目标检测的性能评价指标

作者： fivetrees
https://zhuanlan.zhihu.com/p/70306015

写在前面：纸上得来终觉浅，绝知此事看源码！看源码是理解最准确的方法，没有之一。
一般来说目标检测问题的评价指标有：map（平均准确度均值），速度指标（FPS即每秒处理的图片数量或者处理每张图片所需的时间，当然必须在同一硬件条件下进行比较），下面对这2个评价指标仔细介绍。如有不对的地方，还望告知。

1.map（平均准确度均值）

1.1 mAP定义及相关概念

- mAP: mean Average Precision, 即各类别AP的平均值
- AP: PR曲线下面积，后文会详细讲解
- PR曲线: Precision-Recall曲线
- Precision: TP / (TP + FP)
- Recall: TP / (TP + FN)
- TP: IoU>0.5的检测框数量（同一Ground Truth只计算一次）
- FP: IoU<=0.5的检测框，或者是检测到同一个GT的多余检测框的数量
- FN: 没有检测到的GT的数量

注意：（1）一般来说mAP针对整个数据集而言的；AP针对数据集中某一个类别而言的；而percision和recall针对单张图片某一类别的。

1.2 mAP的具体计算

- 不同数据集map计算方法

由于map是数据集中所有类别AP值得平均，所以我们要计算map，首先得知道某一类别的AP值怎么求。不同数据集的某类别的AP计算方法大同小异，主要分为三种：

（1）在VOC2010以前，只需要选取当Recall >= 0, 0.1, 0.2, ..., 1共11个点时的Precision最大值，然后AP就是这11个Precision的平均值，map就是所有类别AP值的平均。

（2）在VOC2010及以后，需要针对每一个不同的Recall值（包括0和1），选取其大于等于这些Recall值时的Precision最大值，然后计算PR曲线下面积作为AP值，map就是所有类别AP值的平均。

（3）COCO数据集，设定多个IOU阈值（0.5-0.95,0.05为步长），在每一个IOU阈值下都有某一类别的AP值，然后求不同IOU阈值下的AP平均，就是所求的最终的某类别的AP值。

- 计算某一类别的AP

由上面概念我们知道，我们计算某一类别AP需要需要绘画出这一类别的PR曲线，所以我们要计算数据集中每张图片中这一类别的percision和reacll。由公式：

- Precision: TP / (TP + FP)
- Recall: TP / (TP + FN)

