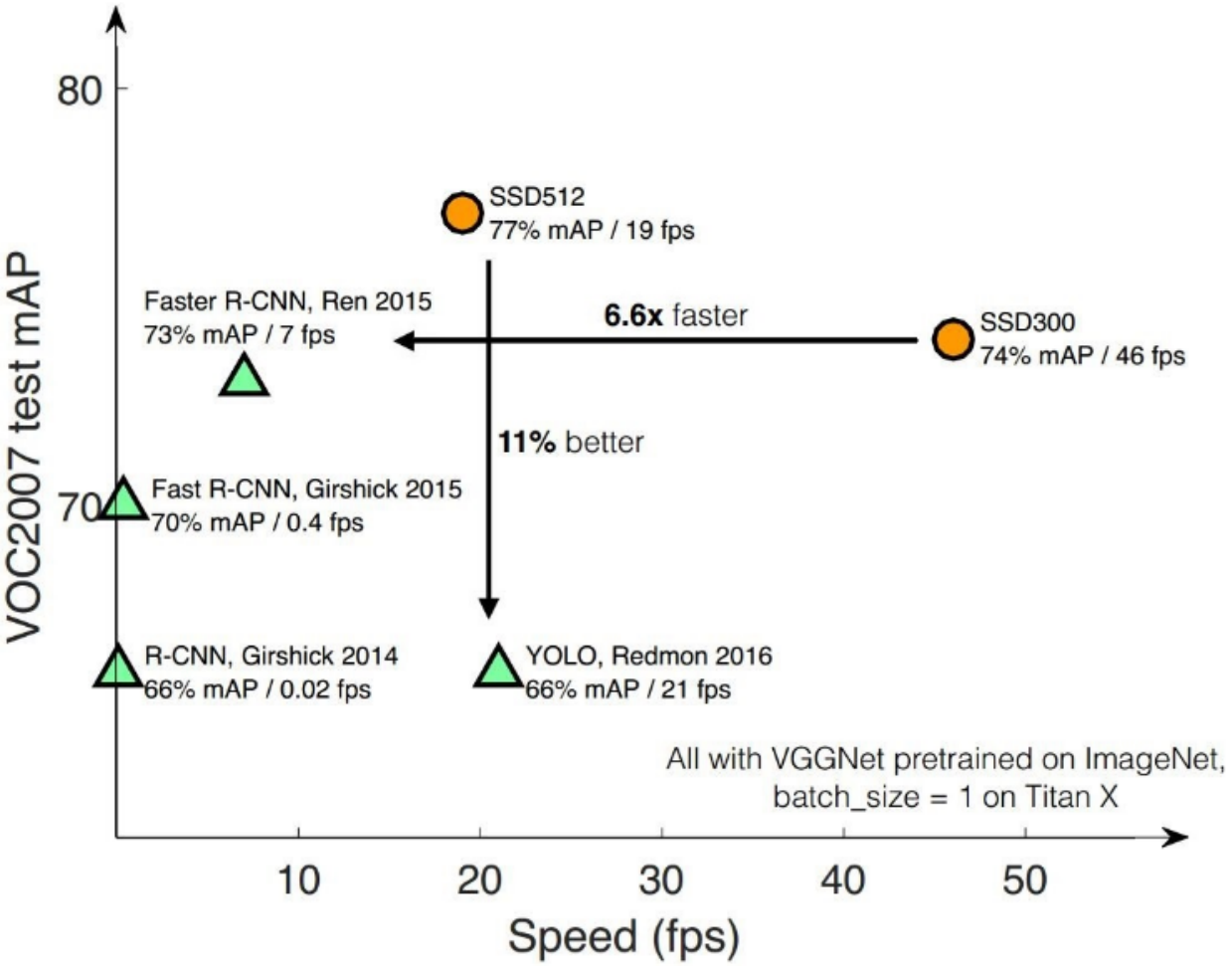
SSD: Single Shot MultiBox Detector

第一部分 绪论

1.1 什么是SSD?

SSD，全称Single Shot MultiBox Detector，是Wei Liu在ECCV 2016上提出的一种目标检测算法，截至目前是主要的检测框架之一，相比Faster RCNN有明显的速度优势，相比YOLO又有明显的mAP优势。

1.2 为什么使用SSD?



