2.3 YOLO入门教程：YOLOv2(3)-基于kmeans聚类方法的先验框、正样本制作与损失函数

Kissrabbit
业余写手/机器人/深度学习/计算机视觉

关注他

一、使用kmeans聚类方法获取先验框

前面已经介绍过，YOLOv2的先验框是使用kmeans聚类的方法来获得的，而非人工设计的。因此，在开展后续的工作之前，我们需要先使用kmeans聚类法为我们的YOLOv2+网络获得适用于VOC数据集的5个先验框。

这一部分的代码已放在项目中的 `generate_ab_kmeans.py` 文件中，读者可以打开查看。下好VOC数据集后，读者即可运行此代码来获取先验框。运行命令如下：

```
python generate_ab_kmeans.py -d voc -na 5 -size 416
```

其中，参数 `-d` 是用来选择数据集，如VOC和COCO；参数 `-na` 是用来确定先验框的个数。在YOLOv2中，先验框的个数为5；参数 `-size` 是输入图像的大小，默认使用416。更多的参数详情，还请读者自行打开代码文件去查看。

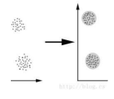
我们先在VOC数据集上聚类出5个先验框尺寸，所得到的结果为{(1.19, 1.98), (2.79, 4.59), (4.53, 8.92), (8.06, 5.29), (10.32, 10.65)}。

注意，以上先验框尺寸是相对于网格的尺寸13×13而言的，算出来的中心点坐标和宽高还要乘以一个变量 `stride` ，这个变量就是网络最后一层的特征成的降采样倍数，其目的就是为了将坐标从网格的尺度映射回输入图像的尺度。

另外，笔者也提供了在COCO数据集上聚类出的先验框尺寸，其结果为{(0.53, 0.79), (1.71, 2.36), (2.89, 6.44), (6.33, 3.79), (9.03, 9.74)}。同上，这5个尺寸也是相对于网格的尺度，而非输入图像的尺度。从先验框的聚类结果便不能发现，COCO上的先验框要略小于VOC，这一点间接表明了COCO所包含的较小的目标的比例要高于VOC。

多说一句，本项目所使用的 `generate_ab_kmeans.py` 文件中的代码参考自下篇博客，感兴趣的读者可以跳转过去，了解一下具体的技术细节。

K-means 计算 anchor boxes
blog.csdn.net/hrsstudy/article/details/71...



二、基于anchor box机制的正样本制作方法

事实上，YOLOv2的检测机制和YOLOv1是一样的，仍是先获得所有边界框的预测，然后与真实框计算IoU，只有IoU最大的边界框会去计算损失，其余的忽略掉。这里，我们稍做改动，这种改动也更加符合当下的基于anchor box机制（即anchor-based方法）的目标检测工作的做法，同时也更加便于理解。

2.1 计算中心点所落在的网格

这一点与我们的YOLOv1+是一样的，我们仍用YOLOv1中的方法确定网格点坐标。

2.2 基于IoU的正样本选择

在YOLOv1+中，由于每个网格只预测一个边界框，无需做选择，但现在每个网格处有5个先验框，需要我们从中选择适合做正样本的先验框。选择原则非常简单，首先，计算这5个先验框与此处的真实框之间的IoU，分别记作 IoU_{B_1} 、IoU_{B_2} 、IoU_{B_3} 、IoU_{B_4} 和 IoU_{B_5} ，然后设定一个IoU阈值 θ 。接下来，我们会遇到两种情况。

情况1：**所有的IoU都小于阈值。**此时，为了不丢失掉这个真实样本，我们选择这当中IoU值最大的那个先验框，不失一般性的，我们假设是 B_1 ，则这个先验框 B_1 将作为正样本，然后使用公式(1)至(4)计算出边界框的位置参数的学习标签。

(1) $t_x = \frac{c_x}{stride} - grid_x$

(2) $t_y = \frac{c_y}{stride} - grid_y$

(3) $t_w = log\left(\frac{w_s}{p_w}\right)$

(4) $t_h = log\left(\frac{h_s}{p_h}\right)$

其中，$w_s = \frac{w}{stride}$ ，$h_s = \frac{h}{stride}$ ，即将真实框的宽高 w 和 h 映射到网格的尺度，因为我们的先验框的尺寸就是在网格的尺度上，如果不做此映射，很显然是有问题的。然后，其余的先验框 B_2 至 B_5 全部作为负样本。不同于正样本，负样本只参与边界框的置信度损失计算，且学习标签是0，不参与类别损失和边界框的 t_x 、t_y 、t_w 和 t_h 的损失计算。

