



fastai lecture 9

2018年7月



上海交通大學

SHANGHAI JIAO TONG UNIVERSITY



概述



Lecture nine依然承接上一课有关目标检测的内容，但是将单目标识别扩展到多目标的识别问题上。

本次介绍流程会按照训练的三要素依次进行，分别是数据(data)、网络结构(architecture)、损失函数(loss function)。

在本次课程的最后会补充一点相关的进阶技巧，它们可以较好地提升训练的效果。



1

三要素

2

进阶技巧





1

三要素

2

进阶技巧



1、数据



数据增强——扩大数据量的方式

图片+bounding box

翻转、旋转、调整亮度 (rand)

```
In [51]: augs = [RandomFlip(),  
                RandomRotate(30),  
                RandomLighting(0.1,0.1)]
```

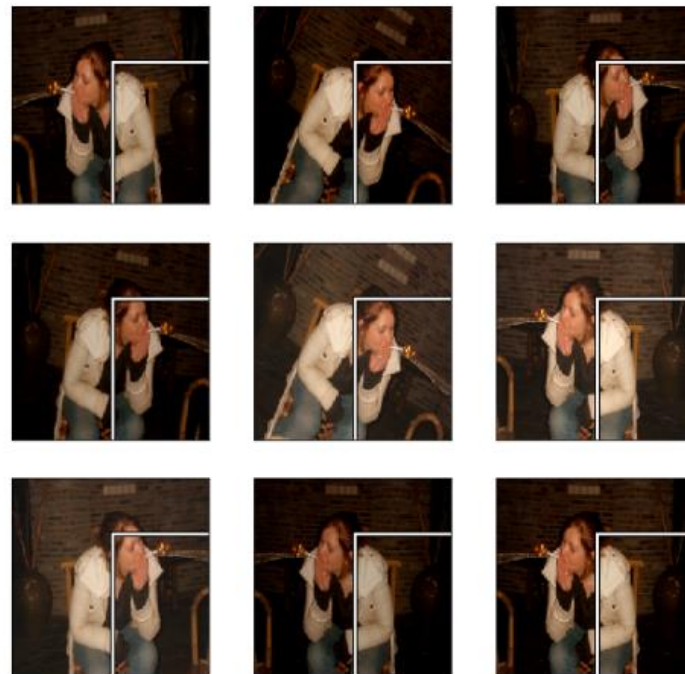
```
In [52]: tfms = tfms_from_model(f_model, sz, crop_type=CropType.NO, aug_tfms=augs)  
md = ImageClassifierData.from_csv(PATH, JPEGs, BB_CSV, tfms=tfms, continuous=True, bs=4)
```

回归问题 (regression)

数据增强



```
In [53]: idx=3
fig, axes = plt.subplots(3,3, figsize=(9,9))
for i, ax in enumerate(axes.flat):
    x,y=next(iter(md.aug_dl))
    ima=md.val_ds.denorm(to_np(x))[idx]
    b = bb_hw(to_np(y[idx]))
    print(b)
    show_img(ima, ax=ax)
    draw_rect(ax, b)
```



数据增强

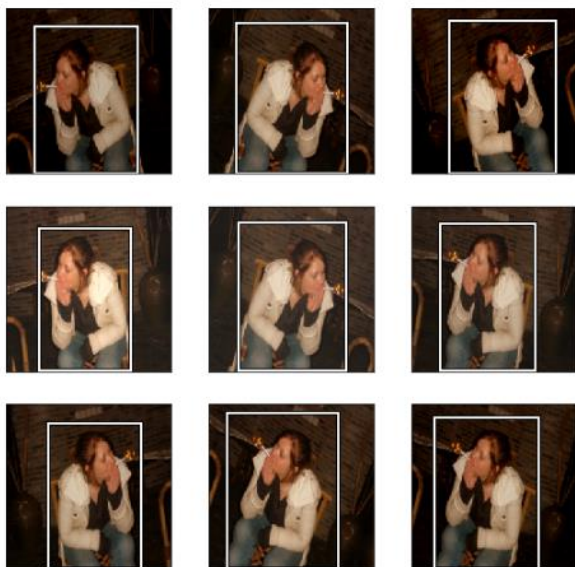


此处代码可以让变换操作一起应用到bounding box

↓

```
In [54]: augs = [RandomFlip(tfm_y=TfmType.COORD),  
               RandomRotate(30, tfm_y=TfmType.COORD),  
               RandomLighting(0.1, 0.1, tfm_y=TfmType.COORD)]
```

```
In [55]: tfms = tfms_from_model(f_model, sz, crop_type=CropType.NO, tfm_y=TfmType.COORD, aug_tfms=augs)  
md = ImageClassifierData.from_csv(PATH, JPEGS, BB_CSV, tfms=tfms, continuous=True, bs=4)
```