1.3 SSD的来龙去脉

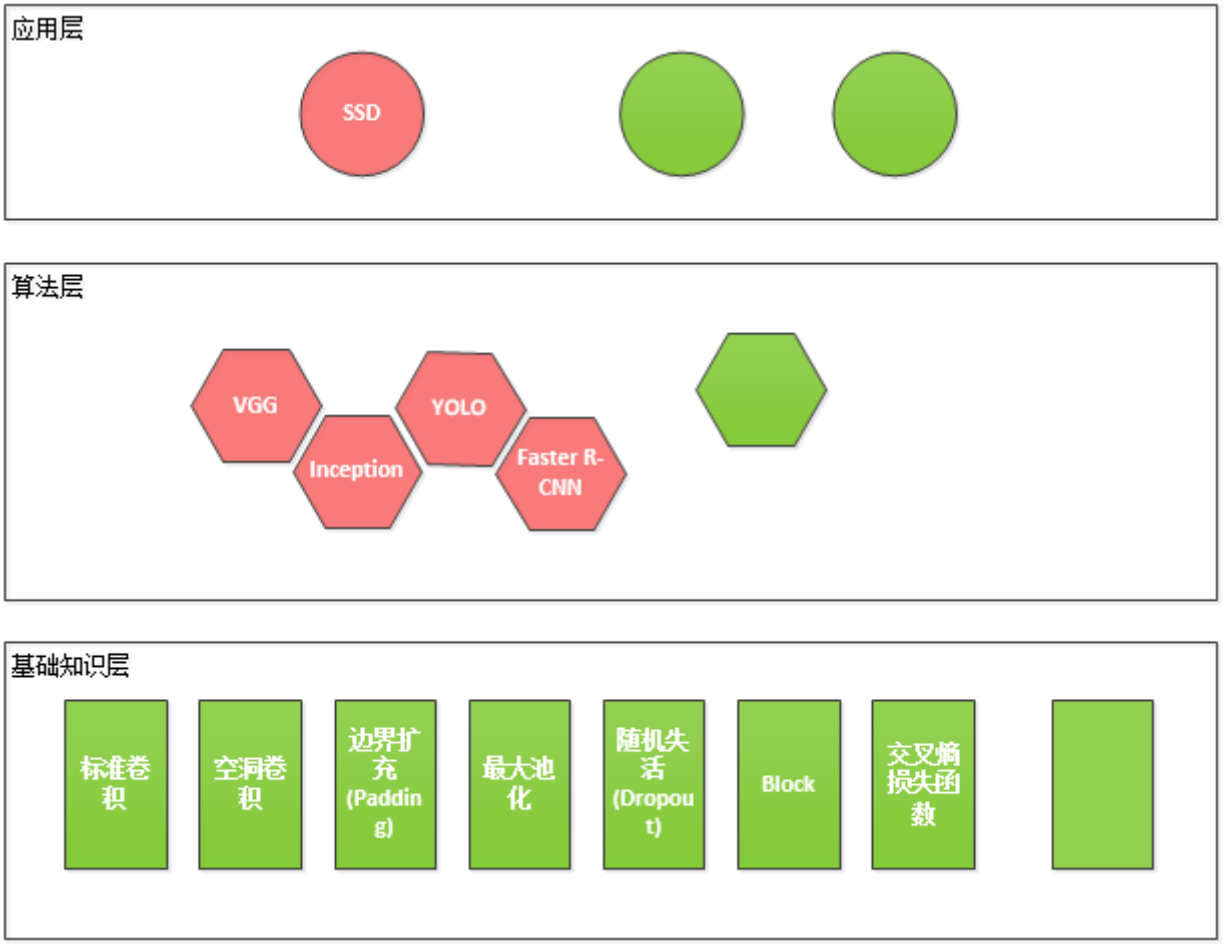
从Faster R-CNN中继承了anchor机制。(详见2.2)

从YOLO中继承了将detection转化为regression的思路，一次即可完成目标检测。（详见2.3)

加入针对对多个尺度feature map的预测。（详见2.1）

1.4 与以往知识体系的结合：

标准卷积、边界扩充(Padding)方式（"SAME","VALID"）、空洞卷积、最大池化、Dropout、Block、Inception网络、交叉熵损失函数



1.4.1 标准卷积：

tf.nn.conv2d(input, filter, strides, padding, use_cudnn_on_gpu=None, name=None)

除去name参数用以指定该操作的name，与方法有关的一共五个参数：

第一个参数input：指需要做卷积的**输入图像**，它要求是一个Tensor，具有[batch, in_height, in_width, in_channels]这样的shape，具体含义是[训练时一个batch的图片数量, 图片高度, 图片宽度, 图像通道数]，注意这是一个4维的Tensor，要求类型为float32和float64其中之一

第二个参数filter：相当于CNN中的**卷积核**，它要求是一个Tensor，具有[filter_height, filter_width, in_channels, out_channels]这样的shape，具体含义是[卷积核的高度，卷积核的宽度，图像通道数，卷积核个数]，要求类型与参数input相同，有一个地方需要注意，第三维in_channels，就是参数input的第四维

第三个参数strides：卷积时在图像每一维的**步长**

第四个参数padding：边界扩充(Padding)的方式，string类型的量，只能是"SAME","VALID"其中之一，这个值决定了不同的卷积方式

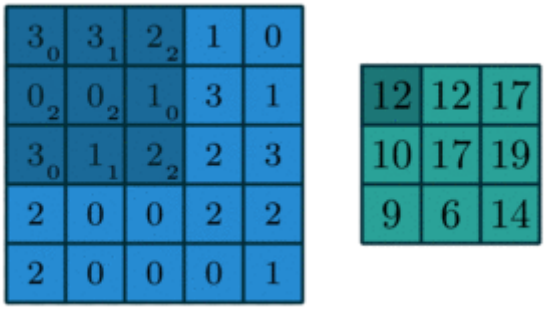
第五个参数：use_cudnn_on_gpu:bool类型，是否使用cudnn加速，默认为true

Return:结果返回一个Tensor，即我们常说的feature map，shape仍然是[batch, height, width, channels]这种形式。

1.4.2 边界扩充(Padding)方式:

Padding="VALID"：without padding

△卷积核为3、步幅为1和Padding = “VALID”的二维卷积结构：



以上为Padding="VALID"时的卷积，此时**输出图像的大小**如下：

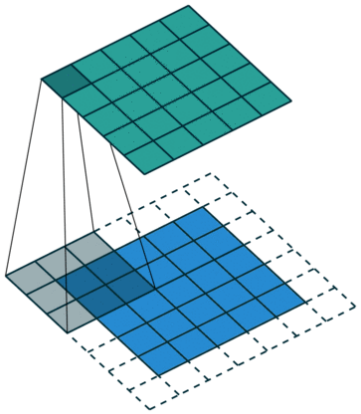
out_width=(in_width-filter_width)/stride+1

out_height=(in_height-filter_height)/stride+1

Padding="SAME"：

输入和输出图像保持相同的大小。

SAME:



out_width=(in_width+2*pad-filter_width)/stride+1

out_height=(in_height+2*pad-filter_height)/stride+1

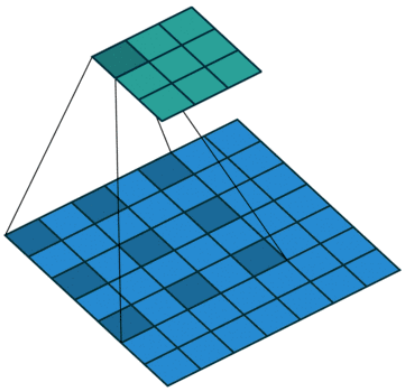
除不尽的情况下怎么办？

对于padding=’VALID’, 166 = math.floor(500/3)　　向下取整

对于padding=’SAME’, 167 =math.ceil(500/3)　　向上取整

1.4.3 空洞卷积：

空洞卷积（atrous convolutions）又名扩张卷积（dilated convolutions），向卷积层引入了一个称为 “扩张率(dilation rate)”的新参数，该参数定义了卷积核处理数据时各值的间距。



