后厂理工学院
houchangtech.com

YOLO

HCT CV Course

主要内容

- YOLOv3算法原理
- YOLOv3源码精讲-讲透Anchor机制
- YOLOv3源码精讲-要点深入理解
- YOLOv4要点

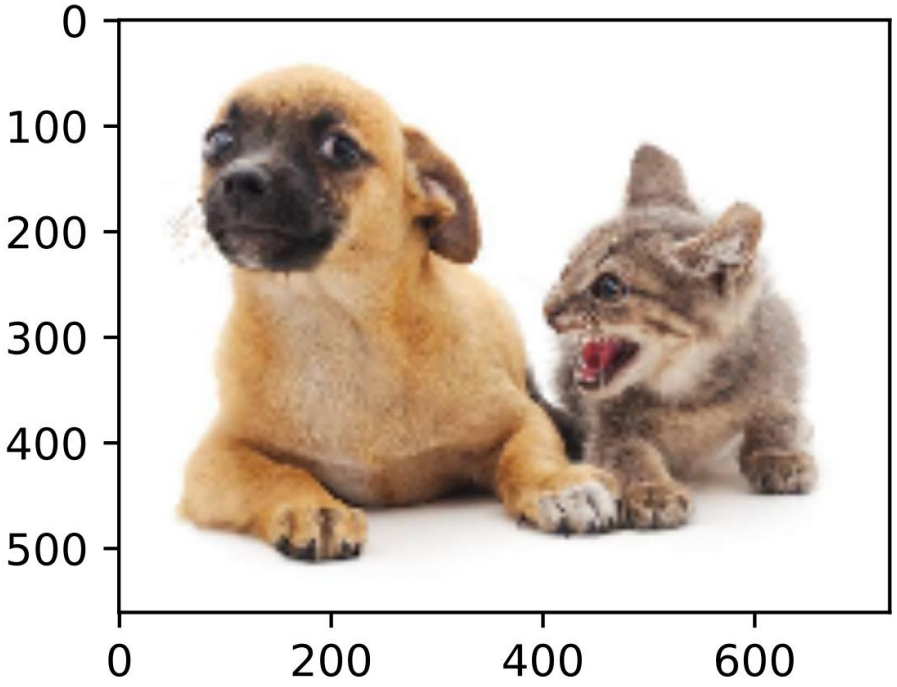
Object Detection

- 应用场景广泛
- 细分领域众多
- 研究非常活跃

Object Detection

- General Detector
- Single Class
  - Face
  - Person
  - Text

Object Detection



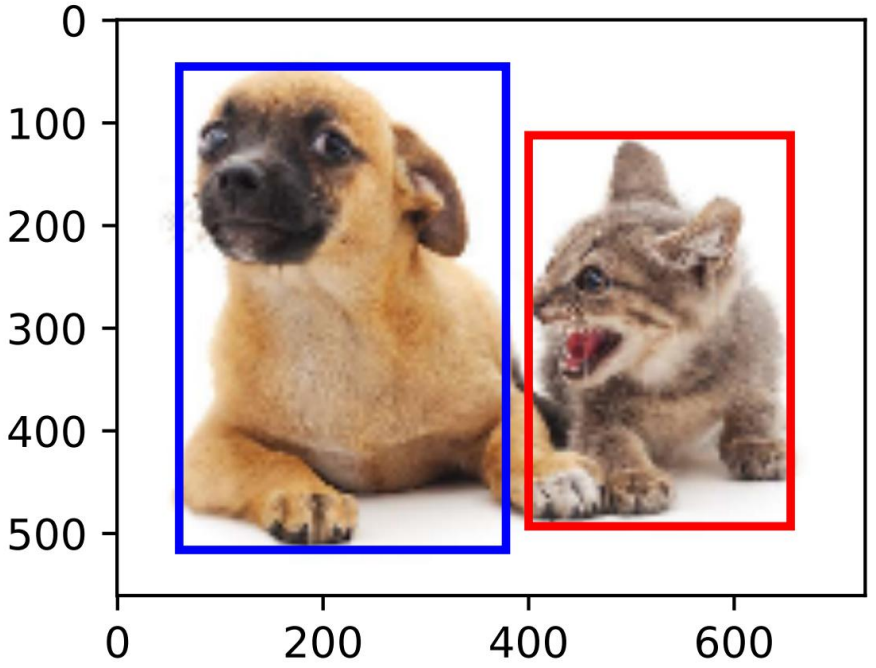


Image Recognition

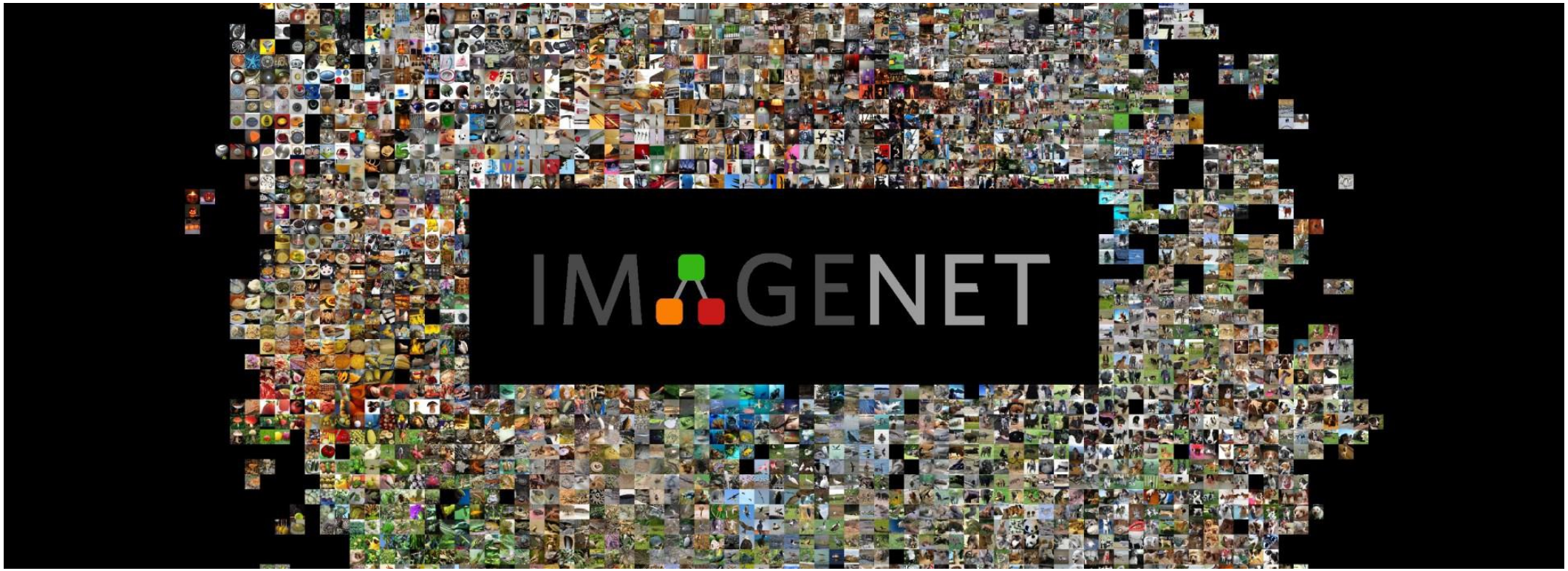
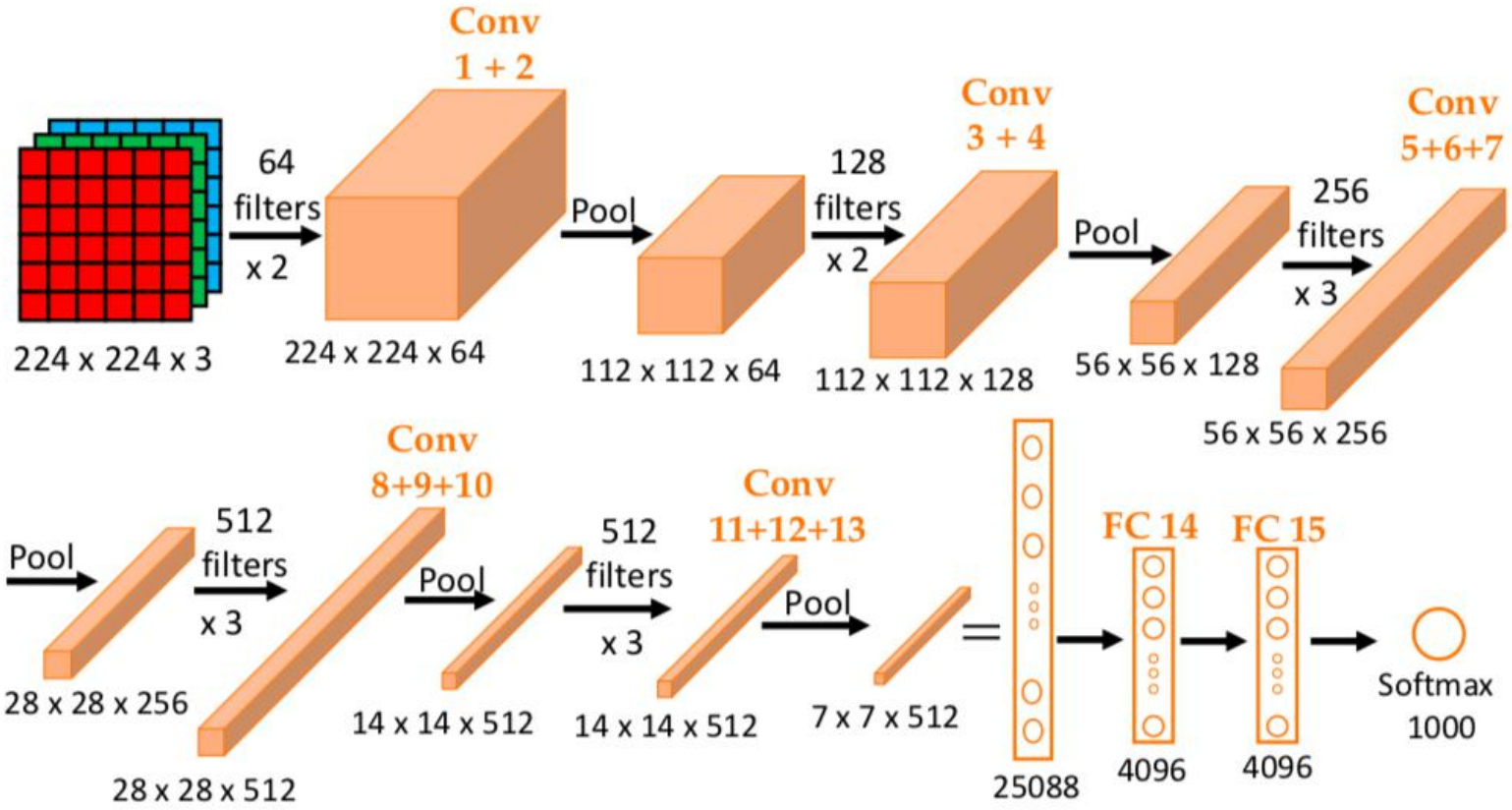


Image Recognition



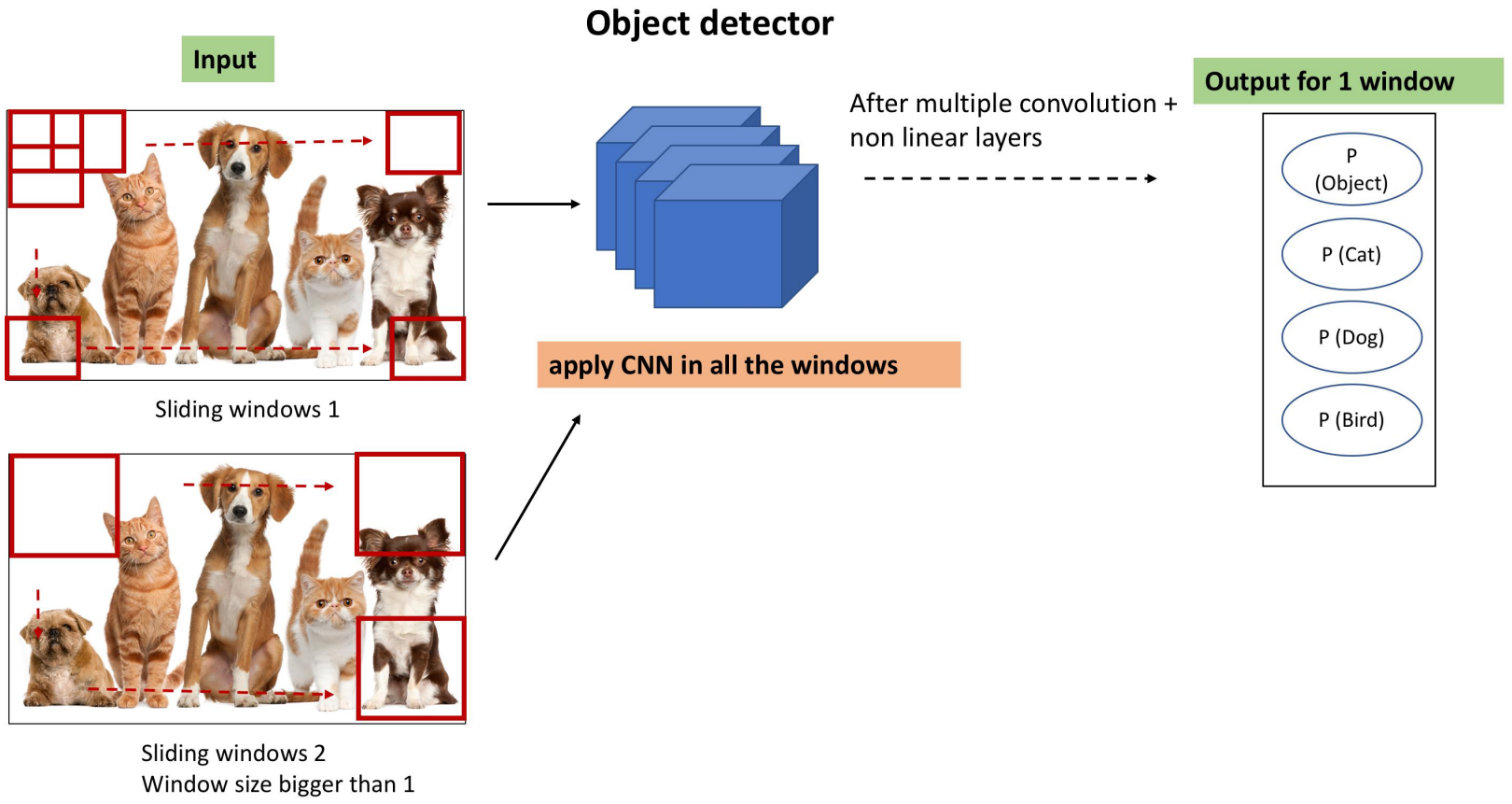
Before YOLO

- Sliding Window
- Region Proposal

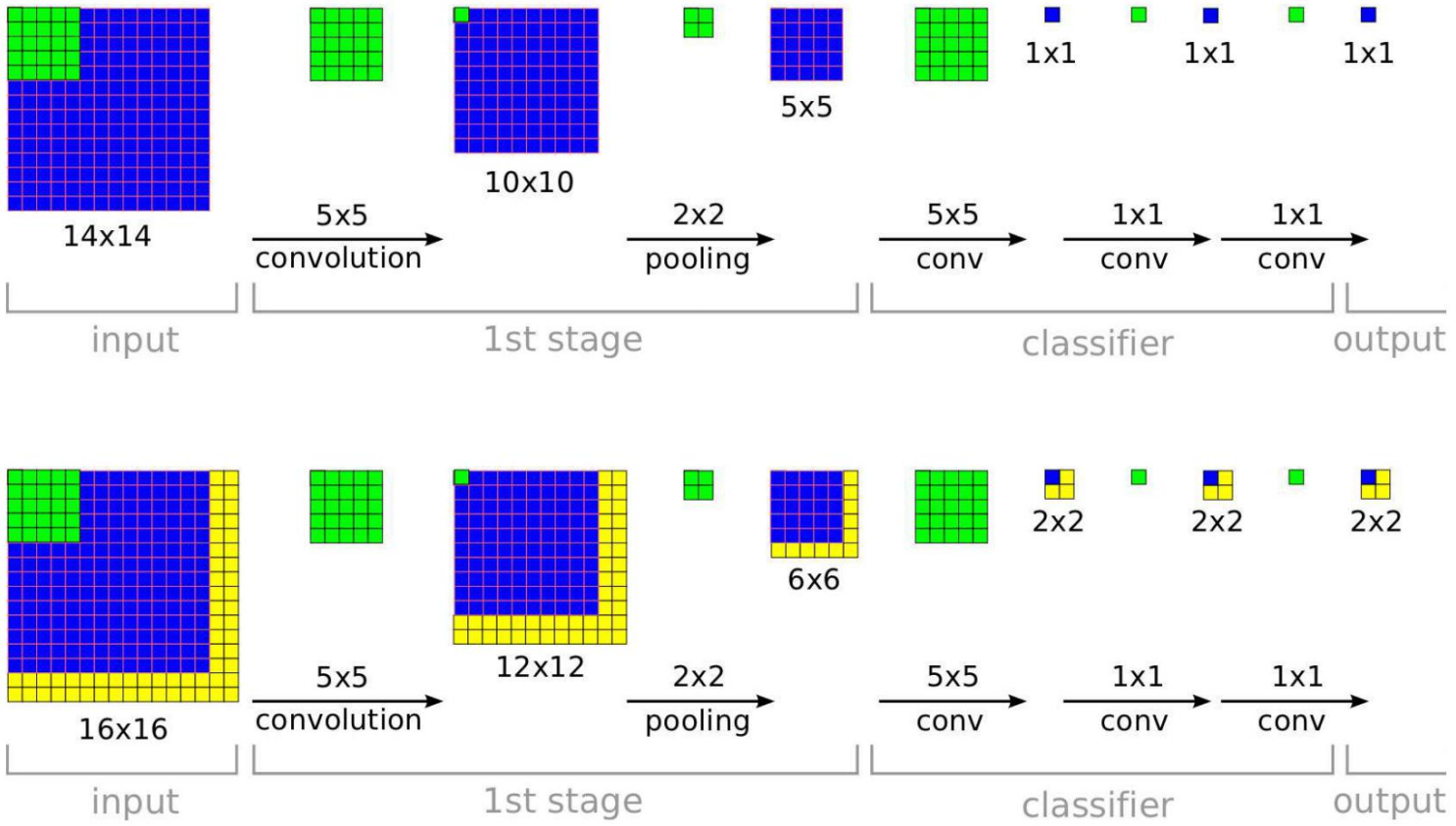
切很多块，对每个块分类，但比较费时

只分类某些可能出现有目标的块

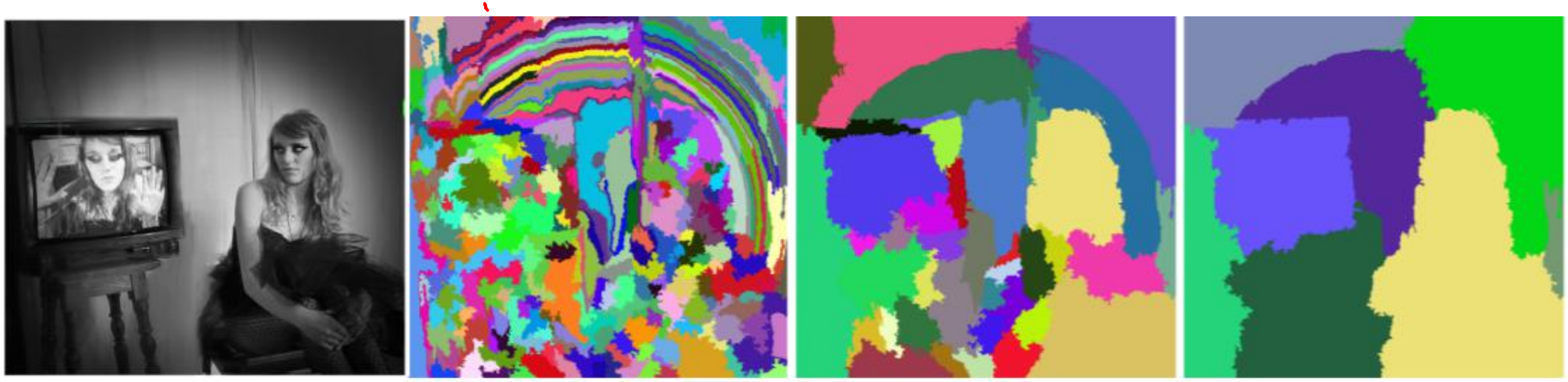
Sliding Window



OverFeat

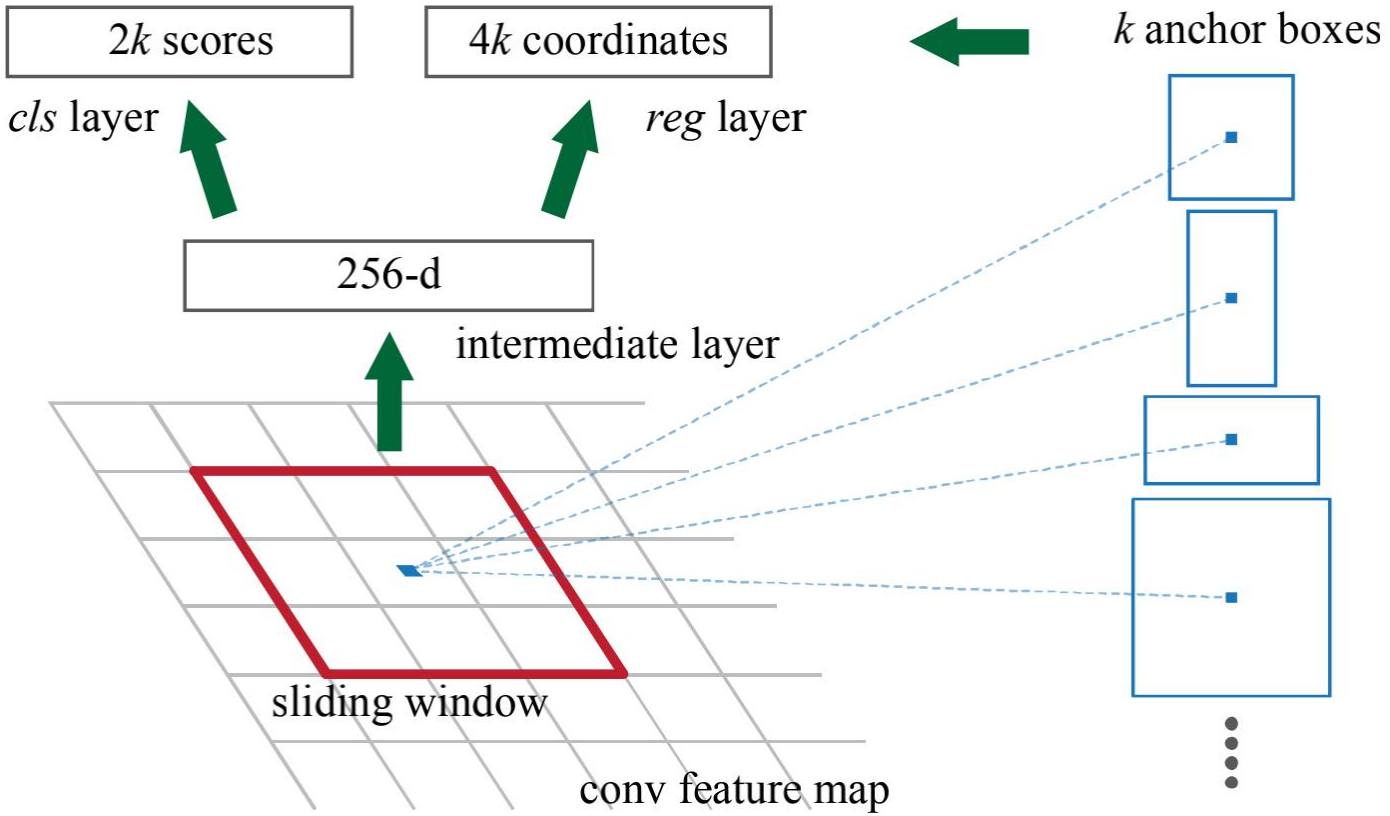


Region Proposal



Input Image | Initial Segmentation | After some iterations | After more iterations

