YOLOv3目标检测算法

模型训练

@tm9161

L2

模型训练

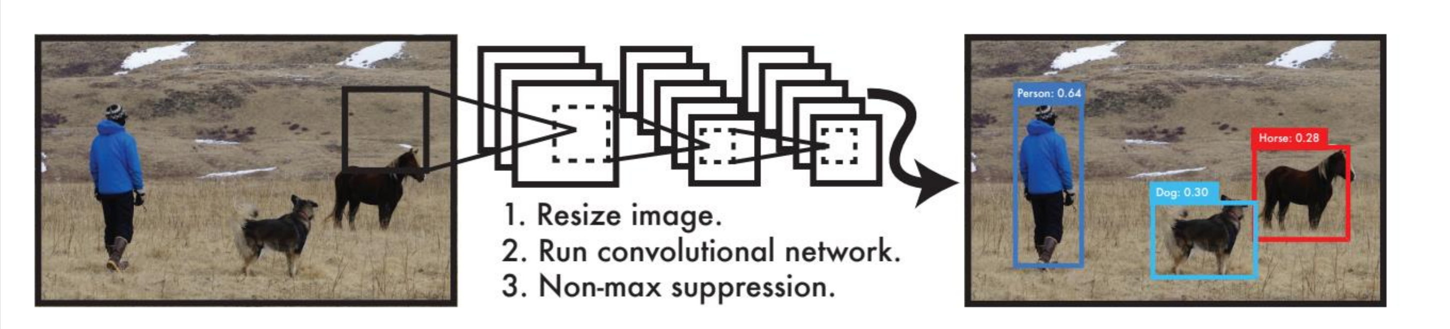
1. 模型网络结构
2. 损失函数计算
3. 数据载入及模型训练

目标检测任务

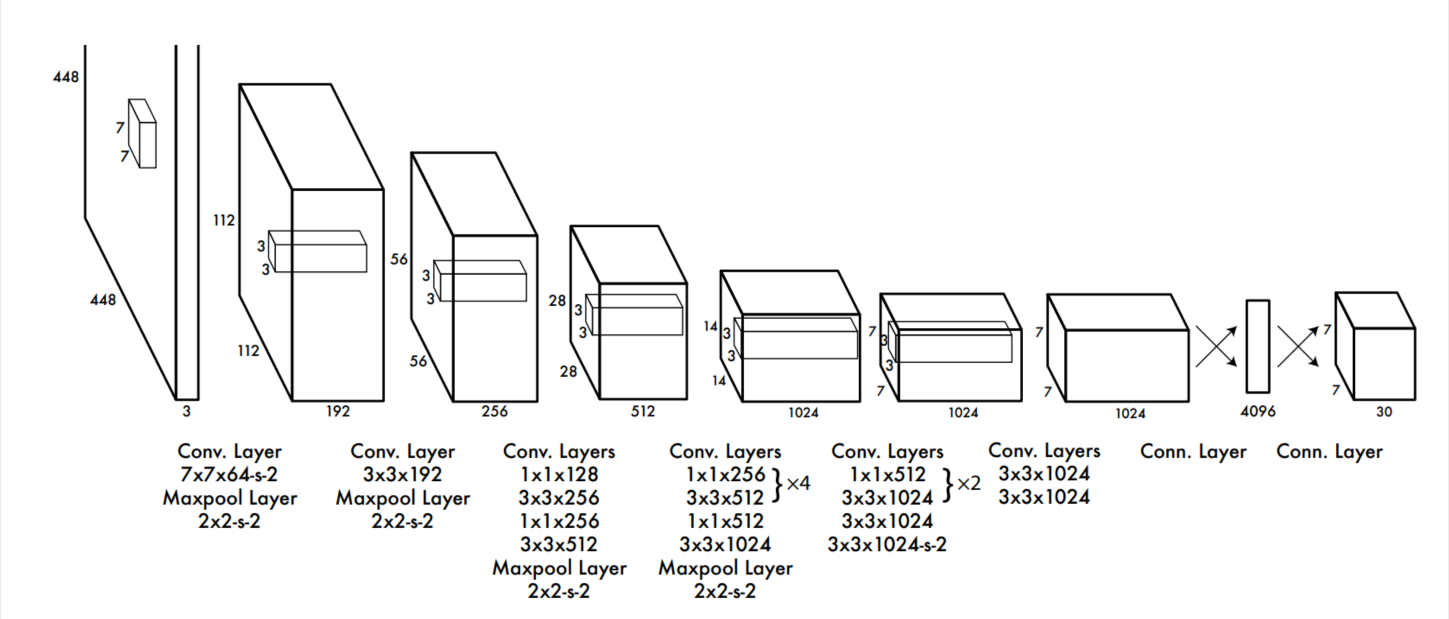
中心点（X，Y）
框宽高（W，H）
置信度
每个类别的概率

输入图片——网络模型——预测框+类别

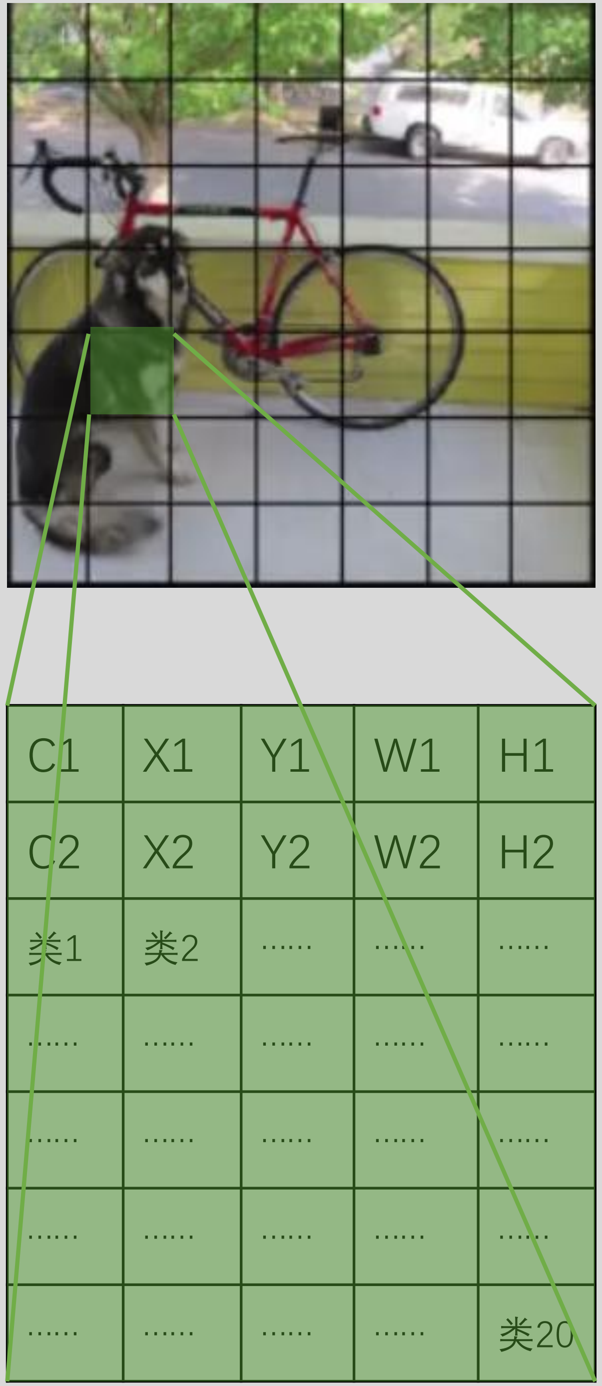
YOLO V1



Person: 0.64
Horse: 0.28
Dog: 0.30
1. Resize image.
2. Run convolutional network.
3. Non-max suppression.

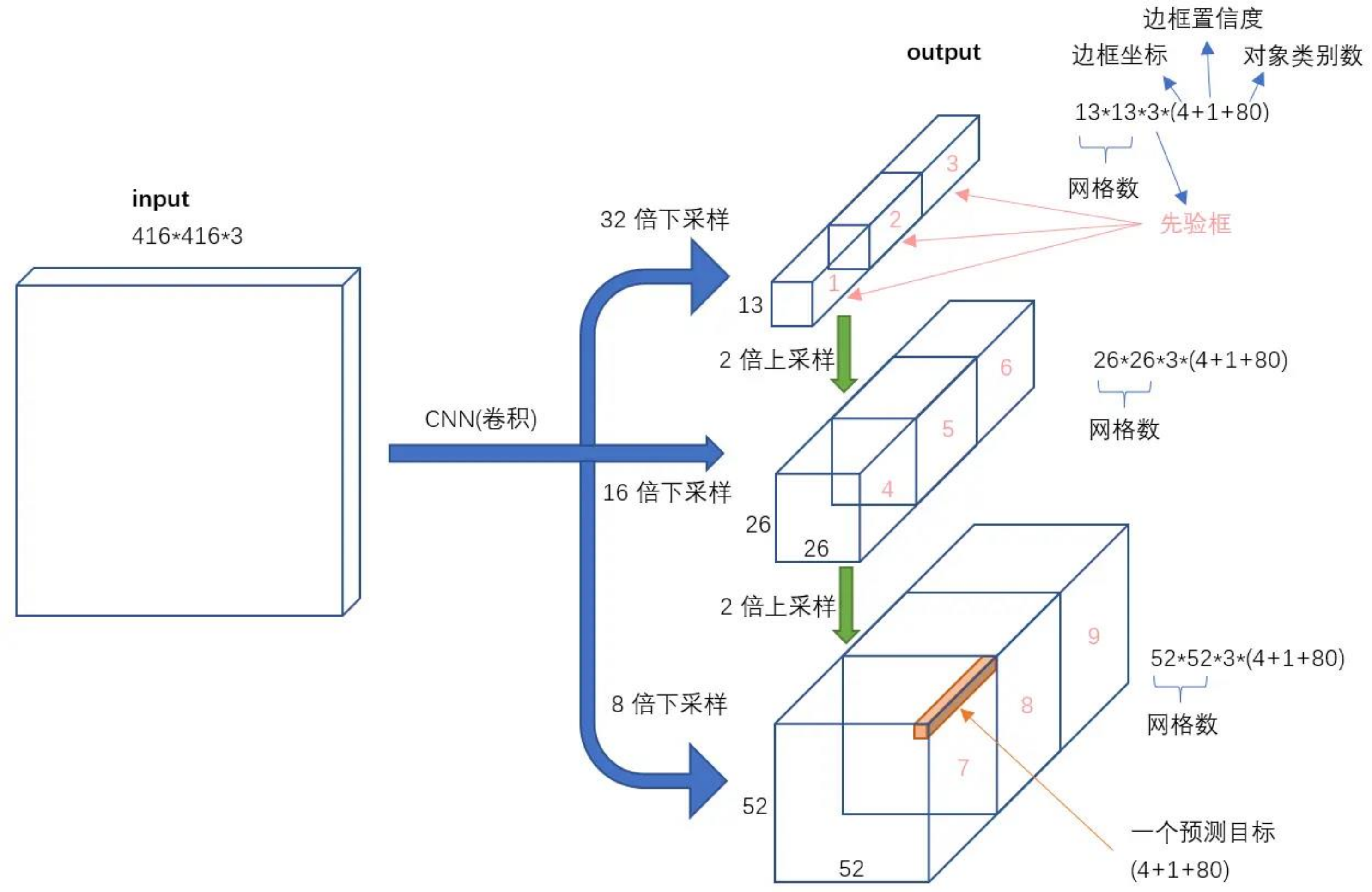


448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer
7x7x64-s-2
Maxpool Layer
2x2-s-2
Conv. Layer
3x3x192
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x128
3x3x256
1x1x256
3x3x512
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x256
3x3x512
}×4
1x1x512
3x3x1024
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x512
3x3x1024
}×2
3x3x1024
3x3x1024-s-2
Conv. Layers
3x3x1024
3x3x1024
Conn. Layer
Conn. Layer

