

一、作者Joseph Redmon概况

二、yolo v1解读

- (一) 、yolo v1几个核心点
- (二) 、yolo v1论文解读: <https://arxiv.org/pdf/1506.02640.pdf>
- (三) 、yolo v1源码解读: <https://github.com/abeardear/pytorch-YOLO-v1>

三、yolo v2解读

- (一) 、yolo v2几个改进点
- (二) 、yolo v2论文解读: <https://arxiv.org/pdf/1612.08242.pdf>
- (三) 、yolo v2源码解读: <https://github.com/allanzelener/YAD2K>

四、yolo v3解读

- (一) 、yolo v3几个改进点
- (二) 、yolo v3论文解读: <https://arxiv.org/pdf/1804.02767.pdf>
- (三) 、yolo v3源码解读: <https://github.com/zzh8829/yolov3-tf2>
- (四) 、yolo v3源码解读: <https://github.com/ultralytics/yolov3>
- (五) 、yolo v3复现: https://github.com/yyccR/yolov3_in_tf2_keras
- (六) 、yolo v3训练coco2017

一、Joseph Redmon



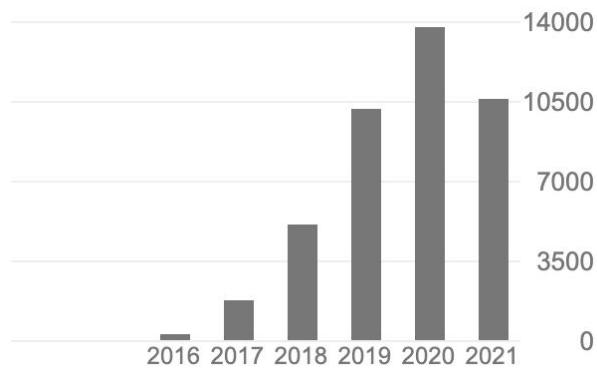
Joseph Redmon

[University of Washington](#)

Verified email at cs.washington.edu - [Homepage](#)

Type Theory Computer Vision Machine Learning

	All	Since 2016
Citations	42288	41894
h-index	13	13
i10-index	14	14



TITLE	CITED BY	YEAR
You only look once: Unified, real-time object detection J Redmon, S Divvala, R Girshick, A Farhadi Proceedings of the IEEE conference on computer vision and pattern ...	18395	2016
YOLO9000: Better, Faster, Stronger. J Redmon, A Farhadi Proceedings of the IEEE conference on computer vision and pattern recognition	9213	2017
Yolov3: An incremental improvement J Redmon, A Farhadi arXiv preprint arXiv:1804.02767	8515	2018
Xnor-net: Imagenet classification using binary convolutional neural networks M Rastegari, V Ordonez, J Redmon, A Farhadi European conference on computer vision, 525-542	3396	2016

一、Joseph Redmon



Joseph Redmon @pjreddie · 2020年2月21日

...

I stopped doing CV research because I saw the impact my work was having.
I loved the work but the military applications and privacy concerns
eventually became impossible to ignore.



Roger Grosse @RogerGrosse · 2020年2月20日

回覆 @skoularidou

What's an example of a situation where you think someone should
decide not to submit their paper due to Broader Impacts reasons?

88

1,196

3,465

↑

20年2月作者在Twitter宣布退出cv领域研究，原因是他的研究被用于军事和隐私研究。

值得关注的是，J.Redmon后来实习的公司 XNOR.ai，是当初XNOR-Net论文第一作者参与创立，20年1月被苹果公司2亿美元收购。

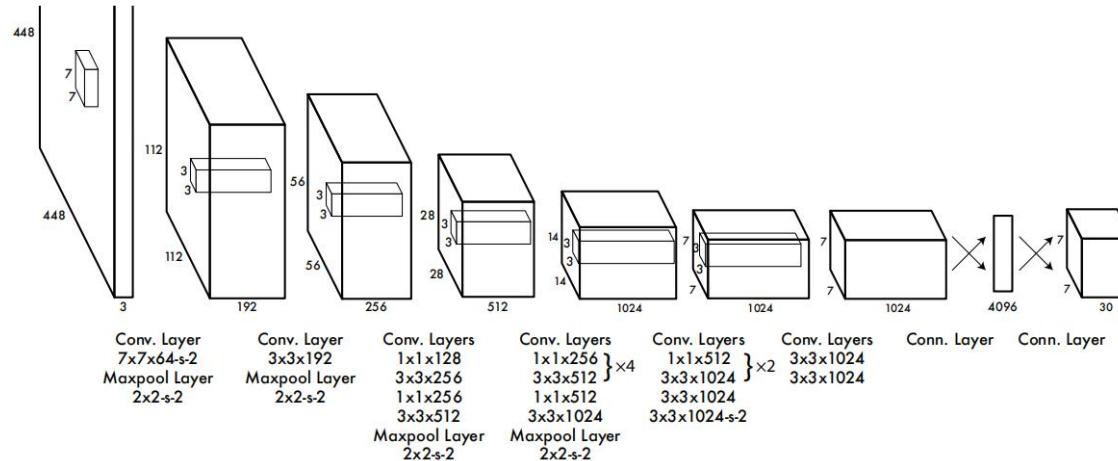
YOLO v1



二、YOLO v1论文解读

(一) YOLO v1两个核心步骤:

1. 卷积网络预测生成边框xywh, 前景概率, 类别概率:



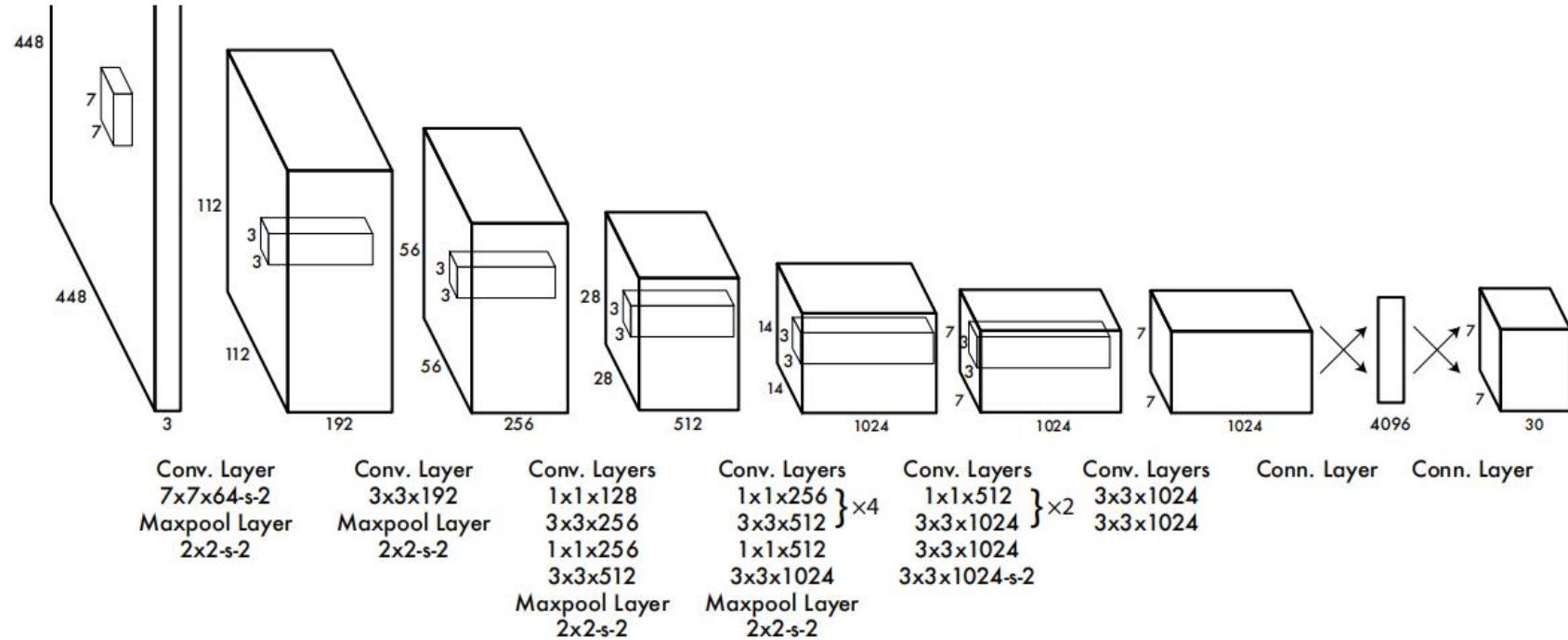
2. 计算5个损失:

$$\begin{aligned}
 & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
 & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
 & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
 & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
 \end{aligned}$$