空洞卷积的好处是不做pooling损失信息的情况下，加大了感受野，让每个卷积输出都包含较大范围的信息。

1.4.4 最大池化：

取区域内所有参数的最大值。

PS：前向传播中不仅要计算pool区域内的最大值，还要记录该最大值所在输入数据中的位置，目的是在反向传播中，把梯度值传到对应最大值所在的位置。

out_width=(in_width-filter_width)/stride+1

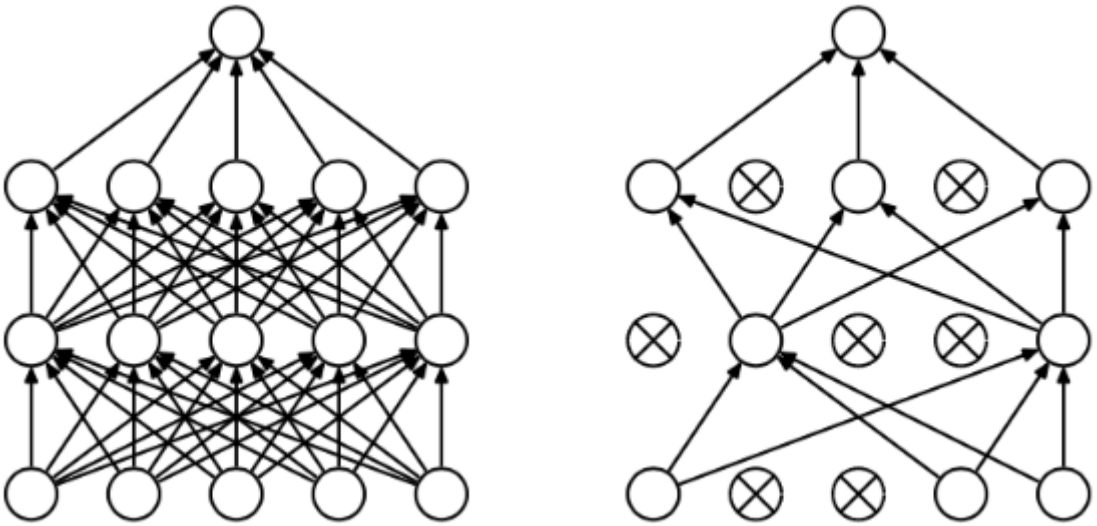
out_height=(in_height-filter_height)/stride+1

池化操作一般向上取整。

1.4.5 Dropout：

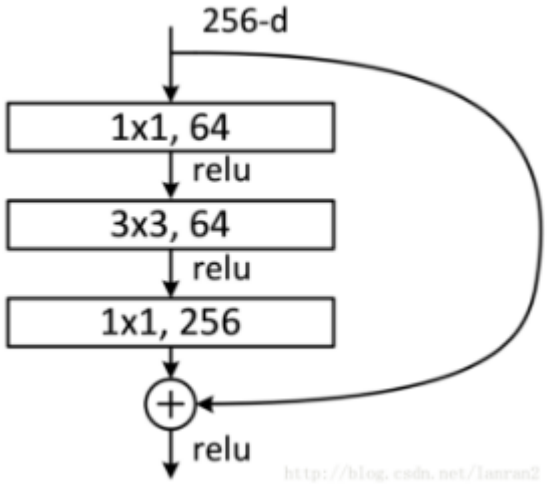
Dropout是指在深度学习网络的训练过程中，对于神经网络单元，按照一定的概率将其暂时从网络中丢弃。注意是暂时，对于随机梯度下降来说，由于是随机丢弃，故而每一个mini-batch都在训练不同的网络。

每次做完dropout，相当于从原始的网络中找到一个更瘦的网络，从而防止了过拟合，同时减少了训练的时间。



1.4.6 Block:

CNN网络最初诞生的时候结构比较简单，都是几个卷积层堆叠一下。但是微软的Resnet和谷歌的Inception系列网络把CNN带到一个设计各种block反复调用的时代。比起传统的CNN网络，新的block设计能够在简化运算的同时保持甚至提高网络的泛化能力。下图是Resnet中的“bottleneck design”。



1.4.7 Inception网络：

Inception中在3*3卷积或者5*5卷积计算之前先进行1*1卷积。通过这种方式能够降低维度，加快计算速度。从而加深网络结构。

1.4.8 交叉熵损失函数：

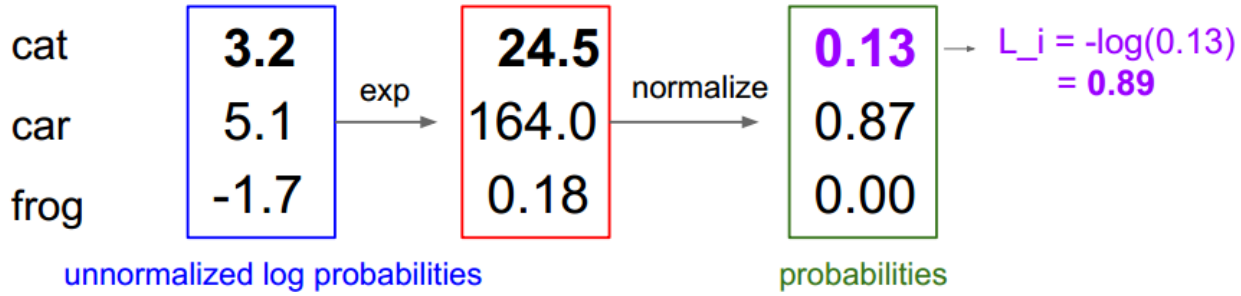
1.从每个类别的得分出发，将这些得分作为e的指数。每个项除以这些项的和，得出每个类的概率。