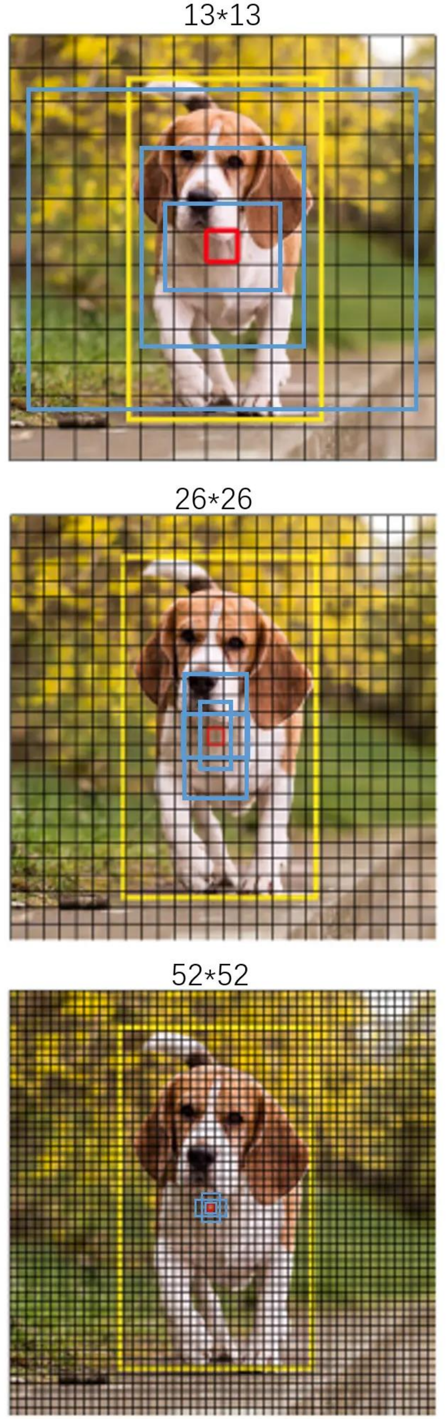


C1
X1
Y1
W1
H1
C2
X2
Y2
W2
H2
类1
类2
……
类20

YOLO V3



图片来源：https://www.jianshu.com/p/d13ae1055302?tdsourcetag=s_pcqq_aiomsg



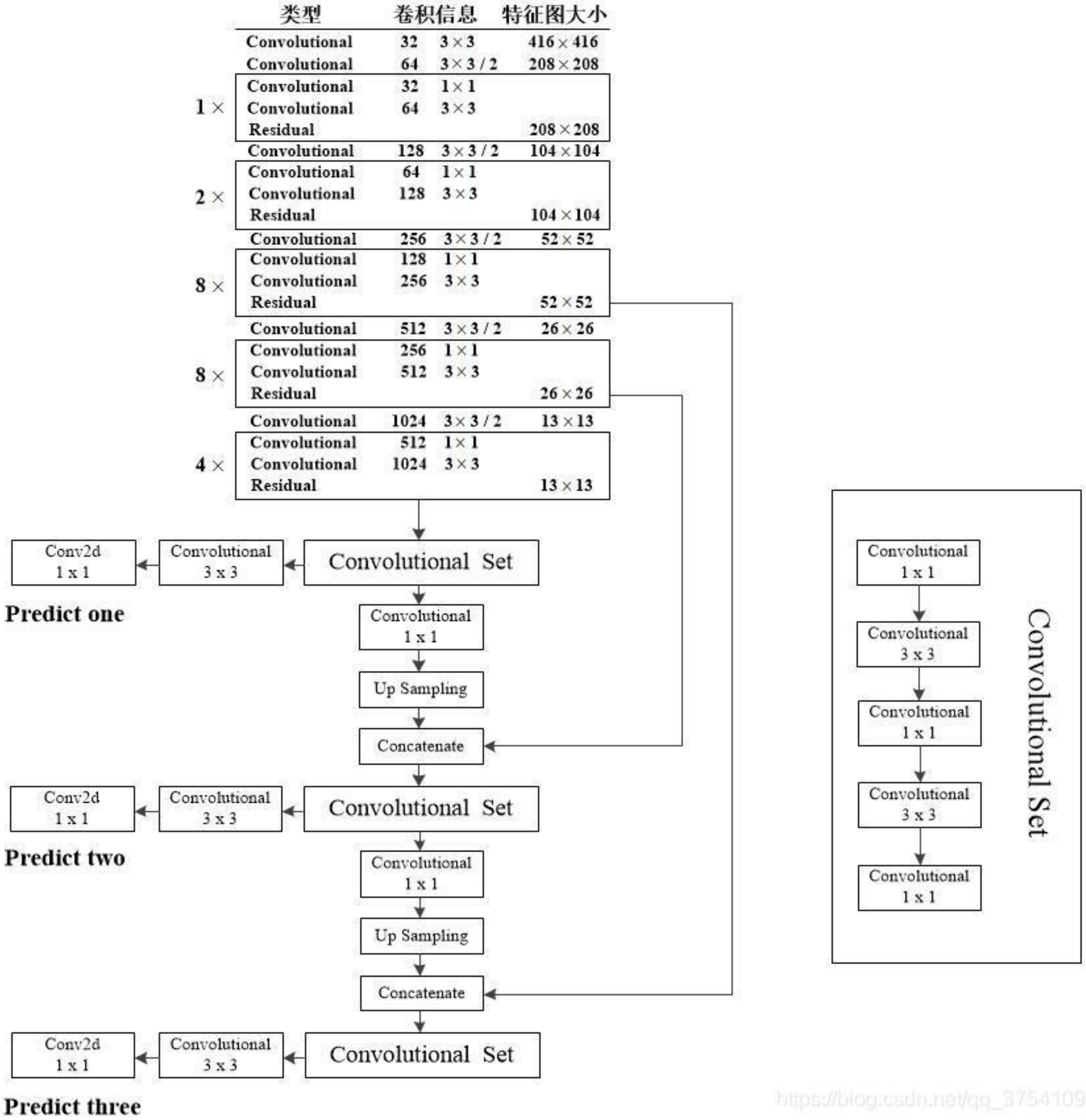
网络结构

1. 卷积 Convolutional

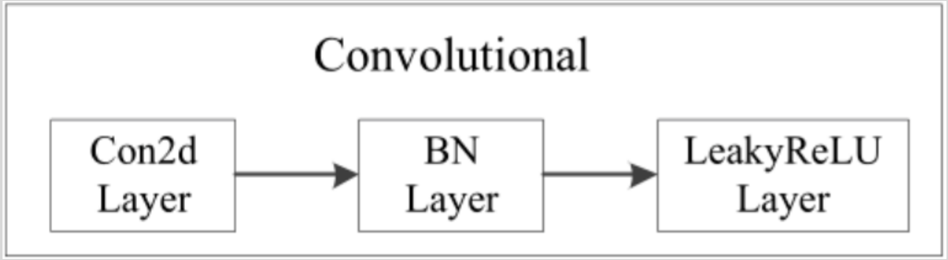
2. 残差 Residual

3. 输出 Convolutional Set

结构图来源：https://www.bilibili.com/video/BV1yi4y1g7ro



1. Convolutional

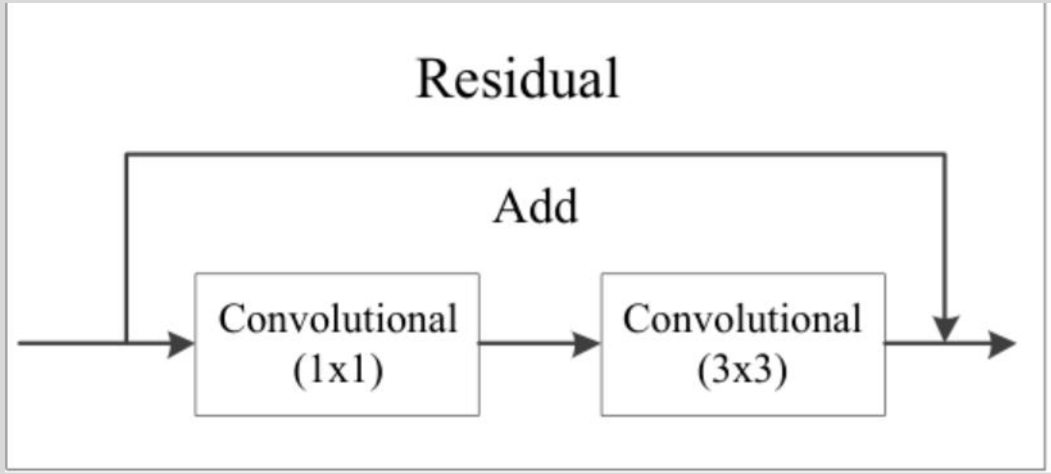



```python
def compose(*funcs):
    """Compose arbitrarily many functions, evaluated left to right.

    Reference: https://mathieularose.com/function-composition-in-python/
    """
    # return lambda x: reduce(lambda v, f: f(v), funcs, x)
    if funcs:
        return reduce(lambda f, g: lambda *a, **kw: g(f(*a, **kw)), funcs)
    else:
        raise ValueError('Composition of empty sequence not supported.')

#依次执行参数对应的函数操作 简化代码
```

```python
def DarknetConv2D(*args, **kwargs):
    """Wrapper to set Darknet parameters for Convolution2D."""
    darknet_conv_kwargs = {'kernel_regularizer': l2(5e-4)} #正则化
    darknet_conv_kwargs['padding'] = 'valid' if kwargs.get('strides') == (2, 2) else 'same' #加边
    darknet_conv_kwargs.update(kwargs) #更新参数