二、YOLO v1论文解读

(一) YOLO v1两个核心步骤：卷积预测

1. 卷积网络结构：



预训练分类阶段输入大小: 224x224

检测训练阶段输入大小: 448x448

二、YOLO v1论文解读

(一) YOLO v1两个核心步骤：卷积预测

2. 卷积网络具体实现：

以Vgg为例：

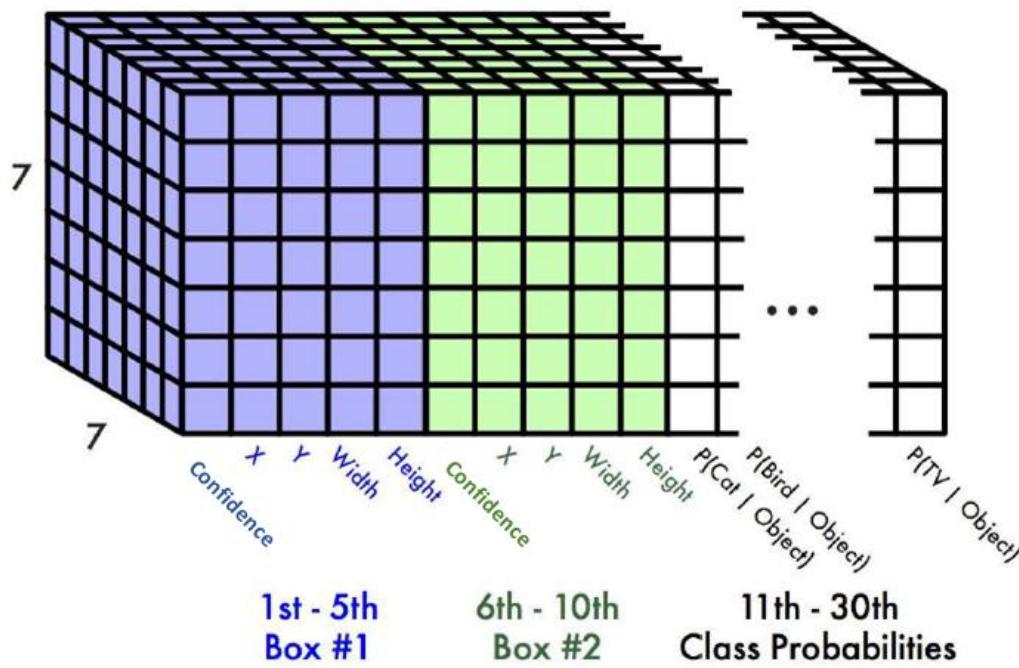
1. 其中nn.Sequential(*layers)可以是任意卷积网络结构
2. yolov1再卷积后面加多2层fc
3. 最后将fc结果处理成(batch,7,7,30)

```
class VGG(nn.Module):
    def __init__(self, image_size=448):
        super(VGG, self).__init__()
        self.features = nn.Sequential(*layers)
        self.image_size = image_size
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 1470),
        )
        self._initialize_weights()
    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        x = F.sigmoid(x)
        x = x.view(-1, 7, 7, 30)
        return x
```

二、YOLO v1论文解读

(一) YOLO v1两个核心步骤：卷积预测

3. 卷积网络预测结果： $7 \times 7 \times 30$:



其中：

1. xy表示预测边框的中心点，它是相对于当前grid坐标的偏移量；
2. width,height表示预测边框宽高；
3. confidence表示预测的边框是否包含目标
4. $P(\text{Cat}|\text{Object})\dots$ 表示当对应边框包含目标，改目标为对应类别的概率。

最终预测：

1. $7 \times 7 \times 2 \times 4$ 个边框坐标
2. $7 \times 7 \times 2$ 个前景目标的概率
3. $7 \times 7 \times 20$ 个类别概率 (voc2012数据为20个类别)

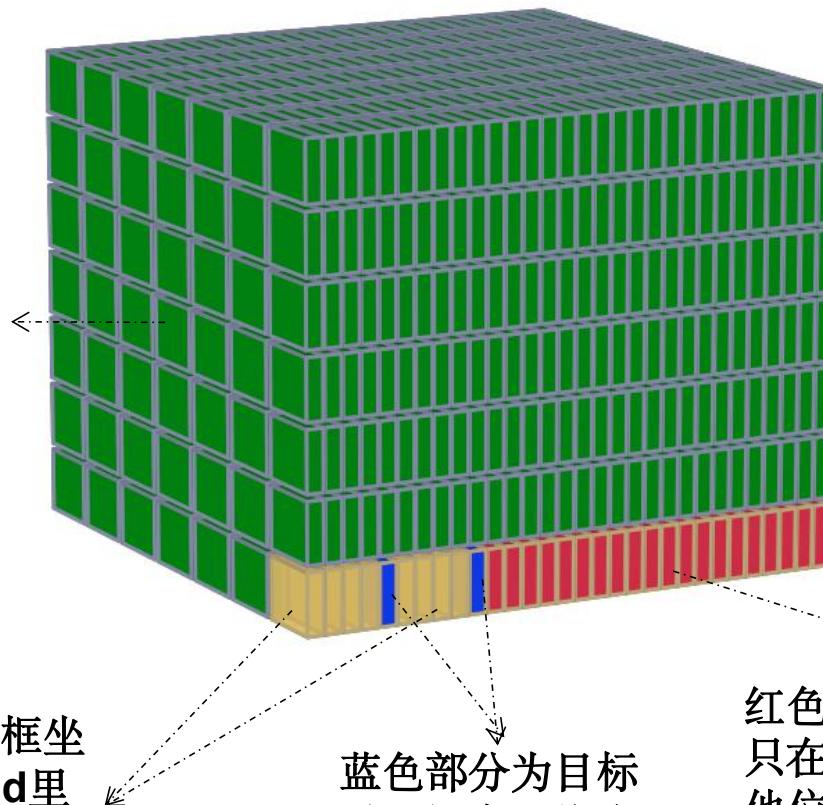
二、YOLO v1论文解读

(一) YOLO v1两个核心步骤：卷积预测

4. 生成目标数据: $7 \times 7 \times 30$:

绿色部分全部为
0, 表示没有目标

黄色部分填补边框坐
标 $xywh$, 同个grid里
面两个边框 $xywh$ 均
相同



蓝色部分为目标
前景概率, 均为1

红色部分为20个类别概率,
只在对应类别上为1, 其
他位置为0

tips:

当只有1个group true
边框时, 只有一个
grid上面有数据, 其
他都是0

二、YOLO v1论文解读

(一) YOLO v1两个核心步骤：卷积预测

```
def encoder(boxes,labels):
    ...
    boxes (tensor) [[x1,y1,x2,y2],[]], 其中xy均已归一化
    labels (tensor) [...]
    return 7x7x30
    ...

    grid_num = 7
    target = torch.zeros((grid_num,grid_num,30))
    # 计算宽高
    wh = boxes[:,2:]-boxes[:,2:]
    # 计算中心点xy
    cxxy = (boxes[:,2:]+boxes[:,2:])/2
    for i in range(cxxy.size()[0]):
        cxxy_sample = cxxy[i]
        # 将归一化坐标中心点映射到7x7grid量纲下，取整得到对应0-7下的坐标
        ij = (cxxy_sample * grid_num).ceil()-1
        # 对应目标概率为1，对应类别概率为1
        target[int(ij[1]),int(ij[0]),4] = 1
        target[int(ij[1]),int(ij[0]),9] = 1
        target[int(ij[1]),int(ij[0]),int(labels[i])+9] = 1
        xy = ij / grid_num
        # 计算xy偏移量
        delta_xy = (cxxy_sample -xy) * grid_num
        target[int(ij[1]),int(ij[0]),2:4] = wh[i]
        target[int(ij[1]),int(ij[0]),:2] = delta_xy
        target[int(ij[1]),int(ij[0]),7:9] = wh[i]
        target[int(ij[1]),int(ij[0]),5:7] = delta_xy
    return target
```

5. 那么目标边框是如何对应到相应的grid上？

二、YOLO v1论文解读

(一) YOLO v1两个核心步骤：损失计算

6. 计算5个损失

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \quad ①$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \quad ②$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \quad ③$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \quad ④$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad ⑤$$

二、YOLO v1论文解读

(一) YOLO v1两个核心步骤：损失计算

7. 目标边框没有包含目标情况下的损失计算，只计算背景概率损失：

$$\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \quad ④$$

其中：

1. $\lambda_{\text{noobj}} = 0.5$ 主要是为了平衡负样本跟正样本比例不均衡问题
2. $\mathbb{1}_{ij}^{\text{noobj}} = 1$ 表示第*i*个grid的第*j*个box不包含目标
3. 两个*C_i*表示对应预测和实际的前景概率，这里*C_i = 0*
4. 由于一个grid会预测两个边框，所以也会预测2个前景概率，不包含目标的情况下，这两个前景概率都会参与最后的损失计算

二、YOLO v1论文解读

(一) YOLO v1两个核心步骤：损失计算

8. 目标边框包含目标情况下的前景概率损失计算：

$$\cdot \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \quad \textcircled{3}$$

其中：

1. $\mathbb{1}_{ij}^{\text{obj}} = 1$ 表示第*i*个grid的第*j*个box包含目标
2. 两个*C_i*表示对应预测和实际的前景概率，这里 $C_i = 1$
3. 由于一个grid会预测两个边框，在包含目标的情况下，计算损失的时候只会选择其中一个跟真实边框iou较高的那个做计算，也就只会拿那个iou较高的前景概率做损失计算。

二、YOLO v1论文解读

(一) YOLO v1两个核心步骤：损失计算

9. 目标边框包含目标情况下的边框损失：

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \quad ①$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \quad ②$$

其中：

1. $\lambda_{\text{obj}} = 5$ 同样是为了平衡负样本跟正样本比例不均衡问题
2. $\mathbb{1}_{ij}^{\text{obj}} = 1$ 表示第*i*个grid的第*j*个box包含目标
3. xywh预测损失计算均采用mae平方差
4. 由于一个grid会预测两个边框，实际上计算损失的时候只会选择其中一个跟真实边框iou较高的那个做计算。

二、YOLO v1论文解读

(一) YOLO v1两个核心步骤：损失计算

9. 目标边框包含目标情况下的类别预测损失：

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad \textcircled{5}$$

其中：

1. $\mathbb{1}_{ij}^{\text{obj}} = 1$ 表示第*i*个grid的第*j*个box包含目标
2. $P_i(C)$ 里只有对应类别的概率为1，其他为0

二、YOLO v1论文解读

yolov1的输入输出(voc数据为例):

inputs:

images(batch, H, W, channels)
targets(batch, 7, 7, 30)

model(training):

```
# [batch,7,7,30]
output_tensors = yolov1(images)

# loss
loc_loss, obj_loss, cls_loss = loss_fn(output_tensors, targets)
```