情况2：**有多个IoU大于或等于阈值 θ** 。不失一般性的，我们假设是 B_1 、B_2 和 B_3 都高于阈值 θ ，其中 B_1 和真实框计算出的 IoU_{B_1} 最大，则这个先验框 B_1 将作为正样本，使用公式(1)至(4)计算出边界框的位置参数的学习标签。而 B_2 和 B_3 被忽略，不赋予学习标签，也不参

与任何损失计算，尽管他们和真实框计算出的IoU都大于阈值θ。剩下的 B_4 和 B_5 则作为负样本。情况2的操作其实是借鉴了后续的YOLOv3的做法。

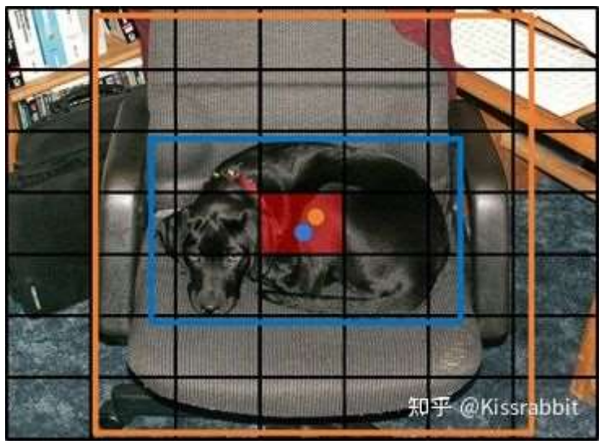


图1 语义歧义

处理完上述两种情况后，我们会发现，理想情况下，**每个真实样本都会匹配到一个先验框**，作为正样本，其余的先验框要么成为负样本，要么被忽略。但是，稍加思考，不难发现这其中有个问题，那就是倘若有两个目标的中心点都落在同一个网格，有没有可能原本分配给目标A的先验框后来又分配给了目标B，导致目标A就没有匹配上的先验框，从而丢失了正样本，这种问题有时被称为“**语义歧义**（semantic ambiguity）”。事实上，如图1所示，这在YOLOv1中也是存在的，当两个目标的都落在同一个网格中，那么这个网格改学哪个目标呢？关于该问题，我们暂且不去做处理，感兴趣的读者可以尝试在制作正样本的代码中添加相应的代码来避免这一问题。

2.3 动手编写正样本制作的代码

确定了正样本的制作规则后，我们就可以编写相应的代码了。这段代码还请读者打开项目中的 `tools.py` 文件，着重阅读 `generate_txtytwth` 函数和 `gt_creator` 函数。