    return Conv2D(*args, **darknet_conv_kwargs)


def DarknetConv2D_BN_Leaky(*args, **kwargs):
    """Darknet Convolution2D followed by BatchNormalization and LeakyReLU."""
    no_bias_kwargs = {'use_bias': False}
    no_bias_kwargs.update(kwargs)
    return compose(
        DarknetConv2D(*args, **no_bias_kwargs),
        BatchNormalization(),
        LeakyReLU(alpha=0.1))
#卷积+BN+激活
```

结构图来源：https://www.bilibili.com/video/BV1yi4y1g7ro

2. Residual



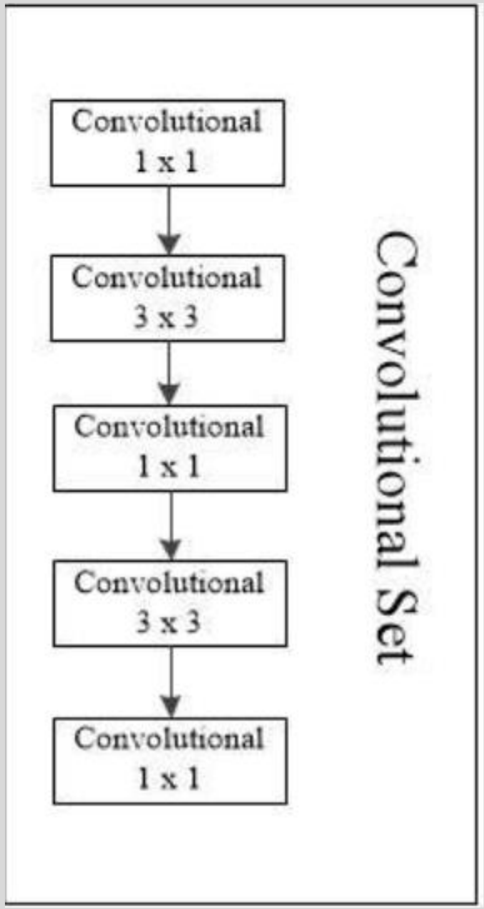

```python
def resblock_body(x, num_filters, num_blocks):
    '''A series of resblocks starting with a downsampling Convolution2D'''
    # Darknet uses left and top padding instead of 'same' mode
    x = ZeroPadding2D(((1, 0), (1, 0)))(x)
    x = DarknetConv2D_BN_Leaky(num_filters, (3, 3), strides=(2, 2))(x)
    for i in range(num_blocks):
        y = compose(
            DarknetConv2D_BN_Leaky(num_filters // 2, (1, 1)),
            DarknetConv2D_BN_Leaky(num_filters, (3, 3)))(x)
        x = Add()([x, y])
    return x
#加边+卷积模块+拼接
```

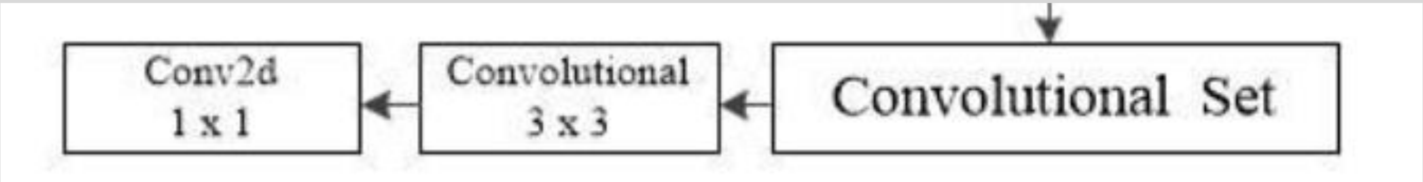
	Convolutional	1024	3 × 3 / 2	13 × 13
4 ×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			13 × 13

```python
x = resblock_body(x, 1024, 4)
```