YOLO v2



三、YOLO v2论文解读

(一) YOLO v2, 几个核心改进点:

	YOLO								YOLOv2
batch norm?	✓	✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?		✓	✓	✓	✓	✓	✓	✓	✓
convolutional?			✓	✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?							✓	✓	✓
passthrough?								✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	78.6

1. 加入batchnorm
2. 提高输入数据分辨率
3. 预测边框引入anchor机制
4. 采用darnet-19新的卷积网络
5. 采用kmeans聚类得到先验框
6. 预测坐标改为偏移缩放
7. 高低维特征融合(passthrough)
8. 多尺度输入训练策略

三、YOLO v2论文解读

(一) YOLO v2, 几个核心改进点:

1. 加入batchnorm

对比v1, 去掉fc与drop_out层, 直接用bn+relu, 提升大概2%mAP

```
class VGG(nn.Module):
    def __init__(self, image_size=448):
        super(VGG, self).__init__()
        self.features = nn.Sequential(*layers)
        self.image_size = image_size
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 1470),
        )
        self._initialize_weights()
    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        x = F.sigmoid(x)
        x = x.view(-1, 7, 7, 30)
        return x
```



```
def yolo_body(inputs, num_anchors, num_classes):
    """Create YOLO_V2 model CNN body in Keras."""
    darknet = Model(inputs, darknet_body()(inputs))
    conv20 = compose(
        DarknetConv2D_BN_Leaky(1024, (3, 3)),
        DarknetConv2D_BN_Leaky(1024, (3, 3)))(darknet.output)

    conv13 = darknet.layers[43].output
    conv21 = DarknetConv2D_BN_Leaky(64, (1, 1))(conv13)
    # TODO: Allow Keras Lambda to use func arguments for output_shape?
    conv21_reshaped = Lambda(
        space_to_depth_x2,
        output_shape=space_to_depth_x2_output_shape,
        name='space_to_depth')(conv21)

    x = concatenate([conv21_reshaped, conv20])
    x = DarknetConv2D_BN_Leaky(1024, (3, 3))(x)
    x = DarknetConv2D(num_anchors * (num_classes + 5), (1, 1))(x)
    return Model(inputs, x)
```

三、YOLO v2论文解读

(一) YOLO v2, 几个核心改进点:

1. 加入batchnorm, 为什么bn和drop_out的通常是二选一?

Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift

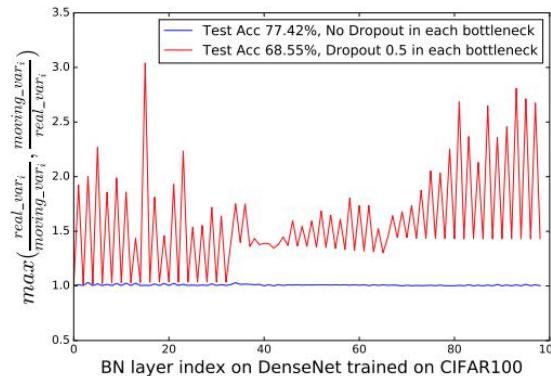
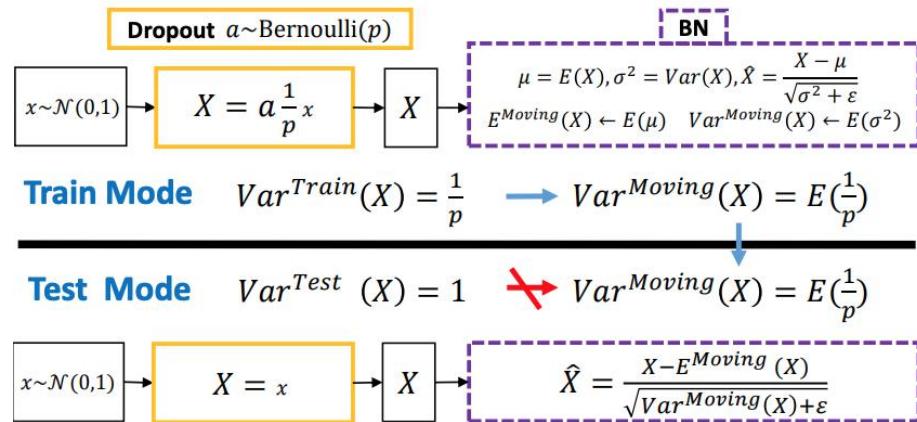
<https://arxiv.org/pdf/1801.05134.pdf>

简单归纳:

训练阶段drop_out如果在bn之前会出现方差偏移, 导致训练和测试结果的bn参数出现不一样, 使得模型失效。

解决方案:

1. 不用drop_out
2. 作者建议drop_out接在bn后
3. 采用高斯Dropout, 又称Uout



三、YOLO v2论文解读

(一) YOLO v2, 几个核心改进点:

2. 提高输入数据分辨率, 3. 预测边框引入anchor机制

对比v1:

1. 预训练分类输入从 224×224 调整为 448×448
2. 检测阶段训练输入从 448×448 调整为 416×416
3. 最终输出从 $7 \times 7 \times (10 + \text{num_classes})$ 调为 $13 \times 13 \times \text{num_anchors} \times (5 + \text{num_classes})$

其中具体实现 $\text{num_anchors}=5$

三、YOLO v2论文解读

(一) YOLO v2, 几个核心改进点:

5. 采用kmeans聚类得到先验框

coco数据集生成的anchors

np.array(

(0.57273, 0.677385), (1.87446, 2.06253), (3.33843, 5.47434),
(7.88282, 3.52778), (9.77052, 9.16828))

voc数据集生成的anchors

np.array(

(1.08, 1.19), (3.42, 4.41), (6.63, 11.38), (9.42, 5.11), (16.62, 10.52))

这里的anchor跟faster-rcnn里的anchor不同点:

1. yolov2的anchor分别对应边框的w, h, 其xy为最后预测的13x13上面的每个grid坐标, 所以yolov2生成的anchor其实不多, 就几个
2. faster-rcnn中anchor是根据backbone网络输出的特征(MxM)大小, 在每个点上生成N个边框, 总共生成MxMxN个边框, 每个边框坐标[x1,y1,x2,y2]
3. yolov2里的anchor大小都是通过聚类生成的, faster-rcnn里是硬编码固定大小生成的

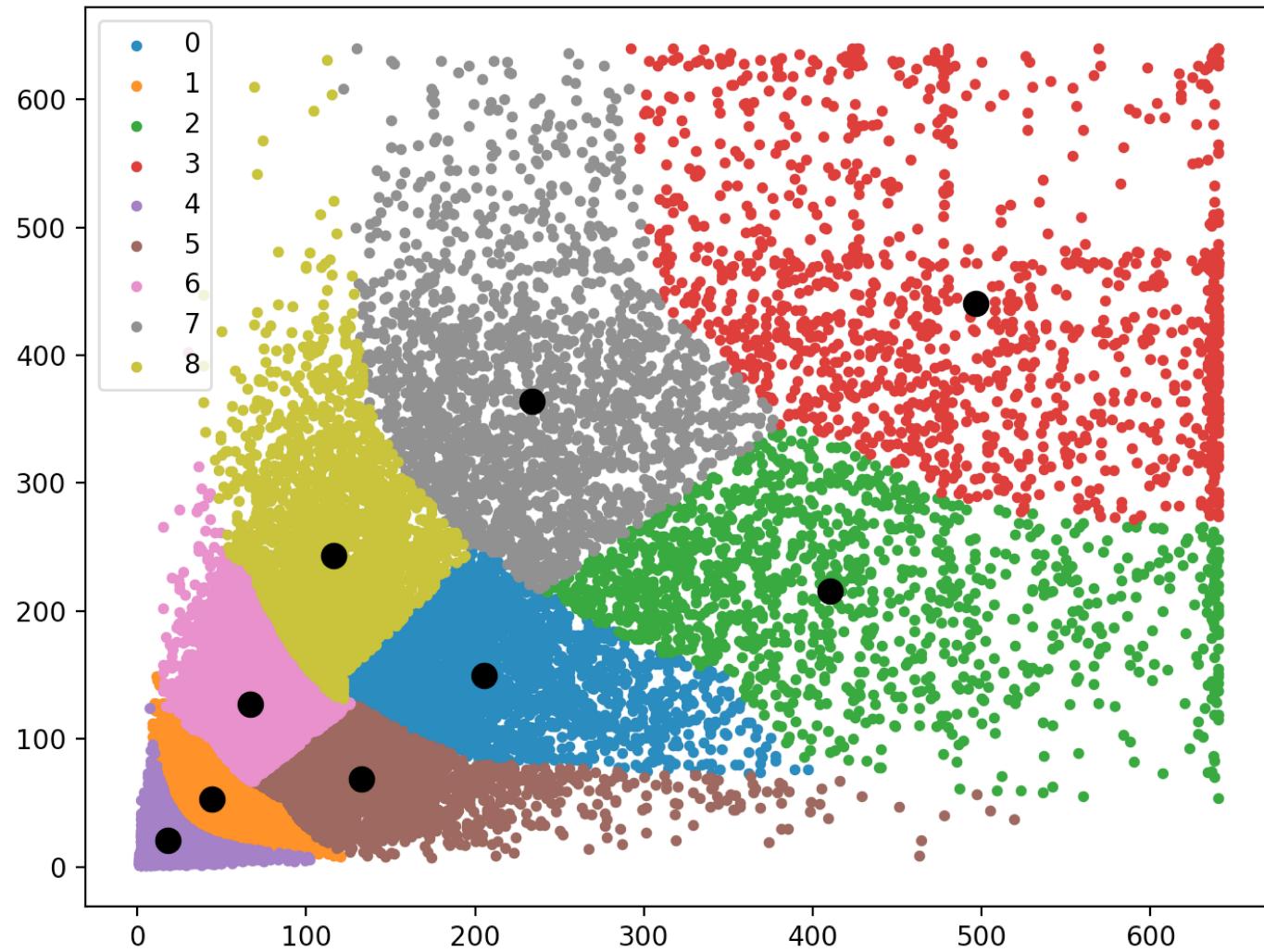
三、YOLO v2论文解读

(一) YOLO v2, 几个核心改进点:

