.6, iou\_threshold=0.5):  
 *"""  
 将YOLO编码的输出（很多框）转换为预测框以及他们的分数、框坐标和类* ***:param*** *yolo\_outputs: 编码模型的输出（对于维度为608\*608\*3的图片），包含4个tensor类型的变量：  
 box\_confidence:--tensor类型，shape of (None,19,19,5,1)  
 box\_xy:--tensor类型，shape of (None,19,19,5,2)  
 box\_wh:--tensor类型，shape of (None,19,19,5,2)  
 box\_class\_probs:--tensor类型， shape of (None,19,19,5,80)* ***:param*** *image\_shape:--tensor类型，shape of (2, )，包含了输入的图像的维度，这里是(608, 608)* ***:param*** *max\_boxes:--integer,预测的锚框数量的最大值* ***:param*** *score\_threshold:--real value，可能的阈值* ***:param*** *iou\_threshold:--real value,交并比阈值* ***:return****:  
 scores:--tensor类型，shape of (None, ),每个锚框的预测的可能值  
 boxes:--tensor类型，shape of (None,4),预测锚框的坐标  
 classes:--tensor类型，shape of (None, ),每个锚框的预测的分类  
 """* # 获取YOLO模型的输出  
 box\_confidence, box\_xy, box\_wh, box\_class\_probs = yolo\_outputs  
  
 # 中心点转换为边角  
 boxes = yolo\_boxes\_to\_corners(box\_xy, box\_wh)  
  
 # score过滤，第一个过滤器  
 scores, boxes, classes = yolo\_filter\_boxes(box\_confidence,boxes,box\_class\_probs, score\_threshold)  
  
 # 缩放锚框，以适应原始图像  
 boxes = scale\_boxes(boxes, image\_shape)  
  
  # 使用非最大值抑制，第二个过滤器  
 scores, boxes, classes = yolo\_non\_max\_suppression(scores, boxes, classes, max\_boxes, iou\_threshold)  
 return scores, boxes, classes

**3、测试已经训练好的YOLO模型**

这部分，我们将使用一个预先训练好的模型并在汽车检测数据集上进行测试。首先创建一个会话来启动计算图

sess = K.get\_session()

**3.1 定义分类，锚框和图像维度**

之前提到，我们去检查80个类别，并且使用5个anchor box，这里有两个文件"coco\_classes.txt" and "yolo\_anchors.txt"，包括了80类和5个anchor box的信息，我们将这些数据加载到模型中

class\_names = read\_classes('coco\_classes.txt文件路径')  
anchors = read\_anchors('yolo\_anchors.txt路径')  
image\_shape = (720., 1280.) #测试数据集中图像维度是（720.，1280.）我们需要预处理成（608,608）

**3.2 加载已经训练好的模型**

训练一个YOLO模型需要很长时间，并且需要为大量目标类提供大量的带标签边框数据集，所以加载存储在“YOLO .h5”中的现有的经过预处理的Keras YOLO模型

yolo\_model = load\_model('yolo.h5模型地址')

这将加载模型的权重，通过下面的代码，可以看到每一层的summary

yolo\_model.summary()

**3.3将模型的输出装换为边界框**

yolo\_model的输出是一个（m，19,19,5,85）的tensor变量，它需要进行处理和转换，如下

yolo\_outputs = yolo\_head(yolo\_model.output, anchors, len(class\_names))

**3.4 过滤anchor boxes**

yolo\_outputs为我们提供了yolo\_model的所有预测框的正确的格式，现在可以执行过滤并仅选择最佳的锚框，调用之前实现的yolo\_eval()

scores, boxes, classes = yolo\_eval(yolo\_outputs, image\_shape)

**3.5在实际图像中运行计算图**

我们已经创建好了回话sess，回顾一下:

（1）yolo\_model.input是yolo\_model的输入，yolo\_model.output是yolo\_model的输出。

（2）yolo\_model.output会让yolo\_head进行处理,这个函数最后输出yolo\_outputs

（3） yolo\_outputs会让一个过滤函数yolo\_eval进行处理，然后输出预测：scores、 boxes、 classes

现在我们要实现predict()函数，使用它来对图像进行预测，需要运行TensorFlow的Session会话，然后在计算图上计算scores、 boxes、 classes，下面的代码可以帮你预处理图像