结构图来源：https://www.bilibili.com/video/BV1yi4y1g7ro

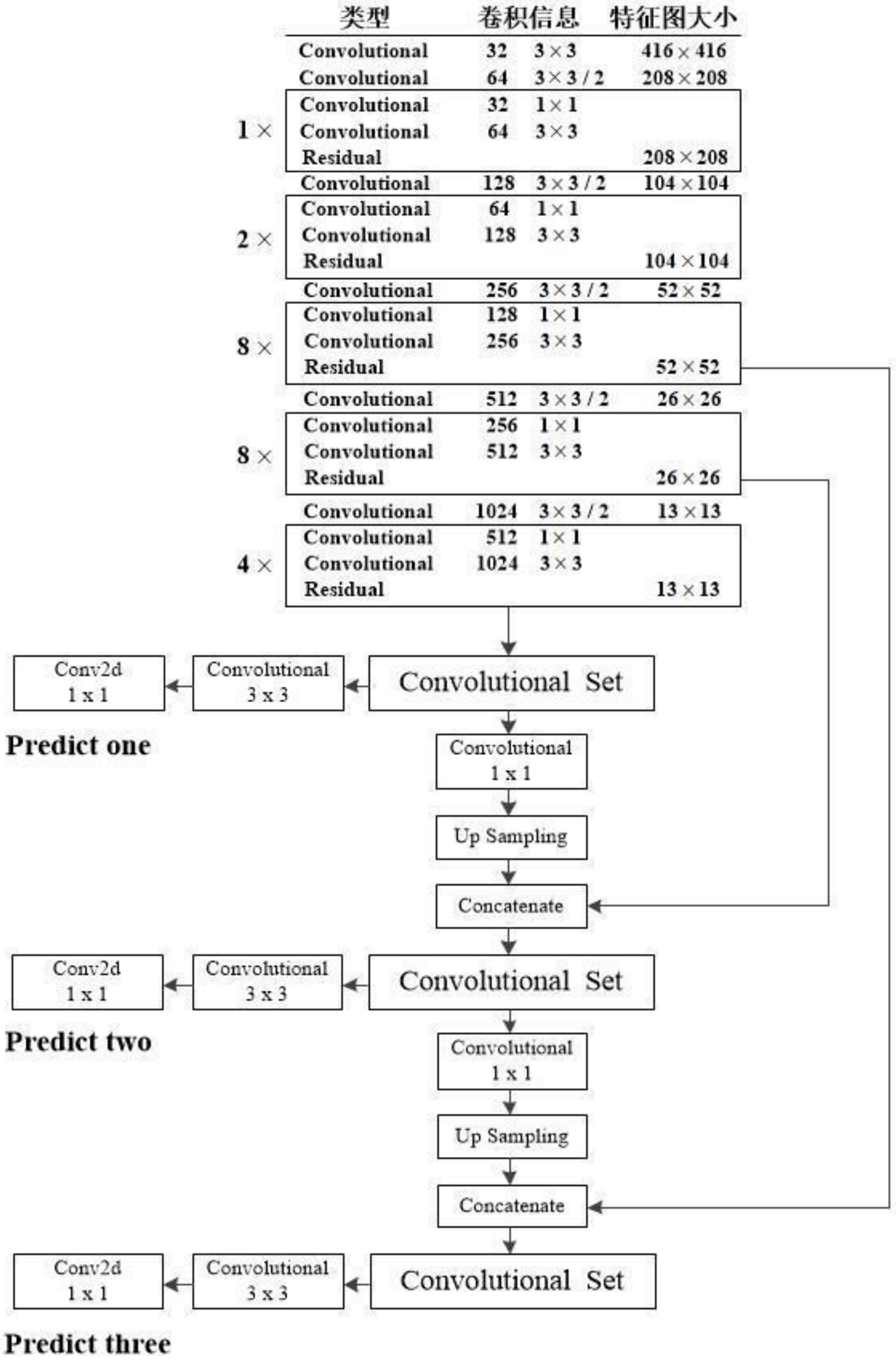
3. Convolutional Set






```
def make_last_layers(x, num_filters, out_filters):
    '''6 Conv2D_BN_Leaky layers followed by a Conv2D_linear layer'''
    x = compose(
        DarknetConv2D_BN_Leaky(num_filters, (1, 1)),
        DarknetConv2D_BN_Leaky(num_filters * 2, (3, 3)),
        DarknetConv2D_BN_Leaky(num_filters, (1, 1)),
        DarknetConv2D_BN_Leaky(num_filters * 2, (3, 3)),
        DarknetConv2D_BN_Leaky(num_filters, (1, 1)))(x)
    y = compose(
        DarknetConv2D_BN_Leaky(num_filters * 2, (3, 3)),
        DarknetConv2D(out_filters, (1, 1)))(x)
    return x, y

#输出部分
```

结构图来源：https://www.bilibili.com/video/BV1yi4y1g7ro




```
def darknet_body(x):
    '''Darknent body having 52 Convolution2D layers'''
    x = DarknetConv2D_BN_Leaky(32, (3, 3))(x)
    x = resblock_body(x, 64, 1)
    x = resblock_body(x, 128, 2)
    x = resblock_body(x, 256, 8)
    x = resblock_body(x, 512, 8)
    x = resblock_body(x, 1024, 4)
    return x
```

```
def yolo_body(inputs, num_anchors, num_classes):
    """Create YOLO_V3 model CNN body in Keras."""
    darknet = Model(inputs, darknet_body(inputs))
    x, y1 = make_last_layers(darknet.output, 512, num_anchors * (num_classes + 5))

    x = compose(
        DarknetConv2D_BN_Leaky(256, (1, 1)),
        UpSampling2D(2))(x)
    x = Concatenate()([x, darknet.layers[152].output])
    x, y2 = make_last_layers(x, 256, num_anchors * (num_classes + 5))

    x = compose(
        DarknetConv2D_BN_Leaky(128, (1, 1)),
        UpSampling2D(2))(x)
    x = Concatenate()([x, darknet.layers[92].output])
    x, y3 = make_last_layers(x, 128, num_anchors * (num_classes + 5))

    return Model(inputs, [y1, y2, y3])
```