Region Proposal

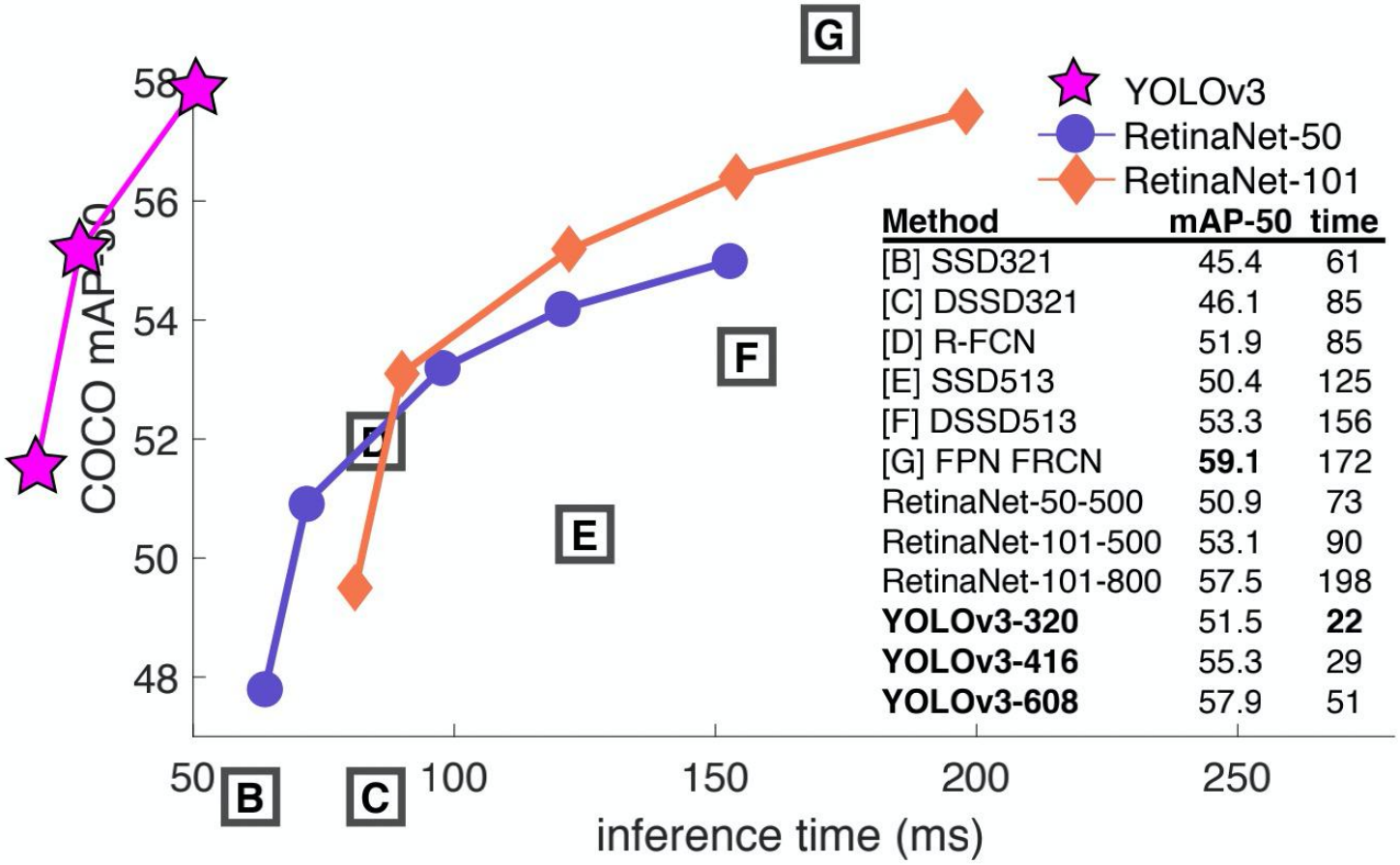


Notes

YOLOv?

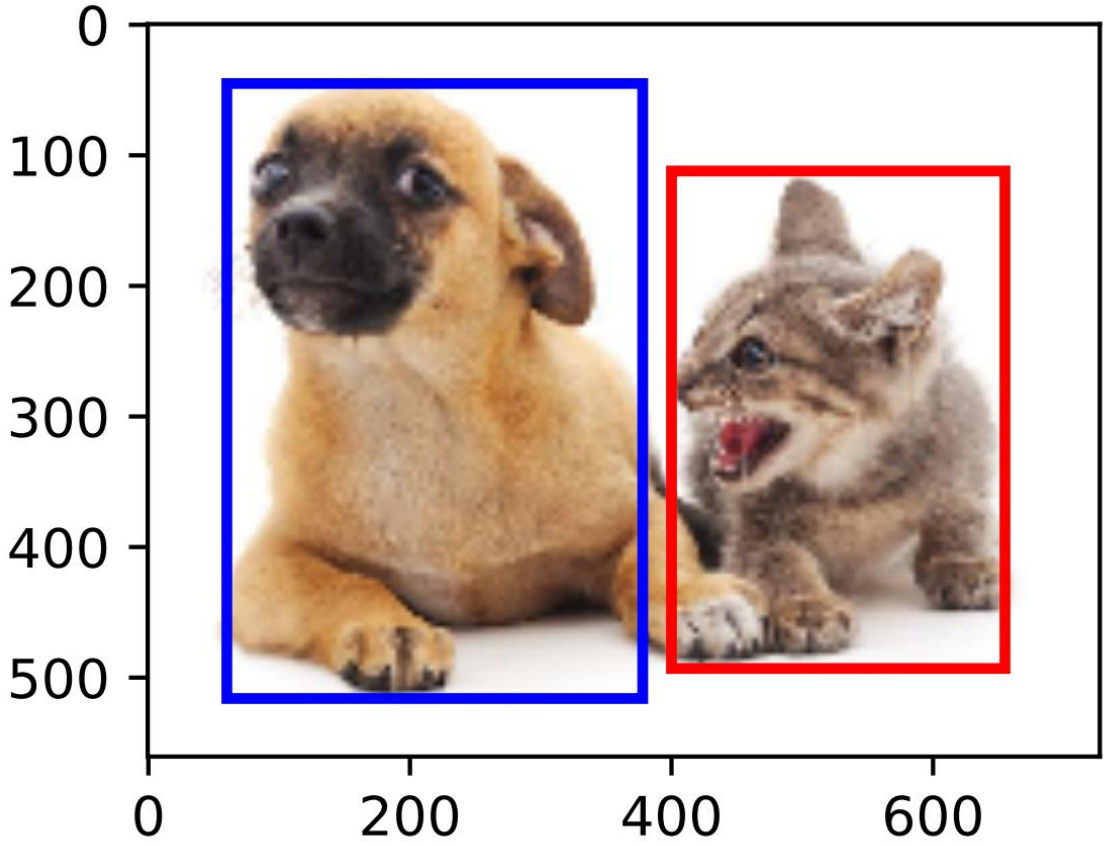
- YOLOv4是最新版本
- YOLOv1/v2/v3是基础

YOLOv3



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

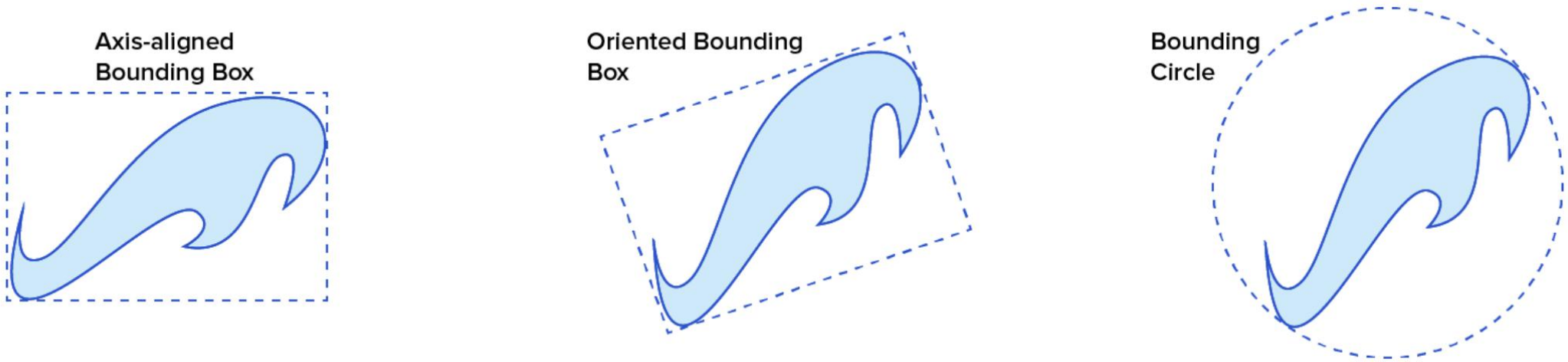
Bounding Box



Bounding Box

- Groud Truth(GT)
- Prediction
- Anchor

Axis-aligned Bounding Box



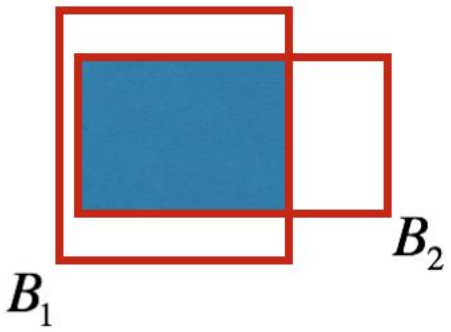
Bounding Box

- 绝对坐标
- 尺度归一化

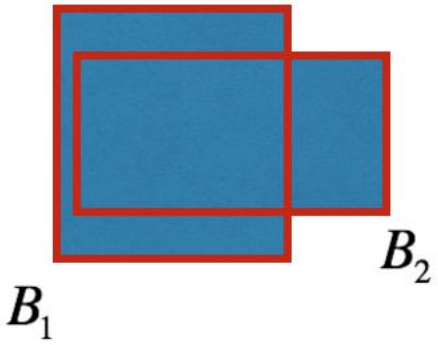
IoU/Jaccard index

- invariant to the scale

Intersection

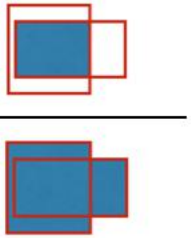


Union

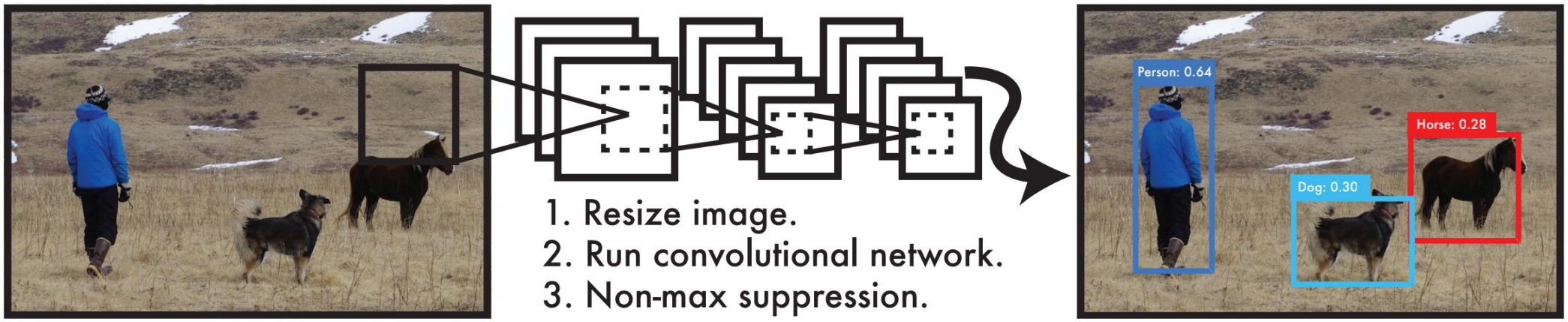


Intersection over Union

$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2} =$$



YOLO



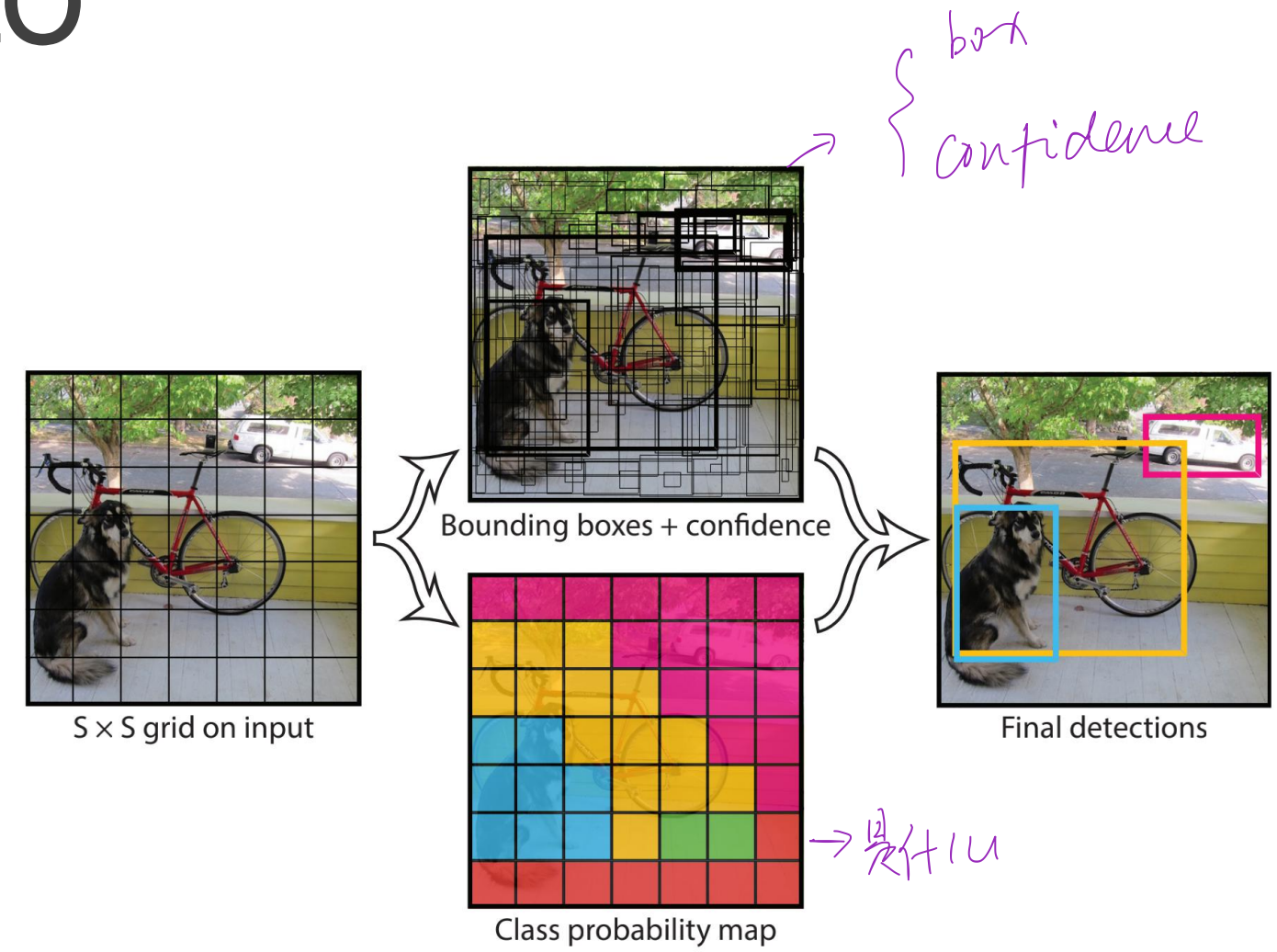
Resize image

Resize image

```python
def letterbox_image(img, inp_dim):
    '''resize image with unchanged aspect ratio using padding'''
    img_w, img_h = img.shape[1], img.shape[0]
    w, h = inp_dim
    new_w = int(img_w * min(w/img_w, h/img_h))
    new_h = int(img_h * min(w/img_w, h/img_h))
    resized_image = cv2.resize(img, (new_w,new_h), interpolation = cv2.INTER_CUBIC)

    canvas = np.full((inp_dim[1], inp_dim[0], 3), 128)

    canvas[(h-new_h)//2:(h-new_h)//2 + new_h,(w-new_w)//2:(w-new_w)//2 + new_w,  :] = resized_image

    return canvas
```

调整尺度至指定

YOLO

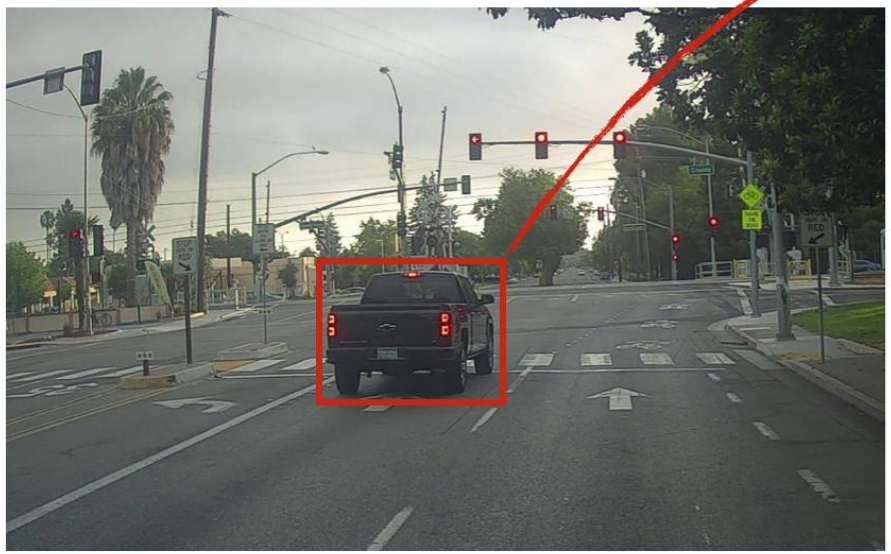


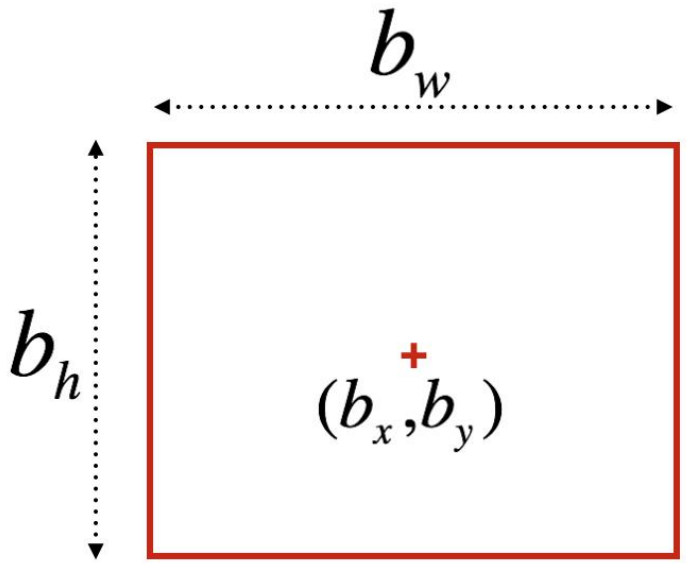
self-driving car

有无物体的概率

bounding box.