5. 采用kmeans聚类得到先验框

在coco2017-val数据集上，计算所有打
标边框的wh

其中k=9的
kmeans聚类
结果，黑点
即为最终
anchor



三、YOLO v2论文解读

(一) YOLO v2, 几个核心改进点:

5. 采用kmeans聚类得到先验框, 聚类相似度计算: $d(\text{box}, \text{centroid}) = 1 - \text{IOU}(\text{box}, \text{centroid})$

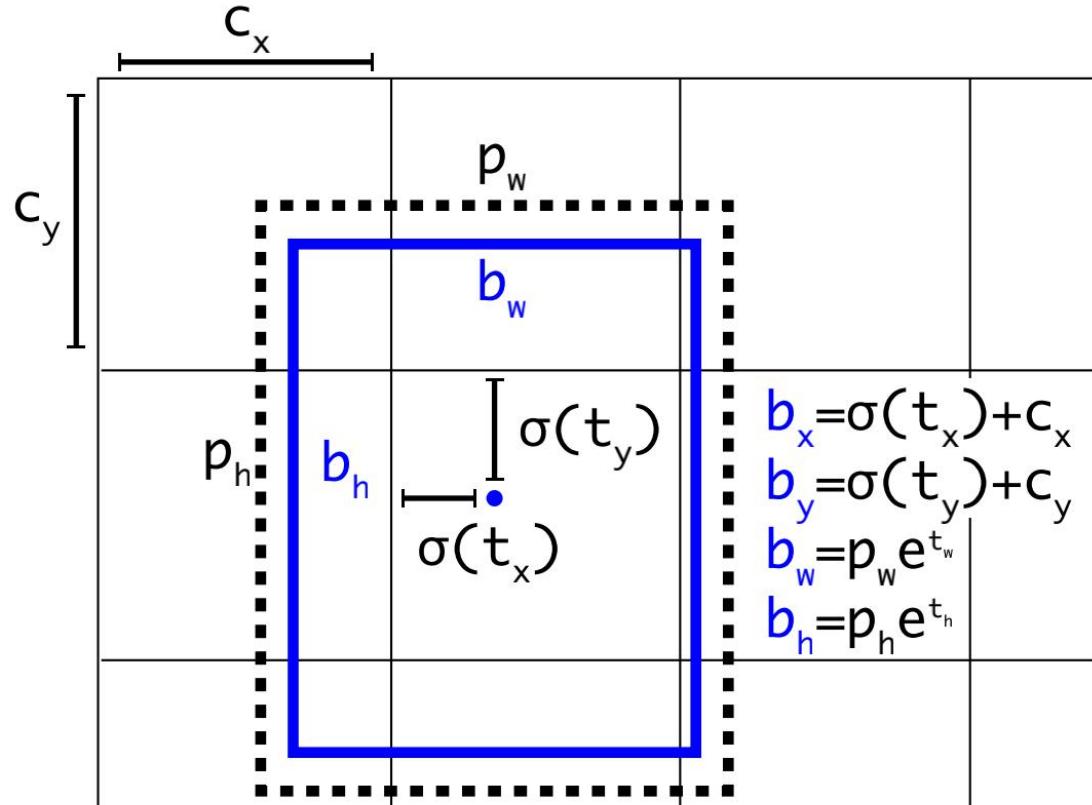
```
def kmeans(x, K):
    # 随机选K个中心点作为初始值
    centers = x[np.random.choice(a=x.shape[0], size=K, replace=False)]
    pre_max_center_ids = np.zeros(np.shape(x)[0], dtype=np.int64)
    step = 0
    while True:
        iou = compute_iou(x=x, centers=centers)
        max_center_ids = np.argmax(a=iou, axis=1)
        # 如果聚类重新分配簇id没有变化则退出
        if np.all(max_center_ids == pre_max_center_ids):
            # 根据面积大小重新排序输出centers
            sort_centers = sorted(centers, key=lambda v:v[0]*v[1], reverse=False)
            sort_centers = np.array(sort_centers, dtype=np.int64)
            return sort_centers
        centers = np.zeros_like(centers, dtype=np.float32)
        # 根据新的类簇id, 重新计算类簇中心
        for j in range(K):
            target_x_index = max_center_ids == j
            target_x = x[target_x_index,]
            centers[j] = np.sum(target_x, axis=0) / np.sum(target_x_index)
        pre_max_center_ids = max_center_ids.copy()
        step += 1
```

具体实现:

三、YOLO v2论文解读

(一) YOLO v2, 几个核心改进点:

6. 预测坐标改为偏移缩放



其中:

1. t_x, t_y, t_w, t_h 为模型预测输出
2. $\sigma(*)$ 为sigmoid函数, 归一化 t_x, t_y
3. p_w, p_h 为anchor宽高
4. c_x, c_y 为当前grid的坐标
5. b_x, b_y, b_w, b_h 为最终预测目标边框

对比yolov1:

$$\begin{aligned} b_x &= t_x + c_x \\ b_y &= t_y + c_y \\ b_w &= t_w \\ b_h &= t_h \end{aligned}$$

三、YOLO v2论文解读

(一) YOLO v2, 几个核心改进点:

4. 采用darknet-19新的卷积网络

yolo darkne-19 body
这部分无论是在训练分类，还是训练检测阶段都是一样

这里卷积+avgpool是做分类训练，检测训练这里采用passthrough结构

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

三、YOLO v2论文解读

(一) YOLO v2, 几个核心改进点:

7. 高低维特征融合(passthrough)

```
def yolo_body(inputs, num_anchors, num_classes):
    """Create YOLO_V2 model CNN body in Keras."""
    darknet = Model(inputs, darknet_body()(inputs))
    conv20 = compose(
        DarknetConv2D_BN_Leaky(1024, (3, 3)),
        DarknetConv2D_BN_Leaky(1024, (3, 3)))(darknet.output)

    conv13 = darknet.layers[43].output
    conv21 = DarknetConv2D_BN_Leaky(64, (1, 1))(conv13)
    # TODO: Allow Keras Lambda to use func arguments for output_shape?
    conv21_reshaped = Lambda(
        space_to_depth_x2,
        output_shape=space_to_depth_x2_output_shape,
        name='space_to_depth')(conv21)

    x = concatenate([conv21_reshaped, conv20])
    x = DarknetConv2D_BN_Leaky(1024, (3, 3))(x)
    x = DarknetConv2D(num_anchors * (num_classes + 5), (1, 1))(x)
    return Model(inputs, x)
```

passthrough思想:
将模型最后一层与第43层合并

最后一层
conv20: [batch, 13, 13, 1024]

第43层
conv13: [batch, 26, 26, 512]
↓
conv21_reshape:
[batch, 13, 13, 256]

合并
output: [batch, 13, 13, 1280]

三、YOLO v2论文解读

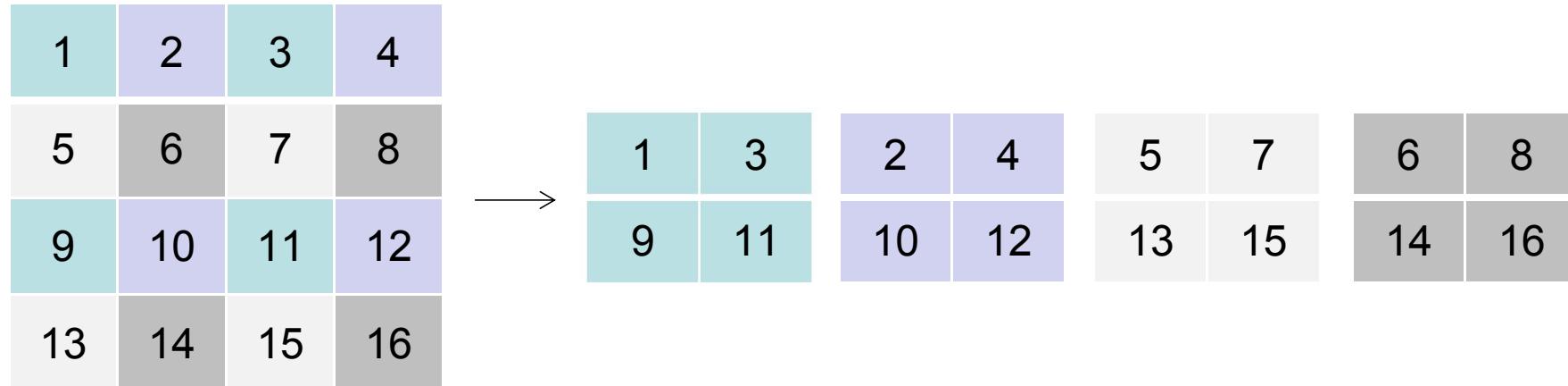
(一) YOLO v2, 几个核心改进点:

7. 高低维特征融合(passthrough), space_to_depth细节:

举个栗子:

`x.shape = [1, 4, 4, 1]`

`tf.nn.space_to_depth(x, block_size=2).shape = [1, 2, 2, 4]`



总结就是space_to_depth把tensor的h,w两个维度的数据转移到channel上。

三、YOLO v2论文解读

(一) YOLO v2, 几个核心改进点:

8. 多尺度输入训练策略

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288×288	2007+2012	69.0	91
YOLOv2 352×352	2007+2012	73.7	81
YOLOv2 416×416	2007+2012	76.8	67
YOLOv2 480×480	2007+2012	77.8	59
YOLOv2 544×544	2007+2012	78.6	40