$$P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}$$

2.通过使用交叉熵，正确的概率越小，Lost越大

$$L_i=-\log P(Y=y_i|X=x_i)$$

通过这个过程，最大化概率对数，最小化负的概率对数

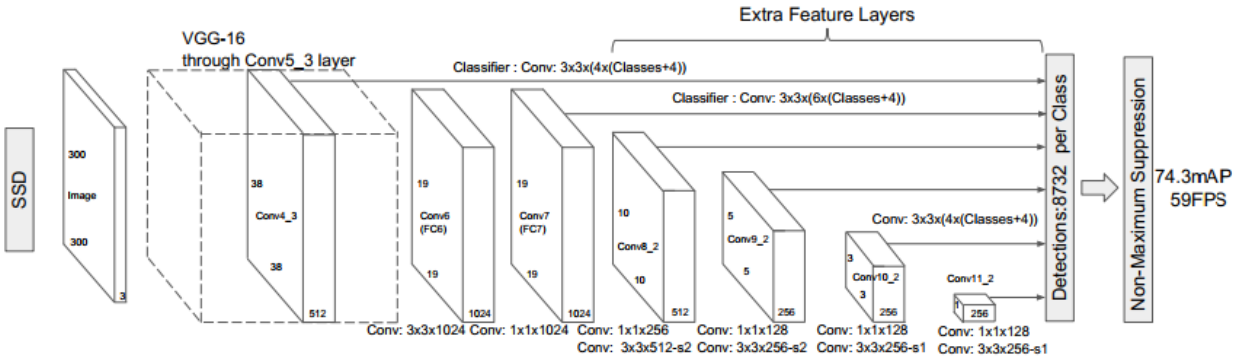


第二部分 基本原理

SSD在每张特征图上设置一系列不同大小和长宽比的框（被称为Default Boxes，或Anchors），在预测阶段产生每个anchor的类别得分和位置偏移量。筛选掉类别得分小于某个阈值的anchors，和相互重叠的anchors（只保留类别得分最大的）。

2.1 SSD的网络结构

对应ssd_vgg_300.py中ssd_net函数。



Block	Layer	kernel	note
1	1,2	3*3*64	VGG 3*3 convolution 每个 block 后加一个 2*2 maxpool（最后一个 block 后加 3*3，stride=1 的 maxpool）
2	3,4	3*3*128	
3	5,6,7	3*3*256	
4(box)	8,9,10	3*3*512	
5	11,12,13	3*3*512	
6	14	3*3*1024 Rate=6	Additional SSD 每个 block 后加 dropout
7(box)	15	1*1*1024	
8(box)	16	1*1*256	1*1 and 3*3 convolutions, Padding=valid,前两个 stride=2
	17	3*3*512	
9(box)	18	1*1*128	
	19	3*3*256	
10(box)	20	1*1*128	
	21	3*3*256	
11(box)	22	1*1*128	
	23	3*3*256	

input_shape:300*300

feat_shape=[(38,38),(19,19),(10,10),(5,5),(3,3),(1,1)]

对于每张特征图，采用3×3卷积计算anchors的四个位置偏移量和21个类别置信度。比如block7，anchors（default boxes）数目为6，每个anchors包含4个偏移位置和21个类别置信度（4+21）。因此，block7的最后输出为(19*19)*6*(4+21)。

为什么使用多个feature map?

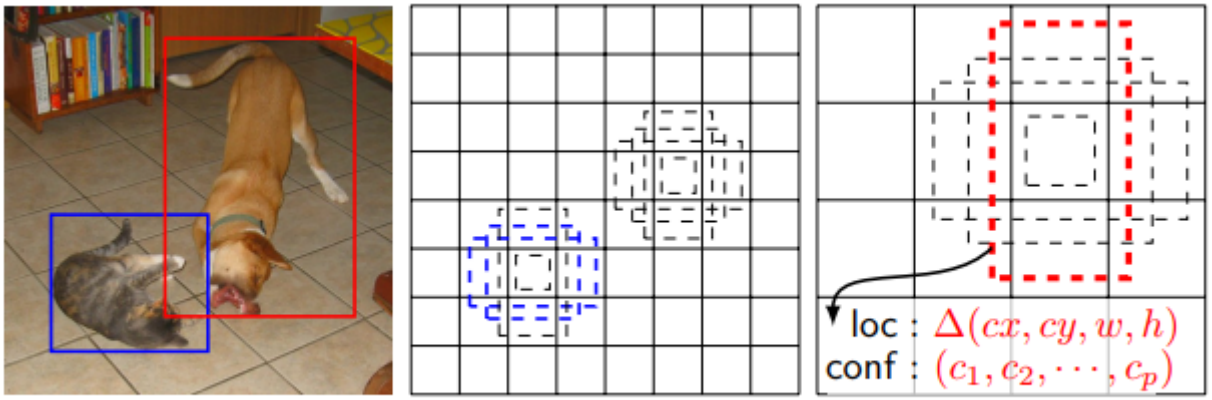
这些feature map的大小逐渐减小，feature map上参数的感受野逐渐增大。因此通过多个feature map可以进行多尺度预测。

2.2 生成anchors

对应ssd_vgg_300.py中ssd_anchor_one_layer函数。

把feature_map根据feat_shape进行划分。比如第一层feature_map，feat_shape=38*38，就将整个feature_map划分成38*38个cell。

在每个feat_shape的每个cell上，以cell的中心作为anchors的中心，设置大小或长宽比不同的4-6个anchors。一五六层为4个，二三四层为6个。



(a) Image with GT boxes (b) 8 × 8 feature map (c) 4 × 4 feature map

anchors的位置由X,Y,W,H确定。

（X,Y）为anchors的中心坐标，等于feature map上每个cell的中心。

W、H由feature map对应的anchor_sizes（简称S_k）和anchor_ratios（简称R）确定。

1、ratio=1的两个anchors。

#W=H=S_k

#W=H=sqr(S_k*S_(k+1))