$$y = (p_c, b_x, b_y, b_h, b_w, c)$$





$p_c = 1$: confidence of an object being present in the bounding box

$c = 3$: class of the object being detected (here 3 for “car”)

FCN

- YOLO makes use of only convolutional layers, making it a fully convolutional network (FCN).

output is a feature map

- 1 x 1 convolutions
- each cell can predict a fixed number of bounding boxes.
- (B x (5 + C)) entries in the feature map

Architecture

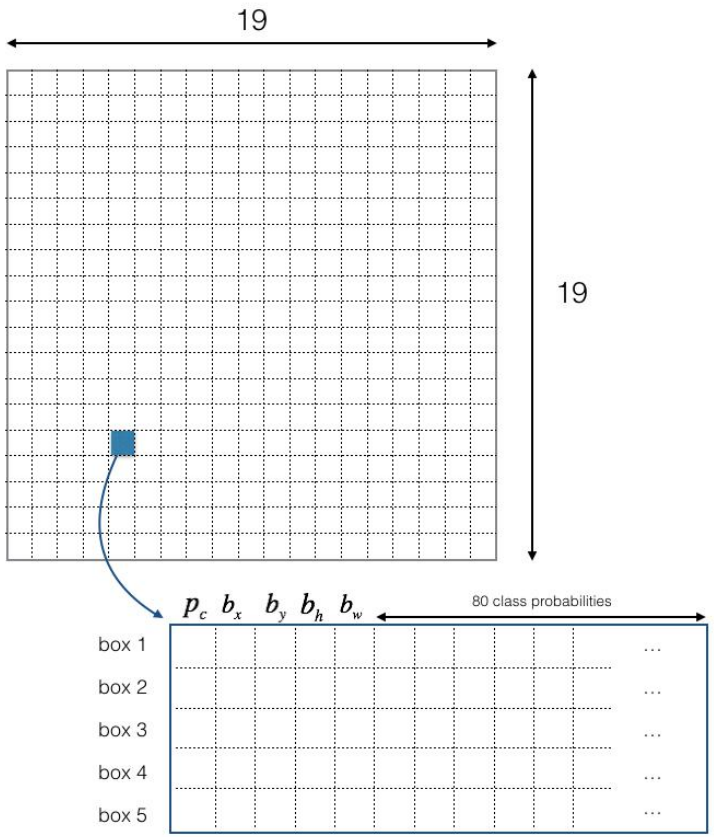
preprocessed image
(608, 608, 3)

Deep CNN

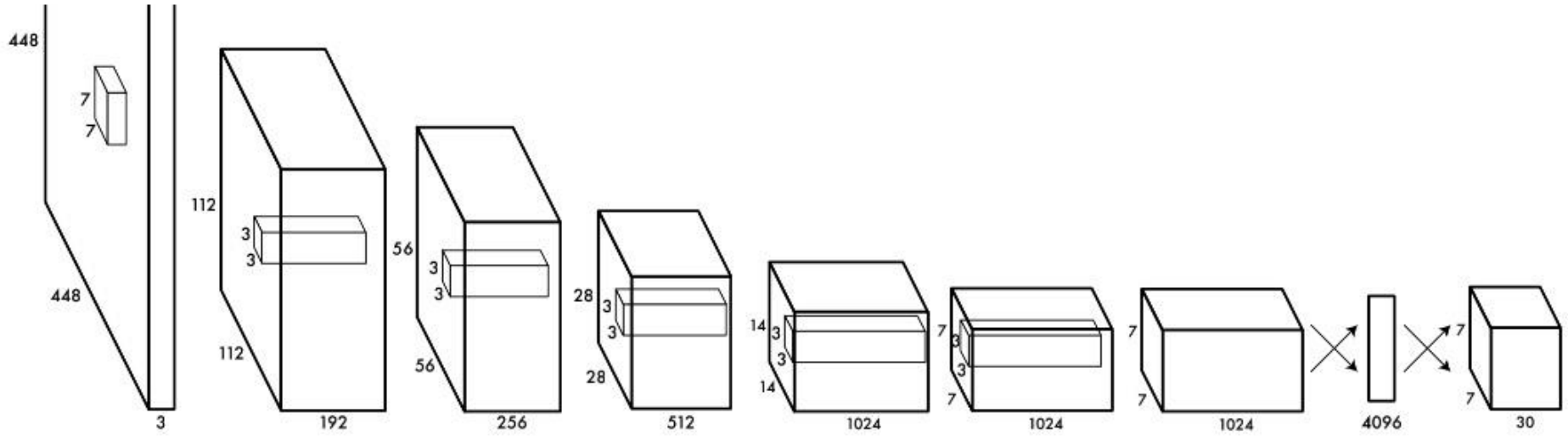
reduction
factor: 32

encoding
(19,19, 5, 85)

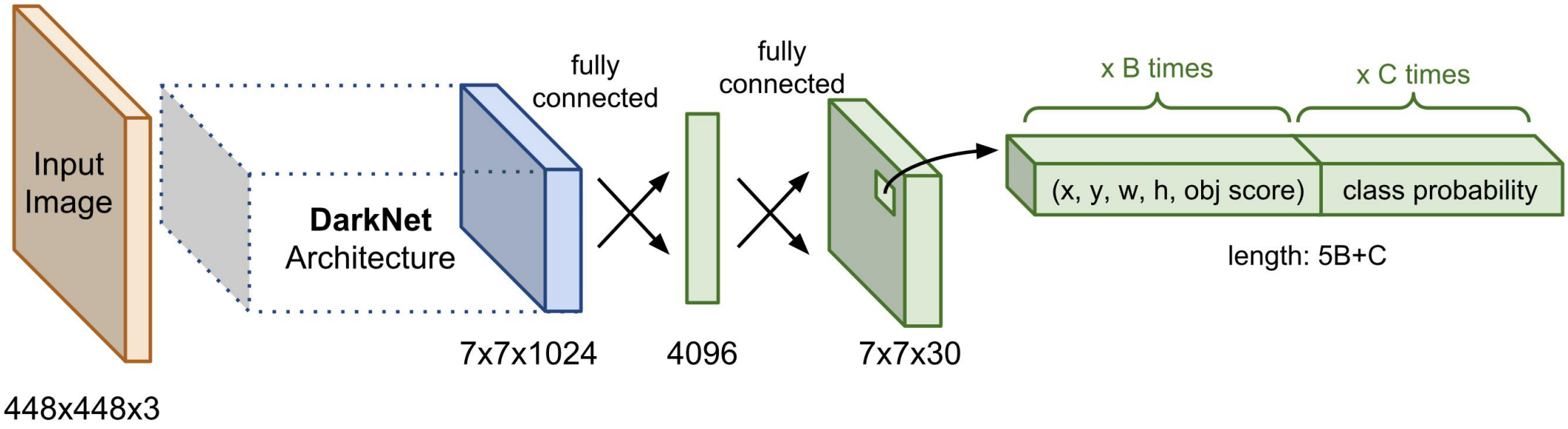
1+4+80.

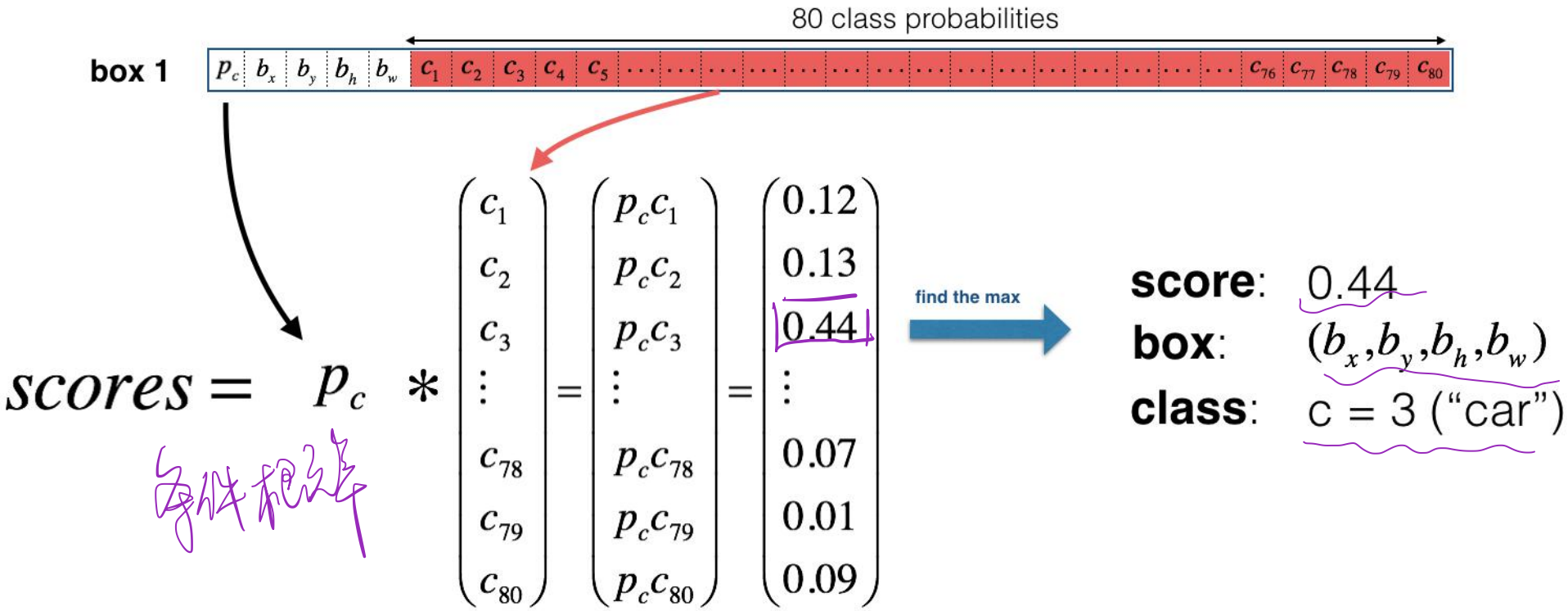


Architecture



Architecture





the box (b_x, b_y, b_h, b_w) has detected c = 3 ("car") with probability score: 0.44

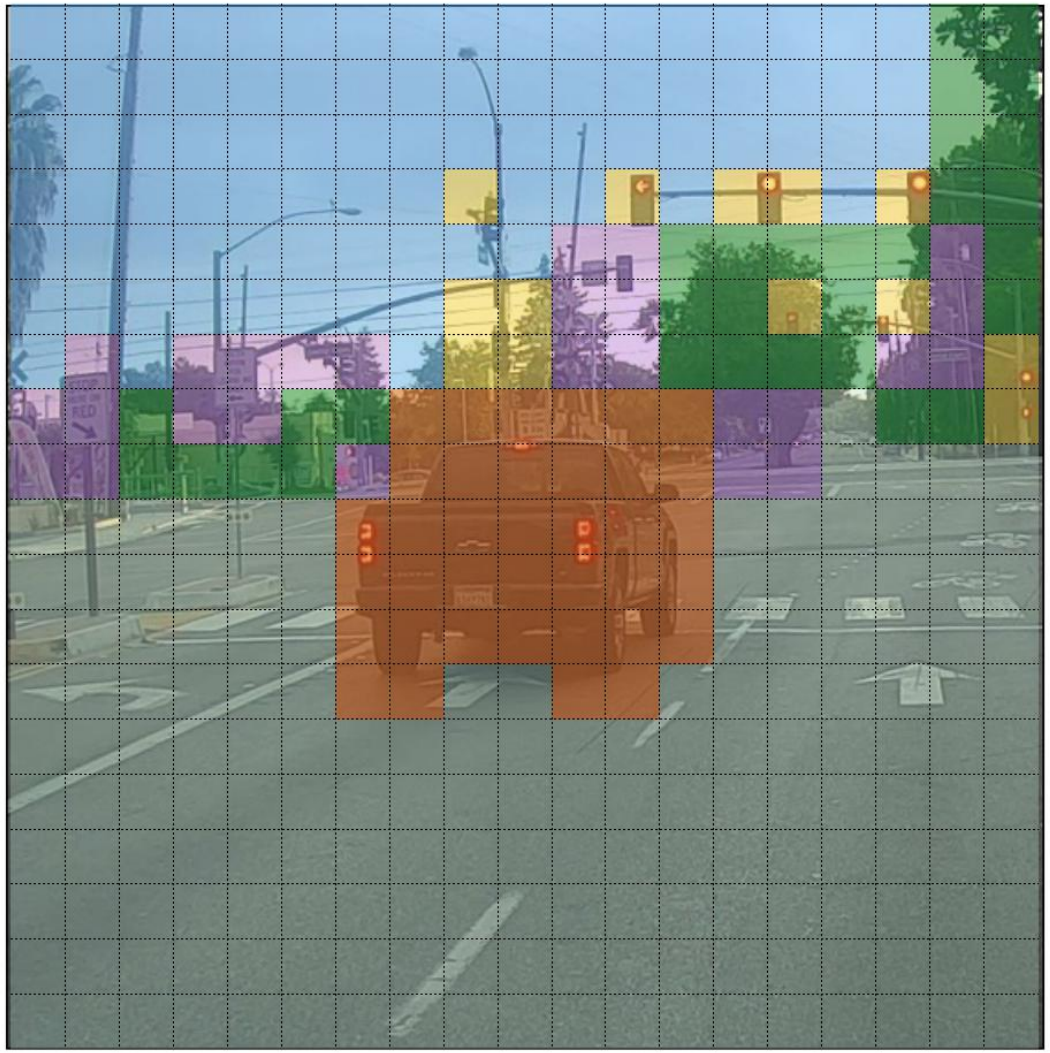
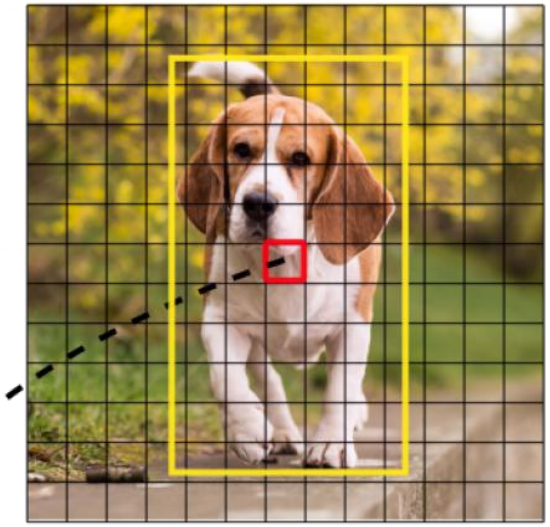
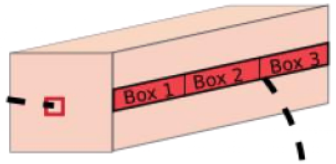


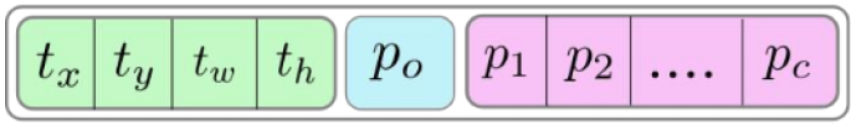
Image Grid. The Red Grid is responsible for detecting the dog



Prediction Feature Map



Attributes of a bounding box



x B

Box Co-ordinates | Objectness Score | Class Scores

Notes

Bounding box regression

- surrogate regression losses

YOLOv1 Loss

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

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

坐标预测

判断第i个网格中的第j个box是否负责这个object

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

含object的box的confidence预测

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

不含object的box的confidence预测

判断是否有object中心落在网格i中

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

类别预测

Loss

- λcoord and λnoobj
- predict the square root of the bounding box width and height instead of the width and height directly

YOLOv3 Loss

$$loss(object) = \sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{obj}(2-w_i*h_i)(-x_i*log(\hat{x_i})-(1-x_i)*log(1-x_i^*))+$$

$$\sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{obj}(2-w_i*h_i)(-y_i*log(\hat{y_i})-(1-y_i)*log(1-y_i^*))+$$

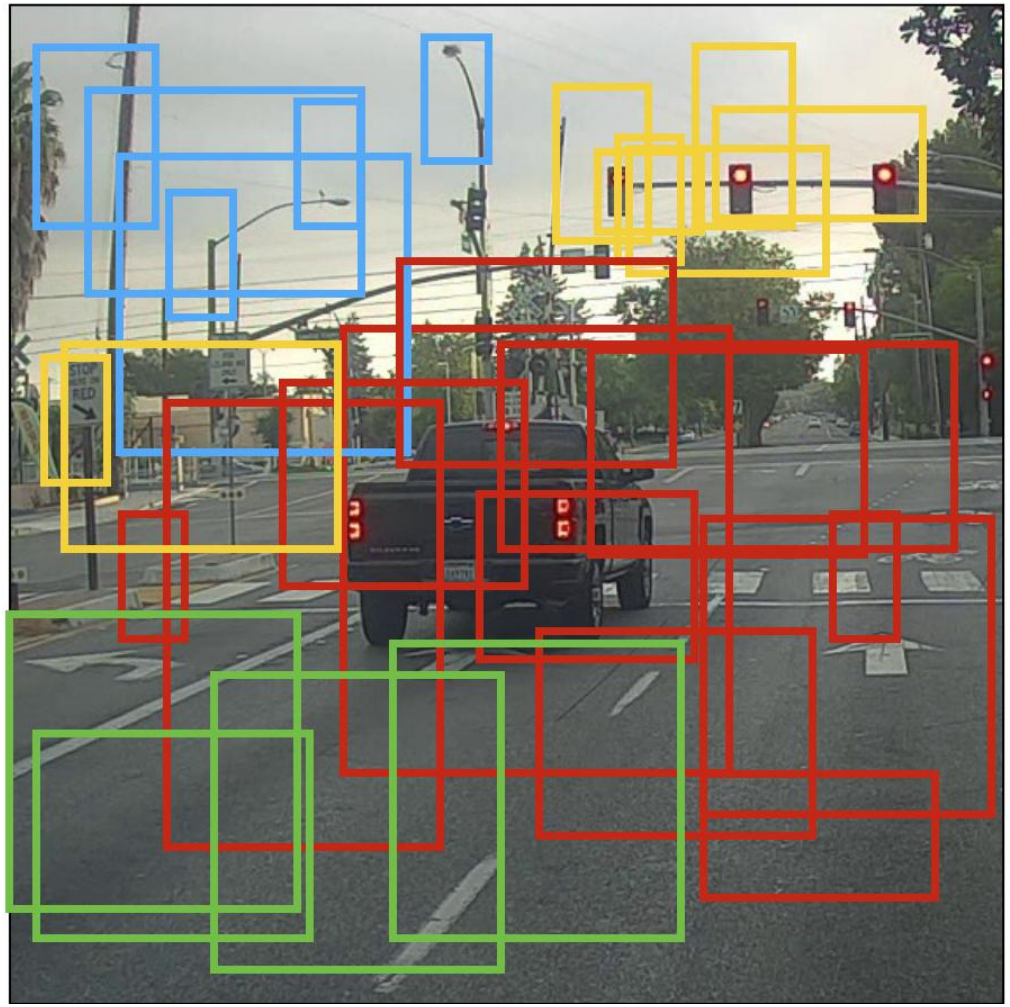
$$\sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{obj}(2-w_i*h_i)[(w_i-w_i^*)^2+(h_i-h_i^*)^2]-$$

$$\sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{obj}[C_i log(C_i^*)+(1-C_i)log(1-C_i^*)]-$$

$$\sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{noobj}[C_i log(C_i^*)+(1-C_i)log(1-C_i^*)]-$$

$$\sum_{i=0}^{K\times K}\sum_{j=0}^{M} I_{ij}^{obj}\sum_{c\in classes}[p_i(c)log(p_i^*(c))+(1-p_i(c))log(1-p_i^*(c))]$$