首先，我们编写制作正样本的代码，如下方代码所示。


```
def generate_txtytwth(gt_label, w, h, s, anchor_size):
    xmin, ymin, xmax, ymax = gt_label[:-1]
    # 计算真实边界框的中心点和宽高
    c_x = (xmax + xmin) / 2 * w
    c_y = (ymax + ymin) / 2 * h
    box_w = (xmax - xmin) * w
    box_h = (ymax - ymin) * h

    if box_w < 1e-4 or box_h < 1e-4:
        # print('not a valid data !!!')
        return False

    # 将真是边界框的尺寸映射到网格的尺度上去
    c_x_s = c_x / s
```

```
c_y_s = c_y / s
box_ws = box_w / s
box_hs = box_h / s

# 计算中心点所落在的网格的坐标
grid_x = int(c_x_s)
grid_y = int(c_y_s)

# 获得先验框的中心点坐标和宽高，
# 这里，我们设置所有的先验框的中心点坐标为0
anchor_boxes = set_anchors(anchor_size)
gt_box = np.array([[0, 0, box_ws, box_hs]])

# 计算先验框和真实框之间的IoU
iou = compute_iou(anchor_boxes, gt_box)

# 只保留大于ignore_thresh的先验框去做正样本匹配，
iou_mask = (iou > ignore_thresh)

result = []
if iou_mask.sum() == 0:
    # 如果所有的先验框算出的IoU都小于阈值，那么就将IoU最大的那个先验框分配给正样本.
    # 其他的先验框统统视为负样本
    index = np.argmax(iou)
    p_w, p_h = anchor_size[index]
    tx = c_x_s - grid_x
    ty = c_y_s - grid_y
    tw = np.log(box_ws / p_w)
    th = np.log(box_hs / p_h)
    weight = 2.0 - (box_w / w) * (box_h / h)

    result.append([index, grid_x, grid_y, tx, ty, tw, th, weight, xmin, ymin, xmax, y

    return result

else:
    # 有至少一个先验框的IoU超过了阈值.
    # 但我们只保留超过阈值的那些先验框中IoU最大的，其他的先验框忽略掉，不参与loss计算。
    # 而小于阈值的先验框统统视为负样本。
    best_index = np.argmax(iou)
    for index, iou_m in enumerate(iou_mask):
        if iou_m:
            if index == best_index:
                p_w, p_h = anchor_size[index]
                tx = c_x_s - grid_x
                ty = c_y_s - grid_y
                tw = np.log(box_ws / p_w)
                th = np.log(box_hs / p_h)
```

```
                    weight = 2.0 - (box_w / w) * (box_h / h)

                    result.append([index, grid_x, grid_y, tx, ty, tw, th, weight, \
                                   xmin, ymin, xmax, ymax])
                else:
                    # 对于被忽略的先验框，我们将其权重weight设置为-1
                    result.append([index, grid_x, grid_y, 0., 0., 0., 0., -1.0, 0., 0., 0
        return result

ef gt_creator(input_size, stride, label_lists, anchor_size):
    # 必要的参数
    batch_size = len(label_lists)
    s = stride
    w = input_size
    h = input_size
    ws = w // s
    hs = h // s
    anchor_number = len(anchor_size)
    gt_tensor = np.zeros([batch_size, hs, ws, anchor_number, 1+1+4+1+4])
    # 制作正样本
    for batch_index in range(batch_size):
        for gt_label in label_lists[batch_index]:
            # get a bbox coords
            gt_class = int(gt_label[-1])
            results = generate_txtytwth(gt_label, w, h, s, anchor_size)
            if results:
                for result in results:
                    index, grid_x, grid_y, tx, ty, tw, th, weight, wxmin, ymin, xmax, yma
                    if weight > 0.:
                        if grid_y < gt_tensor.shape[1] and grid_x < gt_tensor.shape[2]:
                            gt_tensor[batch_index, grid_y, grid_x, index, 0] = 1.0
                            gt_tensor[batch_index, grid_y, grid_x, index, 1] = gt_class
                            gt_tensor[batch_index, grid_y, grid_x, index, 2:6] = np.array
                            gt_tensor[batch_index, grid_y, grid_x, index, 6] = weight
                            gt_tensor[batch_index, grid_y, grid_x, index, 7:] = np.array(
                    else:
                        # 对于那些被忽略的先验框，其gt_obj参数为-1，weight权重也是-1
                        gt_tensor[batch_index, grid_y, grid_x, index, 0] = -1.0
                        gt_tensor[batch_index, grid_y, grid_x, index, 6] = -1.0

    gt_tensor = gt_tensor.reshape(batch_size, hs * ws * anchor_number, 1+1+4+1+4)
    return gt_tensor
```

gt_creator 函数的整体思路其实和YOLOv1+是一样的，仍是在batch size的维度上去遍历每一个样本，然后为每一个样本的每一个真实框去计算学习标签。在拿到一个真实框的数据后，

`gt_creator` 函数内部会调用 `generate_txtytwth` 函数去完成制作正样本的主要计算，其基本思路是先计算这个真实框的中心点所在的网格坐标，然后计算该真实框与此网格中的个先验框的IoU。

注意，依据YOLOv2工作，在计算IoU时，我们只关心形状参数，不在意先验框的位置，因为我们只考虑中心点所在的网格中的5个先验框。因此，在计算IoU的时候，我们不妨将真实框和边界框的中心点坐标都临时设置为0，只保留各自的宽高信息。这一点，就是大家常说的“*把所有的框都置于图像的左上角点……*”，事实上，我们不难发现，在YOLOv2中，真实框只考虑其中心点所在的网格中的k个先验框，因此可以认为他们的中心点坐标是相同，即便我们不设置0，设置10、100、1000甚至是-1都是可以的，因为都会被抵消掉。设成0是最方便的。

当然，诸如Faster R-CNN、SSD和RetinaNet则考虑所有网格的所有先验框，不单是中心点所在的网格中的，其他网格处的先验框也会考虑进来，因此，在这些工作中，每个先验框的中心坐标就不能是简单地赋予0了，而是等于各自先验框所在的网格坐标（实际上还是在此基础上稍作调整）。两种做法，孰优孰劣，我们暂且不必关心，但显而易见的是，YOLO这种只考虑中心点的做法，处理起来会更简便、更易理解，也更易学习。