2、根据anchor_ratios生成anchors。

```
anchor_ratios=[[2, .5],                          //feature map1
               [2, .5, 3, 1./3],                 //feature map2
               [2, .5, 3, 1./3],
               [2, .5, 3, 1./3],
               [2, .5],
               [2, .5]],
```

对于feature map对应的每个anchor_ratio（R）

#W=S_k*sqr(R),H=S_k/sqr(R)

比如feature map1存在R=2和R=0.5两种情况

X,Y,W,H：X,Y即feature map每个cell中心点的位置。 W,H是根据ratio和size设置的宽和高。

如何得到S_k：

a.论文里

$$s_k = s_{\min} + \frac{s_{\max} - s_{\min}}{m-1}(k-1), \quad k \in [1, m]$$

anchor_size_bounds=[0.15, 0.90],

b.代码里

```
anchor_sizes=[(21., 45.),
              (45., 99.),
              (99., 153.),
              (153., 207.),
              (207., 261.),
              (261., 315.)],
```

为什么使用anchors？

通过anchors预测目标的类别及位置。

2.3 编码解码

anchors坐标：

$$d = (d^{cx}, d^{cy}, d^{w}, d^{h})$$

真实边界框的位置坐标：

$$g = (g^{cx}, g^{cy}, g^{w}, g^{h})$$

编码对应ssd_common.py中tf_ssd_bboxes_encode函数

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^{w} \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^{h}$$

$$\hat{g}_j^{w} = \log\left(\frac{g_j^{w}}{d_i^{w}}\right) \qquad \hat{g}_j^{h} = \log\left(\frac{g_j^{h}}{d_i^{h}}\right)$$

相对于anchors的x坐标偏移，除以anchors的宽度。即边界框距anchors左边的比例。

以及真实边界框距anchors的上边的比例。

真实边界框相对于anchors的宽度高度的比例的对数。

解码对应ssd_common.py中tf_ssd_bboxes_decode函数

及np_methods.py中ssd_bboxes_decode函数

Decode：根据预测的位置偏移localisations(简称l) 对anchor进行调整，得到边界框的真实位置 b。

$$g_j^{cx} = d_i^w \hat{g}_j^{cx} + d_i^{cx} \quad g_j^{cy} = d_i^h \hat{g}_j^{cy} + d_i^{cy}$$

$$g_j^w = d_i^w exp(\hat{g}_j^w) \quad g_j^h = d_i^h exp(\hat{g}_j^h)$$

在代码中，编码后对坐标除以prior_scaling，解码前对坐标乘以prior_scaling

prior_scaling=[0.1, 0.1, 0.2, 0.2]

2.4 预测每个anchor的类别置信度，或位置偏移量

对应ssd_vgg_300.py中ssd_multibox_layer(end_points[layer],num_classes,

anchor_sizes[i],anchor_ratios[i],

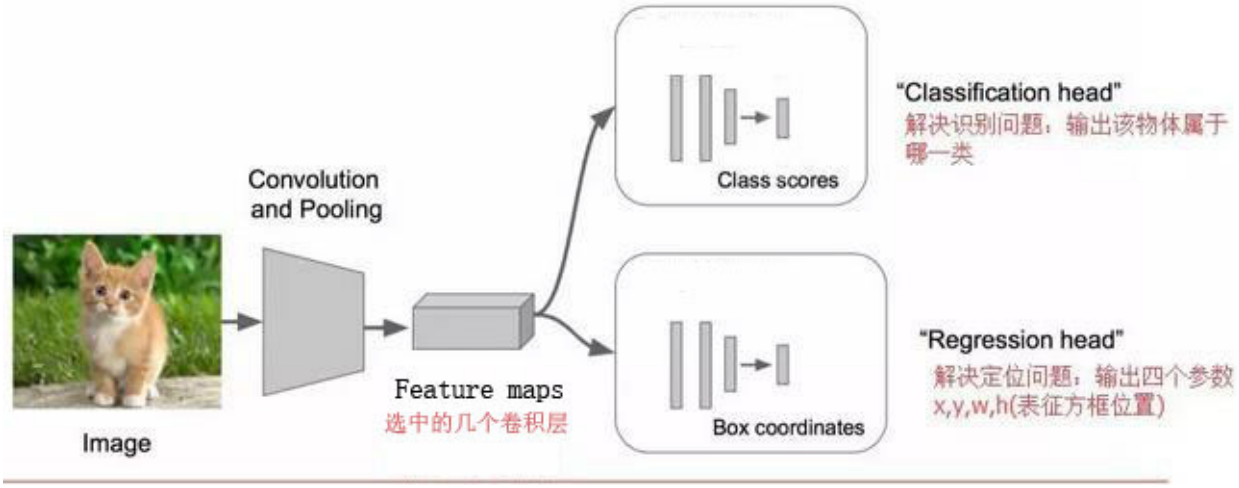
normalizations[i])函数。

输入：

end_points[layer]：特征层　num_classes:类别数

num_anchors = len(sizes) + len(ratios)

if normalization > 0:对end_points[layer]进行l2正则化



SSD在每个feature_layer后面加一个3*3*p的卷积层，用来计算每个anchor的**类别置信度，或位置偏移量**。（p为通道数）

2.4.1 预测类别置信度

class_pred = slim.conv2d(net, num_cls_pred, [3,3], activation_fn=None, scope='conv_cls')

其中num_cls_pred是卷积核的数目=num_anchor*num_classes（对于feature map1为4*21）

对于feature map1，本来大小为1*38*38，卷积后大小为1*38*38*84，将其调整为1*38*38*4*21，即每个cell对应4个先验框，每个框进行21个类别的评分。

通过softmax根据对这21个类别的评分，判断每个类别的概率，作为predictions。

2.4.2 预测位置偏移量

预测的是每个cell中所有的anchor的偏移量，作为locations。

loc_pred = slim.conv2d(net, num_loc_pred, [3, 3], activation_fn=None, scope='conv_loc')

其中num_loc_pred是卷积核的数目，num_anchor*4（对于feature map1为4*4）

对于feature map1，本来大小为38*38，卷积后大小为1*38*38*16，将其调整为1*38*38*4*4，即第一个4对应4个先验框，第二个4对应4个位置偏移量的预测。