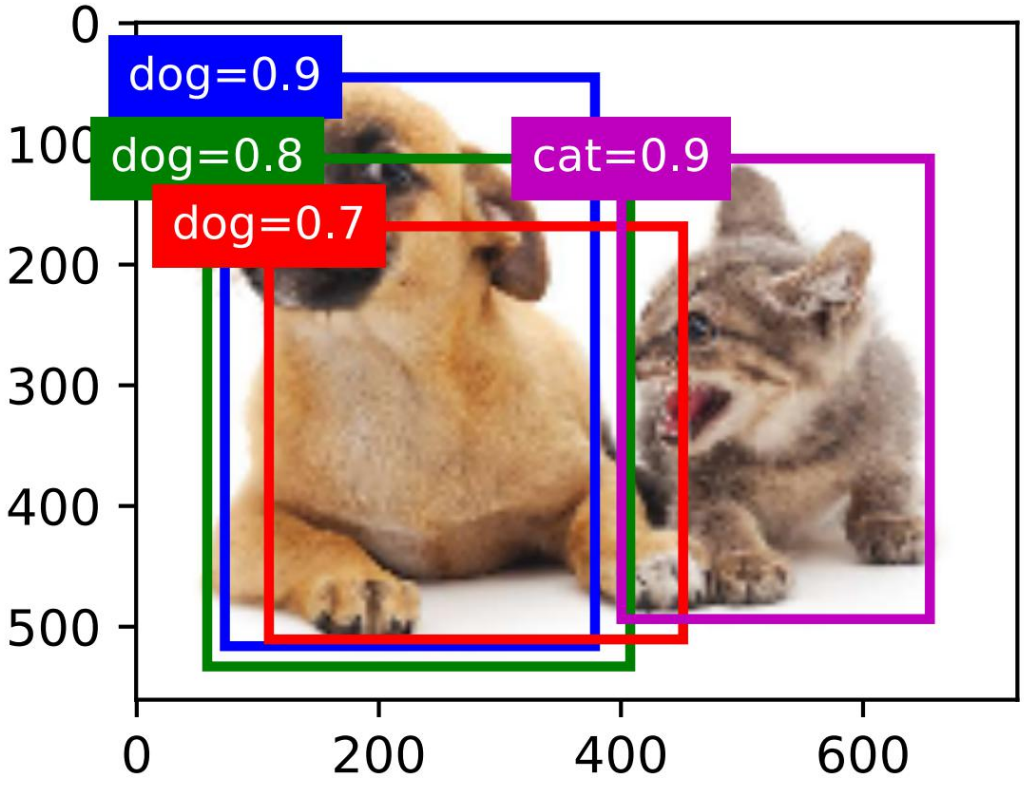
Predictions

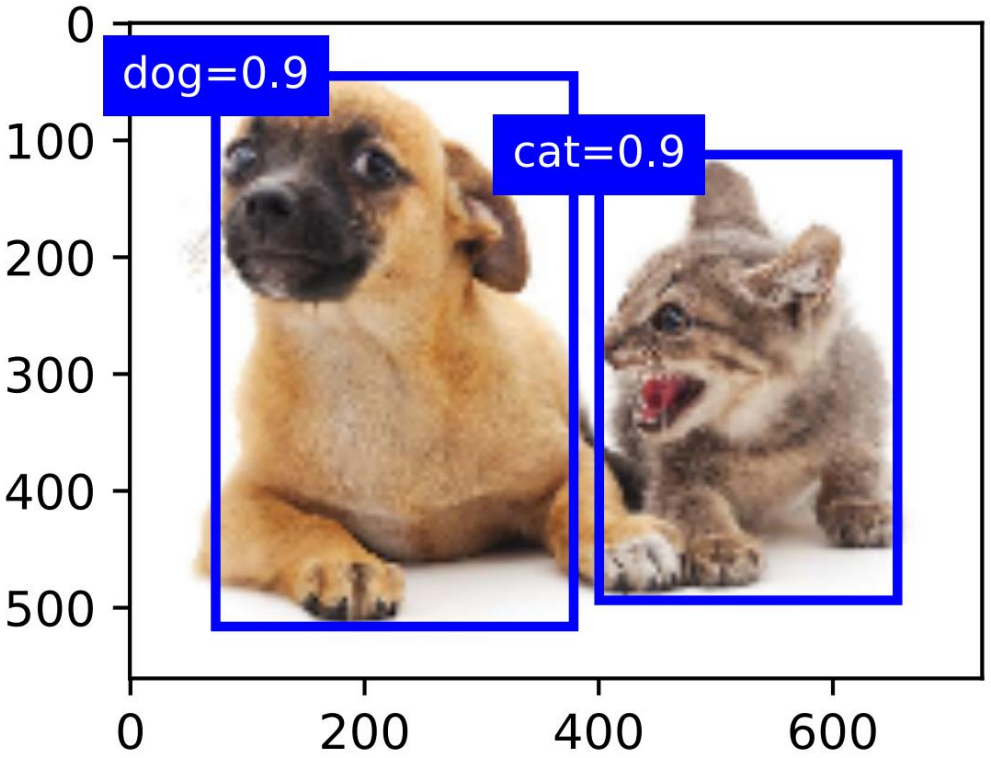


→ NMS →

Non-Maximum Suppression

把概率最大的挑出来，IOU(C)>0.8 去掉





Notes

YOLOv1的缺点

- relatively low recall compared to region proposal-based methods

- a significant number of localization errors

from YOLOv1 to YOLOv2

	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**

项目实战

- 使用YOLO训练自己的检测模型

Convolutional With Anchor Boxes

- Predicting offsets instead of coordinates simplifies the problem and makes it easier for the network to learn

Anchor

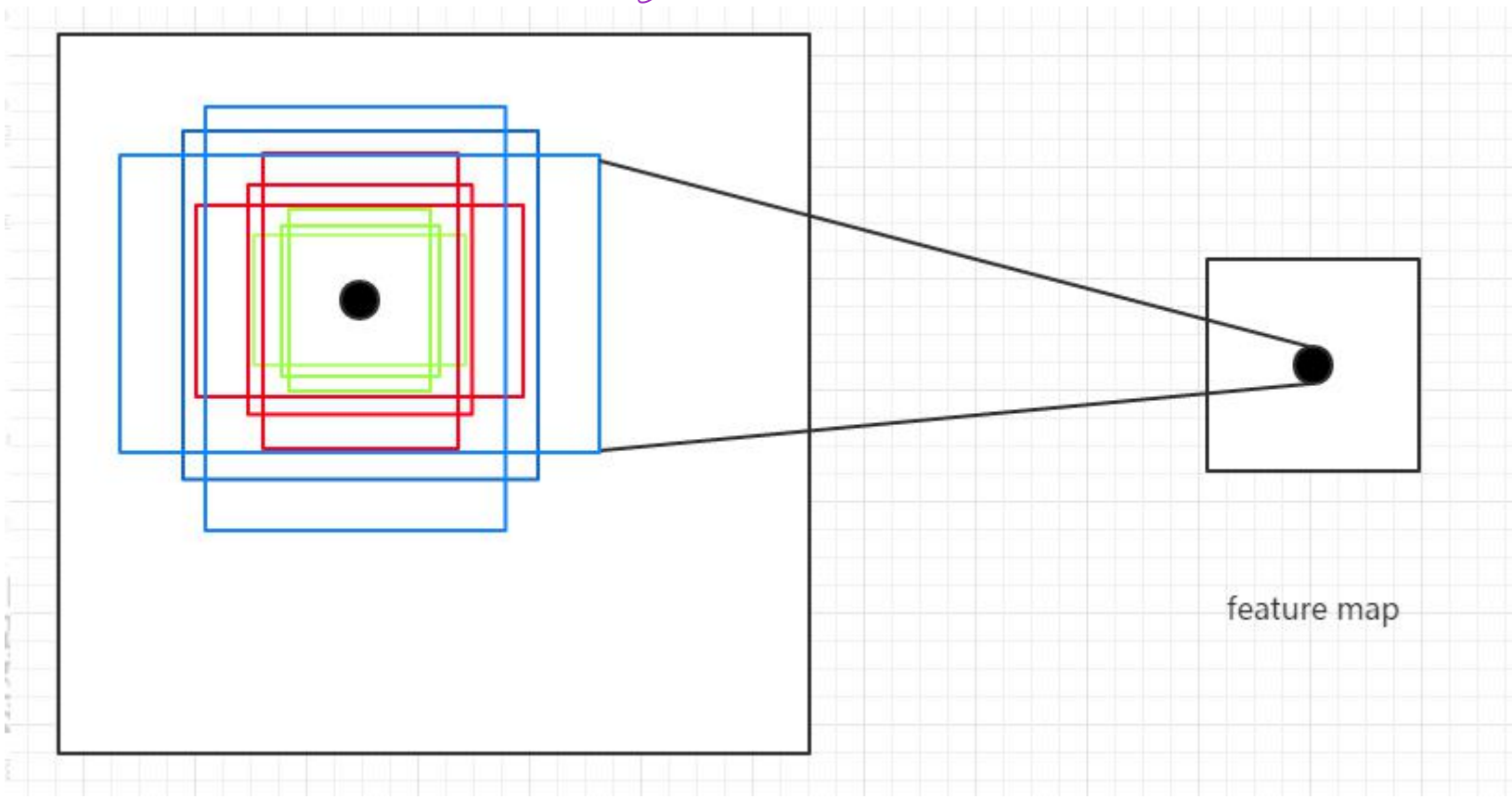
- decouple the class prediction mechanism from the spatial location and instead predict class and objectness

Anchor

- Without anchor boxes our intermediate model gets 69.5 mAP with a recall of 81%
- With anchor boxes our model gets 69.2 mAP with a recall of 88%

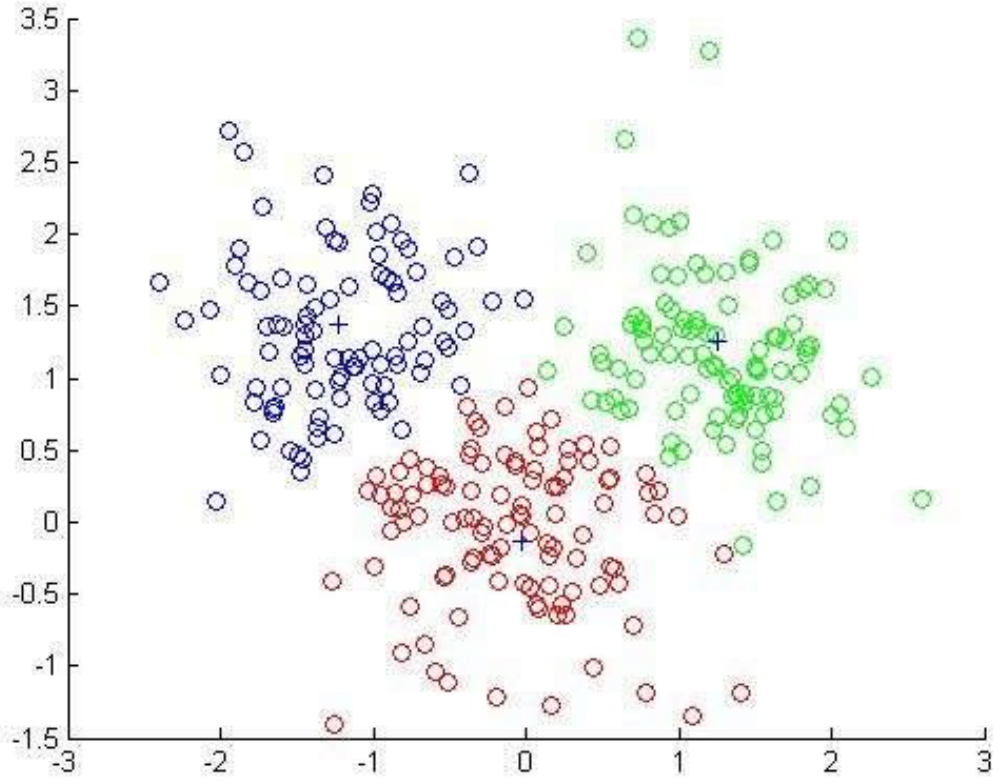
Anchor

sliding window.



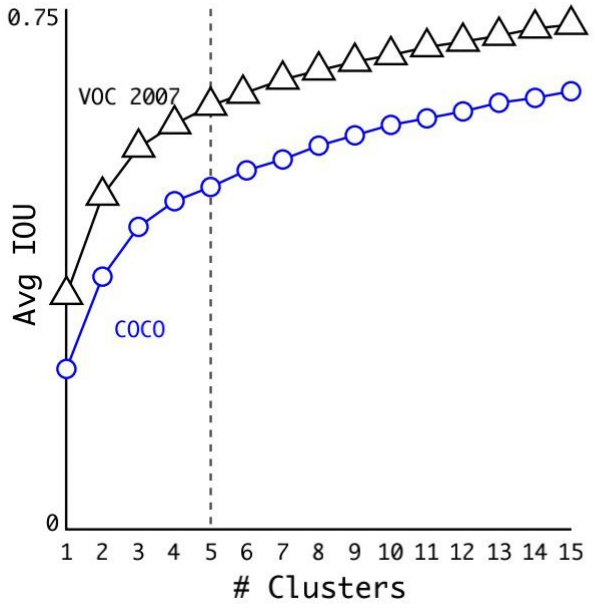
K-Means

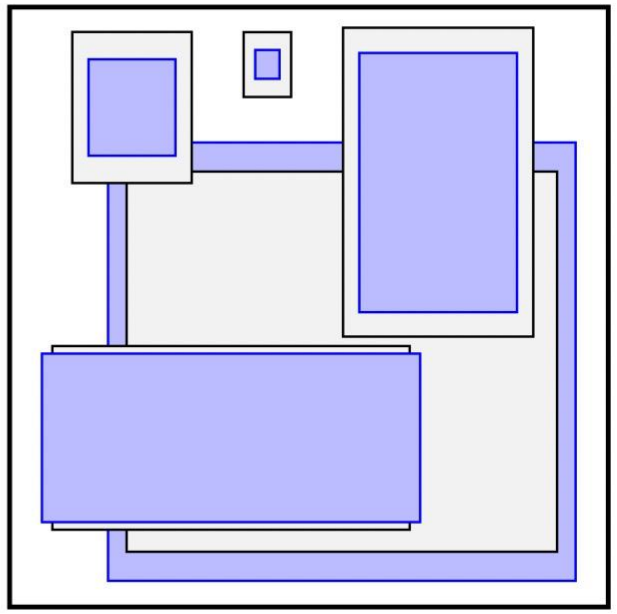
K-Means



Dimension Clusters

- d(box, centroid) = 1- IOU(box, centroid)





YOLOv2 anchor box size

- be relative to the feature size in the last layer. Since the network was downsampling by 32 this means it was relative to 32 pixels so an anchor of 9x9 was actually 288px x 288px.

YOLOv3 anchor box size

- In YOLOv3 anchor sizes are actual pixel values. this simplifies a lot of stuff and was only a little bit harder to implement

Anchor(COCO)

9个anchor，3个不同尺度

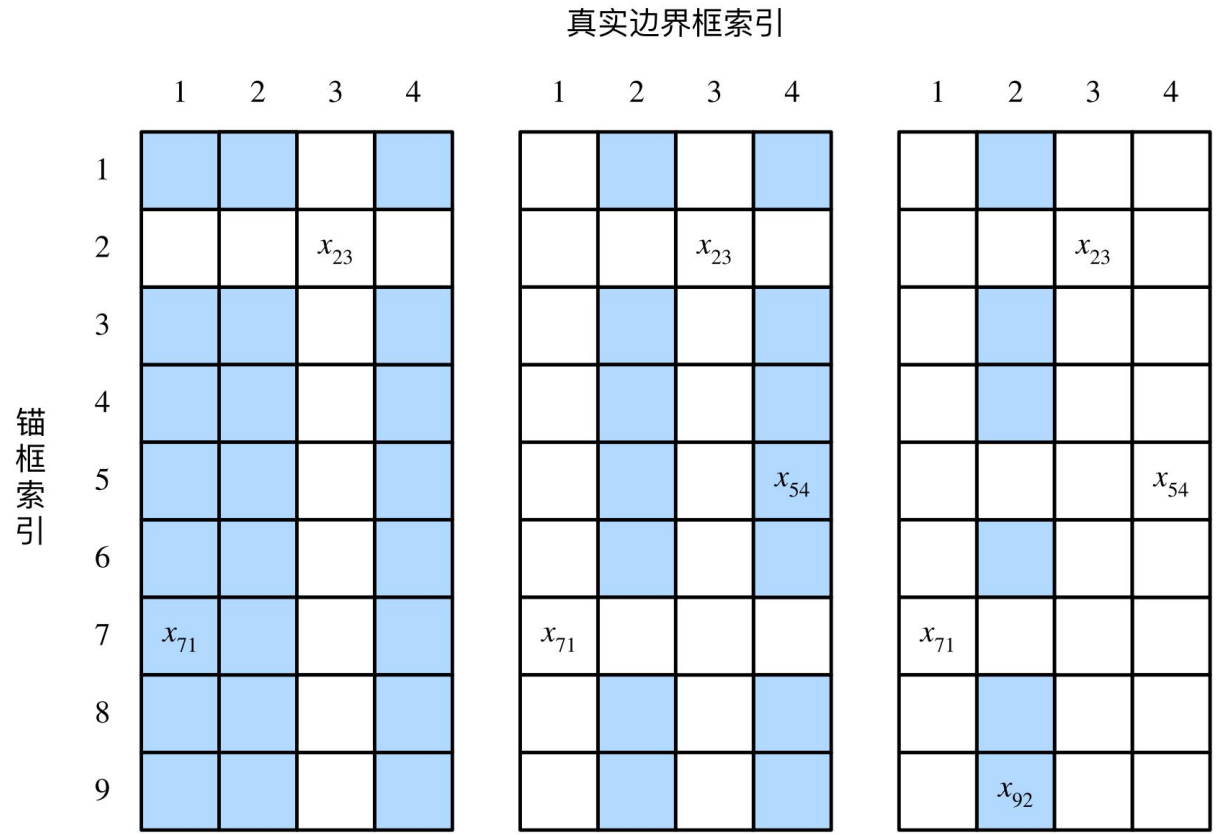
预测不同物体（大小）

[10, 13], [16, 30], [33, 23]

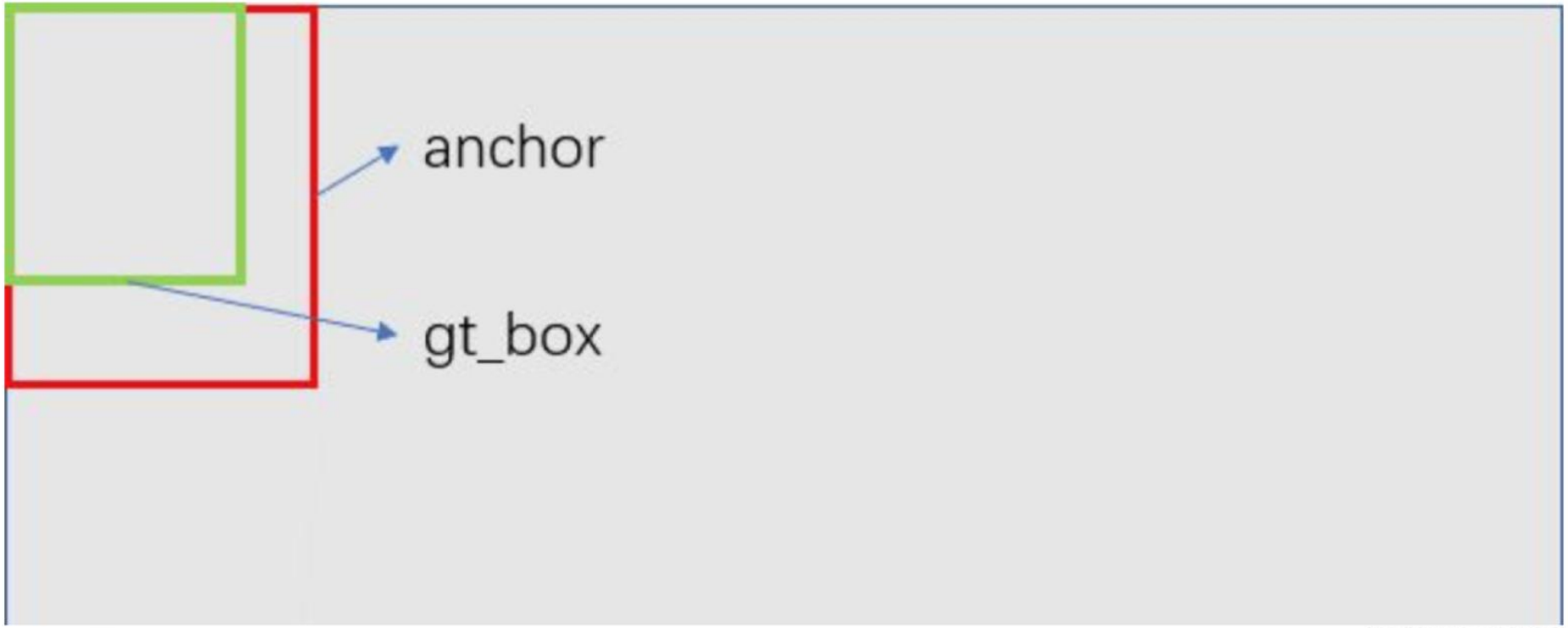
[30, 61], [62, 45], [59, 119]

[116, 90], [156, 198], [373, 326]