计算出IoU后，我们使用变量 `ignore_thresh` （默认设置为0.5）去筛选先验框，得到变量 `iou_mask` 。倘若此变量为空，即对应情况1，此时没有任何一个先验框与真实框匹配上，那么就选择IoU最高的先验框作为正样本，计算标签，其他的先验框作为负样本，只参与边界框的置信度的学习；否则，对应情况2，此时有至少一个先验框与真实框匹配上了，那么我们选择IoU最高的那个先验框作为正样本，其他的那些IoU高于阈值但不是最高的则忽略掉，不参与任何损失计算，剩余的先验框则做为负样本。

注意，对于一个先验框到底是正样本、负样本还是被忽略的样本，我们使用边界框的权重变量 `weight` 来界定。对于正样本，变量 `weight` 就是边界框损失权重；对于负样本，变量 `weight` 为0；对于被忽略的样本，变量 `weight` 则为-1。

确定了正样本后，在计算训练所需的标签，随后返回到外部的 `gt_creator` 函数，将标签信息存放在已定义好的变量 `gt_tensor` 中即可。注意，变量 `gt_tensor` 是一个 `numpy` 库中的 `ndarray` 类型的变量，其形状为 $[B, h_s, w_s, k, 1+1+4+1+4]$ 其中，B对应batch size大小，h_s是网格的高，w_s是网格的宽，k是先验框的个数，最后的$1+1+4+1+4$分别对应着每个边界框的**置信度（1）**、**类别序号（1）**、**边界框位置参数（4）**、**边界框的权重（1）**以及**真实框的坐标信息（4）**。最后的真实框的坐标信息将会被用于计算预测框与真实框之间的IoU，从而得到置信度的学习标签。

总体上，和YOLOv1+的正样本制作代码比起来，我们的YOLOv2+主要就是多了一个真实框和先验框之间的IoU计算以及考虑两种情况下的正样本制作方法，代码的整体思路没有太大的变化。当然，在计算正样本标签时，我们调用了一些其他的函数，如用于计算真实框与先验框之间的IoU的函数，读者不妨自行阅读这些函数的代码，没有复杂的实现过程，这里就不做过多的叙述了。

三、损失函数

我们的YOLOv2+的损失函数与YOLOv1+是一样，读者可以打开tools.py代码文件查看loss函数。该函数的部分代码如下方代码所示。

```
def loss(pred_conf, pred_cls, pred_txtytwth, pred_iou, label, num_classes):
    # 损失函数
    conf_loss_function = MSEWithLogitsLoss(reduction='mean')
    cls_loss_function = nn.CrossEntropyLoss(reduction='none')
    txty_loss_function = nn.BCEWithLogitsLoss(reduction='none')
    twth_loss_function = nn.MSELoss(reduction='none')
    iou_loss_function = nn.SmoothL1Loss(reduction='none')
```

这里，我们做些简单的说明。

首先，置信度的预测仍使用MSE损失函数，尽管置信度是01范围内的，理应用BCE损失函数更加合适，但经过笔者的多次测试，MSE效果要明显好于BCE，故采用MSE损失函数。

其次，类别的损失仍使用cross-entropy损失函数。再次强调一点，PyTorch官方提供的cross entropy损失函数已经内置了softmax操作，因此，不需要我们单独使用softmax函数对类别预测做一次处理。

然后，边界框的四个预测 t_x 、 t_y 、 t_w 和 t_h 仍是使用BCE损失函数和MSE损失函数。再次强调一点，PyTorch官方提供的binary entropy函数已经内置了sigmoid操作，所以我们不需要单独用sigmoid对 t_x 、 t_y 进行处理。

最后，笔者额外添加了一个用于计算IoU Loss的函数，使用的是SmoothL1损失函数。尽管笔者提供了这一段代码，但笔者并未尝试使用，感兴趣的读者不妨尝试来加入IoU Loss去训练我们的YOLOv2+。

