模式识别创新研修课课程总结报告

1170400423 尉前进 2020.6.16

YOLOV3 目标检测算法解读以及应用

一.解读代码流程部分

从 train.py 看起，比较容易些

首先是获得网络需要的 y_true 的过程，因为 ground_truth 不是直接被使用的，它需要找到 Anchor,变成 y_true 的格式才能被真正用于网络训练；

1.首先定义训练数据生成器函数：(train.py_中)

```
def data_generator(annotation_lines, batch_size, input_shape, anchors, num_classes): # annotation_lines是一个列表，它的值从annotation_path中读取，读取的值是从0到数据集中被分为训练
    '''data generator for fit_generator'''
    n = len(annotation_lines)#用于训练的数据总个数
    i = 0
    while True:
        image_data = []
        box_data = []
        for b in range(batch_size):
            if i==0:
                np.random.shuffle(annotation_lines) #将列表随机打乱
            image, box = get_random_data(annotation_lines[i], input_shape, random=True)# 返回值是经过resize的图片和图片中标注的ground_true，都要发生变化，为了了解清楚，要跳转到get_
            image_data.append(image)
            box_data.append(box)
            i = (i+1) % n
        image_data = np.array(image_data) #这里面是形状为416*416，像素值在0-1之间的所有图片，也可以理解为数组
        box_data = np.array(box_data) #这里面是上面对应的416*416的所有图片他里面框的位置信息以及类别信息;相当于修改以后的label;
        y_true = preprocess_true_boxes(box_data, input_shape, anchors, num_classes)
        yield [image_data, *y_true], np.zeros(batch_size)

```

在训练数据生成器中

第一个重点是进行数据增强处理，**做数据增强的意义在于每一个 epoch 训练的时候，输入的信息都是有差别的。**相当于扩充了数据集。(在 utils.py 中)

在数据增强处理中，先把图像 resize 成标准的 416*416 形式，**再做图像随机翻转，图像扭曲**；最后做图像像素的归一化处理。

```
def get_random_data(annotation_line, input_shape, random=True, max_boxes=20, jitter=.3, hue=.1, sat=1.5, val=1.5, proc_img=True):
    '''r实时数据增强的随机预处理'''
    line = annotation_line.split()
    image = Image.open(line[0]) # 把txt文件中的图像用Image函数加载进来，line[0]就指图片，D:\yolo3-keras-master/VOCdevkit/VOC2007/JPEGImages/0000.jpg 173,168,289,309,0
    iw, ih = image.size # 实际图片的尺寸
    h, w = input_shape # 要求的resize后的图片416*416
    box = np.array([np.array(list(map(int,box.split(',')))) for box in line[1:]]) # line[1:]就指ground_truth里面的框的位置和宽高信息以及包含物体的信息，变成数组的格式

    # resize image，随机改变图像的尺寸
    new_ar = w/h * rand(1-jitter,1+jitter)/rand(1-jitter,1+jitter)
    scale = rand(.25, 2)
    if new_ar < 1:
        nh = int(scale*h)
        nw = int(nh*new_ar)
    else:
        nw = int(scale*w)
        nh = int(nw/new_ar)
    image = image.resize((nw,nh), Image.BICUBIC)

    # place image，使图像的尺寸变为规范形式（416*416）
    dx = int(rand(0, w-nw))
    dy = int(rand(0, h-nh))
    new_image = Image.new('RGB', (w,h), (128,128,128))
    new_image.paste(image, (dx, dy))
    image = new_image

    # flip image or not 翻转
    flip = rand()<.5
    if flip: image = image.transpose(Image.FLIP_LEFT_RIGHT)

    # distort image 扭曲图像
    hue = rand(-hue, hue)
    sat = rand(1, sat) if rand()<.5 else 1/rand(1, sat)
    val = rand(1, val) if rand()<.5 else 1/rand(1, val)
    x = rgb_to_hsv(np.array(image)/255.)
    x[..., 0] += hue # 第一个通道
    x[..., 0][x[..., 0]>1] -= 1 #保证所有的值都在0-1之间，大于1的减去1
    x[..., 0][x[..., 0]<0] += 1 #保证所有的值都在0-1之间，小于0的加上1
    x[..., 1] *= sat # 第二个通道
    x[..., 2] *= val # 第三个通道
    x[x>1] = 1 #进一步保证0-1范围内
    x[x<0] = 0 #进一步保证0-1范围内
    image_data = hsv_to_rgb(x) # numpy array, 0 to 1
```