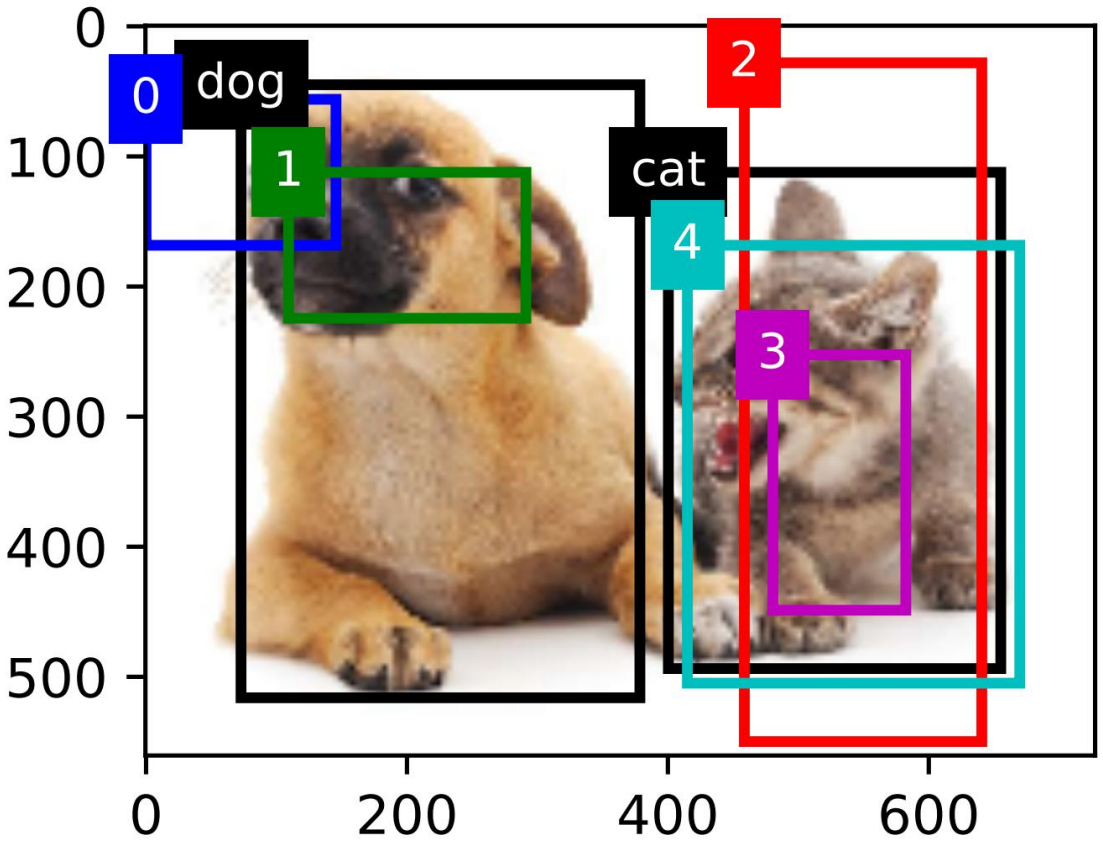
为Anchor分配GT



为Anchor分配GT



为Anchor分配GT

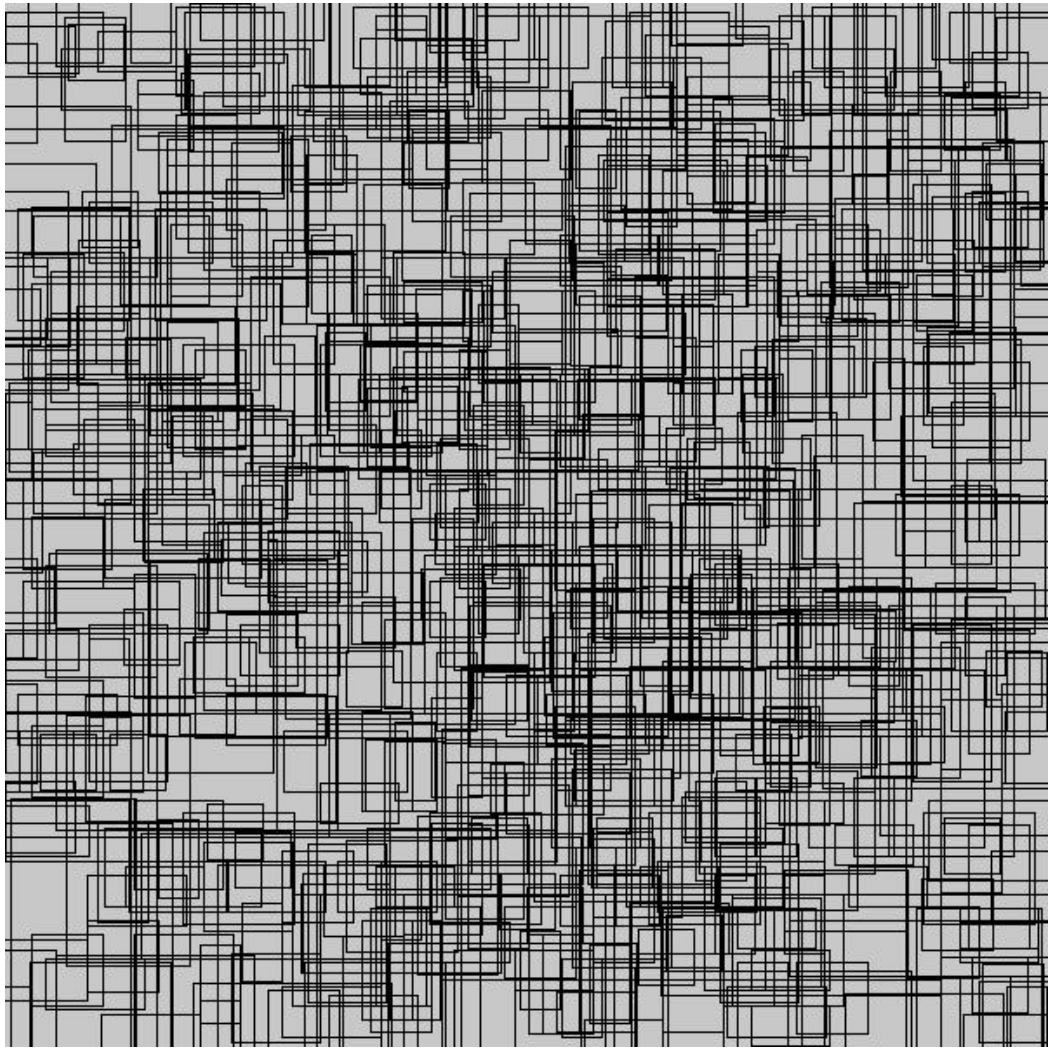


Matching strategy

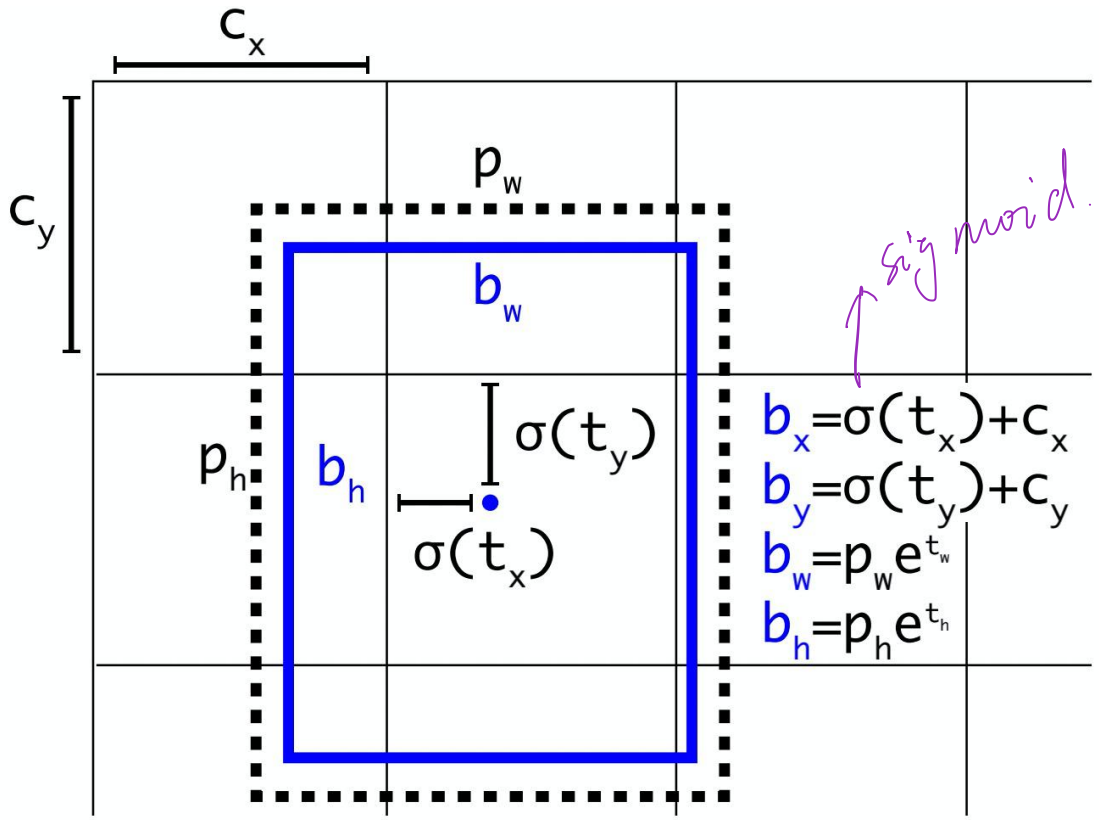
Anchor标注

- 类别
- 偏移量

Anchors

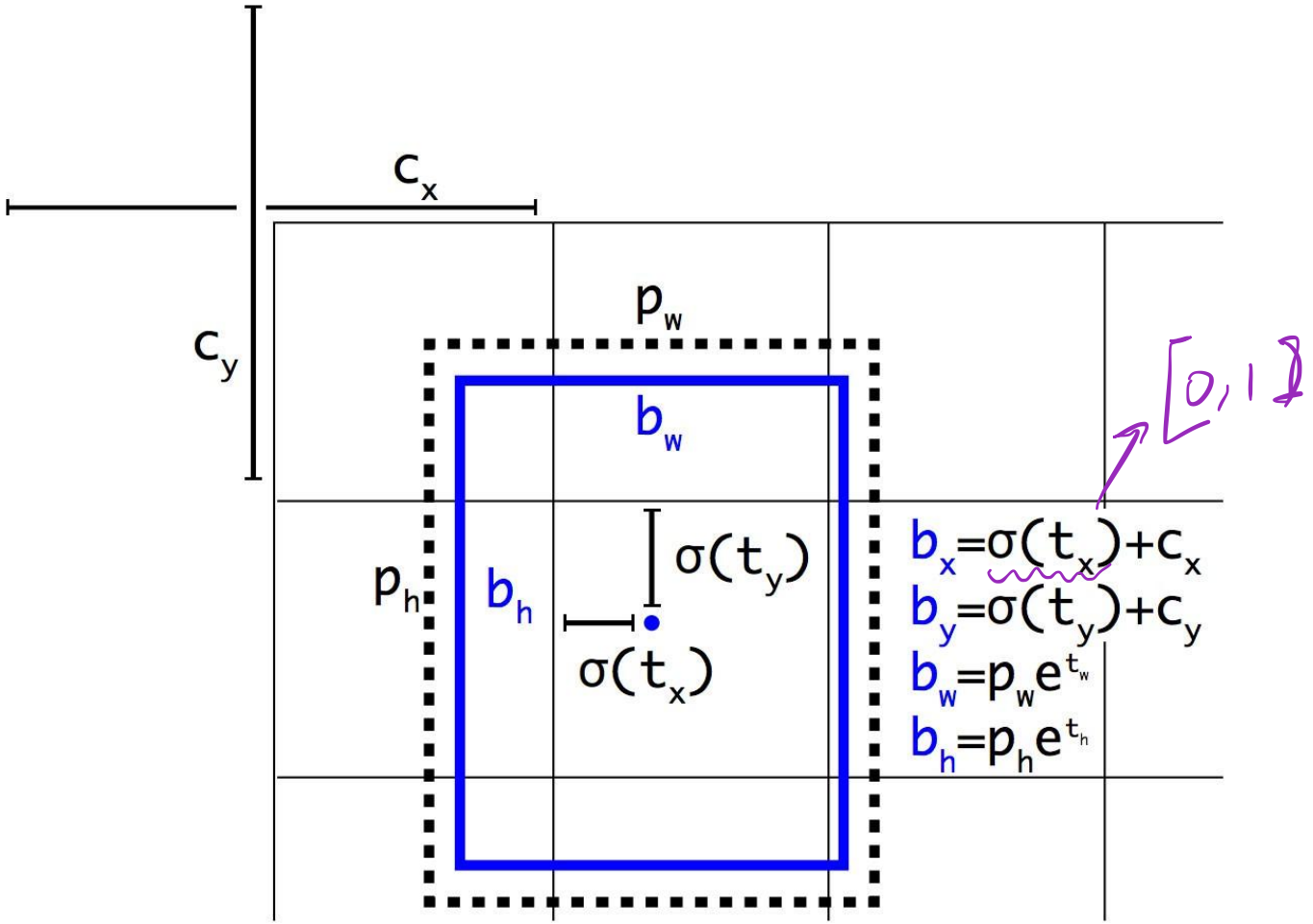


relative to the grid cell

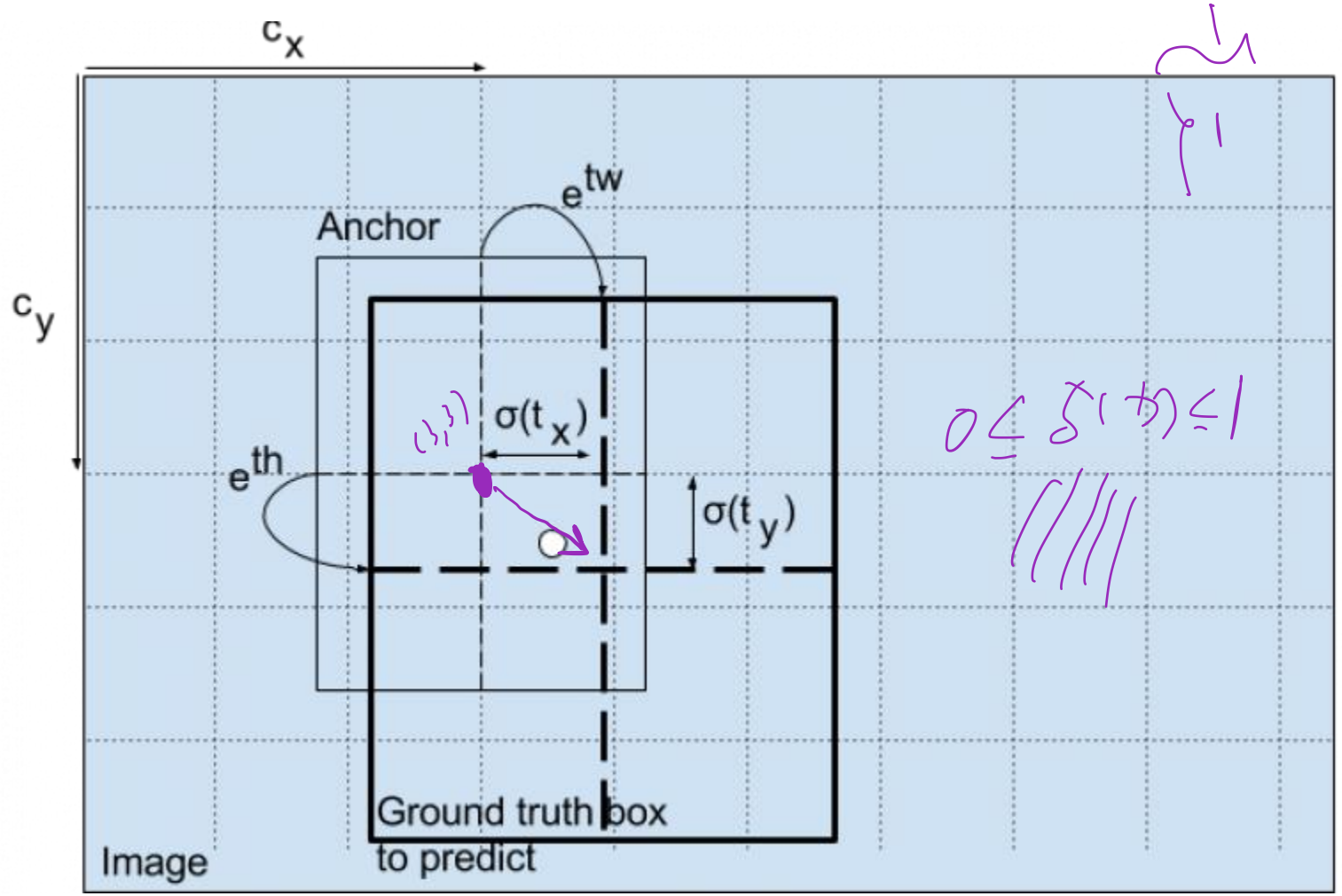


$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

relative to the grid cell



relative to the grid cell



Anchor

- tw/th：尺度缩放
- tx/ty：坐标偏移值

$b_w = p_w t_w$?

$b_w = p_w e^{t_w} \rightarrow p_w e^{\log t_w} = p_w t_w$

$\log t_w$：非负.

优化没有约束的 loss 问题更方便；

因为 t_w 有约束为 >0. 即 $\log t_w > 0$,

这样求 $\log t_w$ 就没有约束).

实际预测为 $\log t_w$ 的话这里要加 e.

Traning

Inference

Direct location prediction

- can end up at any point in the image
- random initialization

$$x = (t_x * w_a) - x_a$$
$$y = (t_y * h_a) - y_a$$

Batch Normalization

- leads to significant improvements in convergence
- eliminating the need for other forms of regularization

High Resolution Classifier

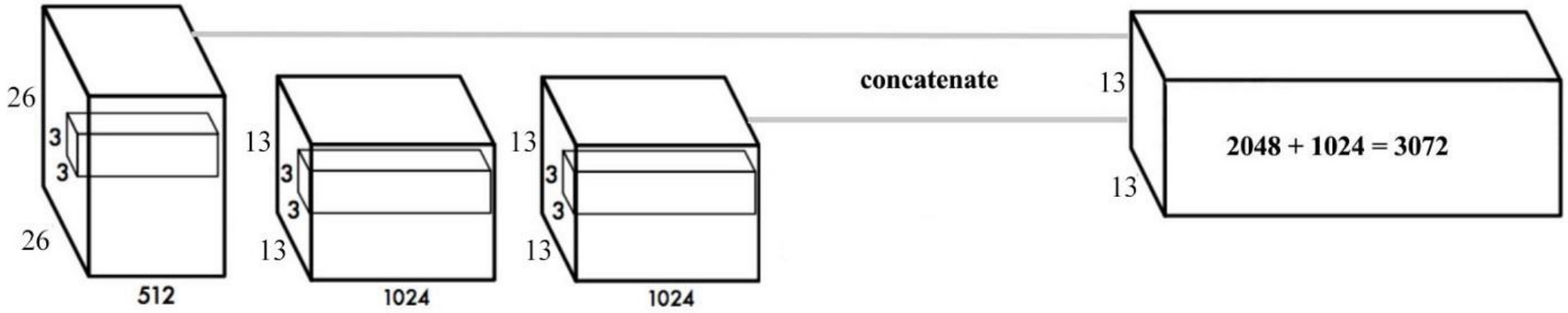
- first fine tune the classification network at the full 448 × 448 resolution
- then fine tune the resulting network on detection

Fine-Grained Features

- 13 × 13 feature map
- finer grained features for localizing smaller objects

passthrough layer

- similar to the identity mappings in ResNet



Multi-Scale Training

- change the network every few iterations
- Every 10 batches randomly chooses a new image dimension size
- multiples of 32: {320, 352, ..., 608}

low resolutions

- a cheap, fairly accurate detector
- ideal for smaller GPUs, high framerate video, or multiple video streams.

Darknet-19

YOLOv3

YOLOv3: An Incremental Improvement

Joseph Redmon Ali Farhadi
University of Washington

Abstract

We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At 320 × 320 YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP_{50} in 51 ms on a Titan X, compared to 57.5 AP_{50} in 198 ms by RetinaNet, similar performance but 3.8× faster. As always, all the code is online at https://pjreddie.com/yolo/.

