在物体检测问题中，主要分为两类检测器模型：one stage detector（SSD,YOLO系列，retinanet）和two stage detector（faster RCNN系列及其改进模型），然而无论是一个阶段的检测器还是两个阶段的检测器，都使用到了anchor机制，即在特征图上密集地画anchor boxes，根据先验知识设定的IOU阈值将这些anchor划分为正样本和负样本，再对于正样本anchor boxes进行位置编码，从而得到训练检测器所需要的ground truth label，其中存在很大的问题在于，

（1）前景anchor和背景anchor类别不平衡：在数量巨大的anchor boxes中，仅仅有少部分的anchor boxes是正样本（前景anchor），大量的anchor boxes是负样本，故而进行classification时会存在严重的类别不平衡问题（前景和背景anchor的类别不平衡）。对此，focal loss采用的方法是在各个类别的loss值之前加上权重，而

（2）负样本anchor太多，如何训练？

focal loss是通过在loss前面加上系数实现的，它能够自动地把更多注意力关注到分类错误的前景anchor和背景anchor上去，OHEM是通过对于所有负样本的classification loss值由大到小排序，取出前面loss较大的损失值（即分类错误程度较大的负样本）。

anchor机制存在的问题	focal loss	OHEM
前景背景类别不平衡	留下所有的正负样本，在每个类别的loss前加权重系数alpha	设定阈值确定anchor正负正负样本1：3比例采样出
如何让optimizer更多关注分类错误的样本	在所有正样本前加上权重，权重数值与(1-pt)正相关，即分类错误的概率值越接近ground truth，则权重值越小	只能让optimizer关注更多的classification loss值出的损失值（即分类错误程

一、focal loss原理及代码实现

kaiming大神的focal loss

以二分类的cross entropy为例：loss=-y*log(p)-(1-y)*log(1-p)，当ground truth label=0时，loss=-log(1-p)，最小化loss值即最大化1-p，则希望p接近于0，当ground truth label=1时，loss=-log(p)，最小化loss值即最大化p，则希望p接近于1.

training (*e.g.*, 1:1000). We introduce the focal loss starting from the cross entropy (CE) loss for binary classification[1]:

$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases} \tag{1}$$

In the above $y \in \{\pm 1\}$ specifies the ground-truth class and $p \in [0, 1]$ is the model's estimated probability for the class with label $y = 1$. For notational convenience, we define p_t:

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise,} \end{cases} \tag{2}$$

and rewrite $\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t)$.

1.focal loss解决class imbalanced问题（alpha）

类别不平衡问题好像现在只能加权重了。代码里面比较好的计算方式是在每个batch size进行训练的时候，自动地统计当前batch size中每个类别的样本数（由于它只想解决正负样本的不平衡，并没有涉及到所有正样本前景类别的不平衡，故而可以动态地统计每个batch size内的正负样本数），并计算每个类别的频率，然后以一定的函数关系式取反比例，得到每个类别的权重（alpha）。

其实很奇怪，focal loss做了很多实验都是设定固定的alpha值来判断效果好，难道不是动态地在每个batch size内的正负样本数，然后再取反比例吗？我之前做过一个multi-class segmentation ，师兄说他是在每个mini-batch内部动态计算频率的，他说动态计算和对于整个数据集事先训练好区别不大，但是总归是要基于数据集本身决定各个类别权值的。作者经过实验观察得到，alpha=0.75时得到的效果最好，也就是（所有类别的）前景正样本anchor classification loss权重为0.75，而负样本的权重为0.25.感觉这莫名地和OHEM中的选择正负样本比例1：3不谋而合了。

3.1. Balanced Cross Entropy

A common method for addressing class imbalance is to introduce a weighting factor $\alpha \in [0, 1]$ for class 1 and $1 - \alpha$ for class -1. In practice α may be set by inverse class frequency or treated as a hyperparameter to set by cross validation. For notational convenience, we define α_t analogously to how we defined p_t. We write the α-balanced CE loss as:

$$\mathrm{CE}(p_t) = -\alpha_t \log(p_t). \tag{3}$$

This loss is a simple extension to CE that we consider as an experimental baseline for our proposed focal loss.