由于图片被 resize 成标准形式，**它对应的框也要变成标准形式**；代码如下：

```
    box_data = np.zeros((max_boxes,5))
    if len(box)>0:
        np.random.shuffle(box)
        box[:, [0,2]] = box[:, [0,2]]*nw/iw + dx
        box[:, [1,3]] = box[:, [1,3]]*nh/ih + dy
        if flip: box[:, [0,2]] = w - box[:, [2,0]]
        box[:, 0:2][box[:, 0:2]<0] = 0
        box[:, 2][box[:, 2]>w] = w
        box[:, 3][box[:, 3]>h] = h
        box_w = box[:, 2] - box[:, 0]
        box_h = box[:, 3] - box[:, 1]
        box = box[np.logical_and(box_w>1, box_h>1)] # discard invalid box
        if len(box)>max_boxes: box = box[:max_boxes]
        box_data[:len(box)] = box # 一幅图像(1 个 ground_truth,1个2007_train.txt,图像是从txt文件里解析出来的)中最大框的个数为20个，从0到len(box)是指有这么多框(5列)是有值的，其他为0
```

第二个重点是编码过程，即得到 y_true 的过程

对应的函数为：(在 train.py 中)

```
#   读入xml文件，并输出y_true
#---------------------------------------------------#
def preprocess_true_boxes(true_boxes, input_shape, anchors, num_classes):

    assert (true_boxes[..., 4]<num_classes).all(), 'class id must be less than num_classes' #true

    # 一共有三个特征层数
```

- xml 标注的格式是矩形主对角线上两点的坐标值，首先将其转换成矩形的中心点坐标和宽高信息；
- 然后再除以 416 变成归一化的坐标格式
- 对 Anchor 和修改过的 ground_truth(即未经过归一化的宽和高)进行处理后，计算真实框和哪个 Anchor 的 IOU 最大，就取这个 Anchor,这里计算的时候不分层，等确定了 Anchor 后便通过 enumerate 函数得到这个 Anchor 位于哪一层
- 最后，得到归一化的 bx,by,bw,bh,但是反解的 tx,ty,th,tw 在 Loss 中才求解得到

```
    # 为Anchor标注的过程，反解公式里的tx,ty,tw,th,但是没看出来咋写得(这里没有反解)
    for t, n in enumerate(best_anchor):#这个函数是啥意思？返回的参数是啥意思？
        for l in range(num_layers):
            if n in anchor_mask[l]:# 如果n在这一层，才会进行下面的过程，不在则继续搜索，但是还不太明白这个n是个啥？
                # floor用于向下取整，中心点在原图的位置，这里面返回到原图了416*416的图
                i = np.floor(true_boxes[b,t,0]*grid_shapes[l][1]).astype('int32')
                j = np.floor(true_boxes[b,t,1]*grid_shapes[l][0]).astype('int32')
                # 找到真实框在特征层l中第b副图像对应的位置
                k = anchor_mask[l].index(n)
                c = true_boxes[b,t, 4].astype('int32')
                y_true[l][b, j, i, k, 0:4] = true_boxes[b,t, 0:4] #true_boxes[b,t,]这里没有做反解，只是确定了groung_truth对应的Anchor在13*13or26*26or52*52的尺度下
                y_true[l][b, j, i, k, 4] = 1
                y_true[l][b, j, i, k, 5+c] = 1
                # 其他位置全是0
    return y_true #坐标和长宽是归一化的值，但是不是真正的y_true,而是归一化的bx,by,bw,bh，真正的y_true在求解Loss的时候才反解出来！
```

到此，编码过程结束，得到了真正的标签 y_true.（没有反解的，不算真的。2020.6.15 修改）

2. 接下来便是训练过程中的前向传播过程

首先在 train.py 中看到加载 Yolo 模型的地方，这便是 Model 的搭建过程

```
    # 创建yolo模型
    print('Create YOLOv3 model with {} anchors and {} classes.'.format(num_anchors, num_classes))
    model_body = yolo_body(image_input, num_anchors//3, num_classes)
```