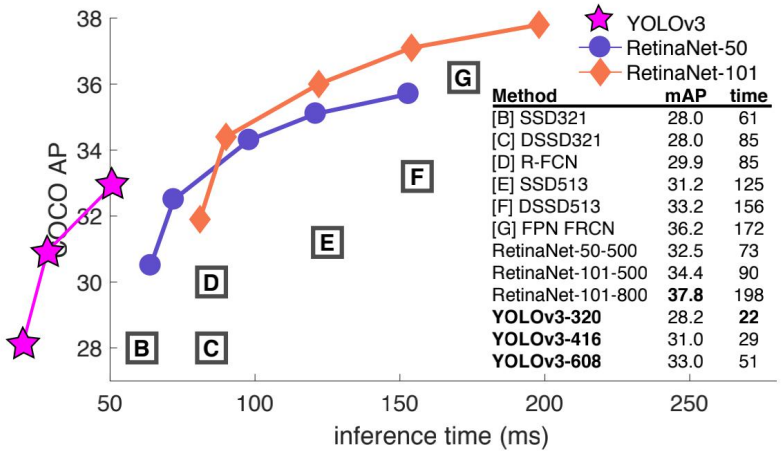


Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan

cfg

```
[net]
# Testing
# batch=1
# subdivisions=1
# Training
batch=64
subdivisions=16
width=608
height=608
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1
```

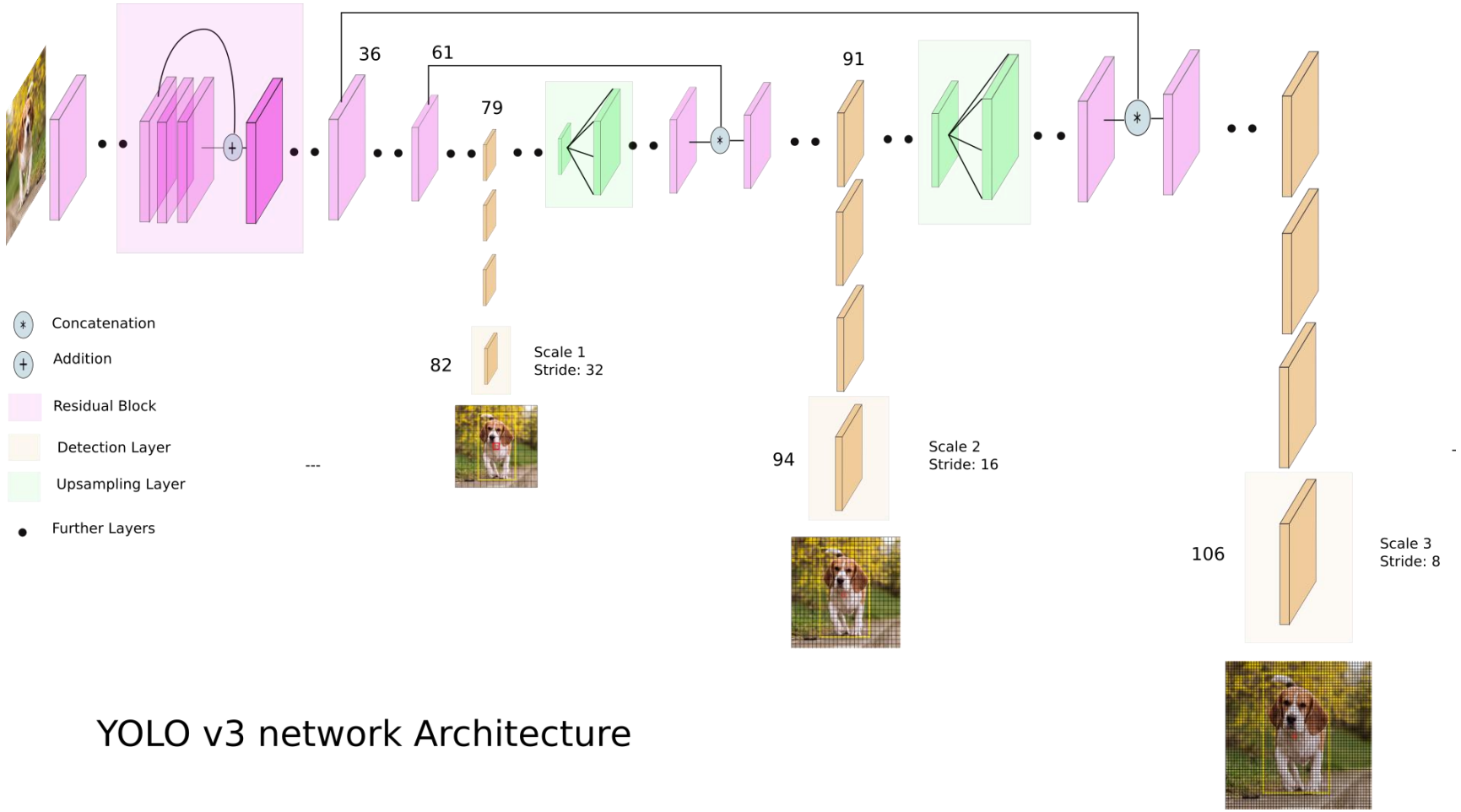
Darknet-53

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Predictions Across Scales

- YOLOv3 makes detection in 3 different scales in order to accommodate different objects size by using strides of 32, 16 and 8. This means, if we feed an input image of size 416 x 416, YOLOv3 will make detection on the scale of 13 x 13, 26 x 26, and 52 x 52.

Predictions Across Scales



YOLO v3 network Architecture

Darknet-53

```
 0 conv     32  3 x 3 / 1   608 x 608 x   3   ->   608 x 608 x  32  0.639 BFLOPs
 1 conv     64  3 x 3 / 2   608 x 608 x  32   ->   304 x 304 x  64  3.407 BFLOPs
 2 conv     32  1 x 1 / 1   304 x 304 x  64   ->   304 x 304 x  32  0.379 BFLOPs
 3 conv     64  3 x 3 / 1   304 x 304 x  32   ->   304 x 304 x  64  3.407 BFLOPs
 4 res    1                 304 x 304 x  64   ->   304 x 304 x  64
 5 conv    128  3 x 3 / 2   304 x 304 x  64   ->   152 x 152 x 128  3.407 BFLOPs
 6 conv     64  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x  64  0.379 BFLOPs
 7 conv    128  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x 128  3.407 BFLOPs
 8 res    5                 152 x 152 x 128   ->   152 x 152 x 128
 9 conv     64  1 x 1 / 1   152 x 152 x 128   ->   152 x 152 x  64  0.379 BFLOPs
10 conv    128  3 x 3 / 1   152 x 152 x  64   ->   152 x 152 x 128  3.407 BFLOPs
11 res    8                 152 x 152 x 128   ->   152 x 152 x 128
12 conv    256  3 x 3 / 2   152 x 152 x 128   ->    76 x  76 x 256  3.407 BFLOPs
13 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
14 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
15 res   12                  76 x  76 x 256   ->    76 x  76 x 256
16 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
17 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
18 res   15                  76 x  76 x 256   ->    76 x  76 x 256
19 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
20 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
21 res   18                  76 x  76 x 256   ->    76 x  76 x 256
22 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
23 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
24 res   21                  76 x  76 x 256   ->    76 x  76 x 256
25 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
26 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
27 res   24                  76 x  76 x 256   ->    76 x  76 x 256
28 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
29 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
30 res   27                  76 x  76 x 256   ->    76 x  76 x 256
```

Darknet-53

```
31 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
32 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
33 res   30                   76 x  76 x 256   ->    76 x  76 x 256
34 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
35 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
36 res   33                   76 x  76 x 256   ->    76 x  76 x 256
37 conv    512  3 x 3 / 2    76 x  76 x 256   ->    38 x  38 x 512  3.407 BFLOPs
38 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
39 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
40 res   37                   38 x  38 x 512   ->    38 x  38 x 512
41 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
42 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
43 res   40                   38 x  38 x 512   ->    38 x  38 x 512
44 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
45 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
46 res   43                   38 x  38 x 512   ->    38 x  38 x 512
47 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
48 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
49 res   46                   38 x  38 x 512   ->    38 x  38 x 512
50 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
51 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
52 res   49                   38 x  38 x 512   ->    38 x  38 x 512
53 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
54 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
55 res   52                   38 x  38 x 512   ->    38 x  38 x 512
56 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
57 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
58 res   55                   38 x  38 x 512   ->    38 x  38 x 512
59 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
60 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
```

Darknet-53

```
61 res    58                   38 x  38 x 512   ->    38 x  38 x 512
62 conv   1024  3 x 3 / 2      38 x  38 x 512   ->    19 x  19 x1024  3.407 BFLOPs
63 conv    512  1 x 1 / 1      19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
64 conv   1024  3 x 3 / 1      19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
65 res    62                   19 x  19 x1024   ->    19 x  19 x1024
66 conv    512  1 x 1 / 1      19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
67 conv   1024  3 x 3 / 1      19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
68 res    65                   19 x  19 x1024   ->    19 x  19 x1024
69 conv    512  1 x 1 / 1      19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
70 conv   1024  3 x 3 / 1      19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
71 res    68                   19 x  19 x1024   ->    19 x  19 x1024
72 conv    512  1 x 1 / 1      19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
```

Darknet-53

```
 73 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
 74 res   71                   19 x  19 x1024   ->    19 x  19 x1024
 75 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
 76 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
 77 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
 78 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
 79 conv    512  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 512  0.379 BFLOPs
 80 conv   1024  3 x 3 / 1    19 x  19 x 512   ->    19 x  19 x1024  3.407 BFLOPs
 81 conv    255  1 x 1 / 1    19 x  19 x1024   ->    19 x  19 x 255  0.189 BFLOPs
 82 yolo
 83 route  79
 84 conv    256  1 x 1 / 1    19 x  19 x 512   ->    19 x  19 x 256  0.095 BFLOPs
 85 upsample            2x    19 x  19 x 256   ->    38 x  38 x 256
 86 route  85 61
 87 conv    256  1 x 1 / 1    38 x  38 x 768   ->    38 x  38 x 256  0.568 BFLOPs
 88 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
 89 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
 90 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
 91 conv    256  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 256  0.379 BFLOPs
 92 conv    512  3 x 3 / 1    38 x  38 x 256   ->    38 x  38 x 512  3.407 BFLOPs
 93 conv    255  1 x 1 / 1    38 x  38 x 512   ->    38 x  38 x 255  0.377 BFLOPs
 94 yolo
 95 route  91
 96 conv    128  1 x 1 / 1    38 x  38 x 256   ->    38 x  38 x 128  0.095 BFLOPs
 97 upsample            2x    38 x  38 x 128   ->    76 x  76 x 128
 98 route  97 36
 99 conv    128  1 x 1 / 1    76 x  76 x 384   ->    76 x  76 x 128  0.568 BFLOPs
100 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
101 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
102 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
103 conv    128  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 128  0.379 BFLOPs
104 conv    256  3 x 3 / 1    76 x  76 x 128   ->    76 x  76 x 256  3.407 BFLOPs
105 conv    255  1 x 1 / 1    76 x  76 x 256   ->    76 x  76 x 255  0.754 BFLOPs
106 yolo
```

yolov3.cfg

- Darknet构建网络架构不是通过代码直接堆叠，而是通过解析cfg文件进行生成的

Convolutional

```
[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
size=3
stride=1
pad=1
filters=256
activation=leaky
```

Convolutional

```
if module_def["type"] == "convolutional":
    bn = int(module_def["batch_normalize"])
    filters = int(module_def["filters"])
    kernel_size = int(module_def["size"])
    pad = (kernel_size - 1) // 2
    modules.add_module(
        f"conv_{module_i}",
        nn.Conv2d(
            in_channels=output_filters[-1],
            out_channels=filters,
            kernel_size=kernel_size,
            stride=int(module_def["stride"]),
            padding=pad,
            bias=not bn,
        ),
    )
    if bn:
        modules.add_module(f"batch_norm_{module_i}", nn.BatchNorm2d(filters, momentum=0.9, eps=1e-5))
    if module_def["activation"] == "leaky":
        modules.add_module(f"leaky_{module_i}", nn.LeakyReLU(0.1))
```

Shortcut

```
[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=512
size=3
stride=1
pad=1
activation=leaky

[shortcut]
from=-3
activation=linear
```

Route
Upsample

```
[route]
layers = -4

[convolutional]
batch_normalize=1
filters=128
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 36
```

EmptyLayer

```
class EmptyLayer(nn.Module):
    """Placeholder for 'route' and 'shortcut' layers"""

    def __init__(self):
        super(EmptyLayer, self).__init__()
```

Route/upsample/shortcut

```
elif module_def["type"] == "maxpool":
    kernel_size = int(module_def["size"])
    stride = int(module_def["stride"])
    if kernel_size == 2 and stride == 1:
        modules.add_module(f"_debug_padding_{module_i}", nn.ZeroPad2d((0, 1, 0, 1)))
    maxpool = nn.MaxPool2d(kernel_size=kernel_size, stride=stride, padding=int((kernel_size - 1) // 2))
    modules.add_module(f"maxpool_{module_i}", maxpool)

elif module_def["type"] == "upsample":
    upsample = Upsample(scale_factor=int(module_def["stride"]), mode="nearest")
    modules.add_module(f"upsample_{module_i}", upsample)

elif module_def["type"] == "route":
    layers = [int(x) for x in module_def["layers"].split(",")]
    filters = sum([output_filters[1:][i] for i in layers])
    modules.add_module(f"route_{module_i}", EmptyLayer())

elif module_def["type"] == "shortcut":
    filters = output_filters[1:][int(module_def["from"])]
    modules.add_module(f"shortcut_{module_i}", EmptyLayer())
```

Upsample

```
class Upsample(nn.Module):
    """ nn.Upsample is deprecated """

    def __init__(self, scale_factor, mode="nearest"):
        super(Upsample, self).__init__()
        self.scale_factor = scale_factor
        self.mode = mode

    def forward(self, x):
        x = F.interpolate(x, scale_factor=self.scale_factor, mode=self.mode)
        return x
```

YOLOLayer

```
[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,
classes=80
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

YOLOLayer

```
class YOLOLayer(nn.Module):
    """Detection layer"""

    def __init__(self, anchors, num_classes, img_dim=416):
        super(YOLOLayer, self).__init__()
        self.anchors = anchors
        self.num_anchors = len(anchors)
        self.num_classes = num_classes
        self.ignore_thres = 0.5
        self.mse_loss = nn.MSELoss()
        self.bce_loss = nn.BCELoss()
        self.obj_scale = 1
        self.noobj_scale = 100
        self.metrics = {}
        self.img_dim = img_dim
        self.grid_size = 0  # grid size

    def compute_grid_offsets(self, grid_size, cuda=True):
```

Pytorch实现

- https://github.com/eriklindernoren/PyTorch-YOLOv3
- https://github.com/Ray-Luo/YOLOV3-PyTorch

IoU实现

答疑

- 竹篱:留个问题：一般每一类要训练多少张？
- YH:问题：tx,ty,tw,th是网络直接输出吗?
- Mine:pytorch有聚类anchor的代码吗？
- delete:问题：最后的loss 公式可以讲一下吗？
- Alina:问题： yolov3多尺度的预测是体现在anchors的多尺度吗？

主要内容

- 1、实战作业讲解

- 数据收集和标注
- 生成训练文件
- anchor clustering
- 修改配置文件
- 预训练权重
- 训练和部署

主要内容

- 2、YOLOv3 知识总结

- 损失函数
- Anchor
- 多尺度

主要内容

- 3，YOLOv4知识拓展

- YOLOv4知识点列表
- DIoU/CIoU
- Mosaic
- Mish激活函数

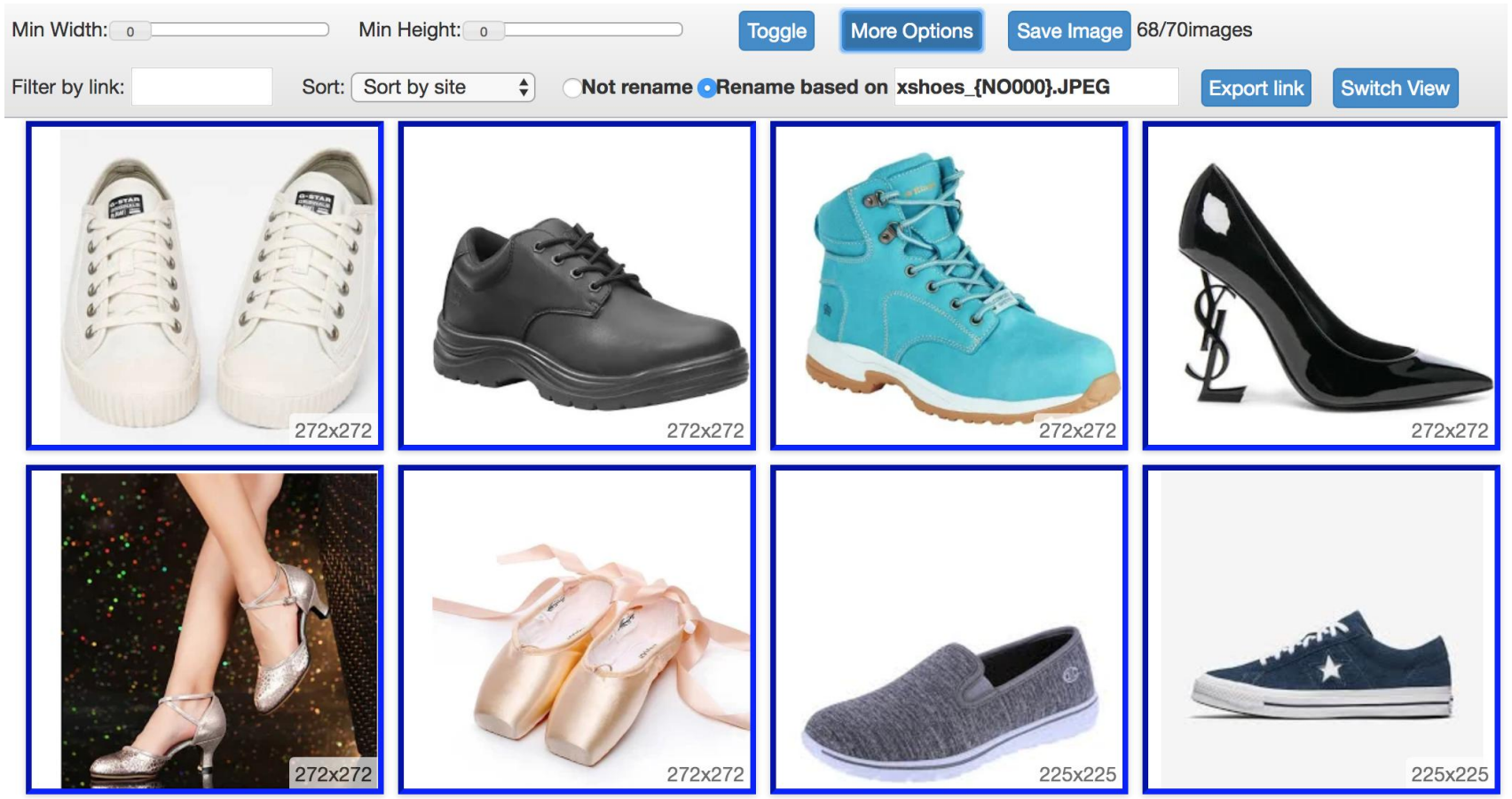
实现自己的YOLO检测器

- 准备数据集
- 训练YOLO模型
- 测试检测效果
- 部署模型

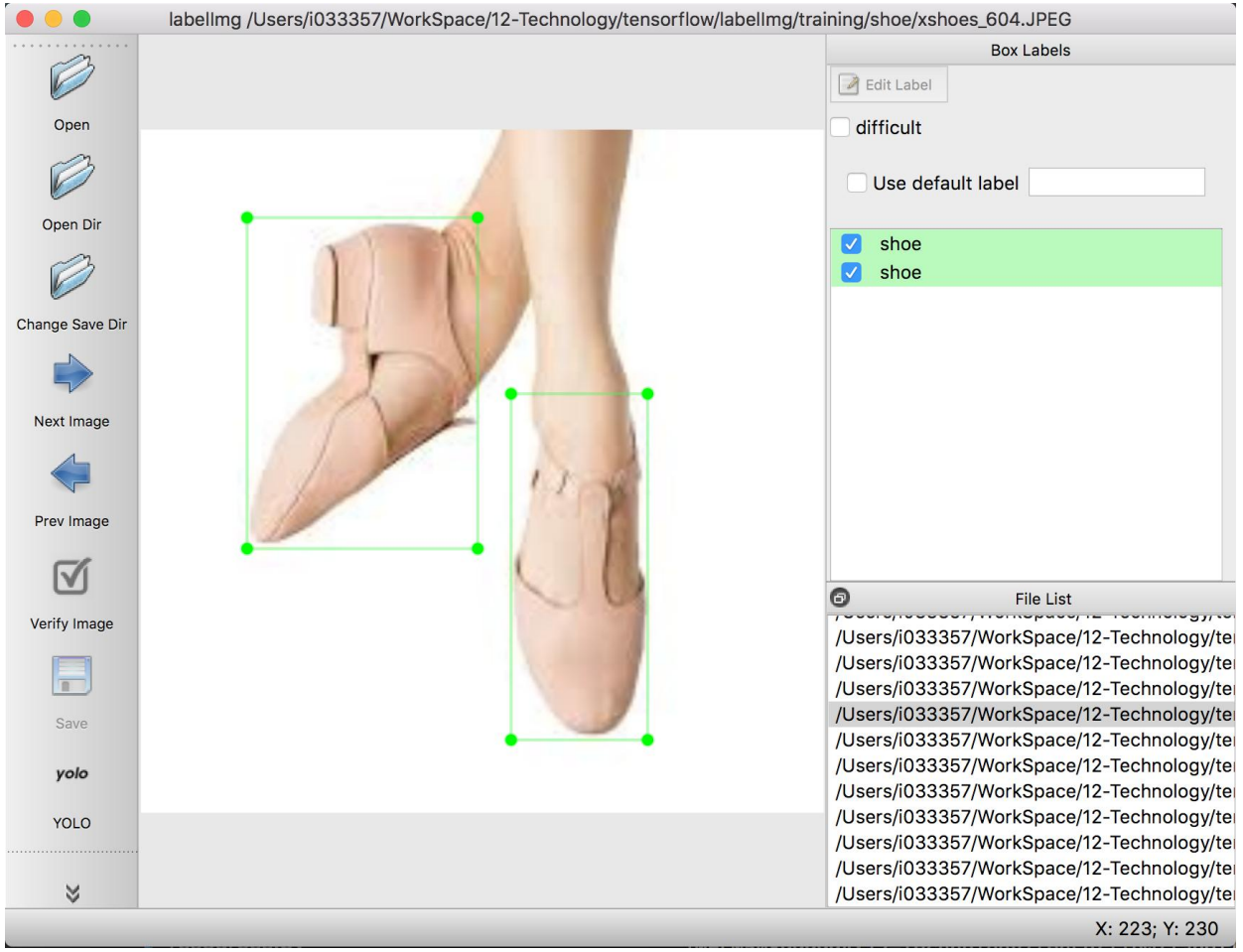
准备数据集

- 下载数据
- 标注数据
- 生成训练文件

下载数据


Min Width: 0
Min Height: 0
Toggle
More Options
Save Image
68/70images
Filter by link:
Sort: Sort by site
Not rename
Rename based on
xshoes_{NO000}.JPEG
Export link
Switch View
272x272
272x272
272x272
272x272
272x272
272x272
225x225
225x225

LabelImg



VOC格式

- XML

```
<annotation>
        <folder>Downloads</folder>
        <filename>clay.jpg</filename>
        <path>/Users/david/Downloads/clay.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>750</width>
                <height>500</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>hand</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>196</xmin>
                        <ymin>57</ymin>
                        <xmax>635</xmax>
                        <ymax>374</ymax>
                </bndbox>
        </object>
</annotation>
```

YOLO格式