只需要统计出TP，FP，FN个数就行了。下面放下代码再作解释（代码原址）：

```
# first load gt
if not os.path.isdir(cachedir):
 os.mkdir(cachedir)
cachefile = os.path.join(cachedir, '%s_annots.pkl' % imagesetfile)
# read list of images
with open(imagesetfile, 'r') as f:
 lines = f.readlines()
imagenames = [x.strip() for x in lines]

if not os.path.isfile(cachefile):
 # load annotations
 recs = {}
 for i, imagename in enumerate(imagenames):
  recs[imagename] = parse_rec(annopath.format(imagename))
  if i % 100 == 0:
   print('Reading annotation for {:d}/{:d}'.format(
    i + 1, len(imagenames)))
 # save
 print('Saving cached annotations to {:s}'.format(cachefile))
```

```
  with open(cachefile, 'wb') as f:
    pickle.dump(recs, f)
else:
  # load
  with open(cachefile, 'rb') as f:
    try:
      recs = pickle.load(f)
    except:
      recs = pickle.load(f, encoding='bytes')

# extract gt objects for this class
class_recs = {}
npos = 0
for imagename in imagenames:
  R = [obj for obj in recs[imagename] if obj['name'] == classname]
  bbox = np.array([x['bbox'] for x in R])
  difficult = np.array([x['difficult'] for x in R]).astype(np.bool)
  det = [False] * len(R)
  npos = npos + sum(~difficult)
  class_recs[imagename] = {'bbox': bbox,
                           'difficult': difficult,
                           'det': det}

# read dets
detfile = detpath.format(classname)
with open(detfile, 'r') as f:
  lines = f.readlines()

splitlines = [x.strip().split(' ') for x in lines]
image_ids = [x[0] for x in splitlines]
confidence = np.array([float(x[1]) for x in splitlines])
BB = np.array([[float(z) for z in x[2:]] for x in splitlines])

nd = len(image_ids)
tp = np.zeros(nd)
fp = np.zeros(nd)

if BB.shape[0] > 0:
  # sort by confidence
  sorted_ind = np.argsort(-confidence)
```

```
sorted_scores = np.sort(-confidence)
BB = BB[sorted_ind, :]
image_ids = [image_ids[x] for x in sorted_ind]

# go down dets and mark TPs and FPs
for d in range(nd):
  R = class_recs[image_ids[d]]
  bb = BB[d, :].astype(float)
  ovmax = -np.inf
  BBGT = R['bbox'].astype(float)

  if BBGT.size > 0:
    # compute overlaps
    # intersection
    ixmin = np.maximum(BBGT[:, 0], bb[0])
    iymin = np.maximum(BBGT[:, 1], bb[1])
    ixmax = np.minimum(BBGT[:, 2], bb[2])
    iymax = np.minimum(BBGT[:, 3], bb[3])
    iw = np.maximum(ixmax - ixmin + 1., 0.)
    ih = np.maximum(iymax - iymin + 1., 0.)
    inters = iw * ih

    # union
    uni = ((bb[2] - bb[0] + 1.) * (bb[3] - bb[1] + 1.) +
           (BBGT[:, 2] - BBGT[:, 0] + 1.) *
           (BBGT[:, 3] - BBGT[:, 1] + 1.) - inters)

    overlaps = inters / uni
    ovmax = np.max(overlaps)
    jmax = np.argmax(overlaps)

  if ovmax > ovthresh:
    if not R['difficult'][jmax]:
      if not R['det'][jmax]:
        tp[d] = 1.
        R['det'][jmax] = 1
      else:
        fp[d] = 1.
  else:
    fp[d] = 1.
```

```
# compute precision recall
fp = np.cumsum(fp)
tp = np.cumsum(tp)
rec = tp / float(npos)
# avoid divide by zero in case the first detection matches a difficult
# ground truth
prec = tp / np.maximum(tp + fp, np.finfo(np.float64).eps)
ap = voc_ap(rec, prec, use_07_metric)

return rec, prec, ap
```

- 如何判断TP,FP,FN（重要）

拿单张图片来说吧，首先遍历图片中ground truth对象，然后提取我们要计算的某类别的gt objects，之后读取我们通过检测器检测出的这种类别的检测框（其他类别的先不管），接着过滤掉置信度分数低于置信度阈值的框（也有的未设置信度阈值），将剩下的检测框按置信度分数从高到低排序，最先判断置信度分数最高的检测框与gt bbox的iou是否大于iou阈值，若iou大于设定的iou阈值即判断为TP，将此gt_bbox标记为已检测（后续的同一个GT的多余检测框都视为FP,这就是为什么先要按照置信度分数从高到低排序，置信度分数最高的检测框最先去与iou阈值比较，若大于iou阈值，视为TP，后续的同一个gt对象的检测框都视为FP），iou小于阈值的，直接规划到FP中去。这里置信度分数不同的论文可能对其定义不一样，一般指分类置信度的居多，也就是预测框中物体属于某一个类别的概率。后面我仔细总结一下。

插一段这一过程的代码:

```
if ovmax > ovthresh:            # 若iou大于阈值
 if not R['difficult'][jmax]:   # 且要检测的gt对象非difficult类型
  if not R['det'][jmax]:        # 且gt对象暂未被检测
   tp[d] = 1.                   # 此检测框记为TP
   R['det'][jmax] = 1           # 并将此gt对象标记为已检测
  else:                         # 若gt对象已被检测，那么此检测框为FP
   fp[d] = 1.
else:                           # iou<=阈值，此检测框为FP
 fp[d] = 1.
```