然后，一切关于训练中前向传播的过程必然在 yolo_body 中（yolo3.py)中

分析过程如下：

```
53
54  # ---------------------------------------------------#
55  #   特征层->最后的输出
56  # ---------------------------------------------------#
57  def yolo_body(inputs, num_anchors, num_classes):
58      # 生成darknet53的主干模型
59      # 三个特征层的形状大小分别为
60      # 52,52,256
61      # 26,26,512
62      # 13,13,1024
63      feat1, feat2, feat3 = darknet_body(inputs) #416*416
64      darknet = Model(inputs, feat3) #Keras里搭建好模型的一个表达方式,设定输入为inputs即416*416,输出为darknet网络的最后一层，即feat3
65
66      # 第一个特征层
67      # y1=(batch_size,13,13,3,25) 85是cooc数据集，25是VOC数据集！，x是对应着5次卷积后的结果
68      x, y1 = make_last_layers(darknet.output, 512, num_anchors*(num_classes+5)) # draknet.output的输出就是feat3
69
70      x = compose(
71              DarknetConv2D_BN_Leaky(256, (1, 1)),
72              UpSampling2D(2))(x)
73      x = Concatenate()([x, feat2])  # 将第一个特征层进行一次卷积和上采样，然后与第二个特征层进行结合
74      # 第二个特征层
75      # y2=(batch_size,26,26,3,25)
76      x, y2 = make_last_layers(x, 256, num_anchors*(num_classes+5))
77
78      x = compose(
79              DarknetConv2D_BN_Leaky(128, (1, 1)),
80              UpSampling2D(2))(x)
81      x = Concatenate()([x, feat1])
82      # 第三个特征层
83      # y3=(batch_size,52,52,3,25)
84      x, y3 = make_last_layers(x, 128, num_anchors*(num_classes+5))
85
86      return Model(inputs, [y1, y2, y3])
87
```

输入的值是 416*416，返回的结果是一个模型 Model(Model(input,[y1,y2,y3]),其中 y1,y2,y3 为我们最后需要的 3 个尺度下的输出，它的格式与 y_true 是一致的。因此便可以计算 Loss 了

然后按着代码读下去，就到了

```
# y_true为13,13,3,85
# 26,26,3,85
# 52,52,3,85
# 这里重新定义一个y_true的意义在哪里？
y_true = [Input(shape=(h//{0:32, 1:16, 2:8}[l], w//{0:32, 1:16, 2:8}[l], \
    num_anchors//3, num_classes+5)) for l in range(3)]

# 输入为*model_body.input, *y_true
# 输出为model_loss
loss_input = [*model_body.output, *y_true]
model_loss = Lambda(yolo_loss, output_shape=(1,), name='yolo_loss',
    arguments={'anchors': anchors, 'num_classes': num_classes, 'ignore_thresh': 0.5})(loss_input)

model = Model([model_body.input, *y_true], model_loss)
```

然后我们去寻找 yolo_loss 这个函数，看如何计算 Loss,在(loss.py)中；
得到 y_pre 的过程：

```
# 将yolo_outputs的特征层输出进行处理
# grid为网格结构(13,13,1,2)，raw_pred为尚未处理的预测结果(m,13,13,3,85)
# 还有解码后的xy，wh，(m,13,13,3,2)
grid, raw_pred, pred_xy, pred_wh = yolo_head(yolo_outputs[l],#yolo_output[1]只有13*13大小，以它为例
     anchors[anchor_mask[l]], num_classes, input_shape, calc_loss=True)#说明如果输出的特征在第1个输出尺度上，则在为它寻找Anchor时就是这个尺度上的Anchor

# 这个是解码后的预测的box的位置
# (m,13,13,3,4)
```