****请注意！****当模型使用BatchNorm（比如YOLO中的情况）时，您需要在feed\_dict 中传递一个额外的占位符{K.learning\_phase()：0}

def predict(sess, image\_file, is\_show\_info=True, is\_plot=True):  
 *"""  
 运行存储在sess的计算图以预测image\_file的边界框，打印出预测图与信息* ***:param*** *sess: 包含了YOLO计算图的TensorFlow/keras的会话* ***:param*** *imagefile: 存储images文件下的图片名称* ***:param*** *is\_show\_info:* ***:param*** *is\_plot:* ***:return****:  
 out\_scores:tensor, (None, ),锚框的预测的可能值  
 out\_boxes:tensor, (None,4),包含了锚框位置信息  
 out\_classes:tensor, (None, ),锚框的预测的分类索引  
 """* image, image\_data = preprocess\_image(image\_file, model\_image\_size =(608, 608))###预处理图像  
 out\_scores, out\_boxes, out\_classes = sess.run([scores,boxes,classes],feed\_dict={yolo\_model.input:image\_data, K.learning\_phase():0})  
 if is\_show\_info:  
 print("在" + str(image\_file)+"中找到"+str(len(out\_boxes))+"个锚框。")  
 colors = generate\_colors(class\_names)  
 draw\_boxes(image, out\_scores, out\_boxes, out\_classes, class\_names, colors)  
 image.save(os.path.join('C:\\Users\\korey\\Desktop\\car',image\_file), quality=90)#保存的路径（可改）  
 if is\_plot:  
 out\_image = plt.imread(os.path.join('C:\\Users\\korey\\Desktop\\car',image\_file))#保存的路径（可改）  
 plt.imshow(out\_image)  
 plt.show()  
 return out\_scores, out\_boxes, out\_classes

实际测试一下

out\_scores, out\_boxes, out\_classes = predict(sess, "test.jpg")

如下结果

在test.jpg中找到了7个锚框。

car 0.60 (925, 285) (1045, 374)

car 0.66 (706, 279) (786, 350)

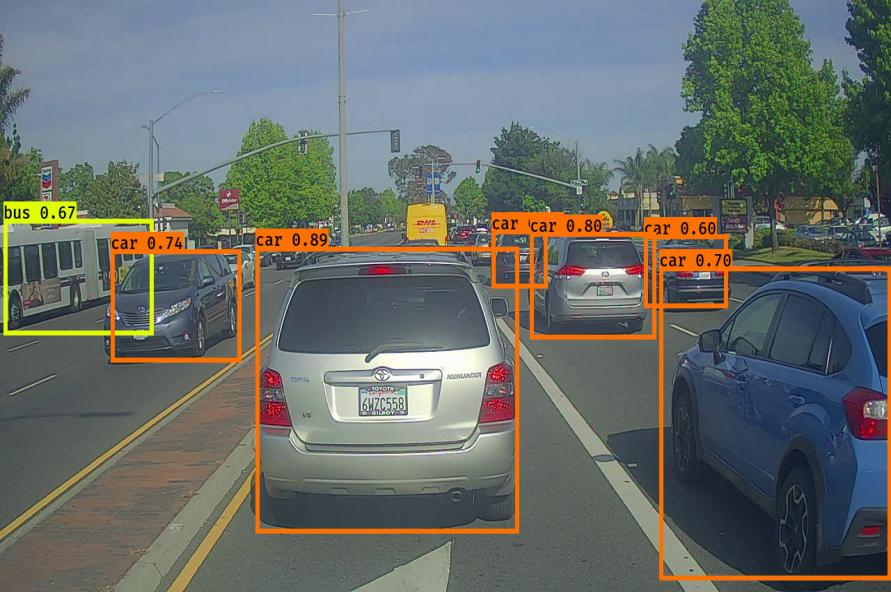
bus 0.67 (5, 266) (220, 407)

car 0.70 (947, 324) (1280, 705)

car 0.74 (159, 303) (346, 440)

car 0.80 (761, 282) (942, 412)

car 0.89 (367, 300) (745, 648)



我们还可以对所有的测试图片进行测试，如下：

rootdir = 'images图片路径'  
for parent,dirnames,filenames in os.walk(rootdir):

#1.parent父目录 2.dirnames所有文件夹名字（不含路径） 3.filenames所有文件名字

for filename in filenames:  
 print('当前图片：'+str( os.path.join(parent, filename)))  
 out\_scores, out\_boxes, out\_classes = predict(sess, os.path.join(parent, filename))