数据增强



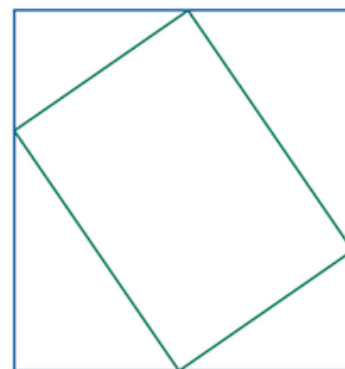
Tips

一定的旋转操作会导致box的增大（以四个角为依据），因此我们实验中设置的旋转参数就不能很大

读入图片变换的概率



Randc






1、数据

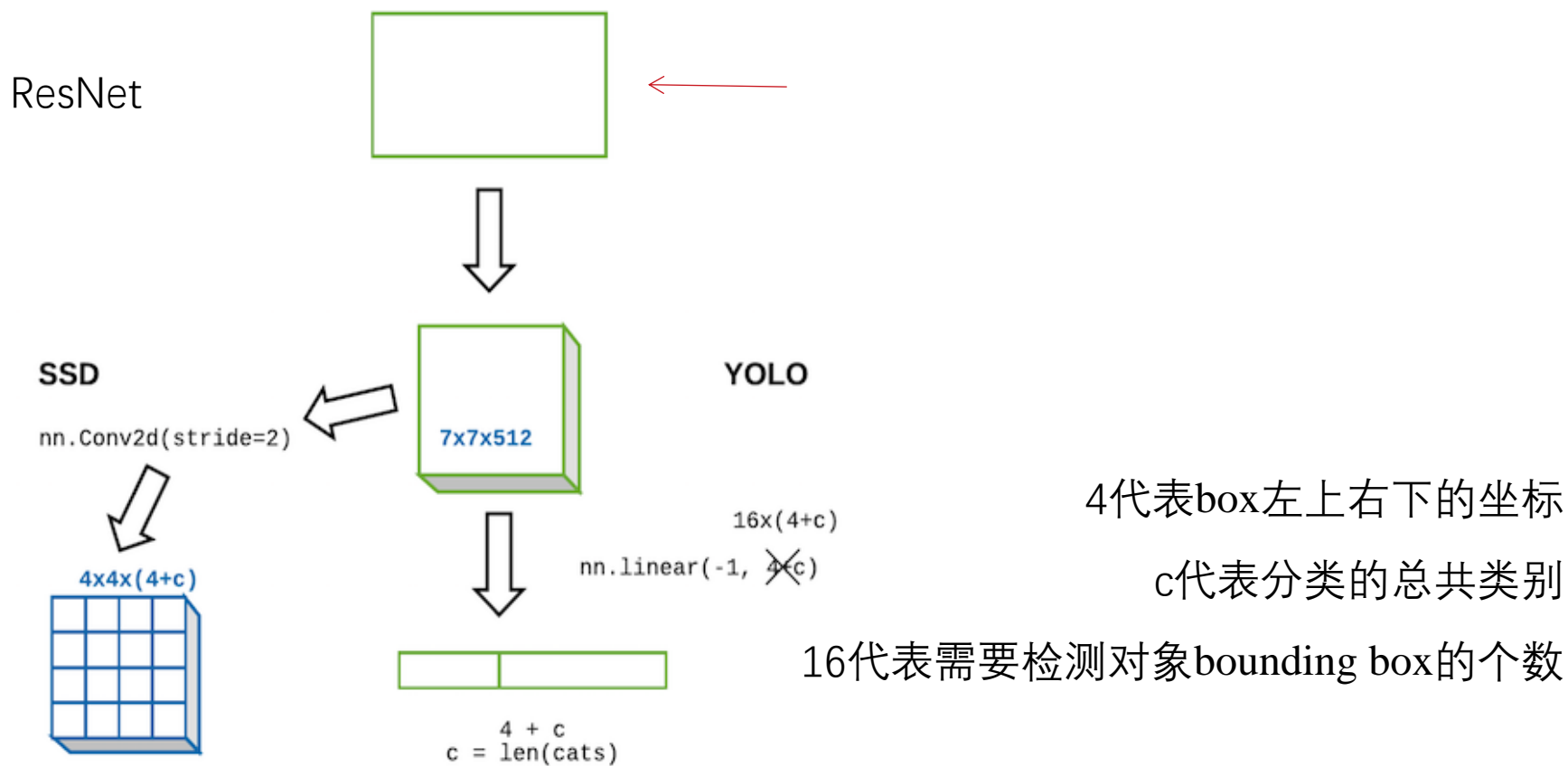


Set函数保证检测出的相同标签只显示一次。



```
In [10]: mc = [set([cats[p[1]] for p in tm_anno[o]]) for o in tm_ids] #将标签集合在一起，且相同的标签只显示一次
mcs = [''].join(str(p) for p in o) for o in mc]
```

2、结构



2、结构



4是regression问题，c为分类问题，整个模型对两个问题基本共享了前面所有的层，除了最后一层。

背景

```
class OutConv(nn.Module): #两个分离的卷积层，公用之前的所有层
    def __init__(self, k, nin, bias):
        super().__init__()
        self.k = k
        self.oconv1 = nn.Conv2d(nin, (len(id2cat)+1)*k, 3, padding=1) #k=1, classifier
        self.oconv2 = nn.Conv2d(nin, 4*k, 3, padding=1) #bounding box regression
        self.oconv1.bias.data.zero_().add_(bias)

    def forward(self, x):
        return [flatten_conv(self.oconv1(x), self.k),
                flatten_conv(self.oconv2(x), self.k)] #分离的tensor结果和activations
```

2、结构



```
class SSD_Head(nn.Module):
    def __init__(self, k, bias):
        super().__init__()
        self.drop = nn.Dropout(0.25)
        self.sconv0 = StdConv(512, 256, stride=1) #不改变图片大小, 只是一层正常卷积
        # self.sconv1 = StdConv(256, 256)
        self.sconv2 = StdConv(256, 256) #output为4x4
        self.out = OutConv(k, 256, bias)

    def forward(self, x):
        x = self.drop(F.relu(x))
        x = self.sconv0(x)
        # x = self.sconv1(x)
        x = self.sconv2(x)
        return self.out(x)

head_reg4 = SSD_Head(k, -3.)
models = ConvnetBuilder(f_model, 0, 0, 0, custom_head=head_reg4)
learn = ConvLearner(md, models)
learn.opt_fn = optim.Adam
```

```
class StdConv(nn.Module):
    def __init__(self, nin, nout, stride=2, drop=0.1):
        super().__init__()
        self.conv = nn.Conv2d(nin, nout, 3, stride=stride, padding=1)
        self.bn = nn.BatchNorm2d(nout)
        self.drop = nn.Dropout(drop)

    def forward(self, x): return self.drop(self.bn(F.relu(self.conv(x))))

def flatten_conv(x, k):
    bs, nf, gx, gy = x.size()
    x = x.permute(0, 2, 3, 1).contiguous()
    return x.view(bs, -1, nf//k)
```

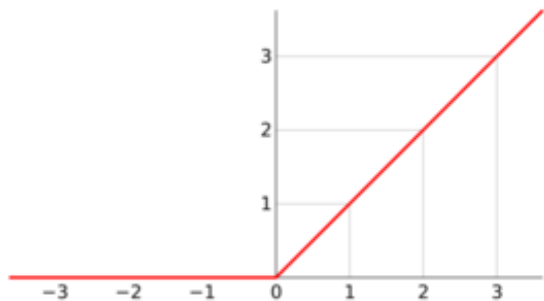
紧接着ResNet后的层

2、结构



Tips

1、实验中batchnorm放在relu后面，从概念上说batchnorm把结果分布到零均值单位方差，如果放前面再经过relu就会导致结果没有负值，但是在实际训练中顺序反过来也可以得到很好的结果。