YOLO头部分

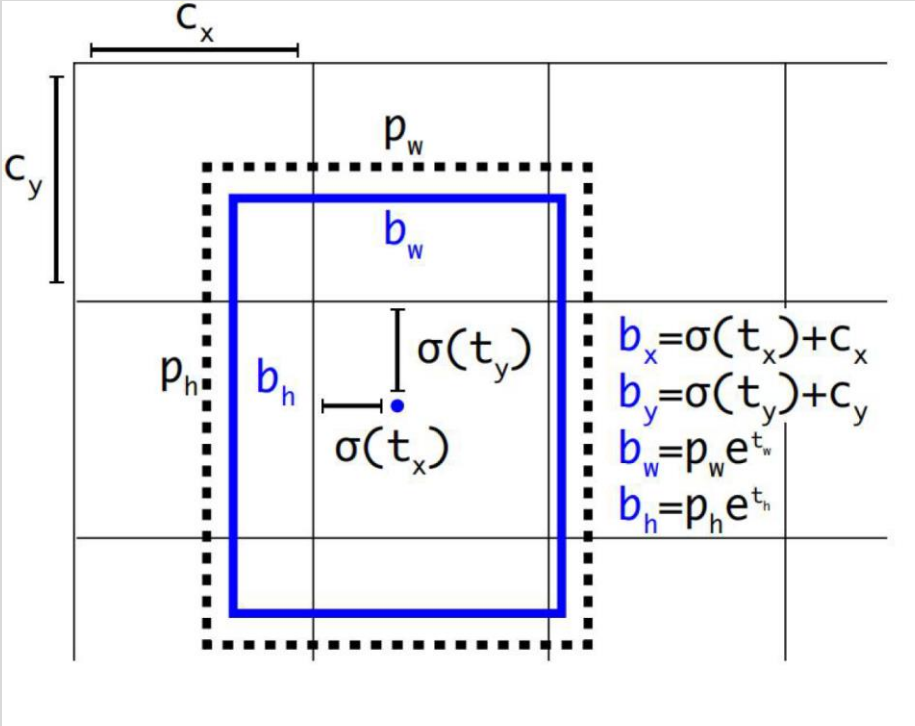
1. 生成预测网格

```
num_anchors = len(anchors)
anchors_tensor = K.reshape(K.constant(anchors), [1, 1, 1, num_anchors, 2])
# Reshape to batch, height, width, num_anchors, box_params.

grid_shape = K.shape(feats)[1:3]  # height, width
grid_y = K.tile(K.reshape(K.arange(0, stop=grid_shape[0]), [-1, 1, 1, 1]),[1, grid_shape[1], 1, 1])
grid_x = K.tile(K.reshape(K.arange(0, stop=grid_shape[1]), [1, -1, 1, 1]),[grid_shape[0], 1, 1, 1])
grid = K.concatenate([grid_x, grid_y])
grid = K.cast(grid, K.dtype(feats))

feats = K.reshape(feats, [-1, grid_shape[0], grid_shape[1], num_anchors, num_classes + 5])
```

2. 生成预测框

```
# Adjust preditions to each spatial grid point and anchor size.
box_xy = (K.sigmoid(feats[..., :2]) + grid) / K.cast(grid_shape[::-1], K.dtype(feats))
box_wh = K.exp(feats[..., 2:4]) * anchors_tensor / K.cast(input_shape[::-1], K.dtype(feats))
box_confidence = K.sigmoid(feats[..., 4:5])
box_class_probs = K.sigmoid(feats[..., 5:])
```

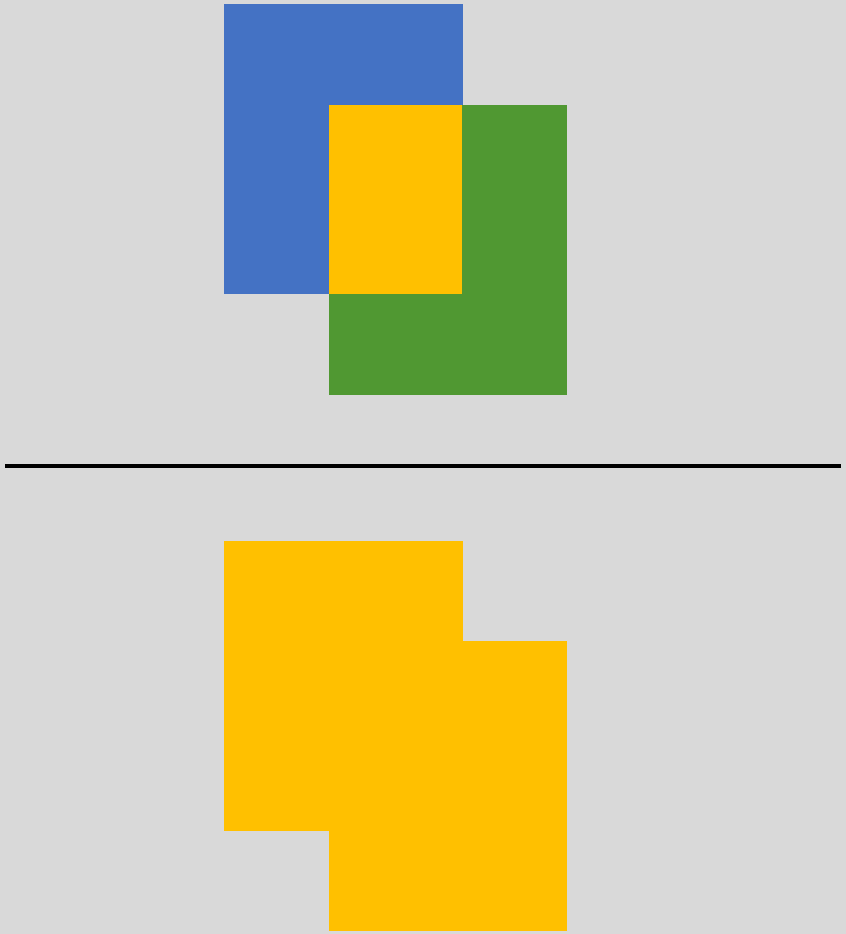
中心点（X，Y）
框宽高（W，H）
置信度
每个类别的概率



结构图来源：https://www.bilibili.com/video/BV1yi4y1g7ro

交并比计算

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$



```python
def box_iou(b1, b2):

    # Expand dim to apply broadcasting.
    b1 = K.expand_dims(b1, -2)
    b1_xy = b1[..., :2]
    b1_wh = b1[..., 2:4]
    b1_wh_half = b1_wh / 2.
    b1_mins = b1_xy - b1_wh_half
    b1_maxes = b1_xy + b1_wh_half

    # Expand dim to apply broadcasting.
    b2 = K.expand_dims(b2, 0)
    b2_xy = b2[..., :2]
    b2_wh = b2[..., 2:4]
    b2_wh_half = b2_wh / 2.
    b2_mins = b2_xy - b2_wh_half
    b2_maxes = b2_xy + b2_wh_half

    intersect_mins = K.maximum(b1_mins, b2_mins)
    intersect_maxes = K.minimum(b1_maxes, b2_maxes)
    intersect_wh = K.maximum(intersect_maxes - intersect_mins, 0.)
    intersect_area = intersect_wh[..., 0] * intersect_wh[..., 1]
    b1_area = b1_wh[..., 0] * b1_wh[..., 1]
    b2_area = b2_wh[..., 0] * b2_wh[..., 1]
    iou = intersect_area / (b1_area + b2_area - intersect_area)

    return iou
```