下一节我们开始训练。

敬请期待！

科普不易，还请各位读者多多支持，十分感谢！

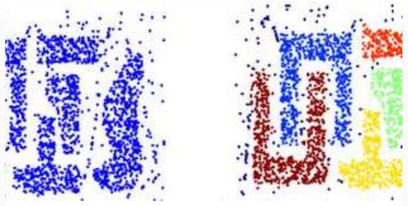
编辑于 2021-12-28 11:17

YOLO算法　目标检测　计算机视觉

文章被以下专栏收录

第二卷-基于YOLO的目标检测入门教程
本专栏将从零开始完整实现YOLOv1至v3模型

推荐阅读



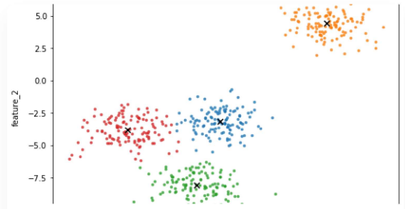
[机器学习基础复习] 常见聚类算法总结

魔法学院的... 发表于机器学习基...

集成学习聚类算法DBSCAN密度聚类算法详解和可视化调参

DBSCAN（Density-Based Spatial Clustering of Applications with Noise，具有噪声的基于密度的聚类方法）是一种很典型的密度聚类算法，和只适用于凸样本集的K-Means聚类相比，DBSCAN既可...

statr 发表于实战统计学



基于轮廓系数确定K-Means聚类中的K

陈罐头 发表于罐头也会数...

基于深度学习的图像实例分割方法研究现状

在计算机视觉领域，一些传统算法，如边缘检测、Adaboost、支持向量机(SVM)等已经获得了成熟的应用，但是在准确率、速度、鲁棒性等方面还有提升的空间。随着深度学习(Deep Learning)技术的不...

AI高级人... 发表于深度人脸识...

11 条评论

切换为时间排序

写下你的评论...

书昨日 2021-11-25

作者写的好棒啊

赞

Kissrabbit (作者) 回复 书昨日 2021-11-26

谢谢支持～😘

赞

Infinite 05-26

对于yolo中制作正样本时候的语义歧义和两个物体的中心落到同一个网格，有哪些处理思路呀

赞

Kissrabbit (作者) 回复 Infinite 05-26

对于纯依赖于手工设计的正样本制作方案，这个问题很难有一个很完美的结局方案。像文章给的做法是一种局部的，因为每次只管当前这个gt的分配，对于今后可能会遇到什

么问题完全不去考虑。所以，解决这个问题，应该从全局的视角，也就是说，我们一次性把所有的gt和正样本放到一块去考虑怎么分配，这就是现在的dynamic label assignment的范式。最近的一个很好的办法是OTA，CVPR2021的一篇文章，旷视提出一个动态label assignment方案，能比较好地解决这个问题。

1

Infinite 回复 Kissrabbit (作者) 05-26

👍

赞

Kemlz 01-26

您好~在本专栏2.1中，有提到“对于宽高比，研究者们通常采取的配置是1：1、1：3以及3：1；对于面积，常用的配置是32、64、128、256以及512。以上述两个配置为例，每一个面积都使用3个长宽比，因此，不难算出共有15个先验框”。这里聚类得到了5个不同的先验框尺寸，是指yolov2的每个网格用了5个先验框吗，怎么和2.1节的话对应呢，不是很明白🤔

赞

Kissrabbit (作者) 回复 Kemlz 01-27

对，yolov2用聚类的方法获得了5个不同尺寸的先验框。而其他方法中，比如faster rcnn ssd这些工作，他们的先验框就采用的手工设置

赞

荏苒 2021-12-15

在generate_txtytwth函数中，最后一个else是否应该和if iou_m对齐，表示其余小于阈值的，而代码中是和if index == best_index对齐，感觉有点不清楚

赞

Kissrabbit (作者) 回复 荏苒 2021-12-15

是这样的，你说的问题其实是对应制作正样本时候的情况2，即使有多个anchor box与gt box 的IoU超过了阈值了。这个时候，我们只保留best_index索引的那个anchor box，也就是IoU最大的，其他的anchor box尽管与gt box的IoU也超过了阈值，但因为不是最大的，所以就忽略了，也就是你所说的else，这里我们要和if index == best_index对齐，将其objectness标签设置为-1，后续计算loss的时候，这些-1是不会参与到loss计算的，也就和原文所讲的内容对应上了。。
我们不需要一个else去和if iou_m对齐，因为iou_m若是0，就表明这是个负样本，无需做任何操作，就忽略了，也就无需一个elsel。

赞

荏苒 回复 Kissrabbit (作者) 2021-12-15

明白啦，谢谢

赞