目标识别：SSD 论文及pytorch代码学习笔记

zxd52csx 2018-09-20 23:18:42

论文：https://arxiv.org/pdf/1512.02325.pdf

代码：https://github.com/amdegroot/ssd.pytorch

目录

一 介绍

SSD--single shot multibox detection，是目标识别领域中对 不同类的物体（汽车，人。。。）的识别，识别的表示方式是对被识别的物体画bounding box（包围框）。除此之外还有其他种类的目标识别，比如下图：

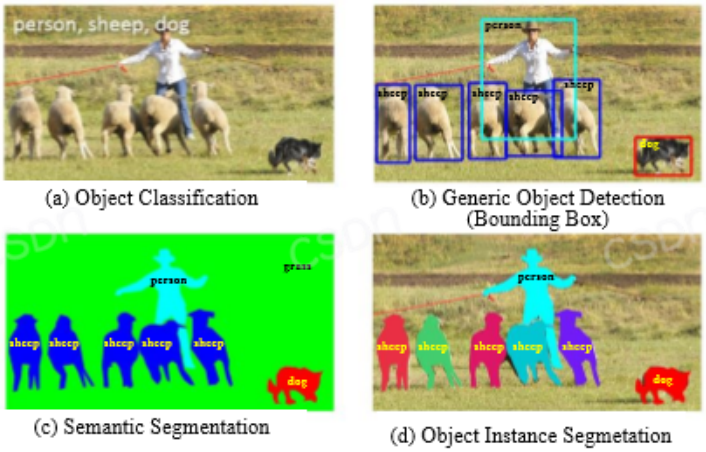


Fig. 3 Recognition problems related to generic object detection. (a) Image level object classification, (b) bounding box level generic object detection, (c) pixel-wise semantic segmentation, (d) instance level semantic segmentation.

SSD 网络的特点是 对不同尺度下的 feature map 中每一个点都设置一些 default box, 这些default box有不同的大小和横纵比例，对这些default box进行分类和边框回归的操作。在预测阶段，SSD会对每个default box 生成一个分类标签（属于每一类的概率构成的向量）以及位置坐标的调整（两个点有四个坐标值）。