为了解决不同大小的目标检测，yolov2在v1基础上加入了多尺度训练方法，其具体为：

1. 保持backbone网络不变
2. 调整输入的图片大小: [320, 352, 384, 416, 448, 480, 512, 544, 576, 608]
3. 每隔10个epoch改变一次输入的大小

对比faster-rcnn以及mask-rcnn中的检测，其采用的特征金字塔(FPN)的结构来处理不同大小目标检测问题。

三、YOLO v2论文解读

(二) YOLO v2, 整体结构:

inputs:

```
image_input = Input(shape=(416, 416, 3))
boxes_input = Input(shape=(None, 5))
# 这里对应下文  $1_{ij}^{ob}$ , 即对应grid-anchor位置有无包含目标
detectors_mask_input = Input(shape=(13, 13, 5, 1))
# 将groud_true边框类别放到对应grid-anchor位置上
matching_boxes_input = Input(shape=(13, 13, 5, 5))
```

model(training):

```
# [batch, 13, 13, 5, 4+1+num_class]
output_tensors = yolov1(images)

# loss
loc_loss, obj_loss, cls_loss =
    loss_fn(output_tensors, boxes_input, detectors_mask_input,
            matching_boxes_input)
```

三、YOLO v2论文解读

(二) 、YOLO v2, 整体结构: 损失计算

$$\begin{aligned}
 & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
 & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
 & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
 & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2
 \end{aligned}$$

yolov1损失

$$\begin{aligned}
 & \lambda_{\text{noobj}} \sum_i^{S^2} \sum_j^{\text{anchors}} \mathbb{1}_{ij}^{\text{noobj}} \mathbb{1}_{ij}^{\max_iou < \text{thres}} (O_{ij}^{\text{pred}})^2 \\
 & + \lambda_{\text{obj}} \sum_i^{S^2} \sum_j^{\text{anchors}} \mathbb{1}_{ij}^{\text{obj}} \mathbb{1}_{ij}^{\max_iou >= \text{thres}} (1 - O_{ij}^{\text{pred}})^2 \\
 & + \lambda_{\text{cls}} \sum_i^{S^2} \sum_j^{\text{anchors}} \mathbb{1}_{ij}^{\text{obj}} \sum_{c \in \text{class}} (p_{ij}^{\text{true}}(c) - p_{ij}^{\text{pred}}(c))^2 \\
 & + \lambda_{\text{coord}} \sum_i^{S^2} \sum_j^{\text{anchors}} \mathbb{1}_{ij}^{\text{obj}} \sum_{l \in [x,y,w,h]} (l_{ij}^{\text{true}} - l_{ij}^{\text{pred}})^2
 \end{aligned}$$

yolov2损失

红色框部分早先写错，实际上没有

三、YOLO v2论文解读

(二) 、YOLO v2, 整体结构: 损失计算

1. 目标边框没有包含目标情况下的损失计算, 只计算背景概率损失

$$\lambda_{noobj} \sum_i^{S^2} \sum_j^{anchors} 1_{ij}^{noobj} 1_{ij}^{max_iou < thres} (O_{ij}^{pred})^2$$

其中:

1. $\lambda_{noobj} = 1$ 主要是为了平衡负样本跟正样本比例不均衡问题
2. $1_{ij}^{noobj} = 1$ 表示第i个grid, 第j个anchor, 不存在目标
3. $1_{ij}^{max_iou < thres} = 1$ 表示第i个grid, 第j个anchor, 预测的边框, 跟所有的真实边框的最大iou小于阈值(具体实现里thres=0.6)
4. O_{ij}^{pred} 表示第i个grid, 第j个anchor模型预测的前景概率(也可以理解为预测与真实边框的iou值)

三、YOLO v2论文解读

(二) YOLO v2, 整体结构: 损失计算

2. 目标边框包含目标情况下的损失计算, 只计算前景概率损失

$$\lambda_{obj} \sum_i^{S^2} \sum_j^{anchors} 1_{ij}^{obj} \boxed{1_{ij}^{max_iou >= thres}} (1 - O_{ij}^{pred})^2$$

其中:

1. $\lambda_{obj} = 5$ 主要是为了平衡负样本跟正样本比例不均衡问题
2. $1_{ij}^{obj} = 1$ 表示第*i*个grid, 第*j*个anchor, 存在目标
3. ~~$1_{ij}^{max_iou >= thres} = 1$ 表示第*i*个grid, 第*j*个anchor, 预测的边框, 跟所有的现实边框的最大iou大于等于阈值(具体实现里*thres = 0.6*)~~
4. O_{ij}^{pred} 表示第*i*个grid, 第*j*个anchor模型预测的前景概率(也可以理解为预测与真实边框的iou值)

三、YOLO v2论文解读

(二) 、YOLO v2, 整体结构: 损失计算

3. 目标边框包含目标情况下的损失计算, 类别损失

$$\lambda_{cls} \sum_i^{S^2} \sum_j^{anchors} 1_{ij}^{obj} \sum_{c \in class} (p_{ij}^{true}(c) - p_{ij}^{pred}(c))^2$$

其中:

1. $\lambda_{cls} = 1$

2. $1_{ij}^{obj} = 1$ 表示第*i*个grid, 第*j*个anchor, 存在目标

3. $c \in classes$ 表示遍历所有类别预测结果

4. $p_{ij}^{pred}(c)$ 表示第*i*个grid, 第*j*个anchor, 对类别*c*的预测概率

5. $p_{ij}^{true}(c)$ 表示第*i*个grid, 第*j*个anchor, 类别*c*的真实概率, 这里真实数据如果是目标类别则为1, 否则为0

三、YOLO v2论文解读

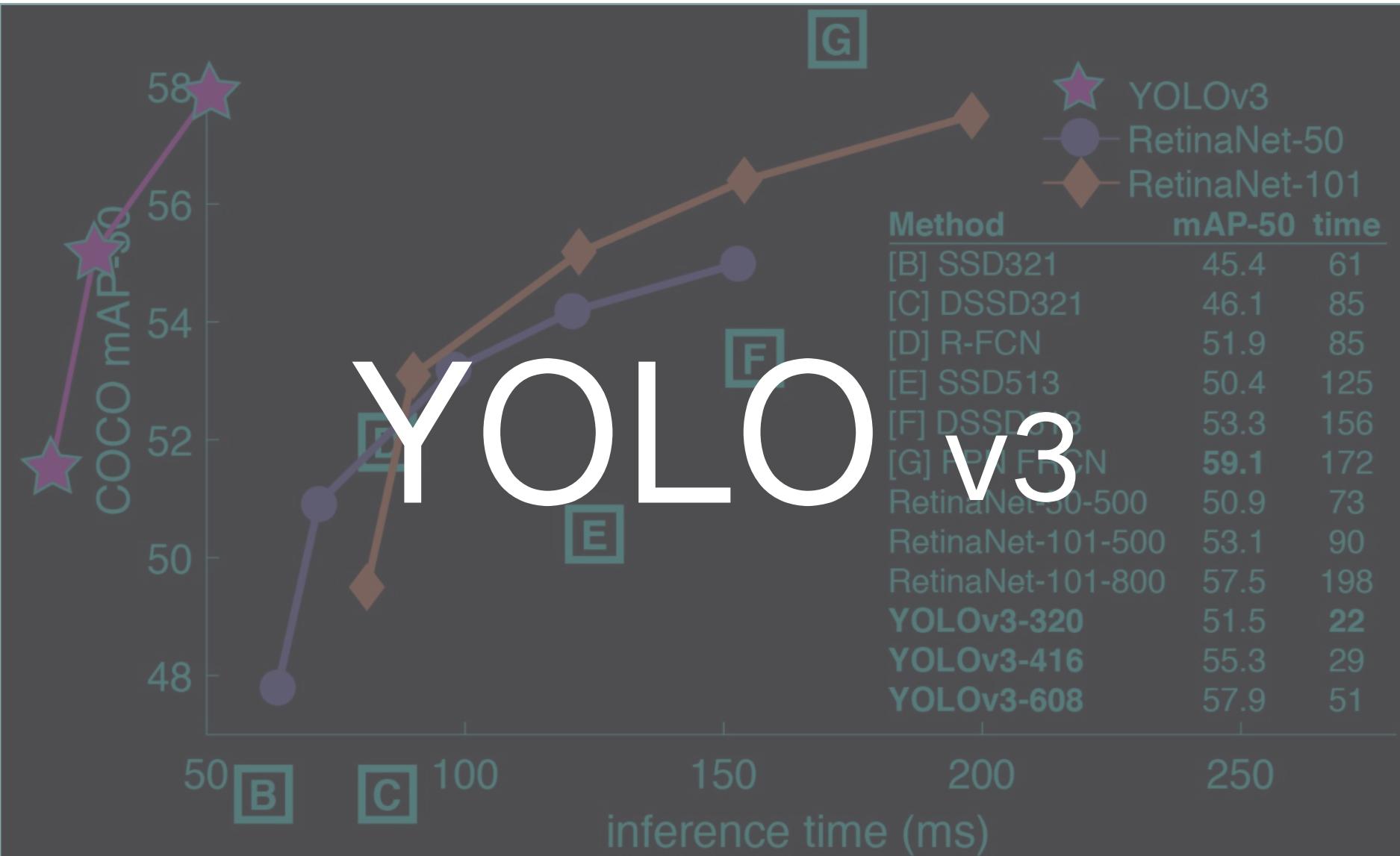
(二) 、YOLO v2, 整体结构: 损失计算

4. 目标边框包含目标情况下的损失计算, 边框损失

$$\lambda_{coord} \sum_i^{S^2} \sum_j^{anchors} 1_{ij}^{obj} \sum_{l \in [x,y,w,h]} (l_{ij}^{true} - l_{ij}^{pred})^2$$