2. focal loss解决训练hard example的问题（gamma）

$$\mathrm{FL}(p_t) = -(1 - p_t)^{\gamma} \log(p_t). \tag{4}$$

The focal loss is visualized for several values of $\gamma \in [0, 5]$ in Figure 1. We note two properties of the focal loss. (1) When an example is misclassified and p_t is small, the modulating factor is near 1 and the loss is unaffected. As $p_t \to 1$, the factor goes to 0 and the loss for well-classified examples is down-weighted. (2) The focusing parameter γ

当p_t接近于0的时候，则-log(pt)接近于无穷大，说明当前样本的classification loss很大，应该花更多的注意力在这里，则(1-p_t)接近于1，则分类严重错误，是hard example困难样本，则会给当前的分类损失值赋予较大的权重；当p_t接近于1的时候，则-log(pt)接近于0，说明当前样本的classification loss很小，分类正确，是easy example简单样本，则(1-p_t)接近于0，会给当前的分类损失值赋予较小的权重，因为简单样本很容易分类正确所以不需要关注太多。

由于我的数据集中存在每个前景类别的不平衡，而在每个batch size中又不能保证每个类别的ground truth boxes都出现，所以决定事先统计出每个类别的

ground truth boxes在原始数据集中的频率，然后计算出每个类别的权重，最后把所有前景类别的权重值加起来除以3，就得到背景类别的权重。

```
1. #coding=gbk
2. import torch
3. import torch.nn as nn
4. import torch.nn.functional as F
5. from torch.autograd import Variable
6. import numpy as np
7. defcompute_class_weights(histogram):
8.     classWeights = np.ones(histogram.shape[0], dtype=np.float32)
9. # print(classWeights.shape)
10.     normHist = histogram / np.sum(histogram)
11. for i in range(histogram.shape[0]):
12.         classWeights[i] = 1 / (np.log(1.10 + normHist[i]))
13. return classWeights
14. classFocalLoss(nn.Module):
15. def__init__(self):
16.         super(FocalLoss, self).__init__()
17. defforward(self, cls_prob,rois_label):
18. '''
19.         :param cls_prob:经过softmax激活函数作用后的，shape [128,8]
num_class=8
20.         :param targets:ground truth class类别
21.         :return:focal loss classification loss
22.         '''
23.         proposal_num=cls_prob.shape[0]
24.         num_class=cls_prob.shape[1]
25.         class_mask=cls_prob.data.new(proposal_num,num_class)
26.         class_mask = Variable(class_mask)
27.         ids=rois_label.view(-1,1)
28.         class_mask.scatter_(1, ids.data, 1.)
29. '''
30.         class_mask  shape [proposal_num,num_class]
31.         表示对于ids进行one-hot编码，对应类别的概率值为1
32.         '''
```

```
33. #alpha 是用来解决物体检测中类别不平衡问题的
34.         frequency=torch.zeros(num_class,1)
35. for ij in range(torch.max(rois_label)+1):
36.             frequency[ij]=torch.sum(rois_label==ij)
37.         frequency=frequency.numpy()
38.         classWeights=compute_class_weights(frequency)
39.
alpha=Variable(torch.from_numpy(classWeights).view(-1,1)).cuda()#shape
[num_class,]
40.         alpha_class=alpha[ids.view(-1)]
41.         gamma = 2
42.         probs=(cls_prob*class_mask).sum(1).view(-1,1)
43.         log_p=probs.log()
44.         batch_loss = -alpha_class * (torch.pow((1 - probs), self.gamma)) *
log_p
45.         loss = batch_loss.mean()
46. return loss
```

