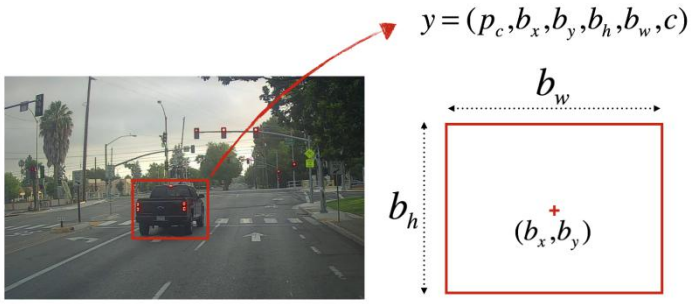
自动驾驶 -- 汽车识别

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：



$p_c=1$: confidence of an object being present in the bounding box
$c=3$: class of the object being detected (here 3 for "car")

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。

2、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

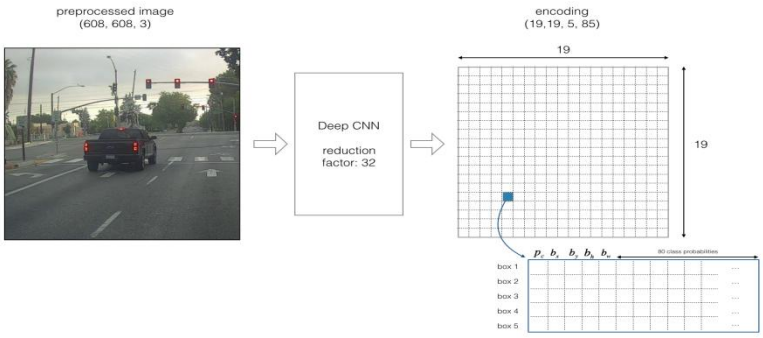
首先要知道的是：

(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)

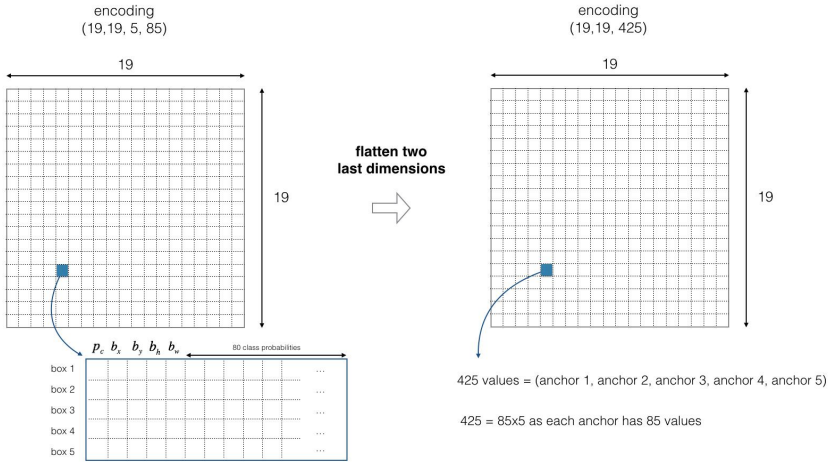
下面是 ENCODING 编码的情况：



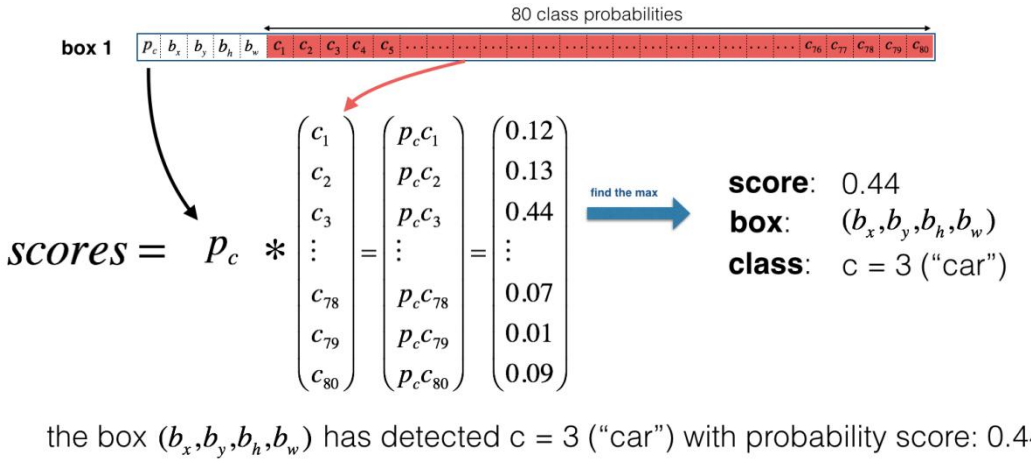
如果对象的中心/中点在单元格内，那么单元格就负责识别该对象。

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（P_c，b_x，b_y，b_h，b_w，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
```

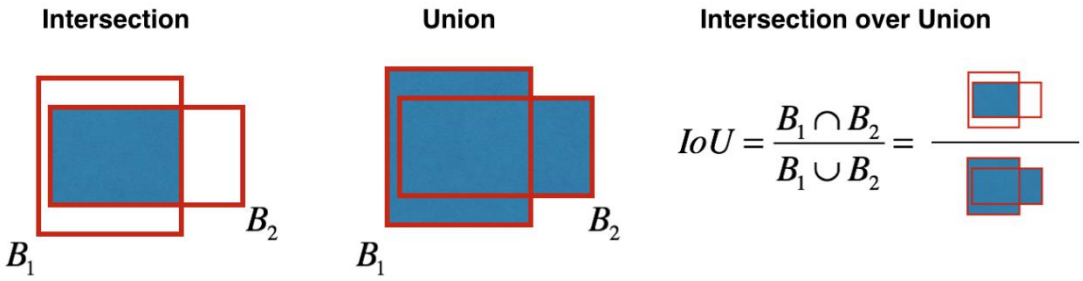
2.3 non_max_suppression

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（IOU）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()
- K.gather()

```
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)

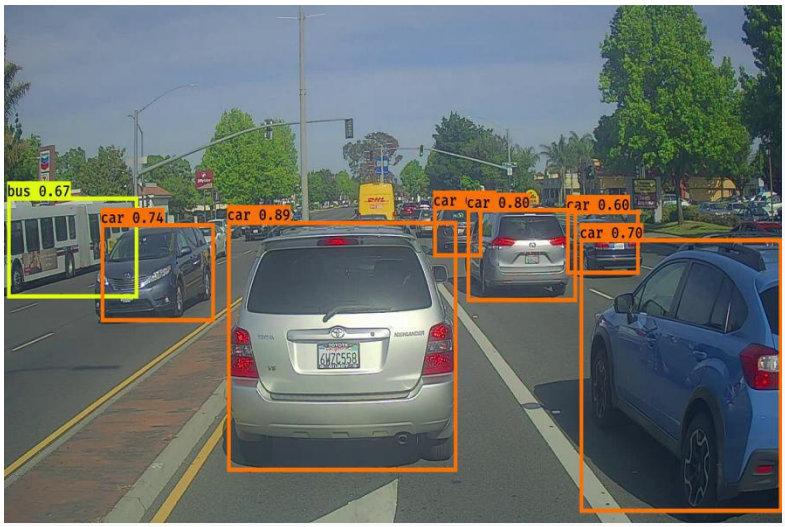
bus 0.67 (5, 266) (220, 407)

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))
```