在 yolo_head()函数中，返回解码的值
yolo_head()中，有两个作用，在训练时和在预测时的返回值不同
在训练时返回的是：
1.grid
2.网络输出的 feats 经过了一次 reshape,变成与 y_true 一致的格式，便于后面求解 Loss，但内容没有发生改变，还是 tx,ty,tw,th,这几个参数
3.还要返回 bx,by,bw,bh 这几个经过论文公式转换的值，在后面会用到

```python
def yolo_head(feats, anchors, num_classes, input_shape, calc_loss=False):
    num_anchors = len(anchors)
    # [1, 1, 1, num_anchors, 2]
    anchors_tensor = K.reshape(K.constant(anchors), [1, 1, 1, num_anchors, 2])# 第一个维度指batch_size,第2、3维度指anchor的shape,第4个维度是anchor的数量，第5个维度指anchor的宽高

    # 获得x，y的网格
    # (13, 13, 1, 2)
    grid_shape = K.shape(feats)[1:3] # height, width 13*13
    grid_y = K.tile(K.reshape(K.arange(0, stop=grid_shape[0]), [-1, 1, 1, 1]),
        [1, grid_shape[1], 1, 1])
    grid_x = K.tile(K.reshape(K.arange(0, stop=grid_shape[1]), [1, -1, 1, 1]),
        [grid_shape[0], 1, 1, 1])
    grid = K.concatenate([grid_x, grid_y]) #整数,就是网格数的值，这部分代码有点难
    grid = K.cast(grid, K.dtype(feats))

    # (batch_size,13,13,3,85) 这一步把yolo_body的输出变成了跟y_true对应的维数
    feats = K.reshape(feats, [-1, grid_shape[0], grid_shape[1], num_anchors, num_classes + 5])#第一个维度是batch_size,第2、3个维度是特征层的shape,...

    # 将预测值调成真实值
    # box_xy对应框的中心点
    # box_wh对应框的宽和高
    # 依据论文里的公式从预测的tx,ty,tw,th(存在于feats[...,:2]和feats[...,2:4])中
    box_xy = (K.sigmoid(feats[..., :2]) + grid) / K.cast(grid_shape[::-1], K.dtype(feats))
    box_wh = K.exp(feats[..., 2:4]) * anchors_tensor / K.cast(input_shape[::-1], K.dtype(feats))
    box_confidence = K.sigmoid(feats[..., 4:5])
    box_class_probs = K.sigmoid(feats[..., 5:])

    # 在计算loss的时候返回如下参数
    if calc_loss == True:
        return grid, feats, box_xy, box_wh
    return box_xy, box_wh, box_confidence, box_class_probs
```

接下来便是**用公式反解**真正的 y_true:

代码如下：

```python
    # 将真实框进行编码，使其格式与预测的相同，后面用于计算loss
    raw_true_xy = y_true[l][..., :2]*grid_shapes[l][:] - grid #由于y_true的坐标值和宽高在0-1之间，为了和公式对应，得到真正的tx,ty,tw,th,
    raw_true_wh = K.log(y_true[l][..., 2:4] / anchors[anchor_mask[l]] * input_shape[::-1])
```

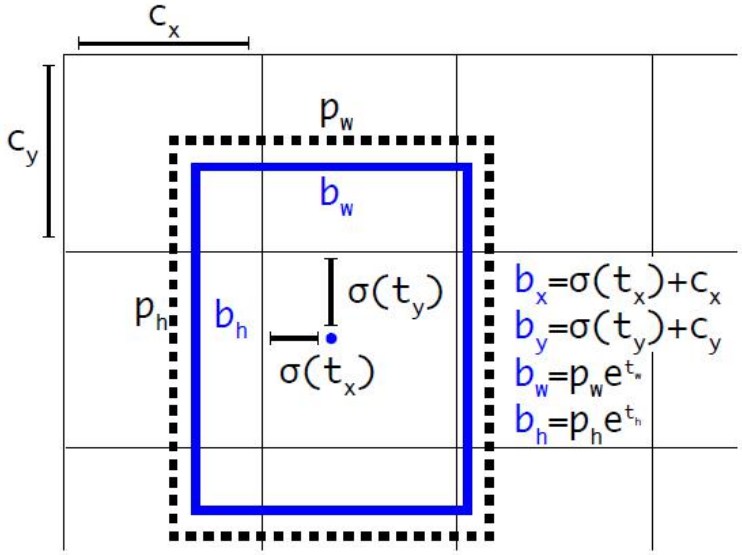


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [15].