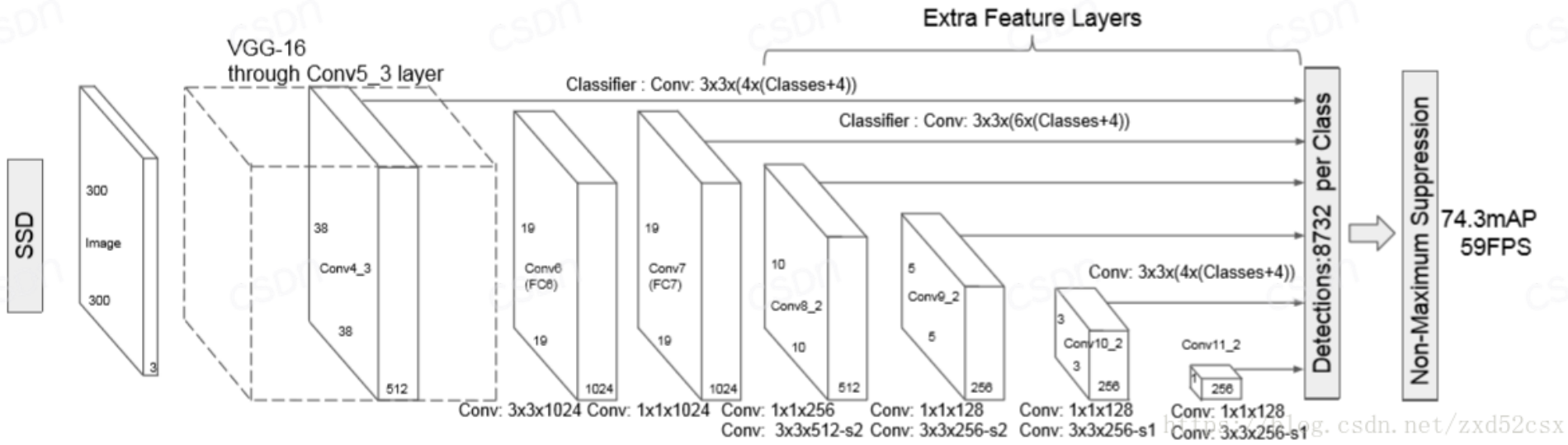
SSD 的特点和贡献：

是当时的最高性能的one stage方法，精确度与two stage的faster cnn基本持平

SSD的核心是对 固定设置的default box（不同尺度feature map中每一个空间位置都设置一组default box，这里只考虑空间位置，不考虑feature 的通道个数）计算属于各类物体的概率以及坐标调整的数值。这个计算方法是 对每层的feature map 做卷积操作，卷积核设定为3*3，卷积核的个数是与default box个数相关。

SSD同时实现了端到端的操作（可以理解为输入是图像，输出是结果而不是什么中间的特征）和比较高的准确率

二 SSD网络结构：



SSD网络由两部分组成，前部分是一些经典的特征提取网络，在这里作为base network，使用的是VGG16。后半部分是作者增加的网络，提取成特征的基础之上处理得到不同尺度的feature map，生成许多组default box进行预测分类和位置调整信息。

两部分网络要想直接连接在一起需要经过一个处理，VGG16的两个全连接层被改掉（FC6,7）。之前用全连接层是因为在这两层要完成分类，改动过后可以与后面的卷积层连接，这个地方需要注意1*1的卷积层，1*1的感受野是1，它对空间位置上的信息没有进行任何改变，它完成的是维度信息的整合。在作者加入的后半部分也用到了1*1的卷积层，并且发挥了1*1卷积层降维的作用。

把全连接层换成1*1的卷积层的另一个好处：输入图像的大小可以变化（虽然SSD300只使用300*300的图片作为输入），因为全连接层的参数由前后两层的元素个数决定，如果输入图像的大小不统一，那么就找不到一个固定大小的全连接层

模型的输入与输出

输入：300*300*3 的图像

default box：六个尺度下的feature map 中每一个点设置多个default box，不同尺度下的feature map决定了default box 的scale，不同的横纵比决定了default box 的形状。

实现对不同尺度feature map进行分类预测和位置调整的方法是进行卷积操作，一个卷积核相当于完成对一种default box的一个信息（比如类别信息，位置信息）进行处理，一个卷积核可以完成对所有空间位置的操作。因此模型图中的filters的计算公式可以理解为：default box形状数量*（分类类别+四个位置信息）。

三 模型训练

matching规则

为了训练网络，需要把网络提出的default box（也可以叫做priors）与标记好的ground truth进行一个比对。如果满足标准就真为是match，那么这个priors就是 一个positive 样本，否则就是negative样本。这个标准是IoU，可以理解为priors与ground truth两个box 的交并比，阈值为0.5.

训练目标（损失函数）

网络的损失函数：

$$L(x,c,l,g)=\frac{1}{N}(L_{conf}(x,c)+\alpha L_{loc}(x,l,g)) \quad (1)$$

其中：

l is predicted box

N is the number of matched default boxes

g is ground truth

位置损失函数：

truth box (g) parameters. Similar to Faster R-CNN [2], we regress to offsets for the center (cx, cy) of the default bounding box (d) and for its width (w) and height (h).

$$L_{loc}(x,l,g) = \sum_{i\in Pos}^{N} \sum_{m\in\{cx,cy,w,h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$
$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h \tag{2}$$
$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

注意这部分的位置损失只计算经过match的正样本，即i属于positive。

g(由ground truth计算出来的):d是priors的w,h，g是ground truth的w,h

l是由feature map经过卷积计算出来的

其中smooth L1的计算方法如下：（参考https://blog.csdn.net/jningwei/article/details/78853669）

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

类别损失函数：

The confidence loss is the softmax loss over multiple classes confidences (c).

$$L_{conf}(x,c) = -\sum_{i\in Pos}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i\in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)} \tag{3}$$