```
0 0.324444 0.371111 0.337778 0.484444
0 0.642222 0.640000 0.200000 0.506667

class_index box_x1_ratio box_y1_ratio box_width_ratio box_height_ratio
0 - The index of object, in my case, only one class - shoe.
0.324444 - box_x1_ratio(box_x1 / image_width)
0.371111 - box_y1_ratio(box_y1 / image_height)
...
```

生成训练文件

- train.txt
- test.txt
- shoe.data

shoe.data

```
1   classes= 1
2   train  = training/train_list.txt
3   valid  = training/test_list.txt
4   names = cfg/custom.names
5   backup = backup
```

训练模型

- 安装DarkNet
- 复制训练数据
- 生成Anchor
- 修改配置文件
- 下载预训练权重-darknet53.conv.74
- 开始训练

安装Darknet

- git clone https://github.com/pjreddie/darknet.git
- cd darknet
- GPU=1 in Makefile
- make
- ./darknet

生成Anchor

- python generate_anchors_yolo_v3.py -filelist train_list.txt -num_clusters 9

预训练权重

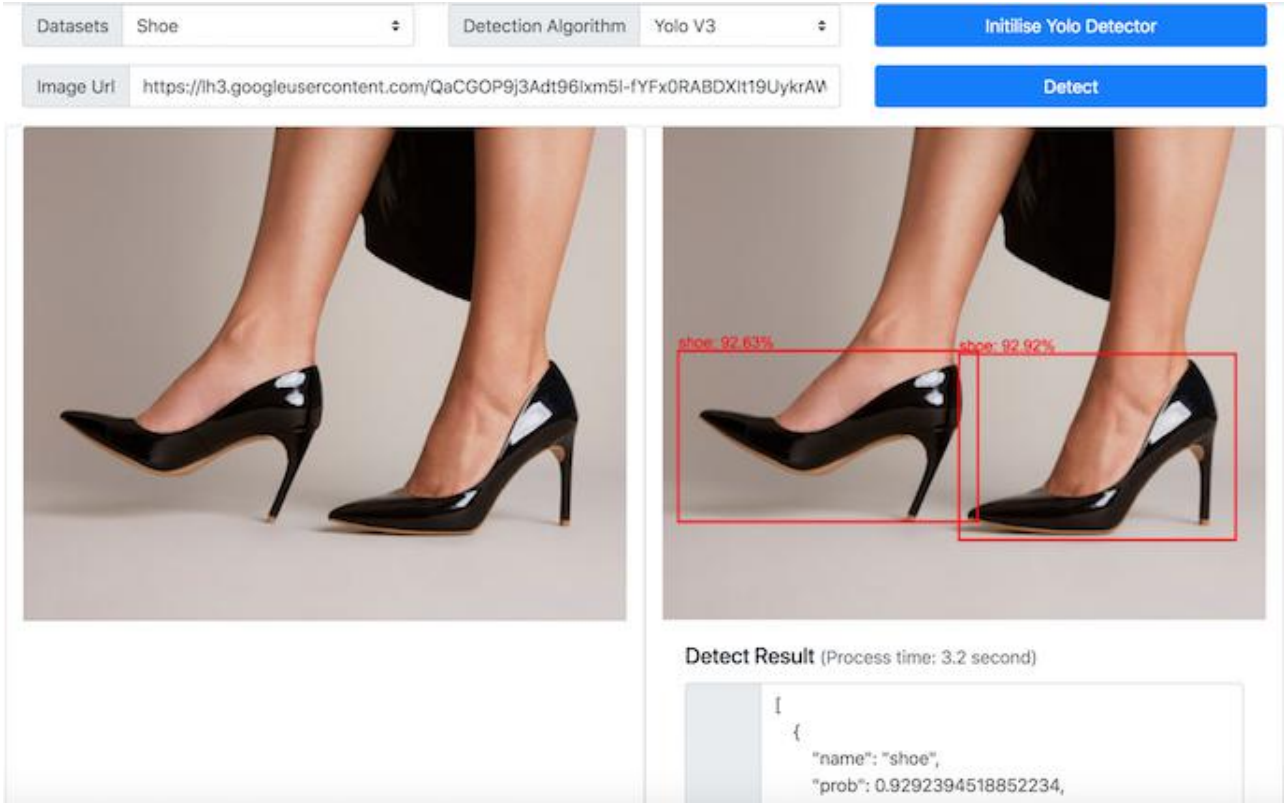
- https://pjreddie.com/media/files/darknet53.conv.74

- ./darknet detector train cfg/shoe_training_config.data cfg/yolov3_shoe.cfg darknet53.conv.74

使用模型

- ./darknet detect cfg/yolo_shoe.cfg backup/yolo_shoe.backup data/s8.jpeg

部署模型


Datasets
Shoe
Detection Algorithm
Yolo V3
Initilise Yolo Detector
Image Url
https://lh3.googleusercontent.com/QaCGOP9j3Adt96lxm5l-fYFx0RABDXIt19UykrAW
Detect
shoe: 92.63%
shoe: 92.92%
Detect Result (Process time: 3.2 second)
[
{
"name": "shoe",
"prob": 0.9292394518852234,

RESTful API(Node.js)