2、尽管Batchnorm已经保证了规则化，但还是需要使用dropout用以进一步降低过拟合。

3、损失函数



单目标检测——一个目标的坐标和分类的损失函数

多目标检测——需要知道检测出的目标分类和位置是否和图片中ground truth中的目标一致，这里还会涉及到一个匹配问题。

grid cell

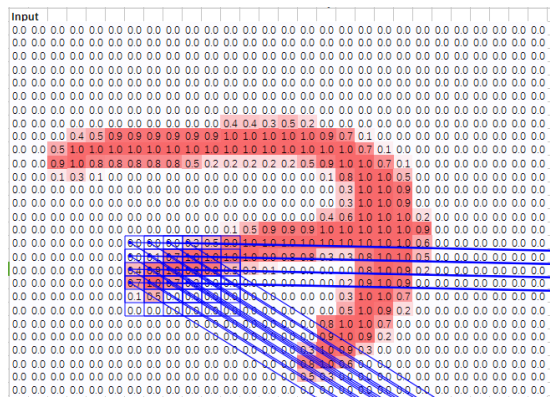
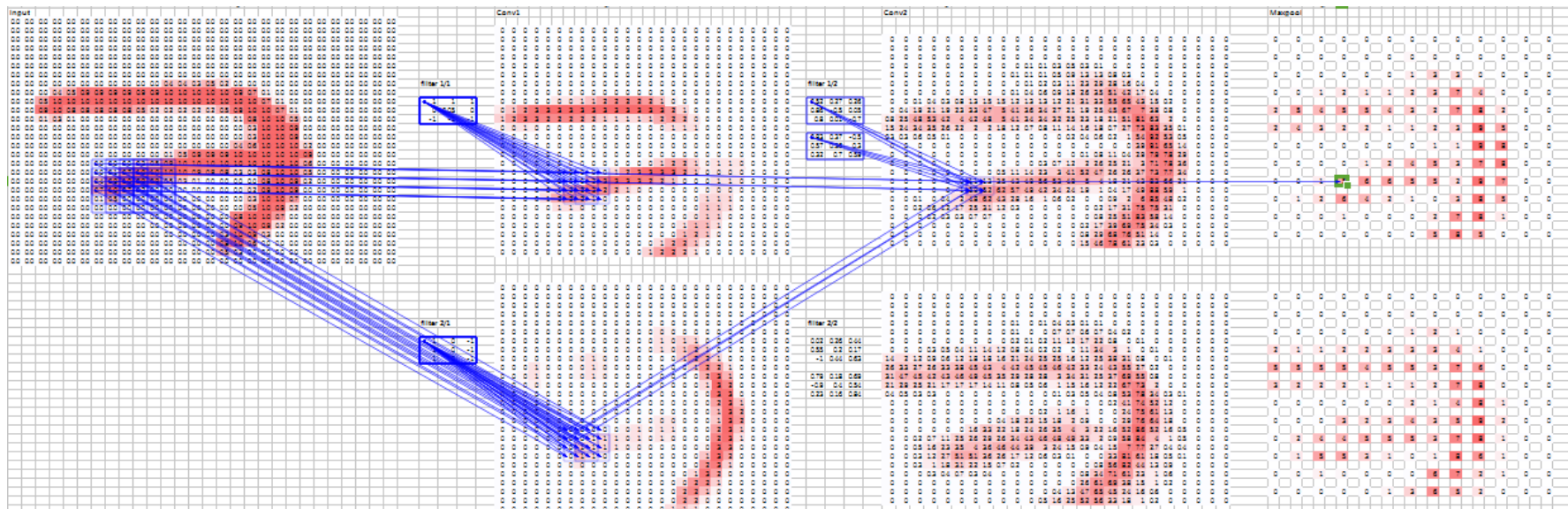


这是一个先验的概念，是它独立于训练的部分，是stride=2的卷积层输出的几何表现。

直观上来说就是人为的将图像分成许多个单元格（grid cell），然后在单个grid cell中进行bounding box的回归，是自然的人类的“分而治之”的思想。

从神经网络上来说，每一个grid cell是由于与其“层层相关”感知域（receptive field）才能够和原图片中对应区域的物体相关联的。

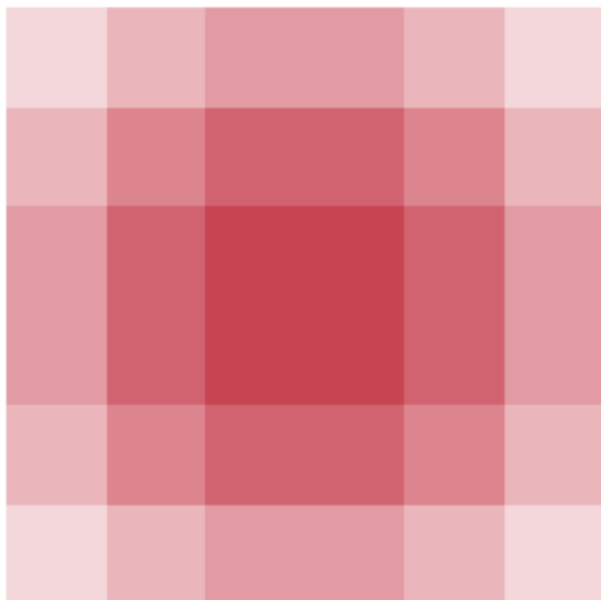
感知域 (receptive field)