损失值计算：

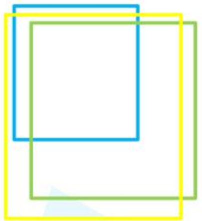
置信度损失

Binary Cross Entropy

$$L_{conf}(o,c) = -\frac{\sum_i (o_i \ln(\hat{c}_i) + (1-o_i)\ln(1-\hat{c}_i))}{N}$$

$$\hat{c}_i = Sigmoid(c_i)$$

可能和原文有出入



其中$o_i \in [0,1]$，表示预测目标边界框与真实目标边界框的IOU，

c为预测值，$\hat{c}_i$为c通过$Sigmoid$函数得到的预测置信度，

N为正负样本个数

YOLOv3 predicts an objectness score for each bounding box using logistic regression. This should be 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. If the bounding box prior

类别损失

Binary Cross Entropy

$$L_{cla}(O,C) = -\frac{\sum_{i \in pos}\sum_{j \in cla}(O_{ij}\ln(\hat{C}_{ij}) + (1-O_{ij})\ln(1-\hat{C}_{ij}))}{N_{pos}}$$

$$\hat{C}_{ij} = Sigmoid(C_{ij})$$

其中$O_{ij} \in \{0,1\}$，表示预测目标边界框i中是否存在第j类目标，

C_{ij}为预测值，$\hat{C}_{ij}$为C_{ij}通过$Sigmoid$函数得到的目标概率

N_{pos}为正样本个数

Each box predicts the classes the bounding box may contain using multilabel classification. We do not use a softmax as we have found it is unnecessary for good performance, instead we simply use independent logistic classifiers. During training we use binary cross-entropy loss for the class predictions.

定位损失

$$L_{loc}(t,g) = \frac{\sum_{i \in pos}(\sigma(t_x^i)-\hat{g}_x^i)^2 + (\sigma(t_y^i)-\hat{g}_y^i)^2 + (t_w^i-\hat{g}_w^i)^2 + (t_h^i-\hat{g}_h^i)^2}{N_{pos}}$$

$$\hat{g}_x^i = g_x^i - c_x^i$$

$$\hat{g}_y^i = g_y^i - c_y^i$$

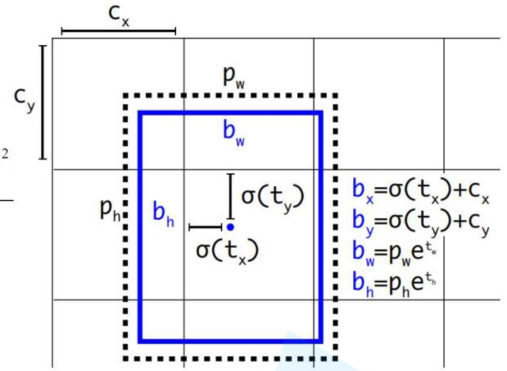
$$\hat{g}_w^i = \ln(g_w^i / p_w^i)$$

$$\hat{g}_h^i = \ln(g_h^i / p_h^i)$$

t_x, t_y, t_w, t_h：为网络预测的回归参数

g_x, g_y, g_w, g_h：为GT中心点的坐标 x, y以及宽度和高度

（映射在Grid网格中的）



During training we use sum of squared error loss. If the ground truth for some coordinate prediction is $\hat{t}_*$ our gradient is the ground truth value (computed from the ground truth box) minus our prediction: $\hat{t}_* - t_*$. This ground truth value can be easily computed by inverting the equations above.

置信度损失　　分类损失　　定位损失

$$L(o,c,O,C,l,g) = \lambda_1 \boxed{L_{conf}(o,c)} + \lambda_2 \boxed{L_{cla}(O,C)} + \lambda_3 \boxed{L_{loc}(l,g)}$$

$\lambda_1, \lambda_2, \lambda_3$为平衡系数

内容来自：https://www.bilibili.com/video/BV1yi4y1g7ro

置信度损失&分类损失

$$L_{conf}(o,c) = -\frac{\sum_i (o_i \ln(\hat{c}_i) + (1-o_i)\ln(1-\hat{c}_i))}{N}$$

```
confidence_loss = object_mask * K.binary_crossentropy(object_mask, raw_pred[..., 4:5], from_logits=True) + \
            (1 - object_mask) * K.binary_crossentropy(object_mask, raw_pred[..., 4:5],
                                                    from_logits=True) * ignore_mask
```

计算预测和真实标签的二元交叉熵损失

包含物体+不包含物体

```
# 寻找忽略掩码
ignore_mask = tf.TensorArray(K.dtype(y_true[0]), size=1, dynamic_size=True)
object_mask_bool = K.cast(object_mask, 'bool')

def loop_body(b, ignore_mask):
    true_box = tf.boolean_mask(y_true[l][b, ..., 0:4], object_mask_bool[b, ..., 0])
    iou = box_iou(pred_box[b], true_box)
    best_iou = K.max(iou, axis=-1)
    ignore_mask = ignore_mask.write(b, K.cast(best_iou < ignore_thresh, K.dtype(true_box)))
    return b + 1, ignore_mask

_, ignore_mask = tf.while_loop(lambda b, *args: b < m, loop_body, [0, ignore_mask])
ignore_mask = ignore_mask.stack()
ignore_mask = K.expand_dims(ignore_mask, -1)
```

```
class_loss = object_mask * K.binary_crossentropy(true_class_probs, raw_pred[..., 5:], from_logits=True)
```

计算预测和真实标签的二元交叉熵损失

包含物体才有类别损失

$$L_{cla}(O,C) = -\frac{\sum_{i\in pos}\sum_{j\in cla}(O_{ij}\ln(\hat{C}_{ij}) + (1-O_{ij})\ln(1-\hat{C}_{ij}))}{N_{pos}}$$