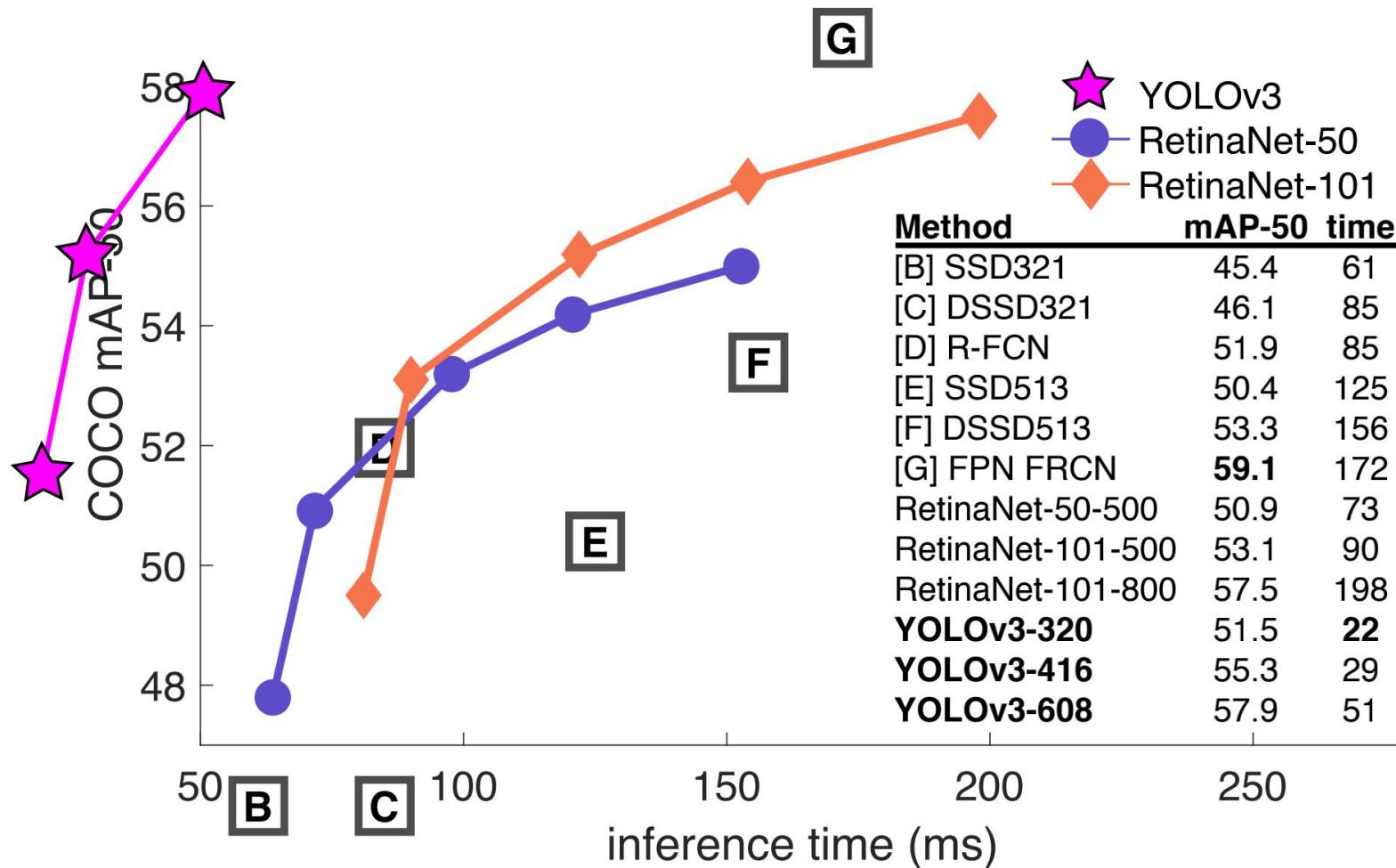
其中:

1. $\lambda_{coord} = 1$
2. $1_{ij}^{obj} = 1$ 表示第*i*个grid, 第*j*个anchor, 存在目标
3. $l \in [x, y, w, h]$ 表示遍历坐标x,y,w,h
4. l_{ij}^{pred} 表示第*i*个grid, 第*j*个anchor, 预测的坐标边框坐标*l*
5. l_{ij}^{true} 表示第*i*个grid, 第*j*个anchor, 真实的坐标边框坐标*l*



四、YOLO v3论文解读

(一) YOLO v3, 性能效果



四、YOLO v3论文解读

(一) 、YOLO v3, 性能效果

	backbone	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
<i>Two-stage methods</i>							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1
<i>One-stage methods</i>							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	40.8	61.1	44.1	24.1	44.2	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

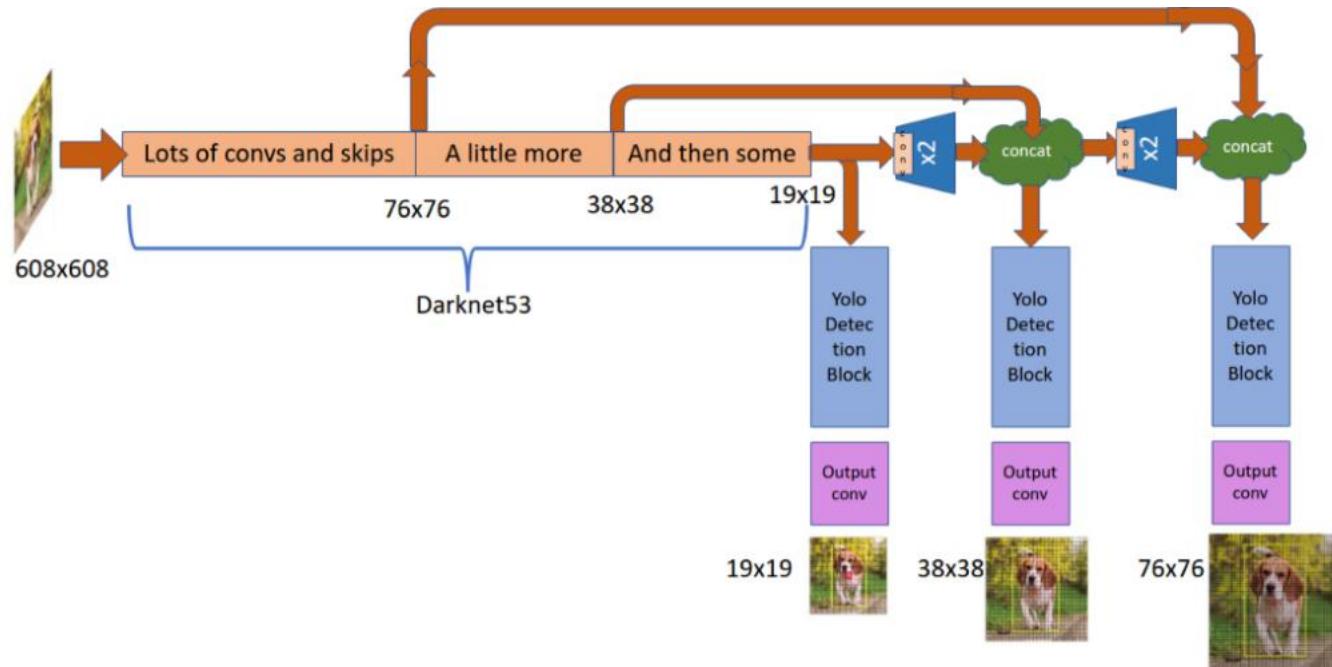
总结:

1. v3的各项mAP均远超v2
2. 对比当时one-stage效果最佳的RetinaNet(ResNetx-101-FPN), 精度方面稍差, 但是速度却大概是其4倍(51ms:198ms)
3. 对比当时two-stage的代表faster-rcnn(ResNet-101-FPN), 精度基本相差无几, 但是速度却大概是其4倍(51ms:200ms)。

四、YOLO v3论文解读

(二) YOLO v3, 几个核心点:

1. 采用darknet-53 + FPN结构



2. 边框预测保持与yolov2一致
3. 沿用yolov2 kmeans生成先验anchors
4. 类别预测改为多分类格式
5. 一些training没有用起来的技巧

四、YOLO v3论文解读

(二) 、YOLO v3, 几个核心点:

1. 采用darknet-53 + FPN结构:

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2 / 2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2 / 2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2 / 2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2 / 2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2 / 2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

darknet-19

Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
1x Convolutional	32	1×1	
Convolutional	64	3×3	
Residual			128×128
Convolutional	128	$3 \times 3 / 2$	64×64
2x Convolutional	64	1×1	
Convolutional	128	3×3	
Residual			64×64
Convolutional	256	$3 \times 3 / 2$	32×32
8x Convolutional	128	1×1	
Convolutional	256	3×3	
Residual			32×32
Convolutional	512	$3 \times 3 / 2$	16×16
8x Convolutional	256	1×1	
Convolutional	512	3×3	
Residual			16×16
Convolutional	1024	$3 \times 3 / 2$	8×8
4x Convolutional	512	1×1	
Convolutional	1024	3×3	
Residual			8×8
Avgpool		Global	
Connected			1000
Softmax			

darknet-53

四、YOLO v3论文解读

(二) 、YOLO v3, 几个核心点:

1. 采用darknet-53 + FPN结构:

darknet-53几个重要点:

1. 采用residual结构, 一定程度上解决了大型网络梯度更新慢问题, 使得模型更容易训练和表达;
2. 采用FPN思想, DarkNet-53最终输出3个中间特征层, 分别为初始输入大小的 $1/8$, $1/16$, $1/32$, 最后将不同尺度的特征两两concat, 一方面使得特征得以重用, 可以加快训练和比较梯度消失, 另一方面可以让下层特征和上层融合, 使得大尺度模板检测的时候能参考小尺度特征。