通过Excel识别踪迹得到的结果

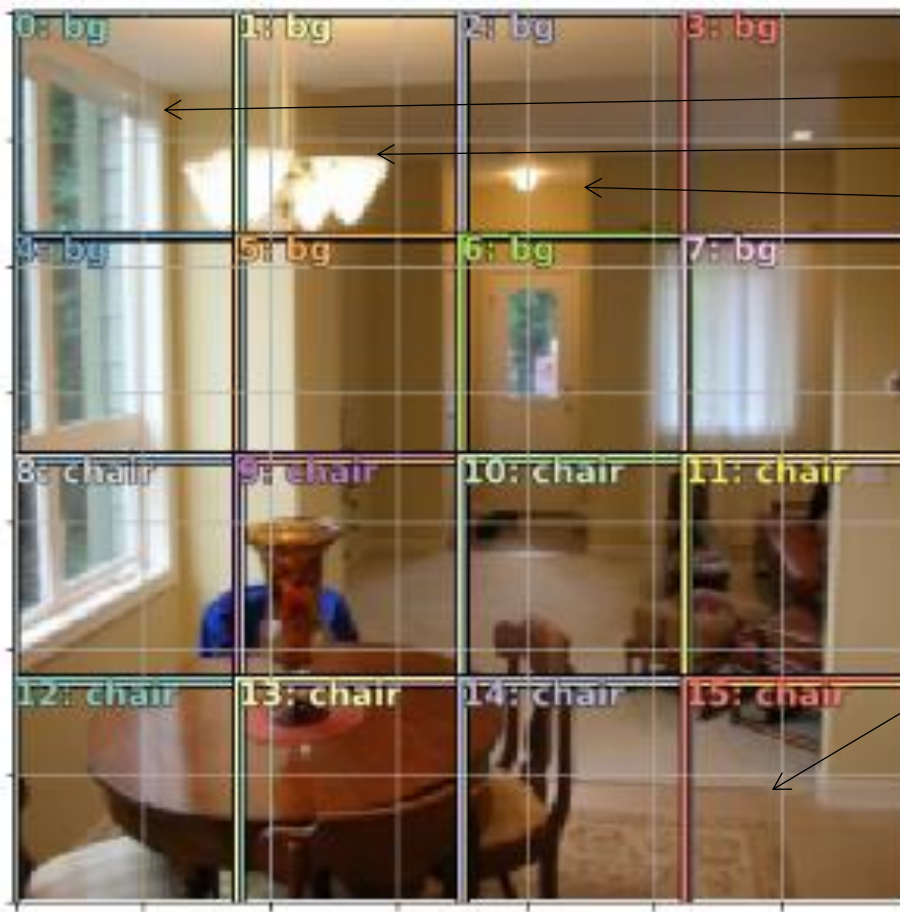
6x6方格像素

感知域 (receptive field)



靠近中间的部分被计算多次，而边缘的部分则仅被计算一次，因此感知域概念不仅指原图中间用以计算最后activation的部分，更进一步是指格子的中心部分。

grid cell



4+c

4+c

4+c

o

o

o

o

o

o

4+c

16

anchor box



每个grid cell中都有一个到几个anchor box，用以检测目标。

但与感知域不同的是，同样从activation中推回，anchor box并不是固定的，它需要根据具体的activation推出，代码如下：

```
def actn_to_bb(actn, anchors):  
    actn_bbs = torch.tanh(actn) #activation经过tanh变换成1到-1的范围  
    actn_centers = (actn_bbs[:, :2] / 2 * grid_sizes) + anchors[:, :2]  
    #每一个预测的bounding box中心可以移动半个grid cell size的范围  
    actn_hw = (actn_bbs[:, 2:] / 2 + 1) * anchors[:, 2:]  
    #预测的box大小可以两倍到一半的范围  
    return hw2corners(actn_centers, actn_hw)
```

ground truth中的bbox



<jaccard index方式>

Ground truth（原始图片）中的bounding box（简记为bbox）

与16个anchor boxes 中重叠部分除以总共的部分，记为IoU，比较16个IoU的大小。

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

fn	bbox
0 000012.jpg	96 155 269 350
1 000017.jpg	61 184 198 278 77 89 335 402
2 000023.jpg	229 8 499 244 219 229 499 333 0 1 368 116 1 2 ...
3 000026.jpg	124 89 211 336
4 000032.jpg	77 103 182 374 87 132 122 196 179 194 228 212 ...

4 4 + 21

f(,) →

ground truth中的bbox



```
In [125]: overlaps = jaccard(bbox.data, anchor_cnr.data)
overlaps #3x16
```

Out[125]:

```
Columns 0 to 9
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0091
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0356 0.0549
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000

Columns 10 to 15
0.0922 0.0000 0.0000 0.0315 0.3985 0.0000
0.0103 0.0000 0.2598 0.4538 0.0653 0.0000
0.0000 0.1897 0.0000 0.0000 0.0000 0.0000
[torch.cuda.FloatTensor of size 3x16 (GPU 0)]
```

```
In [126]: overlaps.max(1) #在row (目标) 上取最大
```

```
Out[126]: (
  0.3985
  0.4538
  0.1897
[torch.cuda.FloatTensor of size 3 (GPU 0)],
 14
 13
 11
[torch.cuda.LongTensor of size 3 (GPU 0)])
```

看到目标和16个
anchor boxess之间的
IoU结果

3x16矩阵，3代表
目标，16代表
anchor box数量

ground truth中的bbox



```
In [127]: overlaps.max(0) #for each anchor box, 0同时表示目标1和没有重叠
```

```
Out[127]: (  
  0.0000  
  0.0000  
  0.0000  
  0.0000  
  0.0000  
  0.0000  
  0.0000  
  0.0000  
  0.0000  
  0.0356  
  0.0549  
  0.0922  
  0.1897  
  0.2598  
  0.4538  
  0.3985  
  0.0000  
  [torch.cuda.FloatTensor of size 16 (GPU 0)],  
  0  
  0  
  0  
  0  
  0  
  0  
  0  
  0  
  0  
  0  
  1  
  1  
  0  
  2  
  1  
  1  
  0  
  0  
  [torch.cuda.LongTensor of size 16 (GPU 0)])
```

需要注意的是当对应的目标为0时，要么是没有重叠，要么是和目标0重叠最大

ground truth中的bbox



```
In [128]: gt_overlap, gt_idx = map_to_ground_truth(overlaps)
          gt_overlap, gt_idx  #将两种max结合得到具体grid cell和目标的对应，行上的最大和列上超过0.5的grid cells给与对应的object，其余为背景
```

```
Out[128]: (
  0.0000
  0.0000
  0.0000
  0.0000
  0.0000
  0.0000
  0.0000
  0.0000
  0.0356
  0.0549
  0.0922
  1.9900
  0.2598
  1.9900
  1.9900
  0.0000
  [torch.cuda.FloatTensor of size 16 (GPU 0)],
  0
  0
  0
  0
  0
  0
  0
  0
  0
  1
  1
  0
  2
  1
  1
  0
  0
  [torch.cuda.LongTensor of size 16 (GPU 0)])
```

- 1 将行上最大值对应的目标赋予；
- 2 在列上IoU大于0.5的目标分类赋予；
- 3 剩下的当做背景（无目标被识别）。

ground truth中的bbox



```
def map_to_ground_truth(overlaps, print_it=False):
    prior_overlap, prior_idx = overlaps.max(1)
    if print_it: print(prior_overlap)
    # pdb.set_trace()
    gt_overlap, gt_idx = overlaps.max(0)
    gt_overlap[prior_idx] = 1.99 #行上最大设置一个大常数, 保证标签绝对赋值
    for i, o in enumerate(prior_idx): gt_idx[o] = i
    return gt_overlap, gt_idx
```

In [129]: gt_clas = clas[gt_idx]; gt_clas #转换成object的classes

```
Out[129]: Variable containing:
      8
      8
      8
      8
      8
      8
      8
      8
      8
     10
     10
      8
     17
     10
     10
      8
      8
[torch.cuda.LongTensor of size 16 (GPU 0)]
```