二、OHEM原理及代码实现

OHEM(在线困难样本挖掘)，通常是 on line hard negative mining，对于困难的负样本进行在线挖掘，它与focal loss的目标一样，都是为了处理物体检测问题中的类别不均衡问题.这里的类别不均衡，是指由于基于anchor 的检测器都是使用密集检测的策略，如RPN以及所有的one-stage detector，则在数量众多的anchor中有大量的anchor框是负样本——背景框，只有少量的anchor框是正样本，如果不使用任何的策略，直接将所有的anchor直接计算cross entropy的classification loss，则由于很多背景anchor的特征对应到输入图像上的感受野部分就是背景，故而是很容易被分类出来的，这一类很容易被分类正确的负样本被称为简单负样本（对应到代码里面就是那些分类损失值比较小的负样本anchor），在所有的负样本anchor中，简单负样本的数量占据了绝大多数，故而网络最终训练好了，分类损失函数值下降，可能就是由于将大量简单负样本分类正确所导致的分类损失函数值小，但这并不是训练detector的目标，我们的目标是检测器能正确区分正负样本并且能够对于所有的正样本进行多个类别的正确划分，故而需要挖掘出那些困难的负样本（其实困难的负样本可以理解为就是和前景ground truth boxes的IOU数值比较大的，但是又没有超过所设定的正样本阈

值的那部分anchor boxes），简单负样本就是那些与前景ground truth boxes的IOU数值较小的，很容易被分类成背景anchor的负样本。所以on line hard example mining是对于负样本进行的，通常的做法是，先对于所有的anchor boxes计算出分类损失值（size_average=False），也就是计算出每个anchor boxes所对应的classification loss，然后将所有正样本的classification loss值取出，将正样本anchor boxes的数量记作pos_num，再对于剩下的所有负样本anchor boxes的classification loss进行从大到小的排序，最后从排序好的负样本anchor boxes的loss中取出前3*num_pos个classification loss，再与所有的正样本分类损失值相加，最终除以的分母是num_pos。