这里面虽然没有 $sigmoid()$ 的反函数，但是它**求解出来的 $\hat{t}_x\ \hat{t}_y$ 的值就是在 0-1 之间**的（经过实际计算发现就是这样的）而 grid 就是论文中的 $c_x c_y$，grid 是一个具有两个值的列表；

而对于宽和高，即 $\hat{t}_w\ \hat{t}_h$,它们的值应该是原 416*416 图像中的值 $b_w b_h$ 反解出来的，而不是归一化的值反解出来的，但是我们之前得到的 y_true 所有值都被归一化了，因此先乘以 416，

再利用论文里的公式反解得到$\hat{t}_w \hat{t}_h$，至此，得到了真正的$\hat{t}_x \hat{t}_y \hat{t}_w \hat{t}_h$，然后网络的输出要逐渐逼近这几个值，得到最优的$t_x t_y t_w t_h$，再由公式得到最优的$b_x b_y b_w b_h$，但是需要注意的是不光$b_x b_y$在 0-1 之间，**而且$b_h b_w$也被归一化到了 0-1 之间，代码是这么写的，可能是为了与 y_true 保持一致，便于后面在训练的过程中计算 IOU**,这几个最优的值是在测试过程中返回的，即 yolo_head 的另一个作用；它在测试的时候又被调用(yolo3.py 中),
这个在测试的时候再具体分析；

再解码后（调用 yolo_head()以后），便可以得到一个真正的 bounding_box(bx,by,bw,bh),虽然在坐标预测值的回归过程中用不到，但是在类别 Loss 的计算中要用到，下面具体分析：
如图：

```
        pred_box = K.concatenate([pred_xy, pred_wh])

        # 找到负样本群组，第一步是创建一个数组，[]
        ignore_mask = tf.TensorArray(K.dtype(y_true[0]), size=1, dynamic_size=True)
        object_mask_bool = K.cast(object_mask, 'bool')

        # 对每一张图片计算ignore_mask
        def loop_body(b, ignore_mask):
            # 取出第b副图内，真实存在的所有的box的参数
            # n,4，下面这句应该是个逻辑运算
            true_box = tf.boolean_mask(y_true[l][b,...,0:4], object_mask_bool[b,...,0])
            # 计算预测结果与真实情况的iou
            # pred_box为13,13,3,4
            # 计算的结果是每个pred_box和其它所有真实框的iou
            # 13,13,3,n
            iou = box_iou(pred_box[b], true_box)# 都是归一化的，计算IOU更方便！

            # 13,13,3,1
            best_iou = K.max(iou, axis=-1)

            # 判断预测框的iou小于ignore_thresh则认为该预测框没有与之对应的真实框
            # 则被认为是这幅图的背景，把IOU小于规定阈值的框全部认为是背景
            ignore_mask = ignore_mask.write(b, K.cast(best_iou<ignore_thresh, K.dtype(true_box)))
            return b+1, ignore_mask

        # 遍历所有的图片
        _, ignore_mask = K.control_flow_ops.while_loop(lambda b,*args: b<m, loop_body, [0, ignore_mask])#这是一个循环语句

        # 将每幅图的内容压缩，进行处理
        ignore_mask = ignore_mask.stack()
        #(m,13,13,3,1,1)
        ignore_mask = K.expand_dims(ignore_mask, -1)
```

它返回了一个 ignore_mask,大致意思就是说把那些有中心点落在区域范围内，但是 Anchor 与 bounding_box 的 IOU 太小的，认为它也是没有物体的，是背景框，这一步在计算是否有物体的 Loss 中得以体现，具体作用还不清楚。
接下来便是 Loss_Function 的构建过程。代码如下：

```
    # object_mask如果真实存在目标则保存其wh值
    # switch接口，就是一个if/else条件判断语句
    raw_true_wh = K.switch(object_mask, raw_true_wh, K.zeros_like(raw_true_wh))
    box_loss_scale = 2 - y_true[l][...,2:3]*y_true[l][...,3:4]

    xy_loss = object_mask * box_loss_scale * K.binary_crossentropy(raw_true_xy, raw_pred[...,0:2], from_logits=True)
    wh_loss = object_mask * box_loss_scale * 0.5 * K.square(raw_true_wh-raw_pred[...,2:4])

    # 如果该位置本来有框，那么计算1与置信度的交叉熵
    # 如果该位置本来没有框，而且满足best_iou<ignore_thresh，则被认定为负样本
    # best_iou<ignore_thresh用于限制负样本数量
    confidence_loss = object_mask * K.binary_crossentropy(object_mask, raw_pred[...,4:5], from_logits=True)+ \
        (1-object_mask) * K.binary_crossentropy(object_mask, raw_pred[...,4:5], from_logits=True) * ignore_mask

    class_loss = object_mask * K.binary_crossentropy(true_class_probs, raw_pred[...,5:], from_logits=True)

    xy_loss = K.sum(xy_loss) / mf
    wh_loss = K.sum(wh_loss) / mf
    confidence_loss = K.sum(confidence_loss) / mf
    class_loss = K.sum(class_loss) / mf
    loss += xy_loss + wh_loss + confidence_loss + class_loss
    if print_loss:
        loss = tf.Print(loss, [loss, xy_loss, wh_loss, confidence_loss, class_loss, K.sum(ignore_mask)], message='loss: ')
return loss
```