In [131]: gt_clas[1-pos] = len(id2cat)
[id2cat[o] if o < len(id2cat) else 'bg' for o in gt_clas.data] #anchor boxes

```
Out[131]: ['bg',
           'bg',
           'bg',
           'bg',
           'bg',
           'bg',
           'bg',
           'bg',
           'bg',
           'bg',
           'bg',
           'bg',
           'bg',
           'sofa',
           'bg',
           'diningtable',
           'chair',
           'bg']
```

ground truth中的bbox



```
In [130]: thresh = 0.5 # 阈值, 大于的为检测到的目标类型
          pos = gt_overlap > thresh
          pos_idx = torch.nonzero(pos)[: , 0]
          neg_idx = torch.nonzero(1-pos)[: , 0]
          pos_idx
```

```
Out[130]:
          11
          13
          14
          [torch.cuda.LongTensor of size 3 (GPU 0)]
```

ground truth中的bbox	预测的anchor Box	grid cell
$m \times (4+c)$	$16 \times (4+c)$	16

loss function



```
def ssd_l_loss(b_c, b_bb, bbox, clas, print_it=False): #for one image
    bbox, clas = get_y(bbox, clas) #去除padding
    a_ic = actn_to_bb(b_bb, anchors) #将activation转换成box
    overlaps = jaccard(bbox.data, anchor_cnr.data)
    gt_overlap, gt_idx = map_to_ground_truth(overlaps, print_it)
    gt_clas = clas[gt_idx]
    pos = gt_overlap > 0.4 #阈值不同
    pos_idx = torch.nonzero(pos)[:,0] #找到匹配的部分
    gt_clas[1-pos] = len(id2cat) #将背景class而是放入
    gt_bbox = bbox[gt_idx]
    loc_loss = ((a_ic[pos_idx] - gt_bbox[pos_idx]).abs()).mean() #L1loss for location
    clas_loss = loss_f(b_c, gt_clas)
    return loc_loss, clas_loss

def ssd_loss(pred, targ, print_it=False):
    lcs, lls = 0., 0.
    for b_c, b_bb, bbox, clas in zip(*pred, *targ):
        loc_loss, clas_loss = ssd_l_loss(b_c, b_bb, bbox, clas, print_it)
        lls += loc_loss
        lcs += clas_loss
    if print_it: print(f'loc: {lls.data[0]}, clas: {lcs.data[0]}')
    return lls+lcs
```


loss function



```
def get_y(bbox, clas): #一次一个minibatch, 一次所有目标类别数
    bbox = bbox.view(-1, 4) / sz
    bb_keep = ((bbox[:, 2] - bbox[:, 0]) > 0).nonzero()[:, 0]
    #fastai自己应对不同的图片中目标的不同数量, 进行补零, 这里是去除0的操作
    return bbox[bb_keep], clas[bb_keep]

class BCE_Loss(nn.Module): #binary cross entropy
    def __init__(self, num_classes):
        super().__init__()
        self.num_classes = num_classes

    def forward(self, pred, targ):
        t = one_hot_embedding(targ, self.num_classes+1) #one hot编码包括背景一类
        t = V(t[:, :-1], contiguous())#.cpu() #除去最后一类, 即全零为背景
        x = pred[:, :-1]
        w = self.get_weight(x, t)
        return F.binary_cross_entropy_with_logits(x, t, w, size_average=False) / self.num_classes

    def get_weight(self, x, t): return None

loss_f = BCE_Loss(len(id2cat))
```

loss function



Tips:

Softmax是不适用的，因为会存在anchor box没有对应的情况。此处如果将没有对应即background作为一个类别，那么softmax是可以使用的，但是background并非可以作为类型直接分别，而是通过排除其他所有可能的目标对应情况下决定的，所以它作为类别将难以判断。

Loss结果如下：

```
In [132]: gt_bbox = bbox[gt_idx]
loc_loss = ((a_ic[pos_idx] - gt_bbox[pos_idx]).abs()).mean() #只计算3个目标的差异
#匹配得到的box和ground truth的bounding boxes求loss平均
clas_loss = F.cross_entropy(b_clasi, gt_clas)
#对于分类使用cross entropy
loc_loss, clas_loss
```

```
Out[132]: (Variable containing:
1.00000e-02 *
7.7381
[torch.cuda.FloatTensor of size 1 (GPU 0)], Variable containing:
1.1878
[torch.cuda.FloatTensor of size 1 (GPU 0)])
```

训练结果



```
In [112]: learn.fit(lr, 1, cycle_len=5, use_clr=(20, 10))
```

```
HBox(children=(IntProgress(value=0, description='Epoch', max=5), HTML(value='')))
```

epoch	trn_loss	val_loss
0	43.041973	32.373425
1	33.690237	28.904852
2	29.329869	27.249815
3	26.588593	26.256581
4	24.385616	25.826601

```
Out[112]: [array([25.8266])]
```

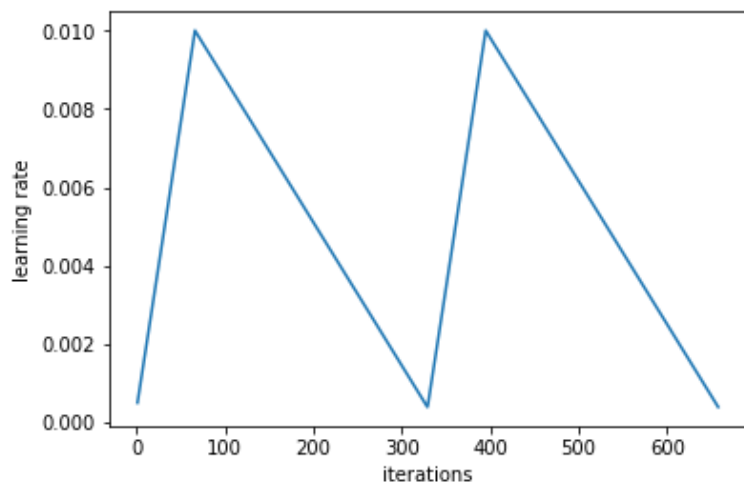
训练结果



Tips:

参数use_clr, use_clr=(clr_div=20,cut_div=4)