关于图片中FN的统计就比较简单了，图片中某类别一共有多少个gt我们是知道的，减去TP的个数，剩下的就是FN的个数了，至此我们已经知道如何计算某一

类别的percision和recall如何计算。AP值计算有3种方式：
（1）在VOC2010以前，只需要选取当Recall >= 0, 0.1, 0.2, ..., 1共11个点时的Precision最大值，然后AP就是这11个Precision的平均值。
（2）在VOC2010及以后，需要针对每一个不同的Recall值（包括0和1），选取其大于等于这些Recall值时的Precision最大值，然后计算PR曲线下面积作为AP值。
（3）COCO数据集，设定多个IOU阈值（0.5-0.95,0.05为步长），在每一个IOU阈值下都有某一类别的AP值，然后求不同IOU阈值下的AP平均，就是所求的最终的某类别的AP值。
最后，mAP值计算就更为简单了，所有类的AP值平均值就是mAP。

题外话：
之前没看源码，看博客上解释只通过iou阈值来判断是否是TP（此处假设gt对象还未被检测），不是很理解，心想iou只能用来判断预测预测的位置准不准确，但是目标检测除了预测位置准不准确外，还需要判断预测的类别和gt类别是否对应呀。现在我拿出一张图片中所有要计算AP的类别的检测框，计算与Gt对象检测框的iou，若iou大于阈值，则视为TP，若假如预测框框中的实际对象不是要计算AP值得那个类别，但预测的标签是要计算AP的那个类别，且预测框与gt对象的框iou也大于阈值，那么这种情况判读为TP，明显是错误的。其实这种情况基本不可能发生，首先源码中：if not R['difficult'][jmax]，将难检测的目标去掉了（个人觉得这里主要去除的是被遮挡面积较大的检测对象，也就是2个不同类别的物体重叠干扰的情况），其次假如框中实际对象与要GT的类别不匹配，你预测的此类别的置信度分数不会很高，然后在统计TP的时候，置信度分数高的检测结果先去判断是否是TP，即使假设你是剩下预测中置信度分数最高的，你与其他目标的GT的IOU还要超过阈值，这三点每一样都基本很难满足，更别说三点都满足了，所以说这样做是一点问题没有的。

- 目标检测中的遮挡问题解决方案

其实现在目标检测问题中的物体对象遮挡问题也是一大难点。一般通过设计优化损失函数来解决这样的问题。
1.对于待检测目标之间的相互遮挡

通过优化损失函数，一方面对proposal向其它目标偏移或其它目标对应的proposals靠近的情况进行惩罚；另一方面设计损失函数使（1）所有的预测框逼近对应的Target框；（2）属于同一Target框的多个预测框尽量集中来解决待检测目标之前的相互遮挡。

2.对于待检测的物体被干扰物体（非检测的类别目标）遮挡，
因为算法只学习待检测的物体的特征，所以第二种遮挡只能通过增加样本来优化检测效果。

- 计算mAP在NMS之后

这一点一定要明确，mAP值计算在NMS之后进行的，mAP是统计我们的检测模型的最终评价指标，是所有操作完成之后，以最终的检测结果作为标准，来计算mAP值的，另外提一点一般只有测试的时候才会作NMS，训练的时候不进行NMS操作，因为训练的时候需要大量的正负样本去学习。ok，下面进入我们的下一个主题，关于目标检测的速度评价指标如何计算。
2.速度指标
2.1 概述
目标检测技术的很多实际应用在准确度和速度上都有很高的要求，如果不计速度性能指标，只注重准确度表现的突破，但其代价是更高的计算复杂度和更多内存需求，对于全面行业部署而言，可扩展性仍是一个悬而未决的问题。一般来说目标检测中的速度评价指标有：（1）FPS，检测器每秒能处理图片的张数（2）检测器处理每张图片所需要的时间
但速度评价指标必须在同一硬件上进行，同一硬件，它的最大FLOPS（每秒运算浮点数代表着硬件性能，此处区分FLOPs）是相同的，不同网络，处理每张图片所需的FLOPs(浮点操作数)是不同的，所以同一硬件处理相同图片所需的FLOPs越小，相同时间内，就能处理更多的图片，速度也就越快，处理每张图片所需的FLOPs与许多因素有关，比如你的网络层数，参数量，选用的激活函数等等，这里仅谈一下网络的参数量对其的影响，一般来说参数量越低的网络，FLOPs会越小，保存模型所需的内存小，对硬件内存要求比较低，因此比较对嵌入式端较友好。