将训练数据编码后的gclasses, glocalisations,gscores，和预测的gscores,localisations输入ssd_losses，计算损失函数。

2.5 正样本与负样本的选择

对应ssd_vgg_300.py中ssd_losses(logits, localisations,
gclasses, glocalisations, gscores,
match_threshold=0.5,
negative_ratio=3.,
alpha=1.,
label_smoothing=0.,
device='/cpu:0',
scope=None)函数

参数：每一个anchors预测的类别得分、预测的位置偏移量、匹配类别、匹配位置、匹配得分

所有匹配得分gscore>0.5的anchor作为正样本，已知对各个类别的预测得分和正确类别，通过学习让正确类别的得分尽可能大。

负样本的匹配得分gscore<0.5，所以作为背景，通过学习让识别为背景的能力变强。

2.5.1 选取IOU>0.5的预测框作为正样本，并计算正样本的数量

每个anchor匹配到一个ground truth，得到该anchor的labels,localizations,scores

(labels:与ground truth类别相同)(localizations:ground truth相对于该anchor的位置偏移)

(scores：该anchor与ground truth的IOU)

（1）将每个anchor匹配到一个ground truth：

论文中的匹配原则：

原则1：对于图片中每个ground truth，找到与其IOU最大的anchor，该anchor与其匹配。

原则2：并计算两者间的jaccard overlap，若大于某个阈值（一般是0.5），那么该anchor也与这个ground truth进行匹配。

代码中实际使用的原则：

每个anchor匹配到与其IOU最大的ground truth。

对应ssd_common.py中tf_ssd_bboxes_encode_layer.body)函数

匹配过程：

对于每一张图片的每一张feature map，执行以下操作：

1.初始化：设置feat_scores记录每个anchor与所有ground truth匹配的最高得分，设置feat_labels记录每个anchor匹配的ground truth对应的label。匹配前将两者都初始化为0。

若feature map shape=[38,38],则对应的feat_scores和feat_labels的shape为[38*38*4](与anchors一一对应)。

2.对于每个ground truth box，判断每个anchors与该box的IOU是否大于与之前box的IOU（保存在feat_scores中），若是将mask对应位置设置为1。（mask与feat_scores同样大小，每个位置对应一个anchor）。判断新的feat_scores是否大于0.5，若小于0.5将mask设置为0。

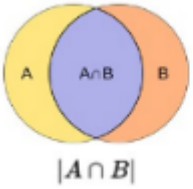
3.若anchors对应的mask=1，在feat_labels上对应的位置将该anchor的label，记录为正在匹配的ground_truth box的label。原label被覆盖。

对于mask=0的anchor，其feat_labels记录的label保持不变。

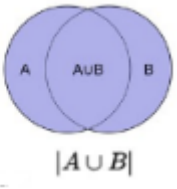
4.如果存在多个ground truth，对于每个ground truth执行一次2、3步骤。

计算jaccard overlap(IOU)：

对应ssd_common.py中tf_ssd_bboxes_encode_layer.jaccard_with_anchors(bbox)函数



$|A \cap B|$



$|A \cup B|$

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

（2）根据匹配后每个anchor对越的gscore选择正样本

对输入数据进行整理后，每一组数据包含一个anchor的信息：logits-预测的类别得分(21)，gclasses-匹配的类别(1)，gscores-匹配的类别得分(1)，localisations-预测的位置偏移(4)，glocalisations-匹配的位置偏移(4)

匹配的类别得分>0.5的anchors的作为正样本。

统计匹配的类别得分>0.5的anchors数量，作为正样本数量。

2.5.2 按照置信度误差（预测背景的置信度越小，误差越大）进行降序排列，选取误差的较大的top-k作为训练的负样本(k:训练负样本数量)-hard negative mining

为了保证正负样本尽量平衡，SSD采用了hard negative mining，就是对负样本进行抽样，抽样时按照置信度误差（预测背景的置信度越小，误差越大）进行降序排列，选取误差的较大的top-k作为训练的负样本，以保证正负样本比例接近1:3

如果-0.5<gscores<0.5,作为负样本。统计全部负样本的数量，记为max_neg_entries。

根据logits（预测的类别得分）通过softmax得到预测的每个类别概率。因为已知负样本为背景，要选出背景概率最小的作为训练负样本。将gscores>0.5样本的背景概率设置为1。然后按照背景概率从小到大的顺序选择k个负样本。（k为训练负样本的数量n_neg，设置n_neg为正样本的三倍+ batch_size）

```
n_neg = tf.minimum(n_neg, max_neg_entries)
```

2.6 损失函数

对应ssd_vgg_300.py中ssd_losses函数

在训练阶段，首先通过预测网络计算出每个anchor的位置偏移量和每个类别的概率。

损失函数由三部分组成：

（1）预测的位置偏移量与匹配偏移量之间的差值作为位置误差。

（2）根据交叉熵损失函数，正确类别的概率越小，正样本类别误差越大。

（3）根据交叉熵损失函数，正确预测为背景的概率越小，负样本类别误差越大。

2.6.1 位置误差

$$L_{loc}(x,l,g)=\sum_{i\in Pos}^{N}\sum_{m\in\{cx,cy,w,h\}}x_{ij}^{k}\text{smooth}_{\text{L1}}(l_i^m-\hat{g}_j^m)$$