SSD训练过程中这里引用torchcv中的ssd_loss.py代码进行解释：

```
1. from __future__ import print_function
2. import torch
3. import torch.nn as nn
4. import torch.nn.functional as F
5. classSSDLoss(nn.Module):#损失函数
6. def__init__(self, num_classes):
7.         super(SSDLoss, self).__init__()
8.         self.num_classes = num_classes#类别总数，对于VOC数据集而言，是21类
9. def_hard_negative_mining(self, cls_loss, pos):
10. '''Return negative indices that is 3x the number as postive indices.
11.         Args:
12.           cls_loss: (tensor) cross entroy loss between cls_preds and cls_targets, sized [N,#anchors]. 分类损失值
13.           pos: (tensor) positive class mask, sized [N,#anchors].
14.         Return:
15.           (tensor) negative indices, sized [N,#anchors].
16.         '''
17.         cls_loss = cls_loss * (pos.float() - 1)
18. #对于正样本，损失值为0，得到对于负样本计算出的损失值，损失值越大的负样本，cls_loss值越小
19. #正样本损失值  0
20. #负样本损失值=之前的负样本损失值*（-1）
```

```
21. #这是因为_hard_negative_mining只返回所有的负样本classification loss
22. #从所有的负样本中采样出前（3*num_positive）个负样本的loss
23. #这些负样本的classification loss最大，是困难的负样本
24.         _, idx = cls_loss.sort(1)  # sort by negative losses
25. '''
26.         cls_loss: [N,#anchors]  正样本的损失值为0，对于负样本，损失值越大，
cls_loss越小
27.         tensor.sort方法返回sort之后的按升序排列的tensor和对应的indices
28.         对每一行，遍历所有的列，则得到的每一行按照升序排列，即对于每个
input images，得到其按照升序排列的分类损失idx
29.         idx同样是[N,#anchors].的tensor，其中的每一行的值范围为  [0,1,2,
.....,8732]
30.         表示当前input image 的所有anchors的负样本的分类损失 由大到小的索引
排序
31.         '''
32.         _, rank = idx.sort(1)       # [N,#anchors]
33.         num_neg = 3*pos.sum(1)  # [N,]
34. #num_neg为长度为batch size 的tensor，其中的每个元素表示3*当前input
image中的正样本个数
35.         neg = rank < num_neg[:,None]   # [N,#anchors]  neg中的数值为1或者
0  如果是hard  negative examples，则对应位置处的值为1
36. '''
37.         对于当前batch size张图像中的每一张（每一张图像中的正样本不同）
38.         找到是当前图像中正样本数量3倍的负样本，并且固定数量的负样本是通过
在线困难样本挖掘得到的
39.         这主要是为了解决计算分类损失函数时样本不均衡的问题，因为比如说
SSD300这种模型中8732个default boxes
40.         中的正样本数量很少（与ground  truth 的overlap大于0.5,在
box_coder.encode函数中设置）
41.         为了保证在同一张图像中的正负样本比例在1：3，故而使用在线困难样本
挖掘（在线指的是在训练过程中，这意味着
42.         在每次训练过程中，每次挖掘到的困难负样本可能是不同的，要根据网络模
型预测的输出值决定）
43.         算法如下：
```

```
44.         首先取出所有的负样本，对于当前batch_size*#anchors ，对于每一行（每张训练图像）的分类损失值进行排序
45.         按照当前图像中正样本的数量的3倍取出loss值排在前面的负样本）
46.         负样本的分类损失值计算：np.log(p) 小  p小，就是说对于负样本预测为背景类的概率值小，就是预测为前景的概率值大
47.         这些是很容易被分类错的负样本，被称为困难负样本，这些样本的loss值很大，对于网络模型的参数更新非常有效
48.         而那些很容易就能被分类正确的负样本对于最终权值更新效果不大，故而舍弃
49.         '''
50. return neg
51. defforward(self, loc_preds, loc_targets, cls_preds, cls_targets):
52. '''Compute loss between (loc_preds, loc_targets) and (cls_preds, cls_targets).
53.         计算分类损失和回归损失
54.         Args:
55.           loc_preds: (tensor) predicted locations, sized [N, #anchors, 4].
56.                     对于当前batch size的图像所预测出来的localization
57.                     N=batch_size
58.                     #anchors表示default boxes的数量
59.           loc_targets: (tensor) encoded target locations, sized [N, #anchors, 4].
60.           cls_preds: (tensor) predicted class confidences, sized [N, #anchors, #classes].
61.                     对于当前batch size的图像所预测出来的classification
62.                     N=batch_size,#anchors表示default boxe数量，
63.                     #classes表示数据集类别总数
64.           cls_targets: (tensor) encoded target labels, sized [N, #anchors].
65.                     batch_size行，#anchors列，
66.                     第i行第j列的元素表示
67.                     对于第i个训练样本图像，SSD预测出来的第j个default boxes的GT类别标号（一个int类型整数）