2.2 FLOPs计算

- FLOPs和FLOPS区分

先解释一下FLOPs：floating point operations 指的是浮点运算次数，理解为计算量，可以用来衡量算法/模型的复杂度。此处区分一下FLOPS（全部大写），FLOPS指的是每秒运算的浮点数，理解为计算速度，衡量一个硬件的标准。我们要的是衡量模型的复杂度的指标，所以选择FLOPs。

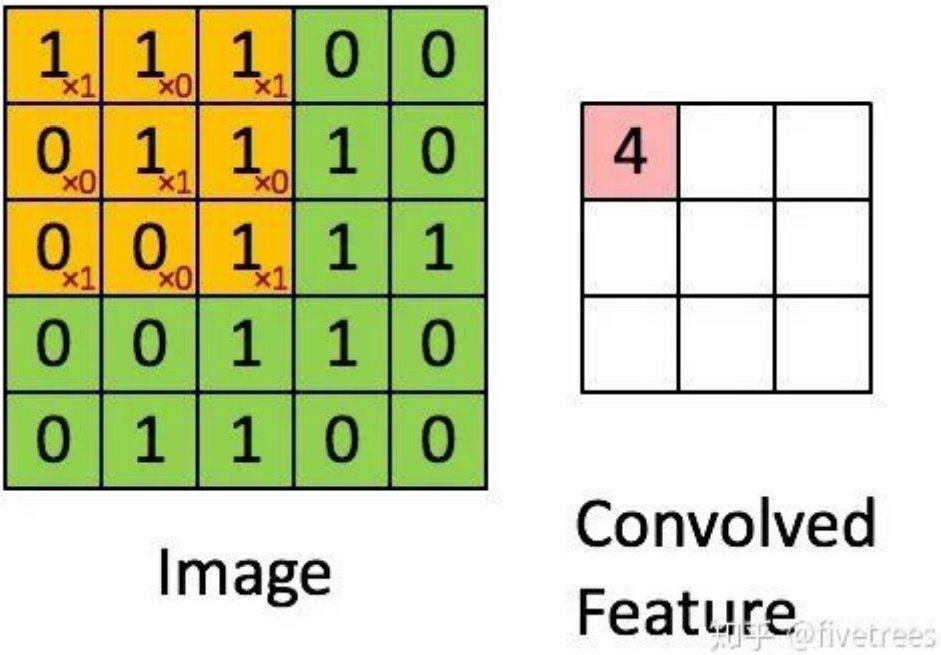
- FLOPs计算（以下计算FLOPs不考虑激活函数的运算）

(1)卷积层

FLOPs=(2*Ci*k*K-1）*H*W*Co（不考虑bias）

FLOPs=(2*Ci*k*K）*H*W*Co（考虑bias）

Ci为输入特征图通道数，K为过滤器尺寸，H,W,Co为输出特征图的高，宽和通道数。



解释一下公式：最后得到的Co张输出特征图，每张特征图上有H*W个像素点，而这其中的每个像素点的值都是由过滤器与输入特征图卷积得到的，过滤器中Ci*k*K个点，每个点都要和输入特征图对应点作一次相乘操作（浮点操作数为ci*k*k），然后将这些过滤器和输入特征图对应点相乘所得的数相加起来（浮点操作数为ci*k*k-1，n个数相加，所需要的浮点操作数为n-1），得到一个值，对应于一张输出特征图中的一个像素，输出特征图有Co张，故有Co个过滤器参与卷积运算，所以卷积层的FLOPs=(2*Ci*k*K-1）*H*W*Co

(Ci为输入特征图通道数，K为过滤器尺寸，H,W,Co为输出特征图的高，宽和通道数)