而 Loss_function 的计算公式为：

$$loss(object) = \sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{obj}(2-w_i*h_i)(-x_i*log(\hat{x_i})-(1-x_i)*log(1-x_i^*))+$$

$$\sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{obj}(2-w_i*h_i)(-y_i*log(\hat{y_i})-(1-y_i)*log(1-y_i^*))+$$

$$\sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{obj}(2-w_i*h_i)[(w_i-w_i^*)^2+(h_i-h_i^*)^2]-$$

$$\sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{obj}[C_i log(C_i^*)+(1-C_i)log(1-C_i^*)]-$$

$$\sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{noobj}[C_i log(C_i^*)+(1-C_i)log(1-C_i^*)]-$$

$$\sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{obj}\sum_{c\in classes}[p_i(c)log(p_i^*(c))+(1-p_i(c))log(1-p_i^*(c))]$$

可见，代码构建的这个有些许不同，具体标准的是哪一个，还需要证实

1. 虽然在坐标预测的过程中这是一个回归问题，但是却用的是交叉熵损失函数，因为它的值是在 0-1 之间的，所以可以用，但是**为什么要用交叉熵损失函数代替均方误差损失函数还不清楚**

2. 在类别的 Loss_Function 中也用交叉熵损失函数而代替 Softmax 损失函数，是因为一个框可以预测多个类别，属于多标签分类？不理解这块儿。

至此，Loss_Function 也构建完毕；

然后利用迁移学习的思想，加载 CoCo 数据集的预训练权重，先对前 249 层（为什么）冻结，只训练后几层，等到 Loss 不再下降了，就解冻全部训练

关于训练参数的设置：

```
# 训练参数设置
logging = TensorBoard(log_dir=log_dir)
checkpoint = ModelCheckpoint(log_dir + 'ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5',
    monitor='val_loss', save_weights_only=True, save_best_only=False, period=2)  # 每过两个周期进行一个保存
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=6, verbose=1)
```

Loss 不下降，就减小学习率；一直不下降，训练就提前停止

关于学习率、epoch、Batch_size 的设置：

```
    # if True:
    #     model.compile(optimizer=Adam(lr=1e-3), loss={
    #         'yolo_loss': lambda y_true, y_pred: y_pred})
    #
    #     batch_size = 16
    #     print('Train on {} samples, val on {} samples, with batch size {}.'.format(num_train, num_val, batch_size))
    #     model.fit_generator(data_generator(lines[:num_train], batch_size, input_shape, anchors, num_classes),
    #             steps_per_epoch=max(1, num_train//batch_size),
    #             validation_data=data_generator(lines[num_train:], batch_size, input_shape, anchors, num_classes),
    #             validation_steps=max(1, num_val//batch_size),
    #             epochs=50,
    #             initial_epoch=0,
    #             callbacks=[logging, checkpoint])
    #     model.save_weights(log_dir + 'trained_weights_stage_1.h5')
```