20代表学习率的最大值和最小值的比值，如图中0.01和0.0005；4代表上升和下降的占epoch的分布，如图中上升1/4，下降3/4。





1

三要素

2

进阶技巧

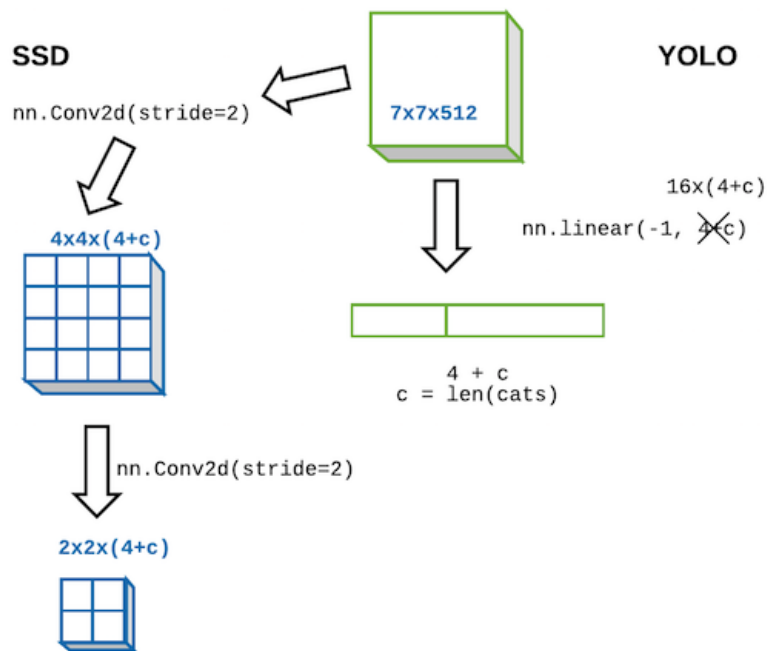


创建更多的anchor boxes



因为IoU的分母是总共的部分，目标的大小肯定会变化，如果没有合适的对应大小的anchor boxes，overlap的阈值就会一直保持小于0.5，导致结果漏检。同时与之对应的grid cell大小也需要变化，同样的道理，过小会导致好几个cells中都有相同的目标，本实验中grid cell设置为4x4，2x2，1x1。

对应的结构如下图：



创建更多的anchors boxes



增加了一层stride=2的卷积层，从而得到了 $2 \times 2 \times (4+c)$ 的结果，简单来说，就是将原图又分成了 2×2 的grid cells，可以进一步再加一个卷积层得到 $1 \times 1 \times (4+c)$ 的activation。

In [149]: drop=0.4

```
class SSD_MultiHead(nn.Module):
    def __init__(self, k, bias):
        super().__init__()
        self.drop = nn.Dropout(drop)
        self.sconv0 = StdConv(512, 256, stride=1, drop=drop)
        self.sconv1 = StdConv(256, 256, drop=drop)
        self.sconv2 = StdConv(256, 256, drop=drop)
        self.sconv3 = StdConv(256, 256, drop=drop)
        self.out0 = OutConv(k, 256, bias)
        self.out1 = OutConv(k, 256, bias)
        self.out2 = OutConv(k, 256, bias)
        self.out3 = OutConv(k, 256, bias)

    def forward(self, x):
        x = self.drop(F.relu(x))
        x = self.sconv0(x)
        x = self.sconv1(x)
        o1c, o1l = self.out1(x) #4x4
        x = self.sconv2(x)
        o2c, o2l = self.out2(x) #2x2
        x = self.sconv3(x)
        o3c, o3l = self.out3(x) #1x1
        return [torch.cat([o1c, o2c, o3c], dim=1),
                torch.cat([o1l, o2l, o3l], dim=1)]

head_reg4 = SSD_MultiHead(k, -4.)
models = ConvnetBuilder(f_model, 0, 0, 0, custom_head=head_reg4)
learn = ConvLearner(md, models)
learn.opt_fn = optim.Adam
```

创建更多的anchors boxes



在anchor boxes本身变换上还可以使用zoom和ratios对应缩放和宽高比。这里k代表的是zoom乘以ratios，在结果的activation上也会变成 $(4 \times 4 + 2 \times 2 + 1 \times 1) * k * (4 + c)$ ，这里为3。

```
In [141]: anc_grids = [4,2,1] #不同的grid, 4x4 2x2 1x1
# anc_grids = [2]
#anc_zooms = [0.7, 1., 1.3]
anc_zooms = [1.] #不同的大小
anc_ratios = [(1.,1.), (1.,0.5), (0.5,1.)] #不同的长宽比
# anc_ratios = [(1.,1.)]
anchor_scales = [(anz*i, anz*j) for anz in anc_zooms for (i,j) in anc_ratios]
k = len(anchor_scales) #zooms乘以ratios
anc_offsets = [1/(o*2) for o in anc_grids]
k
```

Out[141]: 3

创建更多的anchors boxes



64代表batchsize, 63为 $3 \times (4 \times 4 + 2 \times 2 + 1 \times 1)$, 21为分类数量, 4为坐标

```
In [152]: batch[0].size(), batch[1].size()
```

```
Out[152]: (torch.Size([64, 63, 21]), torch.Size([64, 63, 4]))
```