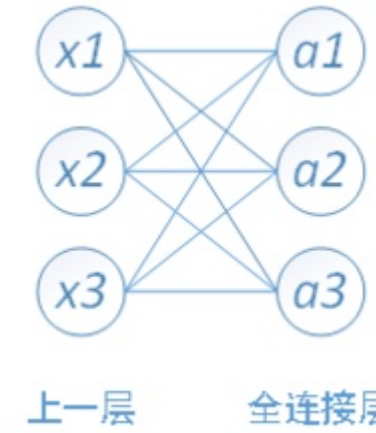
(2)池化层

池化分为最大值池化和均值池化，看别人的博客说网络中一般池化层较少，且池化操作所占用的FLOPs很少，对速度性能影响较小。我在想，最大池化一般是找出输入特征图中过滤器范围内的最大值，以最大值代替，好像不涉及到浮点运算操作。均值池化，要求平均值，先相加再除以总数（输出特征图上一个像素点，需要浮点操作数为：(k*K-1+1)*Ci，求平均值，先k*k个数相加，操作数为,k*k-1，然后除以k*k，浮点操作数为1），输出特征图通道数为Co，所以这里浮点操作数应该为k*k*Ci*H*W*Co(不知道有没有问题，如有大佬知道，还望告知)

（3）全连接层

先解释一下全连接层

> 卷积神经网络的全连接层
>
> 在 CNN 结构中，经多个卷积层和池化层后，连接着1个或1个以上的全连接层．与 MLP 类似，全连接层中的每个神经元与其前一层的所有神经元进行全连接．全连接层可以整合卷积层或者池化层中具有类别区分性的局部信息．为了提升 CNN 网络性能，全连接层每个神经元的激励函数一般采用 ReLU 函数。最后一层全连接层的输出值被传递给一个输出，可以采用 softmax 逻辑回归（softmax regression）进行 分 类，该层也可 称为 softmax 层（softmax layer）．对于一个具体的分类任务，选择一个合适的损失函数是十分重要的，CNN 有几种常用的损失函数，各自都有不同的特点．通常，CNN 的全连接层与 MLP 结构一样，CNN 的训练算法也多采用BP算法
>
> 全连接层的每一个结点都与上一层的所有结点相连，用来把前边提取到的特征综合起来。由于其全相连的特性，一般全连接层的参数也是最多的。例如在VGG16中，第一个全连接层FC1有4096个节点，上一层POOL2是7*7*512 = 25088个节点，则该传输需要4096*25088个权值，需要耗很大的内存



其中，x1、x2、x3为全连接层的输入，a1、a2、a3为输出，

$$a1 = W_{11} * x_1 + W_{12} * x_2 + W_{13} * x_3 + b_1$$
$$a2 = W_{21} * x_1 + W_{22} * x_2 + W_{23} * x_3 + b_2$$
$$a3 = W_{31} * x_1 + W_{32} * x_2 + W_{33} * x_3 + b_3$$

其实现在全连接层已经基本不用了，CNN基本用FCN来代表，可用卷积层来实现全连接层的功能。设I为输入神经元个数，O为输出神经元个数，输出的每个神经元都是由输入的每个神经元乘以权重（浮点操作数为I），然后把所得的积的和相加（浮点操作数为I-1），加上一个偏差（浮点操作数为1）得到了，故FLOPs为：

FLOPs=(I+I-1) * O = (2I-1) * O(不考虑bias)

FLOPs=((I+I-1+1) * O = (2I) * O(考虑bias)

FLOPs和参数量计算小工具

最近在github上找到了一个别人开源的在Pytorch框架中使用的FLOPs和参数量计算的小工具OpCouter，非常好用，再提一点，之前我说的，最大池化不存在浮点操作数，其实是不对的，虽然最大池化没有参数，但存在计算，类似的还有Dropout等。这个工具安装也十分方便，可以直接使用pip简单的完成安装。以下放出作者开源的链接：

https://github.com/Lyken17/pytorch-OpCounter

写在后面：其实模型的轻量化也是现在的一大热门研究方向，后面我将近几年的轻量型网络作一个总结。最后，最重要的事情是：如有错误，一定告知！如有错误，一定告知！如有错误，一定告知！ok，下回见！

扫码添加CVer助手，可申请加入**CVer-目标检测检测学术交流群，同时还可以加入目标检测、图像分割、目标跟踪、人脸检测&识别、OCR、姿态估计、超分辨率、SLAM、医疗影像、Re-ID、GAN、NAS、深度估计、自动驾驶、强化学习、车道线检测和模型剪枝&压缩等群。一定要备注：研究方向+地点+学校/公司+昵称**（如目标检测+上海+上交+卡卡）

▲长按加群

▲长按关注我们

麻烦给我一个在看！