这部分损失函数既考虑正样本也考虑负样本。

; $x_{ij}^{p}=\{1,0\}$ be anindicator for matching the i-th default box to the j-th ground truth box of category p.如果x为1说明两者match，否则就是0，这个x在代码中好像没怎么体现，就是第i个positive default box与其对应的ground truth 的labels进行cross entropy计算。

正样本就是default box与某个ground truth box 的iou超过阈值，这里代码中应该是只选取iou最大的且超过阈值的那个ground truth box的label作为这个default box 的标签，之后与卷积计算出来的这个default box的标签进行cross entropy。

负样本就是用0作为这个default box 的标签与feature map计算出的标签进行cross entropy。

c^是经过了softmax计算（完成概率归一化的操作），这里也有个不理解的地方：c是什么的东西？cross entropy的结果？

强例挖掘（Hard negative example mining）

上面的类别损失函数需要负样本，但是default box 有很多很多（8732），这样正负样本的比例就非常不协调，负样本很多，这样类别损失函数部分就会占据很大的比例，以此训练的效果就不会太好（降低的主要是负样本的损失函数）。因此就需要进行强例挖掘。即按照confidence loss进行排序，按照pos：neg=1：3的比例，找到最高的那些负样本作为最终的负样本进行优化训练。

这个confidence loss的具体计算方法我还不是很清楚，是根据feature map计算出的confidence来进行比较的。

四 pytorch 代码学习

目前只学习了训练相关的代码 也就是train.py以及相关代码

dataset的读取

由于精力没有放在这个上面，所以先放个标题以后来补

网络结构的构建

代码中定义网络结构的部分简介，我个人觉得也需要很强的代码能力，一步步分析：

```
# train.py
# create network object
ssd_net = build_ssd('train', cfg['min_dim'], cfg['num_classes'])
net = ssd_net
```

build_ssd是一个放在ssd.py的函数：

```
# ssd.py
def build_ssd(phase, size=300, num_classes=21):
    base_, extras_, head_ = multibox(vgg(base[str(size)], 3),
                                     add_extras(extras[str(size)], 1024),
                                     mbox[str(size)], num_classes)
    return SSD(phase, size, base_, extras_, head_, num_classes)
```

return回来的是一个类的对象，也就是class SSD(nn.Module)，ssd_net也就是SSD类的一个对象，ssd_net拥有所有class SSD继承于nn.Module以及作者增加方法的所有属性。在SSD这个类中就定义了网络的base部分（修改全连接层后的VGG16）和extras部分（论文作者加入的多尺度feature map）和head部分（对选定的6个尺度下的feature map进行卷积操作得到的每个default box 的每一个分类类别的confidence以及位置坐标的信息）。

SSD网络的建立需要multibox函数生成所需的部分，multibox需要vgg，add_extra,两个函数的输出结果。

vgg：

```
# ssd.py
def vgg(cfg, i, batch_norm=False):
    layers = []  # 用于存放vgg网络的list
    in_channels = i  # 最前面那层的维度--300*300*3，因此i=3
    for v in cfg:  # 代码厉害的地方，循环建立多层，数据信息存放在一个字典中
        if v == 'M':  # maxpooling 时边缘不补
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        elif v == 'C':  # # maxpooling 时边缘补NAN
            layers += [nn.MaxPool2d(kernel_size=2, stride=2, ceil_mode=True)]
        else:  # 卷积前后维度可以通过字典中数据设置好
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    pool5 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)
    conv6 = nn.Conv2d(512, 1024, kernel_size=3, padding=6, dilation=6)
```

```
    #  dilation=卷积核元素之间的间距,扩大卷积感受野的范围，没有增加卷积size
    conv7 = nn.Conv2d(1024, 1024, kernel_size=1)
    layers += [pool5, conv6,
               nn.ReLU(inplace=True), conv7, nn.ReLU(inplace=True)]
    return layers
```