2.6.2 类别误差

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

2.6.3 获得位置误差和类别误差的加权和

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

N is the number of matched default boxes

2.7 数据集的制作与数据的读取

2.7.1 将数据从pascal_voc格式转化为tfrecords

对应pascalvoc_to_tfrecords.py文件中run(dataset_dir, output_dir)函数：

读取xml文件，进行shuffle。

设置TFRecordWriter的output_filename。

根据每个xml文件，找到其对应的jpg文件名(相同) 。

对应_process_image函数：

对图像数据，xml文件中的目标框信息进行提取

对应_convert_to_example函数：

将图像数据及其对应的目标框信息列表打包成一个example

对应_add_to_tfrecord函数：

通过tfrecord_writer.write将exmple转化为tfrecords

2.7.2 制作数据集slim.dataset.Dataset

对应pascalvoc_common.py文件中get_split函数

```
    return slim.dataset.Dataset(
                data_sources=file_pattern,
                reader=reader,
                decoder=decoder,
                num_samples=split_to_sizes[split_name],
                items_to_descriptions=items_to_descriptions,
                num_classes=num_classes,
                labels_to_names=labels_to_names)
 """
    输入：data_sources = file_pattern = tf_records/voc_2007_train_*.tfrecord
          reader = tf.TFRecordReader
          decoder
=slim.tfexample_decoder.TFExampleDecoder(keys_to_features,items_to_handlers)
                    将数据中的特征转化为固定的内容描述项
          num_samples:{'train': 5011,'test': 4952,} [train]
          items_to_descriptions:内容描述项
（image,shape,object/bbox,object/label）
          num_classes = NUM_CLASSES = 20
          labels_to_names: labels_to_names =
dataset_utils.read_label_file(dataset_dir)
  """
```

2.7.3 设置dataset provider，通过dataset provider取出数据（image，shape，labels，bboxes.）

```
provider = slim.dataset_data_provider.DatasetDataProvider(
                    dataset,
                    num_readers=FLAGS.num_readers,
                    common_queue_capacity=20 * FLAGS.batch_size,
                    common_queue_min=10 * FLAGS.batch_size,
                    shuffle=True)
[image, shape, glabels, gbboxes] = provider.get(['image', 'shape',
'object/label', 'object/bbox'])
```

一次读取到一张图片对应的数据

对读取的数据进行预处理和编码，使其数据格式保持一致，之后输入加载队列

2.7.4 设置加载队列

```
r = tf.train.batch(
                tf_utils.reshape_list([image, gclasses, glocalisations,
gscores]),
                batch_size=FLAGS.batch_size,
                num_threads=FLAGS.num_preprocessing_threads,
                capacity=5 * FLAGS.batch_size)
```

数据需要是shapes_is_all_fully_defined

2.7.5 设置预加载队列

```
batch_queue = slim.prefetch_queue.prefetch_queue(
                tf_utils.reshape_list([b_image, b_gclasses, b_glocalisations,
b_gscores]),
                capacity=2 * deploy_config.num_clones)
```

2.8 数据扩增

2.8.1 随机截取图像信息

MIN_OBJECT_COVERED=0.25 图像的裁剪区域必须包含所提供的任一边界框的至少 min_object_covered 的内容

t_ratio_range=（0.6，1.67）图像的裁剪区域的宽高比在（0.6，1.67）之间

```
#使用tf.image.sample_distorted_bounding_box函数随机选取裁剪区域，返回bbox_begin,
bbox_size用于裁剪图像  distort_bbox用于记录裁剪后图像在原图中的坐标
#使用tf.slice(image, bbox_begin, bbox_size)函数裁剪图像
#得到目标框在裁剪后图像中的归一化坐标
#筛选掉与裁剪后图像重叠部分不足0.5的目标框
```

截取后将图像调整到固定的大小-双线性插值法

2.8.2 以一定概率左右翻转

采样得到的patch，其大小为原始图像的[0.1, 1]，宽高比在0.6与1.67之间。当groundtruth box的中心在采样的patch中时，保留重叠部分。经过上述采样之后，将每个采样的patch resize到固定大小，并以0.5的概率对其水平翻转。

2.8.3 随机颜色扰动

随机调整亮度、饱和度、色调、对比度

2.8.4 白化

image = tf_image_whitened(image, [_R_MEAN, _G_MEAN, _B_MEAN])

即减去各个channel的均值。

第三部分：训练与预测过程

3.1 训练过程

（1） 选择GPU参数：GPU块数，是否使用CPU（deploy_config）

replica: 使用多机训练时， 一台机器对应一个replica——复本

tower：使用多GPU训练时， 一个GPU上对应一个tower。

clone: 由于tensorflow里多GPU训练一般是每个GPU上都有完整的模型，各自forward，得到的梯度交给CPU平均然后统一backward，每个GPU上的模型也叫做一个clone。所以clone与tower指的是同一个东西。

parameter server: 多机训练时计算梯度平均值并执行backward操作的参数服务器，功能类比于单机多GPU（也叫单机多卡）时的CPU。（未考证， TODO）

worker server：功能类比于单机多卡中的GPU。（未考证， TODO）

（2） 制作数据集

对应pascalvoc_common.py文件中get_split函数

（3） 搭建ssd_net，并生成anchors

对应preprocessing/ssd_vgg_300.py中SSDNet(object)函数和anchors(image_shape)函数

（4） 选择预处理函数

对应preprocessing/ssd_vgg_preprocessing.py中preprocess_for_train函数

（5） 设置dataset provider，通过dataset provider取出数据（image，shape，labels，bboxes.）

一次取出10 * batch_size数据

（6）对图像进行预处理，得到新的image，glabels，gbboxes

（7）对glabels，gbboxes，ssd_anchors进行编码，得到训练数据的gclasses，glocalisations，以及最大匹配anchor的gscores

（8） 制作预加载队列（batch_queue）

（9）将图像输入预测网络，求解其feature map上的每个位置的num_anchors*21个类别得分(gscores)，num_anchors*4个位置偏移量(localisations)。

（10）将训练数据编码后的gclasses, glocalisations,gscores，和预测的gscores,localisations输入ssd_losses，计算损失函数

（11）设置优化器

```
optimizer = tf.train.AdamOptimizer(
            learning_rate,
            beta1=flags.adam_beta1,
            beta2=flags.adam_beta2,
            epsilon=flags.opt_epsilon)
```