解冻之后的设置：

```
# 解冻后训练
if True:
    model.compile(optimizer=Adam(lr=1e-4), loss={
        'yolo_loss': lambda y_true, y_pred: y_pred})

    batch_size = 4  # 显存问题改小一些！
    print('Train on {} samples, val on {} samples, with batch size {}.'.format(num_train, num_val, batch_size))
    model.fit_generator(data_generator(lines[:num_train], batch_size, input_shape, anchors, num_classes),
            steps_per_epoch=max(1, num_train//batch_size),
            validation_data=data_generator(lines[num_train:], batch_size, input_shape, anchors, num_classes),
            validation_steps=max(1, num_val//batch_size),
            epochs=40,
            initial_epoch=20,
            callbacks=[logging, checkpoint])
    model.save_weights(log_dir + 'last1.h5')
```

然后，就是预测部分：
首先，要看的当然是 predict.py,在里面发现关键是调用了 yolo 这个类，所以我们去寻找它，（在 yolo.py 中）

```
from PIL import Image
import cv2

yolo = YOLO()

while True:
    img = input('Input image filename:')
    try:
        image = cv2.imread(img)
        image = Image.fromarray(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    except:
        print('Open Error! Try again!')
        continue
    else:
        r_image = yolo.detect_image(image)
        r_image.show()
yolo.close_session()
```

首先看一下它的配置默认值：

```
class YOLO(object):
    _defaults = {
        "model_path": 'logs\last1.h5',  # 载入最终的训练模型
        "anchors_path": 'model_data/yolo_anchors.txt',
        "classes_path": 'model_data/voc_classes.txt',  # 载入分类结果，如果要训练自己的数据集，则要在类里进行修改
        "score": 0.5,
        "iou": 0.3,
        "model_image_size": (416, 416)
    }

```

然后，下面就是一堆初始化和数据处理过程；不是很重要
但是网络的输出仍然需要经过解码，才能最终转换为我们想要的框和结果：
在里面有一个很关键的函数：yolo_eval()(在 yolo3.py 中）

```
        self.input_image_shape = K.placeholder(shape=(2, ))
        # 把模型的预测结果转化成图片中实际位置的预测 的一种方式！ Yolo _model.output就是模型的预测结果，尚未处理的信息，未解码的信息！
        boxes, scores, classes = yolo_eval(self.yolo_model.output, self.anchors,
                num_classes, self.input_image_shape,
                score_threshold =self.score, iou_threshold=self.iou)
        return boxes, scores, classes

```

它返回的是得分、框（bbox）、类别
分别看一下它们怎么得到：

对于 boxes、scores 通过 yolo_boxes_and_scores()函数得到

```
    # 对每个特征层进行处理，一共有3个
    for l in range(num_layers):
        _boxes, _box_scores = yolo_boxes_and_scores(yolo_outputs[l], anchors[anchor_mask[l]], num_classes, input_shape, image_shape)
        boxes.append(_boxes)
        box_scores.append(_box_scores)
```

```
def yolo_boxes_and_scores(feats, anchors, num_classes, input_shape, image_shape):
    # 将预测值调成真实值
    # box_xy对应框的中心点
    # box_wh对应框的宽和高
    # -1,13,13,3,2; -1,13,13,3,2; -1,13,13,3,1; -1,13,13,3,80
    box_xy, box_wh, box_confidence, box_class_probs = yolo_head(feats, anchors, num_classes, input_shape)
    # 将box_xy、和box_wh调节成y_min,y_max,xmin,xmax
    boxes = yolo_correct_boxes(box_xy, box_wh, input_shape, image_shape)
    # 获得得分和box
    # 进行一个reshape,把图片变成一维，每个点有4个参数，分别对应着y_min, y_max, xmin, xmax!
    boxes = K.reshape(boxes, [-1, 4])
    box_scores = box_confidence * box_class_probs
    box_scores = K.reshape(box_scores, [-1, num_classes])  # 可能性
    return boxes, box_scores
```