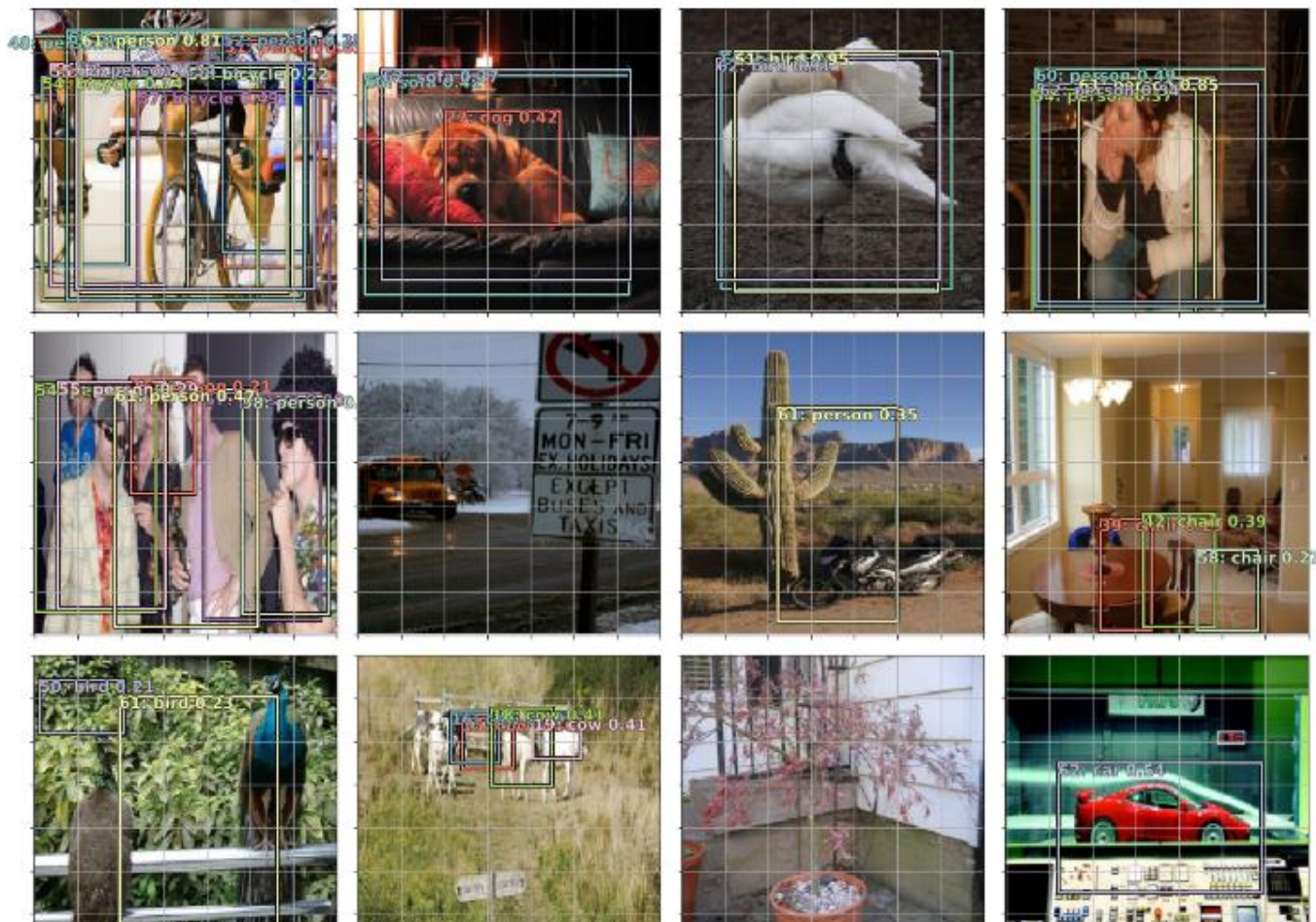
```
In [157]: learn.freeze_to(-2)
learn.fit(lrs/2, 1, cycle_len=4, use_clr=(20,8))
```

```
HBox(children=(IntProgress(value=0, description='Epoch', max=4), HTML(value='')))
```

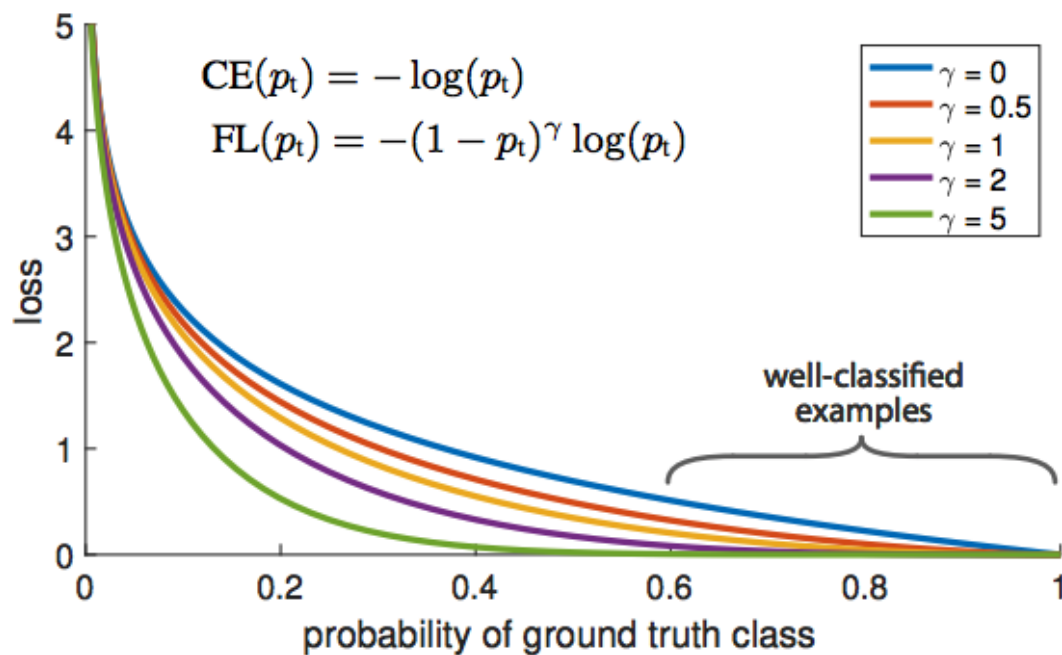
epoch	trn_loss	val_loss
0	45.136695	49.028358
1	42.55799	42.789735
2	38.6295	38.51572
3	34.733755	37.201217

```
Out[157]: [array([37.20122])]
```