add_extra：

```
# ssd.py
def add_extras(cfg, i, batch_norm=False):
    # Extra layers added to VGG for feature scaling
    layers = []
    in_channels = i
    flag = False
    for k, v in enumerate(cfg):  # S代表stride，为2时候就相当于缩小feature map
        if in_channels != 'S':
            if v == 'S':
                layers += [nn.Conv2d(in_channels, cfg[k + 1],
                           kernel_size=(1, 3)[flag], stride=2, padding=1)]
            else:
                layers += [nn.Conv2d(in_channels, v, kernel_size=(1, 3)[flag])]
            flag = not flag
        in_channels = v
    return layers
```

multibox：

```
# ssd.py
# loc_layers的输出维度是default box的种类(4or6)*4
#  conf_layers的输出维度是default box的种类(4or6)*num_class
def multibox(vgg, extra_layers, cfg, num_classes):
    loc_layers = []
    conf_layers = []
    vgg_source = [21, -2]
```

```
    for k, v in enumerate(vgg_source):  # 第21层和倒数第二层
        loc_layers += [nn.Conv2d(vgg[v].out_channels,
                                 cfg[k] * 4, kernel_size=3, padding=1)]
        conf_layers += [nn.Conv2d(vgg[v].out_channels,
                        cfg[k] * num_classes, kernel_size=3, padding=1)]
    for k, v in enumerate(extra_layers[1::2], 2):  # 找到对应的层
        loc_layers += [nn.Conv2d(v.out_channels, cfg[k]
                                 * 4, kernel_size=3, padding=1)]
        conf_layers += [nn.Conv2d(v.out_channels, cfg[k]
                                  * num_classes, kernel_size=3, padding=1)]
    return vgg, extra_layers, (loc_layers, conf_layers)
```

SSD类中继承了nn.Module,同时加入了正向传播的函数：

```
# ssd.py
def forward(self, x):
    """Applies network layers and ops on input image(s) x.

    Args:
        x: input image or batch of images. Shape: [batch,3,300,300].

    Return:(trian)
            list of concat outputs from:
                1: confidence layers, Shape: [batch*num_priors,num_classes]
                default box对应每个分类的confidence
                2: localization layers, Shape: [batch,num_priors*4]
                每一个default box的4个坐标信息
                3: priorbox layers, Shape: [2,num_priors*4]
                计算每个default box在同一尺度下的坐标，用作后面IoU、offset的计算

    """
    sources = list()
    loc = list()
    conf = list()

    # apply vgg up to conv4_3 relu
```

```
    for k in range(23):
        x = self.vgg[k](x)

    s = self.L2Norm(x)
    sources.append(s)

    # apply vgg up to fc7
    for k in range(23, len(self.vgg)):
        x = self.vgg[k](x)
    sources.append(x)

    # apply extra layers and cache source layer outputs
    for k, v in enumerate(self.extras):
        x = F.relu(v(x), inplace=True)
        if k % 2 == 1:
            sources.append(x)

    # apply multibox head to source layers
    for (x, l, c) in zip(sources, self.loc, self.conf):
        loc.append(l(x).permute(0, 2, 3, 1).contiguous())
        conf.append(c(x).permute(0, 2, 3, 1).contiguous())

    loc = torch.cat([o.view(o.size(0), -1) for o in loc], 1)
    conf = torch.cat([o.view(o.size(0), -1) for o in conf], 1)
    if self.phase == "test":
        output = self.detect(
            loc.view(loc.size(0), -1, 4),                   # loc preds
            self.softmax(conf.view(conf.size(0), -1,
                         self.num_classes)),                # conf preds
            self.priors.type(type(x.data))                  # default boxes
        )
    else:
        output = (
            loc.view(loc.size(0), -1, 4),
            # debug show[batch_size,num_priors,4]
            conf.view(conf.size(0), -1, self.num_classes),
            # debug show[batch_size,num_priors,num_class]
            self.priors
```

```
            # debug show[num_priors,4]
        )
    return output
```

在正向传播的时候，使用了SSD类中的forward函数：

```
# train.py
out = net(images)
# 好奇为什么这样写就是调用了forward函数
```