Type	Filters	Size	Output
1×	Convolutional	32	3×3
	Convolutional	64	$3 \times 3 / 2$
	Convolutional	32	1×1
	Convolutional	64	3×3
2×	Residual		128×128
	Convolutional	128	$3 \times 3 / 2$
	Convolutional	64	1×1
	Convolutional	128	3×3
8×	Residual		64×64
	Convolutional	256	$3 \times 3 / 2$
	Convolutional	128	1×1
	Convolutional	256	3×3
8×	Residual		32×32
	Convolutional	512	$3 \times 3 / 2$
	Convolutional	256	1×1
	Convolutional	512	3×3
4×	Residual		16×16
	Convolutional	1024	$3 \times 3 / 2$
	Convolutional	512	1×1
	Convolutional	1024	3×3
	Residual		8×8
	Avgpool		Global
	Connected		1000
	Softmax		

四、YOLO v3论文解读

(二) 、YOLO v3, 几个核心点:

1. dernet-53之residual结构:

为什么residual结构效果好?

resnet论文[1]中对作者对比了几种plain结构的网络:

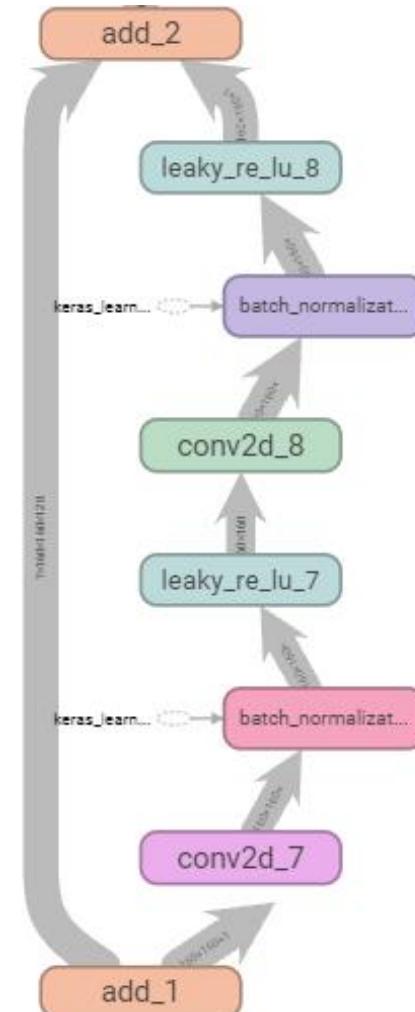
	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	25.03

其中plain结构本身用了bn/relu^[4]模块，梯度本身没有存在弥散，但是越深val-loss越高。

而用了residual结构的resnet越深，val-loss也随之下降，论文[2][3]作者进一步探索了下，总结下来就是浅层权重由于这种连接能得到更快的优化，使得整个模型能训练更好，更快。

参考:

- [1]. [《Deep Residual Learning for Image Recognition》](#)
- [2]. [《Identity Mappings in Deep Residual Networks》](#)
- [3]. [《The Shattered Gradients Problem: If resnets are the answer, then what is the question?》](#)
- [4]. [《Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift》](#)



四、YOLO v3论文解读

(二) YOLO v3, 几个核心点:

1. darknet-53之FPN(top-down+lateral):

FPN效果对比:

RPN	feature	# anchors	lateral?	top-down?	AR ¹⁰⁰	AR ^{1k}	AR _s ^{1k}	AR _m ^{1k}	AR _l ^{1k}
(a) baseline on conv4	C_4	47k			36.1	48.3	32.0	58.7	62.2
(b) baseline on conv5	C_5	12k			36.3	44.9	25.3	55.5	64.2
(c) FPN	$\{P_k\}$	200k	✓	✓	44.0	56.3	44.9	63.4	66.2
<i>Ablation experiments follow:</i>									
(d) bottom-up pyramid	$\{P_k\}$	200k	✓		37.4	49.5	30.5	59.9	68.0
(e) top-down pyramid, w/o lateral	$\{P_k\}$	200k		✓	34.5	46.1	26.5	57.4	64.7
(f) only finest level	P_2	750k	✓	✓	38.4	51.3	35.1	59.7	67.6

论文[1]中作者做了很多实验对比，以FPN(top-down+lateral)为基准对比了其他方案，其中：

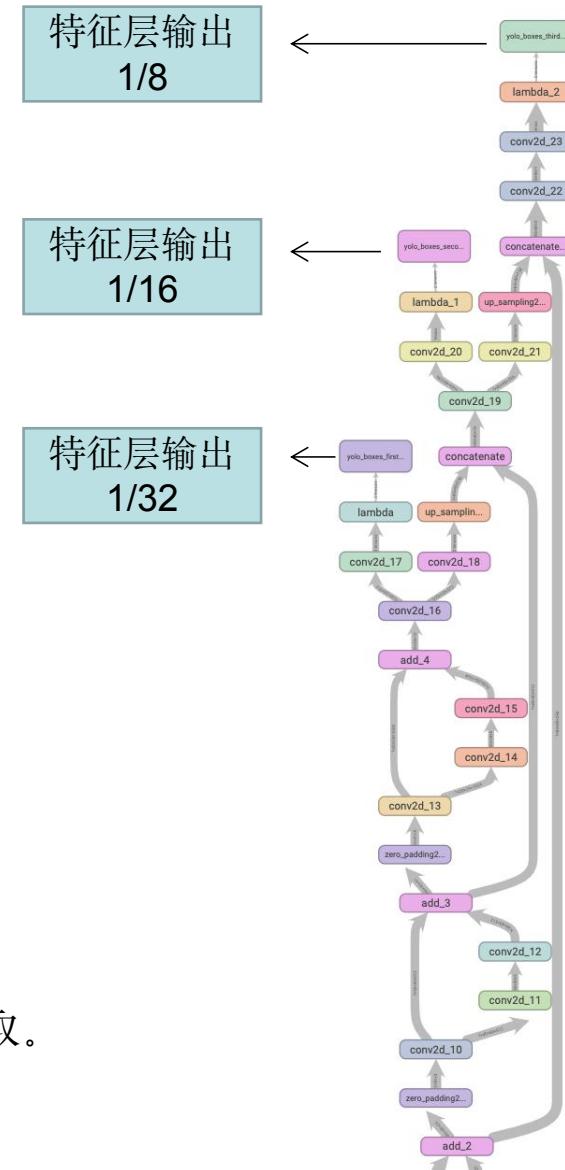
(a)(b)方案只拿C4/C5特征做检测，召回率在小目标和top100/1000上都低于FPN.

(d)不做top-down即没有upsample操作同样AR低

(e)不做lateral即concat或者add,同样AR低

(f)只拿top-dwon+lateral的最后一层，同样AR低

可以看到FPN(top-down+lateral)能提高优化小目标检测特征提取。



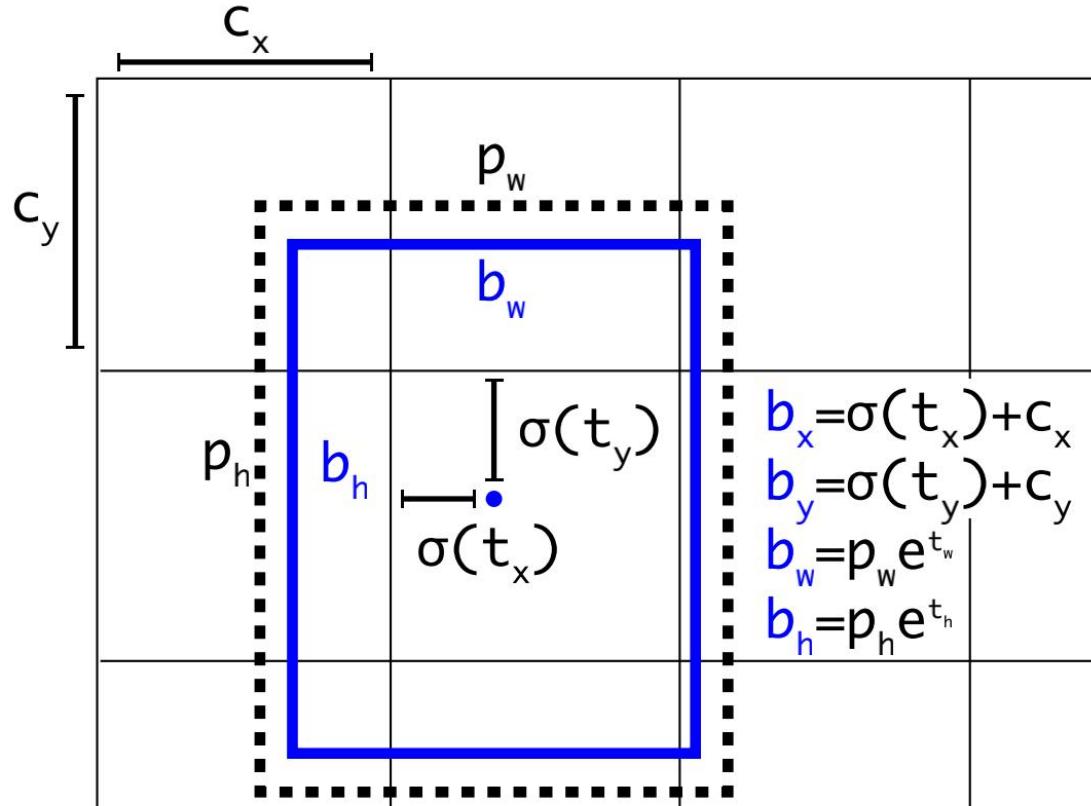
参考:

[1]. [《Feature Pyramid Networks for Object Detection》](#)

四、YOLO v3论文解读

(二) YOLO v3, 几个核心点:

2. 边框预测保持与yolov2一致



其中：

1. t_x, t_y, t_w, t_h 为模型预测输出
2. $\sigma(\cdot)$ 为 sigmoid 函数，归一化 t_x, t_y
3. p_w, p_h 为 anchor 宽高
4. c_x, c_y 为当前 grid 的坐标
5. b_x, b_y, b_w, b_h 为最终预测目标边框

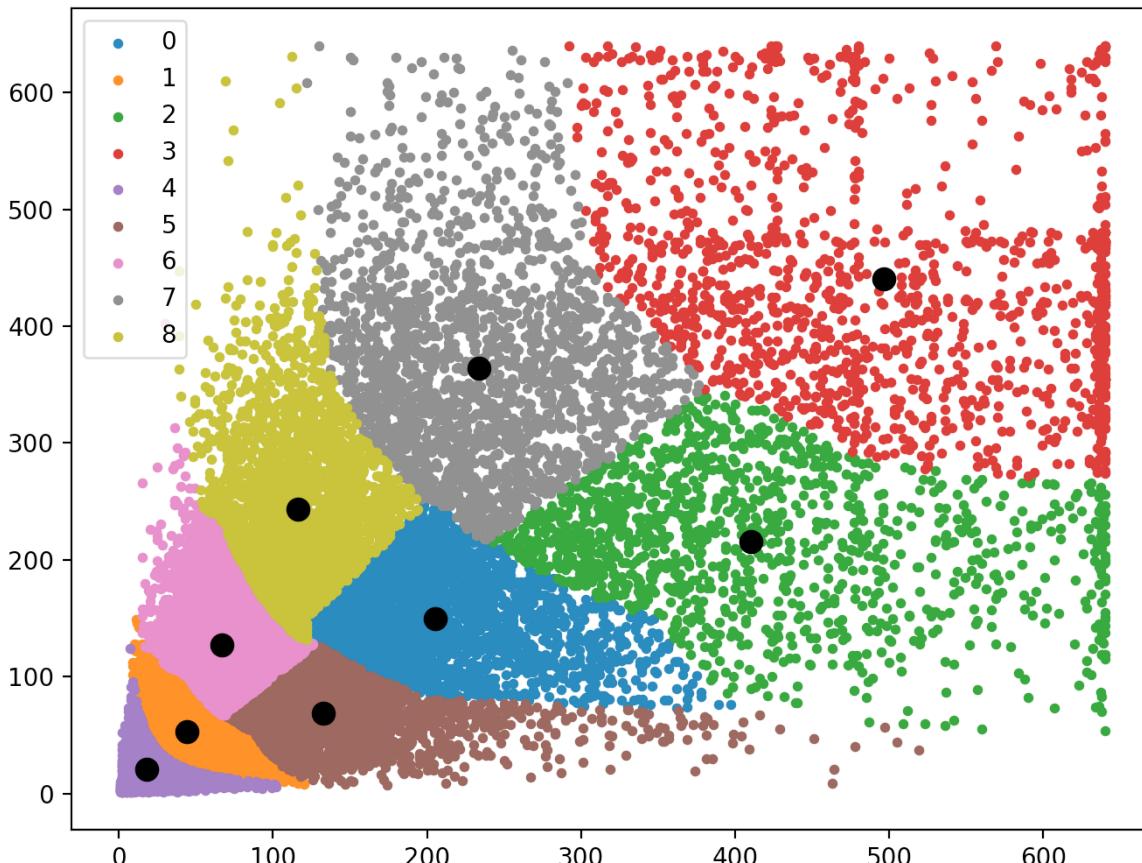
对比 faster-rcnn:

$$\begin{aligned} b_x &= t_x * P_w + P_x \\ b_y &= t_y * P_h + P_y \\ b_w &= P_w * e^{t_w} \\ b_h &= P_h * e^{t_h} \end{aligned}$$

四、YOLO v3论文解读

(二) YOLO v3, 几个核心点:

3. 沿用yolov2 kmeans生成先验anchors



yolov3总共生成9个anchors:

1. 按面积从小到大排序
2. 最小个anchor对应1/32特征
3. 中间三个对应1/16特征
4. 最大三个对应1/8特征

如此，小anchor对应小目标，
大anchor对应大目标检测。

以coco2017-val为例，聚类得到的9个anchor:

```
[[17, 20], [43, 52], [66, 127],  
[132, 69], [116, 243], [205,  
149],[233, 363], [410, 216],  
[496, 440]]
```

四、YOLO v3论文解读

(二) 、YOLO v3, 几个核心点:

4. 类别预测改为多分类格式

yolov3类别预测，损失：

```
class_preds = tf.sigmoid(class_probs)
```

```
class_loss = tf.keras.losses.binary_crossentropy(class_true, class_preds)
```

yolov2类别预测，损失：

```
class_preds = tf.keras.layers.Softmax()(class_probs)
```

```
class_loss = tf.keras.losses.sparse_categorical_crossentropy(class_true, class_preds)
```

但是实际上，如果对数据没有多标签需求，还是建议改yolov2的类别预测和损失格式。

四、YOLO v3论文解读

(二) 、YOLO v3, 几个核心点:

5. 一些training没有用起来的技巧

(1) . 预测anchor的x,y偏移量: tx, ty

$$bx = \sigma(tx) + cx$$

$$by = \sigma(ty) + cy$$

$$bw = Pw * e^{tw}$$

$$bh = Ph * e^{th}$$

替换为faster-rcnn的预测格式:

$$bx = tx * Pw + Px$$

$$by = ty * Ph + Py$$

$$bw = Pw * e^{tw}$$

$$bh = Ph * e^{th}$$

结果作者发现这种格式的bx/by导致了模型的不稳定

但是实际上faster-rcnn的作者没有指出这个问题，可能是yolo不适合这个。

四、YOLO v3论文解读

(二) 、YOLO v3, 几个核心点:

5. 一些training没有用起来的技巧

(2) . 预测x,y偏移量激活函数从logistics替换为linear

```
box_xy, box_wh, objectness, class_probs = tf.split(pred, (2, 2, 1, num_classes), axis=-1)  
box_xy = tf.sigmoid(box_xy)
```

替换为:

```
box_xy, box_wh, objectness, class_probs = tf.split(pred, (2, 2, 1, num_classes), axis=-1)  
box_xy = tf.keras.activations.linear(box_xy)
```

结果作者发现linear激活函数导致了模型的mAP降了几个点

四、YOLO v3论文解读

(二) 、YOLO v3, 几个核心点:

5. 一些training没有用起来的技巧

(3) . 采用focal loss

focal loss的定义可以参考论文 [《Focal Loss for Dense Object Detection》](#)

$$\text{cross_entropy} = \begin{cases} -y\log(p) & \text{if } y = 1 \\ -(1-y)\log(1-p) & \text{if } y = 0 \end{cases}$$

$$\text{focal_loss} = \begin{cases} -\alpha(1-p)^\lambda \log(p) & \text{if } y = 1 \\ -(1-\alpha)p^\lambda \log(1-p) & \text{if } y = 0 \end{cases}$$

上面的公式是普通的交叉熵，下面是focal loss交叉熵，多了一个指数因子 λ 和 α ，focal loss提出的主要目的是解决类别不均衡问题，特别是yolo v3中objness的正负样本不均衡问题，通常 $\lambda = 2$, $\alpha = 0.25$ 可以使得：

- (1) 当正样本对应的预测概率 p 越低时，相应的权重可以更大
- (2) 当负样本对应的预测概率 p 越低时，相应的权重可以更小

但是作者说采用了这种方法降低了mAP 2%.

四、YOLO v3论文解读

(二) 、YOLO v3, 几个核心点:

5. 一些training没有用起来的技巧

(4) . 双IOU阈值采样

参考faster-rcnn在rpn网络里用到的:

当anchor_box边框跟gt_box的iou<0.3, 则为负样本

当anchor_box边框跟gt_box的iou>0.7, 则为正样本

原始yolov3里没有做这一步采样, 加入这个操作后作者发现对模型效果没有提升, 也就去掉了。

最后, 其实这些技巧作者有提到以上的这些技巧也许都需要再细心微调, 就能得到一个比较好的效果, 不能完全否定这些。

四、YOLO v3论文解读

(三) 、YOLO v3, 整体结构:

inputs:

```
image_input = Input(shape=(416, 416, 3))
boxes_input = Input(shape=(None, 5))
```

```
# ([N, grid, grid, anchors, [x1, y1, x2, y2, obj, class]], [], [])
yolo_targets = data_transform(*)
```

model(training):

```
# ([batch, grid, grid, 3, 4+1+num_class],[], [])
output_tensors = yolov3(images)
```

```
# loss
```

```
loc_loss, obj_loss, cls_loss = loss_fn(output_tensors, yolo_targets)
```

四、YOLO v3论文解读

(四) 、YOLO v3: 损失计算

$$\begin{aligned}
 & \lambda_{noobj} \sum_i^{S^2} \sum_j^{anchors} 1_{ij}^{noobj} 1_{ij}^{max_iou < thres} (O_{ij}^{pred})^2 \\
 & + \lambda_{obj} \sum_i^{S^2} \sum_j^{anchors} 1_{ij}^{obj} \boxed{1_{ij}^{max_iou >= thres}} (1 - O_{ij}^{pred})^2 \\
 & + \lambda_{cls} \sum_i^{S^2} \sum_j^{anchors} 1_{ij}^{obj} \sum_{c \in class} (p_{ij}^{true}(c) - p_{ij}^{pred}(c))^2 \\
 & + \lambda_{coord} \sum_i^{S^2} \sum_j^{anchors} 1_{ij}^{obj} \sum_{l \in [x,y,w,h]} (l_{ij}^{true} - l_{ij}^{pred})^2
 \end{aligned}$$

yolov3损失

红色框部分早先写错，实际上没有