可以看到，得分相当于是一个条件概率；

而 boxes 则来自于另外一个函数：

```python
def yolo_correct_boxes(box_xy, box_wh, input_shape, image_shape):
    box_yx = box_xy[..., ::-1]
    box_hw = box_wh[..., ::-1]
    # 转换类型
    input_shape = K.cast(input_shape, K.dtype(box_yx))
    image_shape = K.cast(image_shape, K.dtype(box_yx))

    new_shape = K.round(image_shape * K.min(input_shape/image_shape))#这是社么意思，它返回了啥？
    offset = (input_shape-new_shape)/2./input_shape
    scale = input_shape/new_shape

    box_yx = (box_yx - offset) * scale#我觉得这块儿有问题，从开始到推到这里，我感觉得到的box_wh，box_xy的值都在0-1之间
    box_hw *= scale
    # 去除两条灰边，把比例进行转换！！
    box_mins = box_yx - (box_hw / 2.)  # 获得矩形框左上角的点
    box_maxes = box_yx + (box_hw / 2.)  # 获得矩形框右下角的点
    boxes =  K.concatenate([
        box_mins[..., 0:1],  # y_min
        box_mins[..., 1:2],  # x_min
        box_maxes[..., 0:1],  # y_max
        box_maxes[..., 1:2]  # x_max
    ])

    boxes *= K.concatenate([image_shape, image_shape])
    return boxes
```

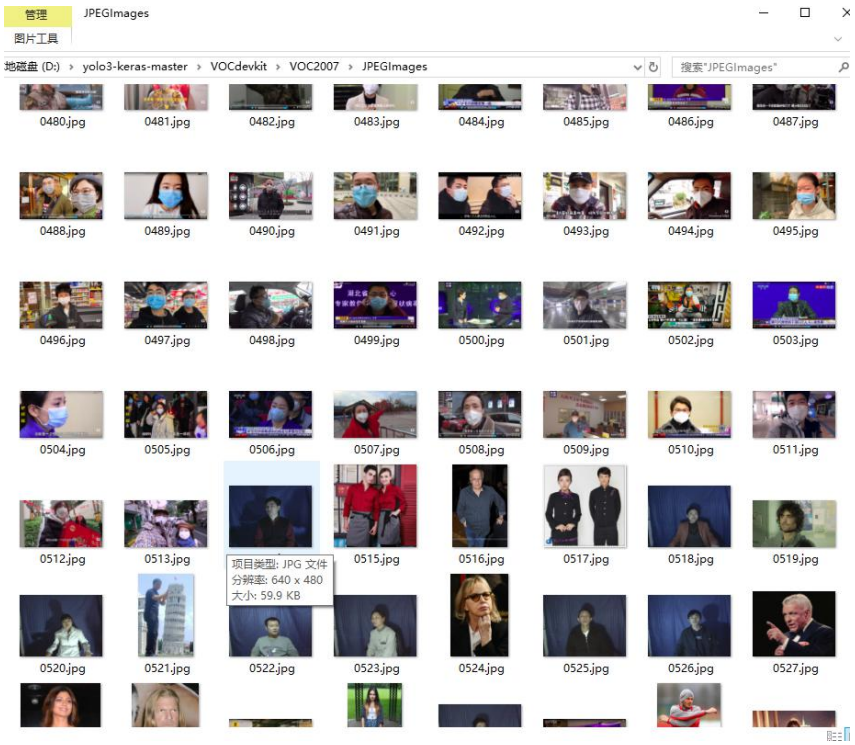
这里还存在一些问题未解决

最后，得到 boxes、score、classes 再进行一个 NMS，便得到了最终的结果。

二. 应用

口罩检测实战

数据集：

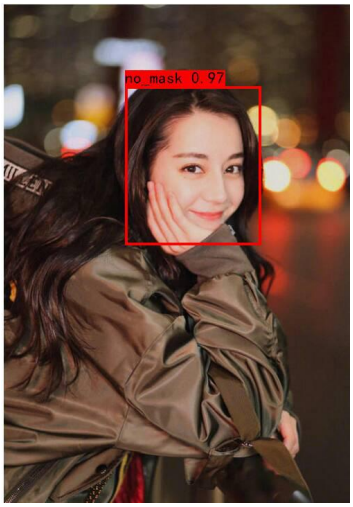


共 513 张正样本，651 张负样本，相当于是一个 2 分类的目标检测的应用

策略:由于这些口罩都在人脸上，所以，如果单纯引入正样本，很可能会导致模型学习到一部分人脸的特征，所以需要将不带口罩的人脸作为负样本，增强模型的识别能力

最后的 Loss 在 10 左右

下面给出检测结果：



下一步计划：

1. 用轻量化的模型提高检测帧率
2. 进一步理解 YoloV3 算法原理
3. 尝试用单类物体的 YoloV3 目标检测