损失函数

```
# train.py
# targets 和image都是读取的训练数据
# images=[batch_size,3,300,300]
# targets=[batch_size,num_object,5]
# num_object代表一张图里面有几个ground truth，5代表四个位置信息和一个label
loss_l, loss_c = criterion(out, targets)
# criterion = MultiBoxLoss(cfg['num_classes'], 0.5, True, 0, True, 3, 0.5,
#                          False, args.cuda)
```

MultiBoxLoss也是一个类，criterion是一个对象。在这个类中也有forward函数，是用来计算位置损失函数和分类信心损失函数的：

```
# multibox_loss.py
def forward(self, predictions, targets):
"""Multibox Loss
Args:
    predictions (tuple): A tuple containing loc preds, conf preds,
    and prior boxes from SSD net.
        conf shape: torch.size(batch_size,num_priors,num_classes)
        loc shape: torch.size(batch_size,num_priors,4)
        priors shape: torch.size(num_priors,4)
```

```
    targets (tensor): Ground truth boxes and labels for a batch,
        shape: [batch_size,num_objs,5] (last idx is the label).
    loc_t,conf_t是由target产生的标签数据
    loc_data,conf_data是feature map计算出来的预测数据
"""
loc_data, conf_data, priors = predictions
num = loc_data.size(0)  #  batch size
priors = priors[:loc_data.size(1), :]  # ????这是做什么
num_priors = (priors.size(0))  # 8732
num_classes = self.num_classes

# match priors (default boxes) and ground truth boxes
loc_t = torch.Tensor(num, num_priors, 4)  # [batch_size,8732,4]
conf_t = torch.LongTensor(num, num_priors)  # [batch_size,8732]
# 这个地方没有把label变为one_hot形式
for idx in range(num):  # every batch process
    truths = targets[idx][:, :-1].data  # position
    labels = targets[idx][:, -1].data  # labels
    defaults = priors.data  # [8732,4] default box在同一尺度下的坐标是不变的，与batch无关
    match(self.threshold, truths, defaults, self.variance, labels,
          loc_t, conf_t, idx)
    # match这个函数给每个ground truth匹配了最好的priors，给每个priors匹配最好的ground truth
    # 经过encode后的offset([g_cxcy, g_wh])->loc_t,top class label for each prior->conf_t
if self.use_gpu:
    loc_t = loc_t.cuda()
    conf_t = conf_t.cuda()
# wrap targets
loc_t = Variable(loc_t, requires_grad=False)
conf_t = Variable(conf_t, requires_grad=False)

pos = conf_t > 0  # if>0 return 1 [4,8732],0->background(IoU<threashold)
num_pos = pos.sum(dim=1, keepdim=True)  # the number of positive bbox

# Localization Loss (Smooth L1)
# loc_loss是只考虑正样本的
# Shape: [batch,num_priors,4]
pos_idx = pos.unsqueeze(pos.dim()).expand_as(loc_data)  # [4,8732,4]
```

```
loc_p = loc_data[pos_idx].view(-1, 4)
# 保留与计算出来的positive的default box所对应的卷积生成的encode offset（相当于预测的）
loc_t = loc_t[pos_idx].view(-1, 4)
# 实际计算出来的，encode offset
loss_l = F.smooth_l1_loss(loc_p, loc_t, size_average=False)

# Compute max conf across batch for hard negative mining
batch_conf = conf_data.view(-1, self.num_classes)
# batch_conf->(batch_size*num_priors,num_classes)
loss_c = log_sum_exp(batch_conf) - batch_conf.gather(1, conf_t.view(-1, 1))
# 这个地方的

# Hard Negative Mining
loss_c[pos] = 0  # filter out pos boxes for now
loss_c = loss_c.view(num, -1)
_, loss_idx = loss_c.sort(1, descending=True)
_, idx_rank = loss_idx.sort(1)
num_pos = pos.long().sum(1, keepdim=True)
num_neg = torch.clamp(self.negpos_ratio*num_pos, max=pos.size(1)-1)
neg = idx_rank < num_neg.expand_as(idx_rank)

# Confidence Loss Including Positive and Negative Examples
pos_idx = pos.unsqueeze(2).expand_as(conf_data)
neg_idx = neg.unsqueeze(2).expand_as(conf_data)
conf_p = conf_data[(pos_idx+neg_idx).gt(0)].view(-1, self.num_classes)
targets_weighted = conf_t[(pos+neg).gt(0)]
loss_c = F.cross_entropy(conf_p, targets_weighted, size_average=False)

# Sum of losses: L(x,c,l,g) = (Lconf(x, c) + αLloc(x,l,g)) / N

N = num_pos.data.sum()
loss_l /= N
loss_c /= N
return loss_l, loss_c
```