位置&尺寸损失

```
box_xy = (K.sigmoid(feats[..., :2]) + grid) / K.cast(grid_shape[::-1], K.dtype(feats))
box_wh = K.exp(feats[..., 2:4]) * anchors_tensor / K.cast(input_shape[::-1], K.dtype(feats))
```

```
# 处理真实框，计算损失
raw_true_xy = y_true[l][..., :2] * grid_shapes[l][::-1] - grid
raw_true_wh = K.log(y_true[l][..., 2:4] / anchors[anchor_mask[l]] * input_shape[::-1])
raw_true_wh = K.switch(object_mask, raw_true_wh, K.zeros_like(raw_true_wh))  # avoid log(0)=-inf
box_loss_scale = 2 - y_true[l][..., 2:3] * y_true[l][..., 3:4]
```

```
xy_loss = object_mask * box_loss_scale * 0.5 * K.square(raw_true_xy - raw_pred[..., 0:2])
wh_loss = object_mask * box_loss_scale * 0.5 * K.square(raw_true_wh - raw_pred[..., 2:4])
```

位置：计算预测和真实标签的平方差
尺寸：计算预测和真实标签的平方差

通过缩放控制权重

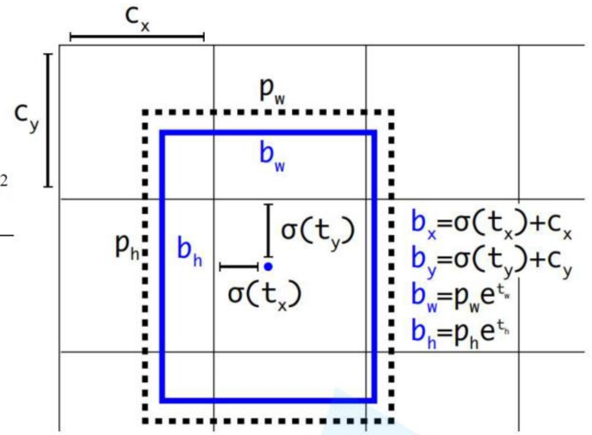
定位损失

$$L_{loc}(t,g)=\frac{\sum_{i\in pos}(\sigma(t_x^i)-\hat{g}_x^i)^2+(\sigma(t_y^i)-\hat{g}_y^i)^2+(t_w^i-\hat{g}_w^i)^2+(t_h^i-\hat{g}_h^i)^2}{N_{pos}}$$

$$\hat{g}_x^i=g_x^i-c_x^i$$
$$\hat{g}_y^i=g_y^i-c_y^i$$
$$\hat{g}_w^i=\ln(g_w^i/p_w^i)$$
$$\hat{g}_h^i=\ln(g_h^i/p_h^i)$$

t_x,t_y,t_w,t_h：为网络预测的回归参数

g_x,g_y,g_w,g_h ：为GT 中心点的坐标 x, y以及宽度和高度
（映射在Grid网格中的）



During training we use sum of squared error loss. If the ground truth for some coordinate prediction is $\hat{t}_*$ our gradient is the ground truth value (computed from the ground truth box) minus our prediction: $\hat{t}_*-t_*$. This ground truth value can be easily computed by inverting the equations above.

损失值计算：

$$L(o,c,O,C,l,g)=\lambda_1\underbrace{\boxed{L_{conf}(o,c)}}_{\text{置信度损失}}+\lambda_2\underbrace{\boxed{L_{cla}(O,C)}}_{\text{分类损失}}+\lambda_3\underbrace{\boxed{L_{loc}(l,g)}}_{\text{定位损失}}$$

$\lambda_1,\lambda_2,\lambda_3$为平衡系数

```
# 将每部分损失求和并除以批次大小，实现损失的归一化
xy_loss = K.sum(xy_loss) / mf
wh_loss = K.sum(wh_loss) / mf
confidence_loss = K.sum(confidence_loss) / mf
class_loss = K.sum(class_loss) / mf
# 将所有损失项累加得到总损失
loss += xy_loss + wh_loss + confidence_loss + class_loss
```

模型创建

数据载入&预处理
生成真实框
构建批训练数据

模型训练

```python
anchors = np.array([[25,39], [38,91], [62,51], [71,136], [123,214], [127,95], [219,293], [250,148], [394,298]])

# 定义要检测的类别
class_names = ['aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'car', 'cat', 'chair', 'cow', 'diningtal
num_classes = len(class_names)  # 类别的数量

input_shape = (416, 416)  # 输入尺寸，必须是32的倍数

# 创建YOLOv3模型
model = create_model(input_shape, anchors, num_classes, freeze_body=2, weights_path='yolov3.h5')

# 设置TensorBoard日志
logging = TensorBoard(log_dir='logs/')

# 设置模型保存的回调函数
checkpoint = ModelCheckpoint('ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5',
                             monitor='val_loss', save_weights_only=True, save_best_only=True, save_freq="epoch"

# 设置学习率衰减的回调函数
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3, verbose=1)

# 读取训练数据
annotation_path = '2007_trainval.txt'
val_split = 0.1  # 验证集的比例
with open(annotation_path) as f:
    lines = f.readlines()

# 分割训练集和验证集
num_val = int(len(lines) * val_split)
num_train = len(lines) - num_val

# 编译模型
model.compile(optimizer=Adam(lr=1e-3), loss={'yolo_loss': lambda y_true, y_pred: y_pred})

# 打印训练和验证样本数量
batch_size = 8
print('Train on {} samples, val on {} samples, with batch size {}.'.format(num_train, num_val, batch_size))

# 开始训练模型
model.fit(x=data_generator(lines[:num_train], batch_size, input_shape, anchors, num_classes),
          steps_per_epoch=max(1, num_train // batch_size),
          validation_data=data_generator(lines[num_train:], batch_size, input_shape, anchors, num_classes),
          validation_steps=max(1, num_val // batch_size),
          epochs=25,
          initial_epoch=0,
          callbacks=[logging, checkpoint])  # reduce_lr为可选项
```

参考资料：

1. 3.1 YOLO系列理论合集(YOLOv1~v3)
https://www.bilibili.com/video/BV1yi4y1g7ro

2. AaronJny/tf2-keras-yolo3
https://github.com/AaronJny/tf2-keras-yolo3

3. Keras 搭建自己的yolo3目标检测平台（Bubbliiiing 深度学习 教程）
https://www.bilibili.com/video/BV1XJ411D7wF

4. keras-yolov3代码解读
https://blog.csdn.net/qq_25800609/article/details/87880651

5.目标检测系列(4)— 在目标检测中不能不聊的 YOLOv3
https://juejin.cn/post/7125356491951276045

6.大语言模型