（12）使用优化器最小化损失函数

3.2 预测过程

对应notebook/ssd_notebook

#预处理

（1）中心化处理，并将图片通过双线性插值调整到固定的尺寸。

（2）生成anchors [y,x,h,w]。

#预测

（3）计算每个anchors的predictions(类别置信度)和localisations(位置偏移量)。

#后处理

（4）根据localisations对anchors坐标进行解码，得到预测框匹配的位置偏移。

（5）选择类别置信度>select_threshold的边界框。

（6）通过NMS（非极大值抑制）算法过滤掉重叠度较大的预测框

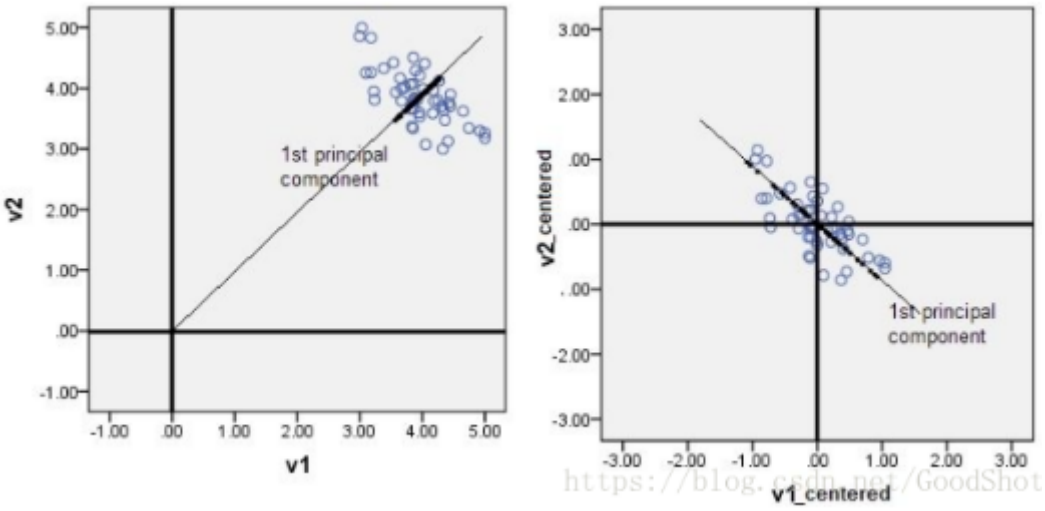
3.2.1 中心化处理，并将图片通过双线性插值调整到固定的尺寸。

得到image_pre(300*300)和bbox_img([0,0,1,1])。

对应preprocessing/ssd_vgg_preprocessing.py中preprocess_for_eval函数

中心化处理：

中心化后能够更好的概括图像的特征。



在ssd300中，将输入图像调整为300*300像素大小。

3.2.2 生成anchors [y,x,h,w]。

3.2.3 计算每一个anchor的predictions(类别置信度)和localisations(位置偏移量)。

通过.ckpt文件使用训练好的模型进行预测。

3.2.4 根据localisations对anchors坐标进行解码，得到边界框的真实位置。

对应np_methods.py中ssd_bboxes_decode函数

一般在解码后还需要做clip，防止预测框位置超出图片。

3.2.5 选择类别置信度>select_threshold的default boxes。

对应np_methods.py中ssd_bboxes_select_layer函数

将predictions和localisations的形状分别调整为[1,-1,21]和[1,-1,4]

如果select_threshold = 0，选择每个数据置信度最大的类别：

判断所有数据的类别，保存每个数据置信度最大的类别及其索引

筛选掉（类别=背景）置信度最大的数据（predictions和localisations）

如果select_threshold > 0:

去除掉背景一行

通过np.where选择类别置信度大于阈值的数据索引（idxes=[索引1，索引2，索引3] 根据索引1和索引2找到每条数据，索引3代表类别）

保留类别置信度大于阈值的数据的类别、类别得分及这条数据对应的位置。

将各个特征层的default box对应的这些数据通过np.concatenate拼接到一起

3.2.6 通过NMS（非极大值抑制）算法过滤掉重叠度较大的预测框

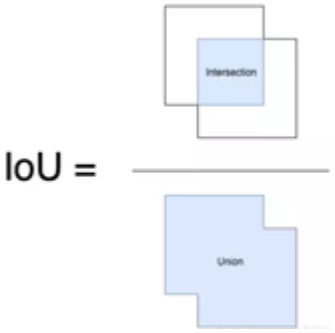
对应np_methods.py中bboxes_nms函数

根据置信度对筛选后的全部数据进行降序排列，仅保留top-k（如400）个预测框。

设置keep_boxes，计算每一个预测框与排序在其后预测框的重叠度（IOU），若其后的边框重叠度<阈值 or 与之前的预测框类别不同，则keep_boxes=1，否则将该预测框keep_boxes设为0。

计算预测框的重叠度（IOU）：

对应np_methods.py中bboxes_jaccard函数



最后根据图像大小对预测框坐标进行归一化

第四部分 SSD中使用的一些tricks

4.1 使用瘦高与宽扁默认框

数据集中目标的开关往往各式各样，因此挑选合适形状的默认框能够提高检测效果。作者实验得出使用瘦高与宽扁默认框相对于只使用正方形默认框有2.9%mAP提升。

4.2 数据扩增

使用随机截取图像信息、随机左右翻转、随机颜色扰动、白化方法对数据进行扩增后，有了2.0%的mAP提升。

4.3 使用atrous卷积

通常卷积过程中为了使特征图尺寸特征图尺寸保持不变，通过会在边缘打padding，但人为加入的padding值会引入噪声，因此，使用atrous卷积能够在保持感受野不变的条件下，减少padding噪声，关于atrous参考。本文SSD训练过程中并且没有使用atrous卷积，但预训练过程使用的模型为VGG-16-atrous，意味着作者给的预训练模型是使用atrous卷积训练出来的。使用atrous版本VGG-16作为预训模型比较普通VGG-16要提高0.7%mAP。

注：COCO数据集，全称Common Objects in Context，是微软团队获取的一个可以用来图像识别、语义分割、目标检测 的数据集。

特点：

COCO数据集包含200,000图片和80 类别。

每张图片上有多个检测目标。

主要从复杂的日常场景中截取。

标签文件标记了每个segmentation的像素精确位置+bounding box的精确坐标，其精度均为小数点后两位。