创建更多的anchors boxes



focal loss



这样网络的训练会更加注重难训练的部分（小目标），而降低容易训练部分的loss占比（背景）

focal loss

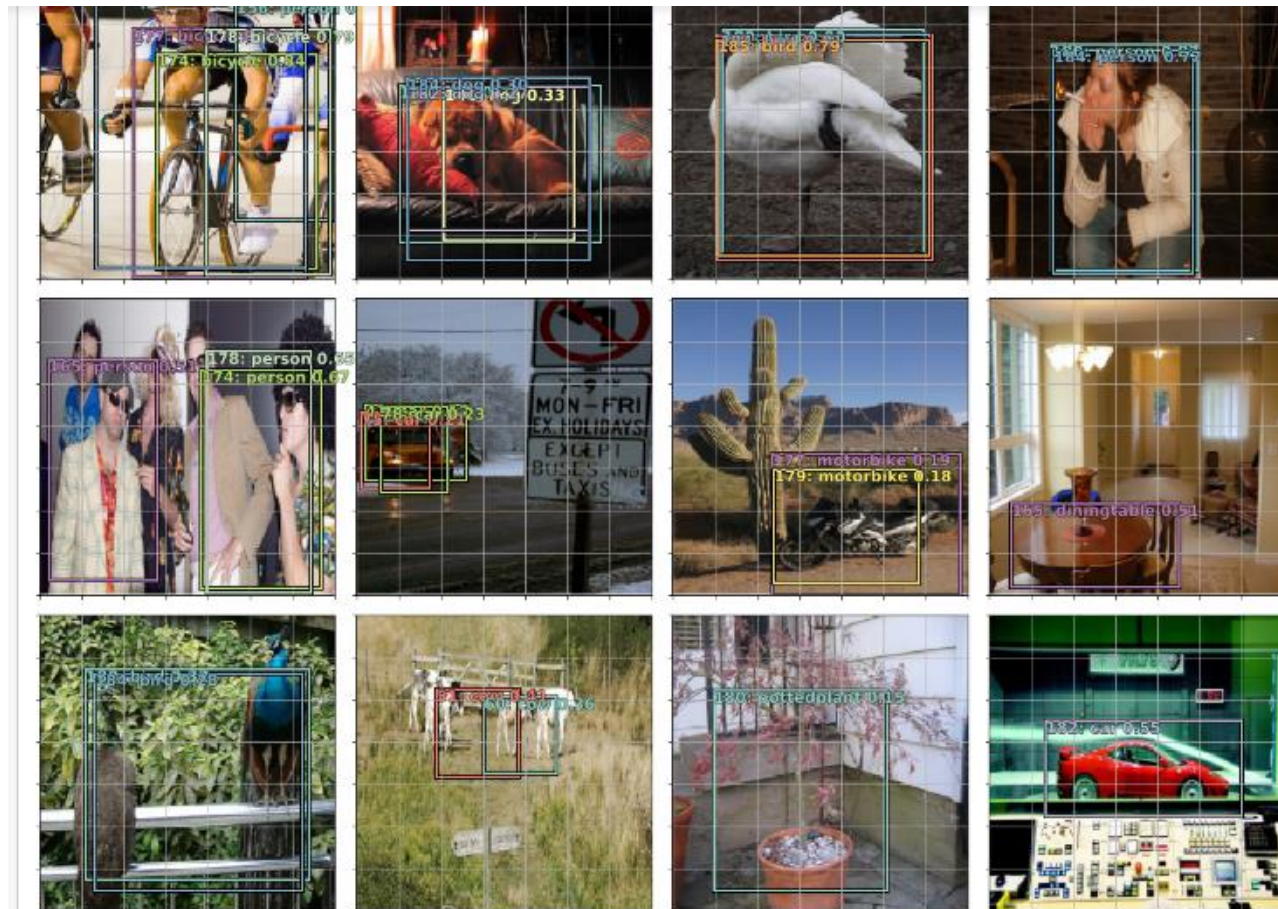


```
In [161]: class FocalLoss(BCE_Loss):  
            def get_weight(self, x, t):  
                alpha, gamma = 0.25, 1  
                p = x.sigmoid()  
                pt = p*t + (1-p)*(1-t)  
                w = alpha*t + (1-alpha)*(1-t)  
                return w * (1-pt).pow(gamma)  
  
            loss_f = FocalLoss(len(id2cat))
```

```
In [167]: learn.freeze_to(-2)  
          learn.fit(lrs/4, 1, cycle_len=10, use_clr=(20,10))  
  
HBox(children=(IntProgress(value=0, description='Epoch', max=10), HTML(value='')))  
  
epoch      trn_loss  val_loss  
0          7.411913  9.913611  
1          7.562427  9.987381  
2          7.455078  9.86398  
3          7.297633  9.754575  
4          7.058716  9.66555  
5          6.803243  9.558953  
6          6.577886  9.667835  
7          6.361771  9.602401  
8          6.156289  9.542483  
9          6.038056  9.51547
```

```
Out[167]: [array([9.51547])]
```

focal loss



读论文的小贴士



$$\text{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t).$$

γ	α	AP	AP ₅₀	AP ₇₅
0	.75	31.1	49.4	33.0
0.1	.75	31.4	49.9	33.1
0.2	.75	31.9	50.7	33.4
0.5	.50	32.9	51.7	35.2
1.0	.25	33.7	52.0	36.2
2.0	.25	34.0	52.5	36.5
5.0	.25	32.2	49.6	34.8

(b) Varying γ for FL (w. optimal α)

在阅读论文过程中，可以看到许多优秀的文章未必是编写了成百上千的代码，其突破可能只是很小的一些地方，一句代码、几个公式等等，但它们却极大的优化了实验结果。对这些文章，重点要关注其中的这些亮点。例如关于focal loss的这篇论文中就只需关注这个公式的相关和它的推导过程，以及各参数的实验结果，这个方法能极大提高你阅读文献的效率。

Non Maximum Suppression(NMS)



看到有好多box都对应同一目标且相互几乎重叠，

使用NMS将相近的box删除取概率最高的，由于这里不涉及深度学习的技巧，就仅贴出图片。



目标检测发展历史



There's a rich history to this idea



Scalable Object
Detection using Deep
Neural Networks
(Multibox)

Faster R-CNN: Towards
Real-Time Object
Detection with Region
Proposal Networks*

You Only Look Once:
Unified, Real-Time
Object Detection

SSD: Single Shot
MultiBox Detector

Focal Loss for Dense
Object Detection
(RetinaNet)

实验常见问题与解决方案

一 代码更新后出现错误

“ 'list' object has no attribute 'data' ”在model.py中

```
def validate(stepper, dl, metrics):  
    batch_cnts, loss, res = [], [], []  
    stepper.reset(False)  
    with no_grad_context():  
        for (*x, y) in iter(dl):  
            preds, l = stepper.evaluate(VV(x), VV(y))  
            if isinstance(x, list): batch_cnts.append(len(x[0]))  
            else: batch_cnts.append(len(x))  
            loss.append(to_np(l))  
            res.append([f(preds.data, y.data) for f in metrics])  
    return [np.average(loss, 0, weights=batch_cnts)] +  
           list(np.average(np.stack(res), 0, weights=batch_cnts))
```

实验常见问题与解决方案

解决方法：

```
if is_listy(y) :
```

```
res.append([f(preds.data, y) for f in metrics])
```

```
else :
```

```
res.append([f(preds.data, y.data) for f in metrics])
```

实验常见问题与解决方案

二 Input type(CUDAFloatTensor) and weight type(CPUFloatTensor) should be the same

```
In [105]: x, y = next(iter(md.val_dl))  
          #x, y = V(x).cpu(), V(y)  
          x, y = V(x), V(y)
```

```
In [106]: for i, o in enumerate(y): y[i] = o.cuda()#o.cpu()  
          learn.model.cuda()#.cpu()
```

代码中修改.cpu()部分改成.cuda()

谢谢！



上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

上海交通大学