在计算损失函数的过程中使用了一些函数进行运算：

```python
# box_utils.py
def match(threshold, truths, priors, variances, labels, loc_t, conf_t, idx):
    """Match each prior box with the ground truth box of the highest jaccard
    overlap, encode the bounding boxes, then return the matched indices
    corresponding to both confidence and location preds.
    Args:
        threshold: (float) The overlap threshold used when mathing boxes.
        truths: (tensor) Ground truth boxes, Shape: [num_obj, num_priors].
        priors: (tensor) Prior boxes from priorbox layers, Shape: [n_priors,4].
        variances: (tensor) Variances corresponding to each prior coord,
            Shape: [num_priors, 4].
        labels: (tensor) All the class labels for the image, Shape: [num_obj].
        loc_t: (tensor) Tensor to be filled w/ endcoded location targets.
        conf_t: (tensor) Tensor to be filled w/ matched indices for conf preds.
        idx: (int) current batch index
    Return:
        The matched indices corresponding to 1)location and 2)confidence preds.
    """
    # jaccard index
    overlaps = jaccard(
        truths,
        point_form(priors)
    )
    # (Bipartite Matching)
    # [1,num_objects] best prior for each ground truth
    best_prior_overlap, best_prior_idx = overlaps.max(1, keepdim=True)
    # [1,num_priors] best ground truth for each prior
    best_truth_overlap, best_truth_idx = overlaps.max(0, keepdim=True)
    best_truth_idx.squeeze_(0)
    best_truth_overlap.squeeze_(0)
    best_prior_idx.squeeze_(1)
    best_prior_overlap.squeeze_(1)
    best_truth_overlap.index_fill_(0, best_prior_idx, 2)  # ensure best prior
    # TODO refactor: index  best_prior_idx with long tensor
    # ensure every gt matches with its prior of max overlap
    for j in range(best_prior_idx.size(0)):
```

```
        best_truth_idx[best_prior_idx[j]] = j  # 给每个prior标记上对应的最好的ground truth
    matches = truths[best_truth_idx]          # Shape: [num_priors,4]
    conf = labels[best_truth_idx] + 1         # Shape: [num_priors] +1是因为0作为背景
    conf[best_truth_overlap < threshold] = 0  # label 0 as background
    loc = encode(matches, priors, variances)  # [g_cxcy, g_wh]
    loc_t[idx] = loc    # [num_priors,4] encoded offsets to learn
    conf_t[idx] = conf  # [num_priors] top class label for each prior
```

```
# box_utils.py
def encode(matched, priors, variances):
    """Encode the variances from the priorbox layers into the ground truth boxes
    we have matched (based on jaccard overlap) with the prior boxes.
    Args:
        matched: (tensor) Coords of ground truth for each prior in point-form
            Shape: [num_priors, 4].
        priors: (tensor) Prior boxes in center-offset form
            Shape: [num_priors,4].
        variances: (list[float]) Variances of priorboxes
    Return:
        encoded boxes (tensor), Shape: [num_priors, 4]
    """

    # dist b/t match center and prior's center
    g_cxcy = (matched[:, :2] + matched[:, 2:])/2 - priors[:, :2]  # cx,cy offsets
    # encode variance
    g_cxcy /= (variances[0] * priors[:, 2:])  # 计算cx,cy,的偏差占框的宽和高的比例
    # match wh / prior wh
    g_wh = (matched[:, 2:] - matched[:, :2]) / priors[:, 2:]
    g_wh = torch.log(g_wh) / variances[1]
    # return target for smooth_l1_loss
    return torch.cat([g_cxcy, g_wh], 1)  # [num_priors,4]
```