68.         loss:
69.           (tensor) loss = SmoothL1Loss(loc_preds, loc_targets) + CrossEntropyLoss(cls_preds, cls_targets).
```

```
                    位置回归损失                    交叉熵分类损失
        '''
        pos = cls_targets > 0# [N,#anchors] pos中的数值是 0 1
'''
        cls_targets是经过编码之后的classification ground truth
        表示与ground truth bounding boxes的IOU值最大或者大于一定的阈值的anchor boxes则会被认为是正样本，为1
        负样本为-1
        在encoder阶段，会计算出当前anchor 与当前输入图像中所有ground truth boxes的IOU，并将anchor与所有gt boxes
        最大的IOU值记作当前anchor的overlap值，如果anchor的overlap值大于阈值0.5，则将anchor记作为正样本
        IOU小于0.5为负样本
        '''
        batch_size = pos.size(0)#每个batch 中包含多少张训练图片
        num_pos = pos.sum().item()#对pos 2-Dtensor求和，得到当前batch size的训练图片中共有多少个anchor boxes为正样本
#当前batch size 数量的输入图像中，positive examples（这里的正样本指的是default boxes而不是一整张图像）的数量

#==========================================================================
# loc_loss = SmoothL1Loss(pos_loc_preds, pos_loc_targets)

#==========================================================================
        mask = pos.unsqueeze(2).expand_as(loc_preds)       # [N,#anchors,4]
        loc_loss = F.smooth_l1_loss(loc_preds[mask], loc_targets[mask], size_average=False)#只对正样本进行回归损失的计算
#mask是# [N,#anchors,4]的3-dimension tensor，扩展的第2维度与之前的数值相同，即对于正样例（batch size中的第i幅图片中的第j个anchors）
#mask[i,j,:]=1,如果为负样本则mask[i,j,:]=0
#mask作下标则表示其中元素值为1的下标，即所有的正样本所在的下标（4）

#==========================================================================
```

```
==============
93. # cls_loss = CrossEntropyLoss(cls_preds, cls_targets)
94.
#==============================================================
==============
95.         cls_loss = F.cross_entropy(cls_preds.view(-1,self.num_classes), \
96.                                    cls_targets.view(-1), reduce=False)  #
[N*#anchors*num_classes,]
97. '''
98.         cls_preds:[N,#anchors,num_classes]   view    cls_preds:
[（N*#anchors）,num_classes]
99.         cls_targets：[N*#anchors,]
100.        计算多分类的交叉损失函数是cross_entropy,reduce参数为false，则返回
值cls_loss维度为(N*#anchors)
101.        分别给出了这一个batch size中每张图像所有anchor boxes的分类损失值
得
102.        '''
103.        cls_loss = cls_loss.view(batch_size, -1)#cls_loss:[N,#anchors]
104.        cls_loss[cls_targets<0] = 0# set ignored loss to 0 现将所有负样本的分
类损失变成0，这是为了使用hard negative mining算法挑选出困难负样本
105.        neg = self._hard_negative_mining(cls_loss, pos)  # [N,#anchors]
106.        cls_loss = cls_loss[pos|neg].sum()
107. '''
108.        正样本具有分类损失和回归损失，SSD中的正样本包括最大的IOU和IOU
值大于0.5的region proposal
109.        一般的负样本没有分类损失，也没有回归损失
110.        hard negative examples具有分类损失，不具有回归损失
111.        实际上训练时采用的正负样本是所有的正样本和所有的hard negative
examples，
112.        '''
113.        print('loc_loss: %.3f | cls_loss: %.3f' % (loc_loss.item()/num_pos,
cls_loss.item()/num_pos), end=' | ')
114.        loss = (loc_loss+cls_loss)/num_pos
115. return loss
```

三、OHEM应用到faster RCNN

如何将OHEM(on line hard example/negative mining)用到faster RCNN中？

可以在原始的faster RCNN代码实现中加入了OHEM，值得注意的是，OHEM是在计算RPN的classification loss时使用的，在计算RCNN的classification loss使用的是全部的2000个region proposal。原始的faster RCNN代码中并没有加入困难样本挖掘，而是：对于所有的anchor boxes，IOU大于0.7为正样本 ，IOU小于0.3为负样本，然后随机在一个batch size的输入图像中采样出128个正样本和128个负样本（比例1：1），这里并没有使用困难样本挖掘，因为IOU小于0.3很大概率是简单负样本。

个人感觉OHEM比较适合false positive 很多的情况，这种就是把背景框划分为前景框了，对于背景框的分类不准确，这是由于训练负样本时都是使用的简单负样本的原因，需要加入更多的困难负样本进行训练。

四、focal loss应用到faster RCNN

由于在现在的实验中，RPN部分的classification loss效果比较好，就是说前景背景二分类准确率在95%左右，故而这里我不在加入任何策略（是用最原始的策略，IOU大于0.7正样本，IOU小于0.3是负样本），故而在RCNN部分的classification loss使用了focal loss。