```
[
    {
        "box": {
            "y": 97.06493854522705,
            "x": 87.51638531684875,
            "w": 122.85414934158325,
            "h": 62.75526809692383
        },
        "name": "shoe",
        "prob": 0.9999971389770508
    }
]
Note: x-top left x, y-top left y, w-width, h-height
```

YOLOv3成绝唱？

- 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.

YOLOv4

YOLOv4: Optimal Speed and Accuracy of Object Detection

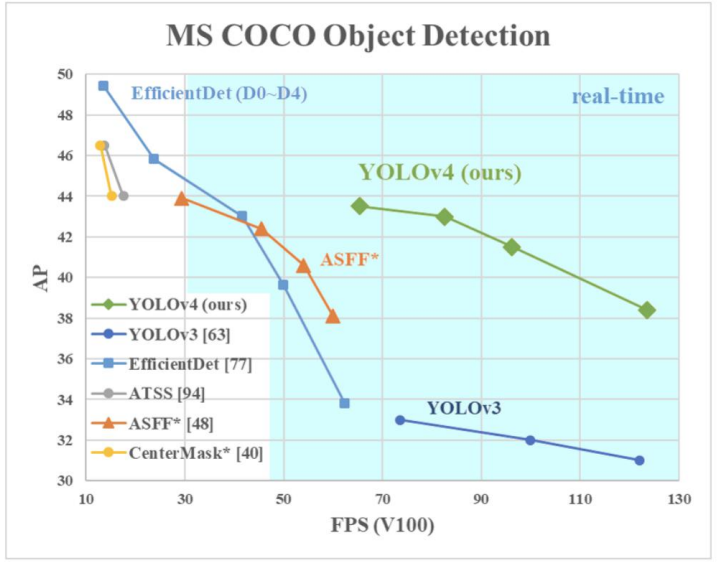
Alexey Bochkovskiy*
alexeyab84@gmail.com

Chien-Yao Wang*
Institute of Information Science
Academia Sinica, Taiwan
kinyiu@iis.sinica.edu.tw

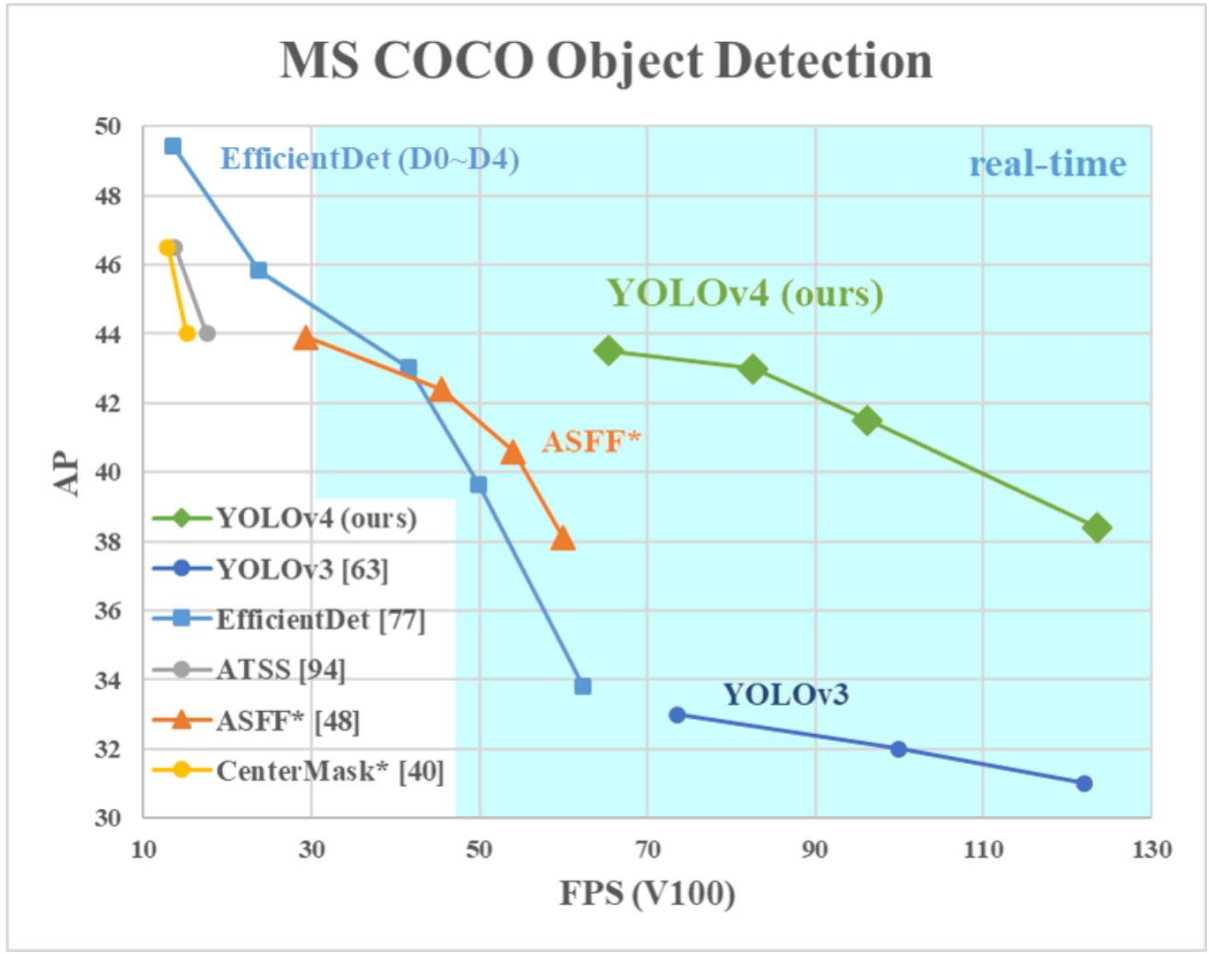
Hong-Yuan Mark Liao
Institute of Information Science
Academia Sinica, Taiwan
liao@iis.sinica.edu.tw

Abstract

There are a huge number of features which are said to improve Convolutional Neural Network (CNN) accuracy. Practical testing of combinations of such features on large datasets, and theoretical justification of the result, is required. Some features operate on certain models exclusively and for certain problems exclusively, or only for small-scale datasets; while some features, such as batch-normalization and residual-connections, are applicable to the majority of models, tasks, and datasets. We assume that such universal features include Weighted-Residual-Connections (WRC), Cross-Stage-Partial-connections (CSP), Cross mini-Batch Normalization (CmBN), Self-adversarial-training (SAT) and Mish-activation. We use new features: WRC, CSP, CmBN, SAT, Mish activation, Mosaic data augmentation,



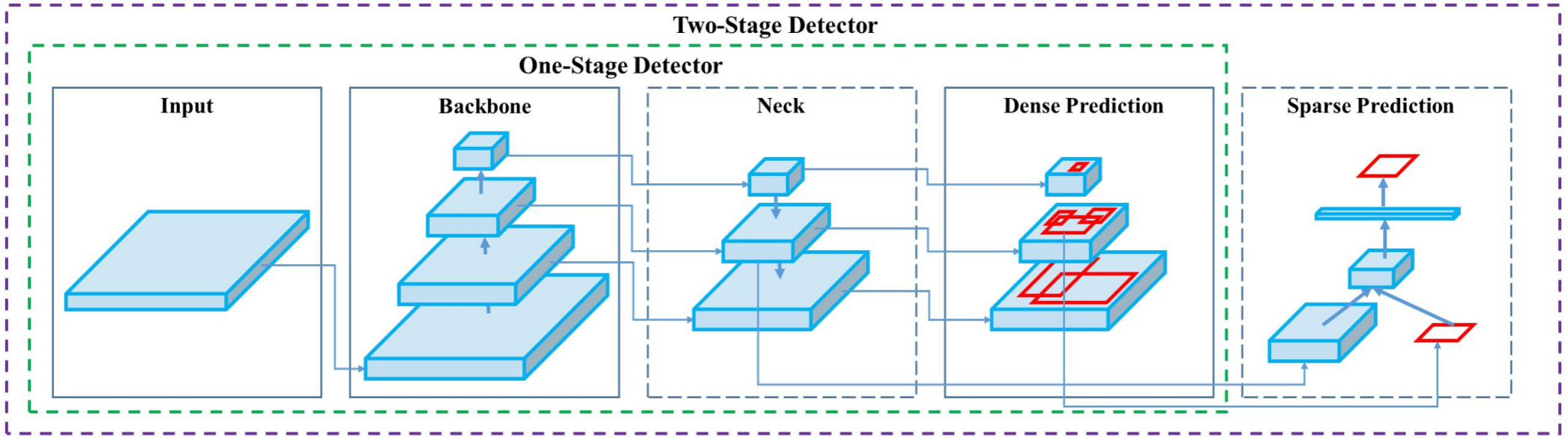
YOLOv4



YOLOv4

- Finally, we choose CSPDarknet53 backbone, SPP additional module, PANet path-aggregation neck, and YOLOv3(anchor based) head as the architecture of YOLOv4.

Object detectors



Bag of freebies

- data augmentation
- distribution bias
- objective function

Bag of specials

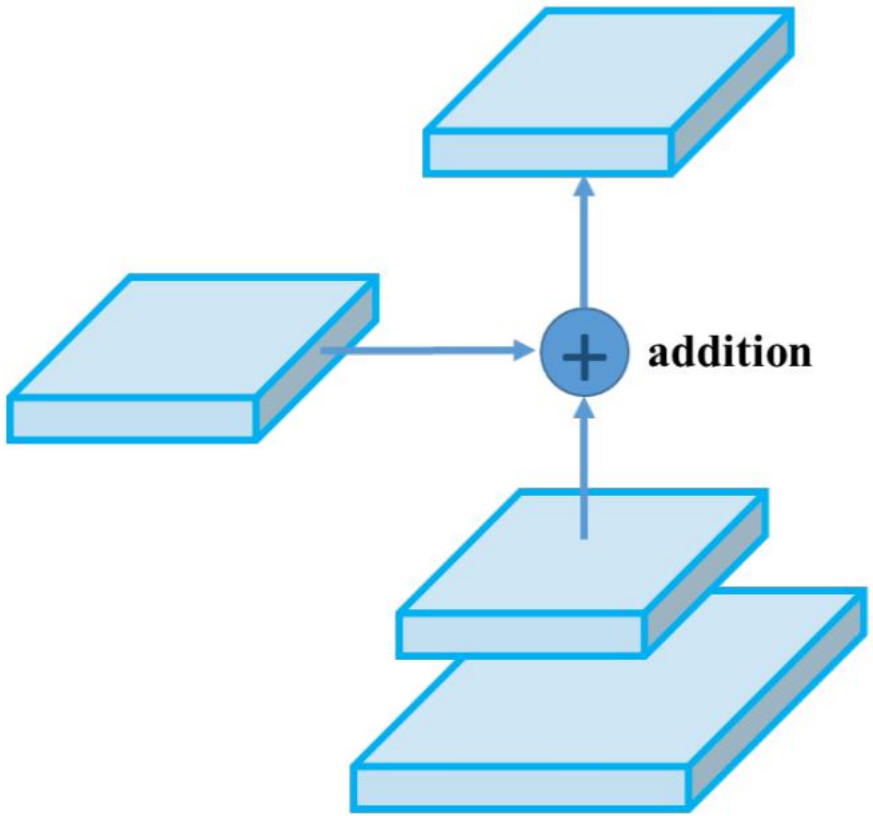
- enlarging receptive field
- introducing attention mechanism
- feature integration
- good activation function
- post-processing method

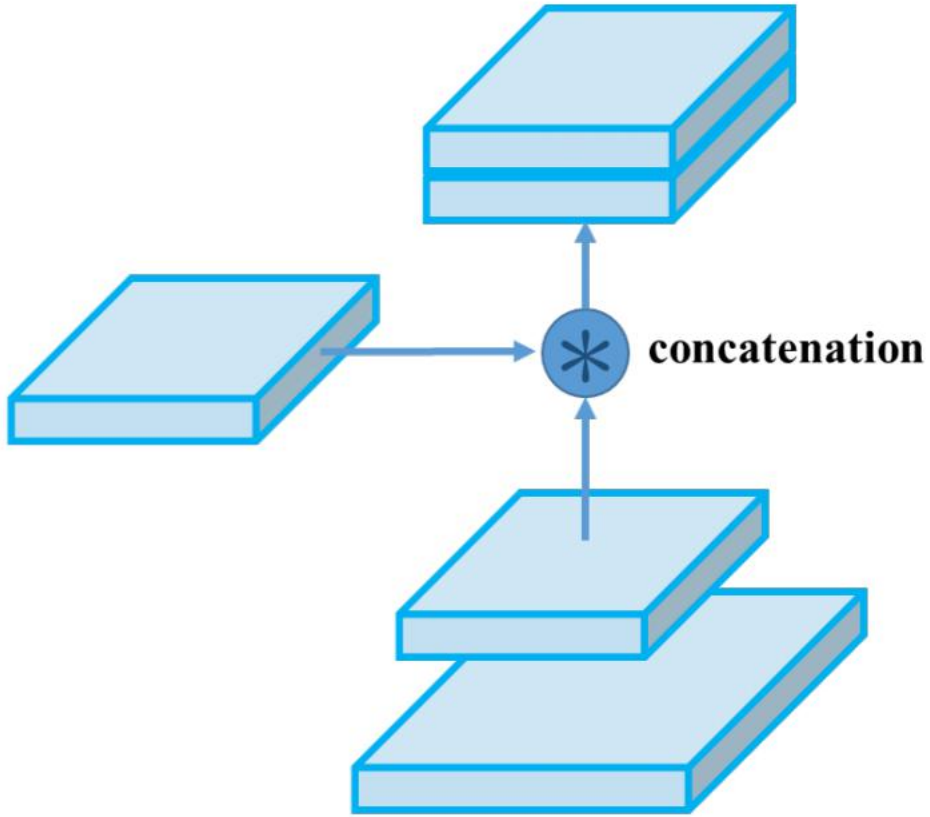
Backbone

Table 1: Parameters of neural networks for image classification.

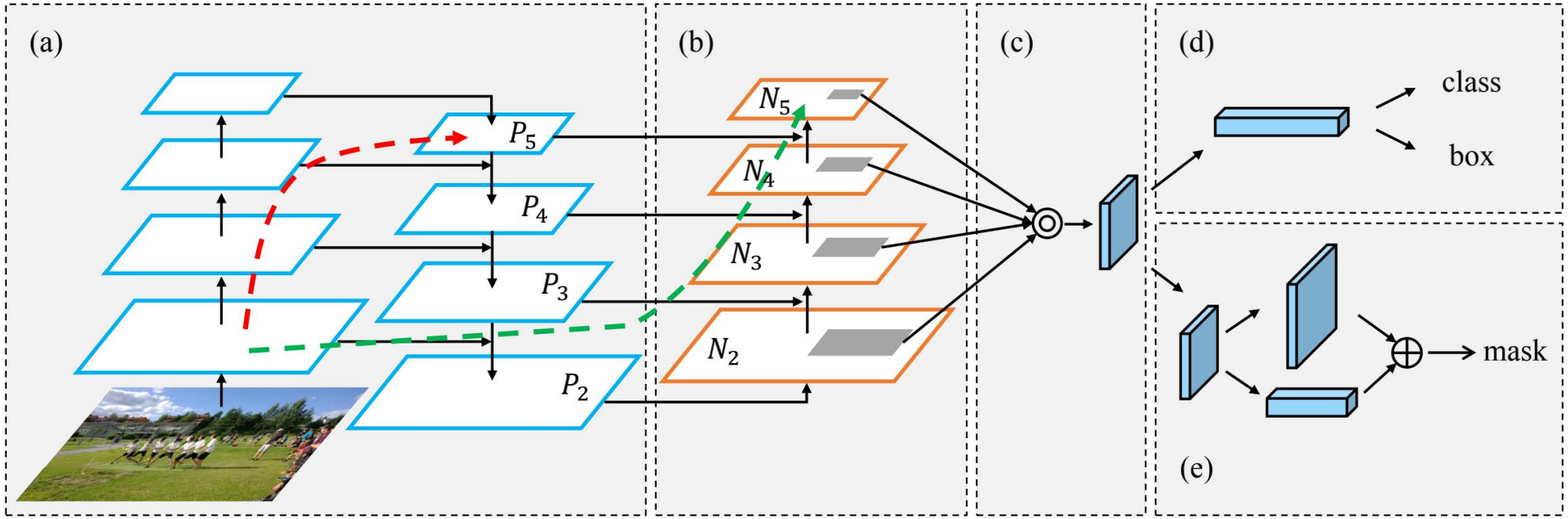
Backbone model	Input network resolution	Receptive field size	Parameters	Average size of layer output (WxHxC)	BFLOPs (512x512 network resolution)	FPS (GPU RTX 2070)
CSPResNext50	512x512	425x425	20.6 M	**1058 K**	31 (15.5 FMA)	62
CSPDarknet53	512x512	725x725	**27.6 M**	950 K	52 (26.0 FMA)	**66**
EfficientNet-B3 (ours)	512x512	**1311x1311**	12.0 M	668 K	11 (5.5 FMA)	26

Modified PAN

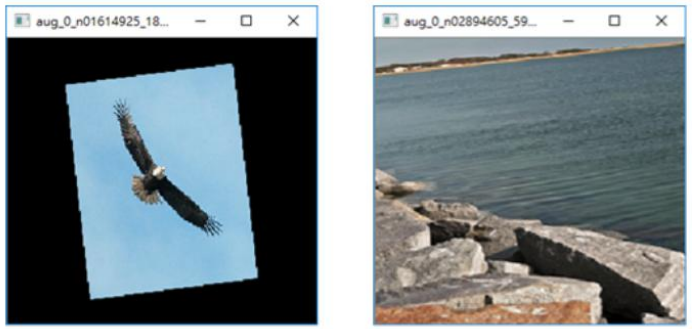


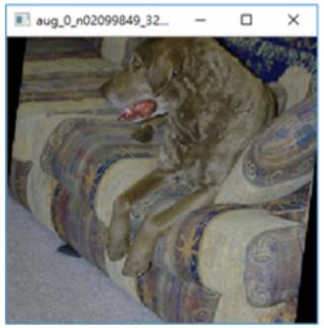


PANet

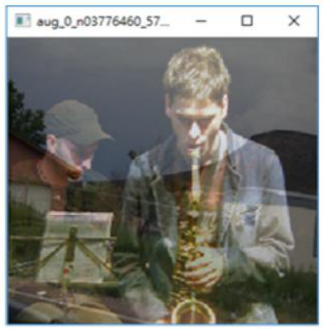


data augmentation

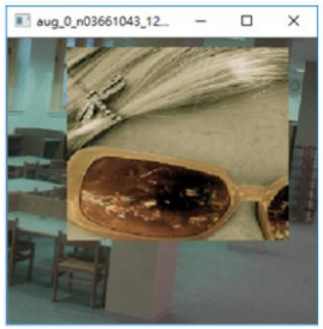




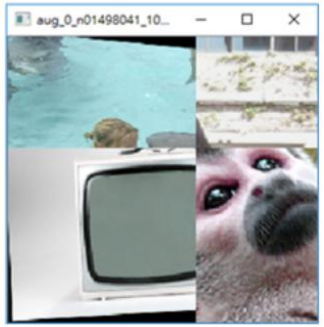
(a) Crop, Rotation, Flip, Hue, Saturation, Exposure, Aspect.



(b) MixUp



(c) CutMix



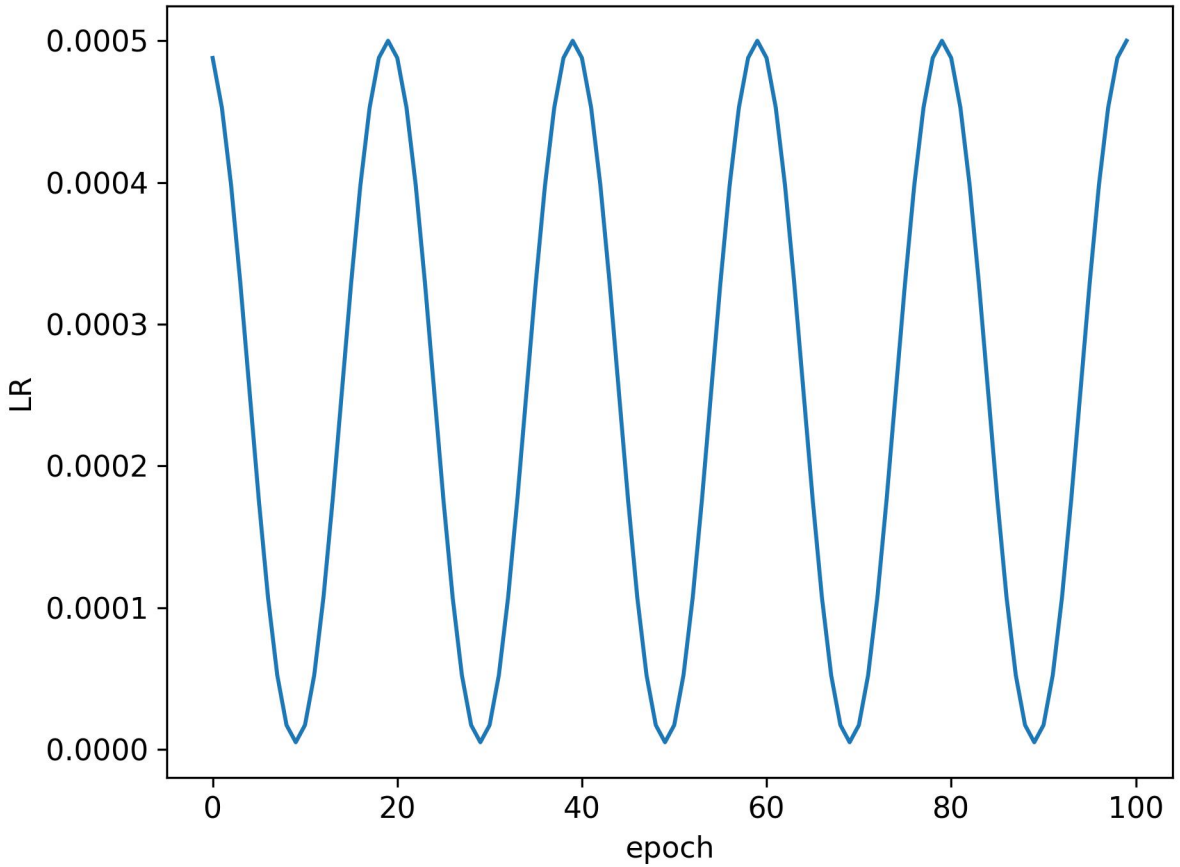
(d) Mosaic

(e) Blur

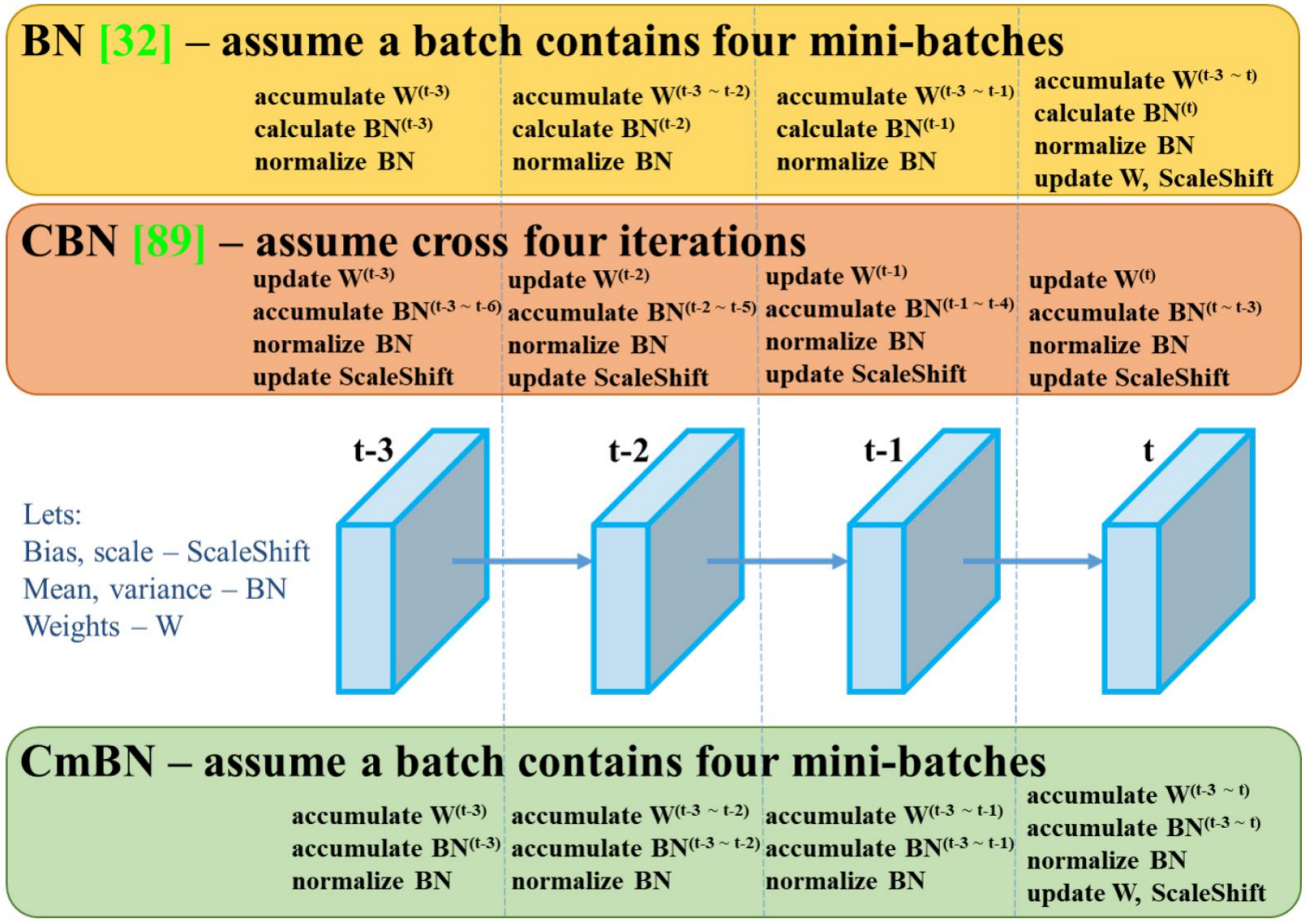
Mosaic

- This significantly reduces the need for a large mini-batch size
- CutMix mixes only 2 input images

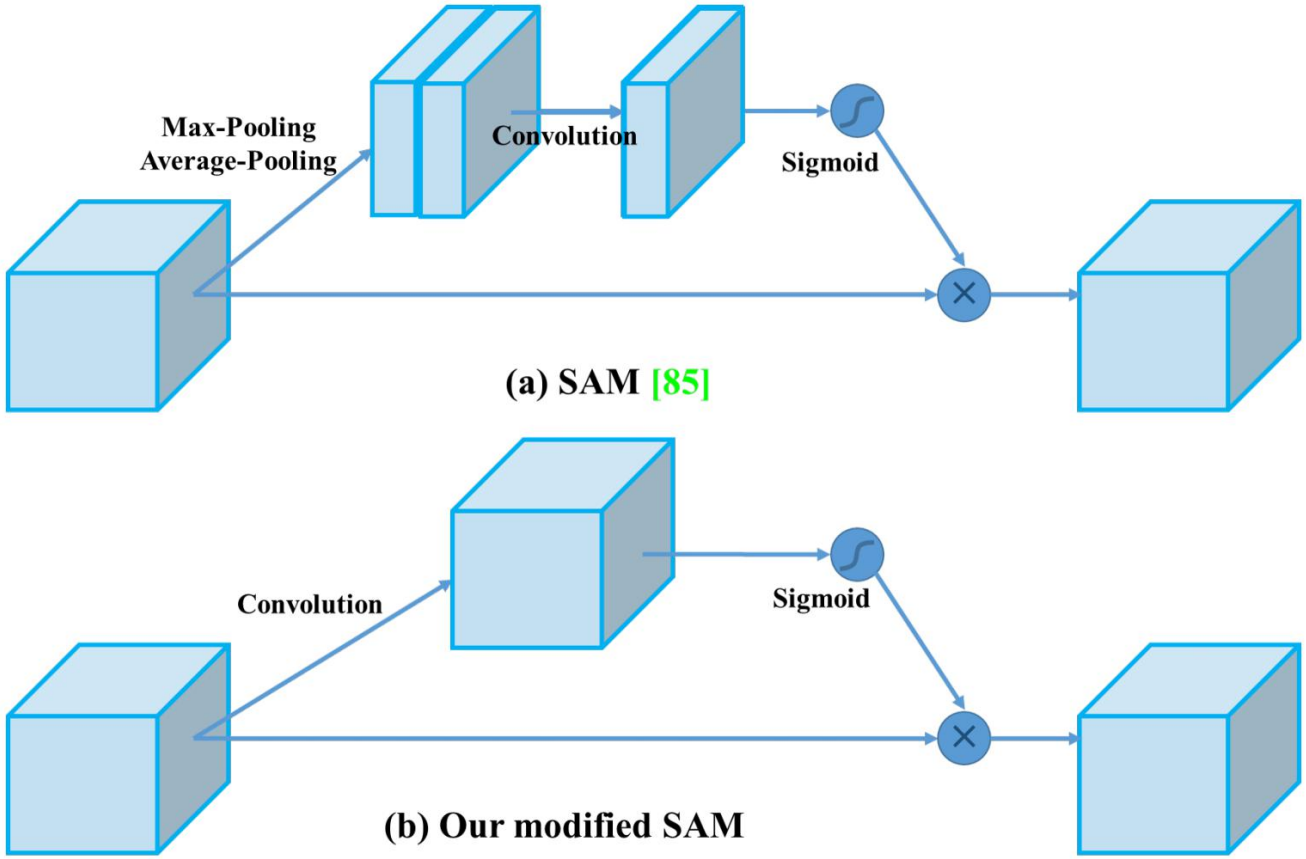
Cosine annealing scheduler



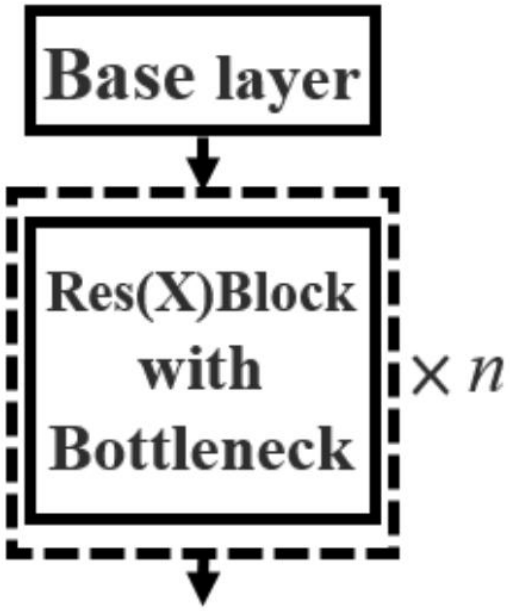
Cross mini-Batch Normalization



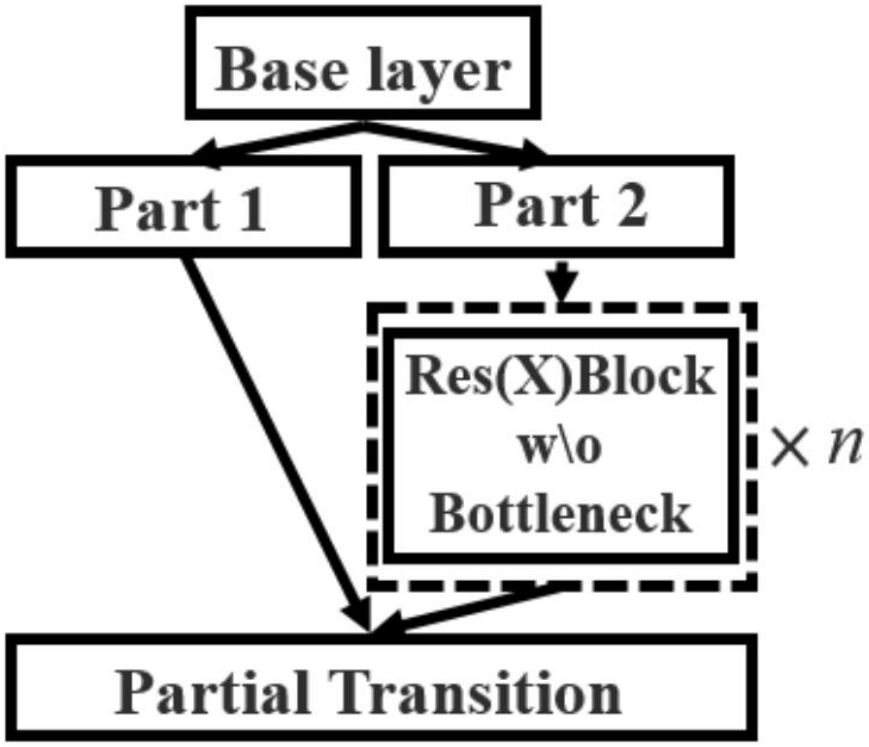
Modified SAM



CSPNet



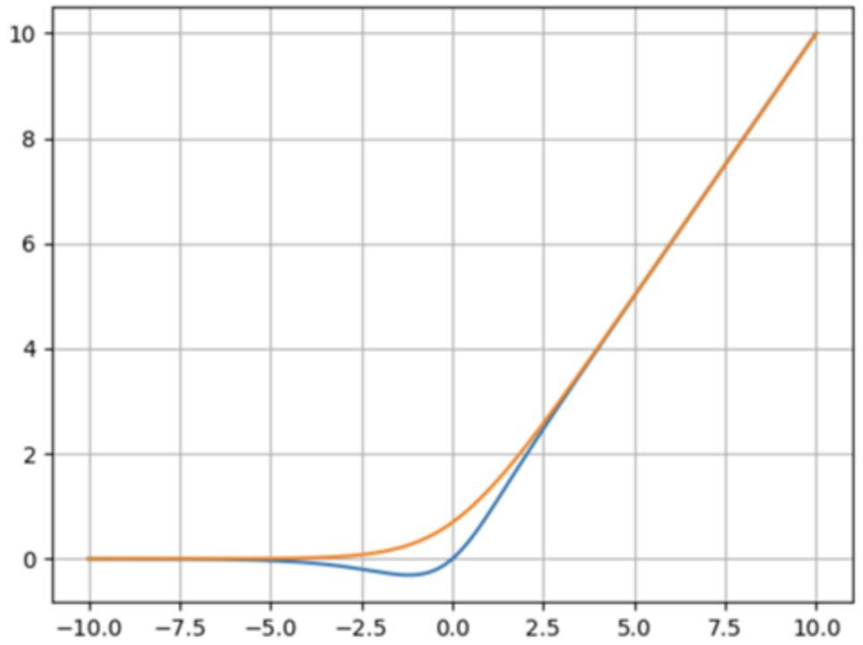
(a) ResNe(X)t



(b) CSPResNe(X)t

Mish/Softplus

Mish=x * tanh(ln(1+e^x)),



Mish

- 对于ResNext-50来说，最好的激活是ReLU，但是速度更快并且需要更少的内存。仅对于DenseNet-121，SENet-18，ShuffleNetv2，尤其是对于SqueezeNet（比ReLU +2.15，比Swish +0.96），Mish明显优于Swish / ReLU。

Mish

ML Review
@ml_review

Follow

Mish: Self Regularized Non-Monotonic Activation Function
By @DigantaMisra1
$f(x)=x\cdot tanh(softplus(x))$
Increased accuracy over Swish/ReLU
Increased performance over Swish

Github
github.com/digantamisra98...
ArXiv
arxiv.org/abs/1908.08681...

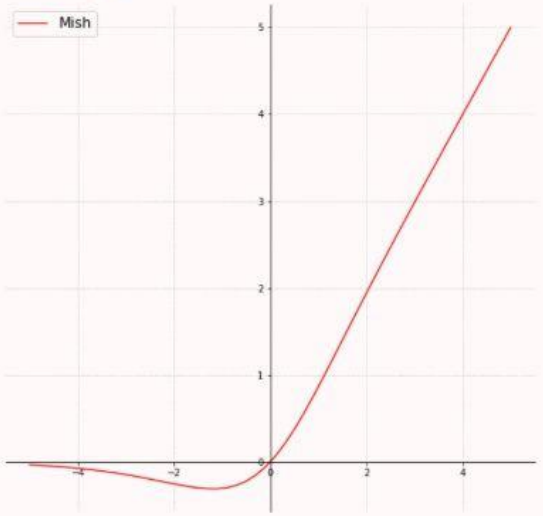
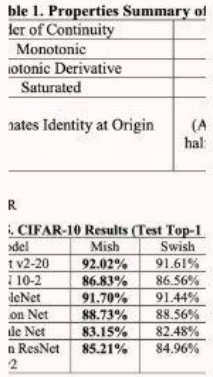


Figure 1. Mish Activation Function



ble 1. Properties Summary of

ler of Continuity	
Monotonic	
otonic Derivative	
Saturated	
nates Identity at Origin	(A hal

R

. CIFAR-10 Results (Test Top-1

odel	Mish	Swish
t v2-20	**92.02%**	91.61%
l 10-2	**86.83%**	86.56%
deNet	**91.70%**	91.44%
on Net	**88.73%**	88.56%
le Net	**83.15%**	82.48%
n ResNet 2	**85.21%**	84.96%

12:30 PM - 13 Oct 2019

82 Retweets **255** Likes

Mish

```python
import math
import numpy as np
from matplotlib import pyplot as plt

def mish(x):
    return x * math.tanh(math.log(1+math.exp(x)))

def ln_e(x):
    return math.log(1+math.exp(x))

x = np.linspace(-10,10,1000)
y=[]
z=[]
for i in x:
    y.append(mish(i))
    z.append(ln_e(i))
plt.plot(x,y)
plt.plot(x,z)
plt.grid()
plt.show()
```

GIoU loss

Algorithm 1: Generalized Intersection over Union

input : Two arbitrary convex shapes: $A, B \subseteq \mathbb{S} \in \mathbb{R}^n$

output: $GIoU$

1 For A and B, find the smallest enclosing convex object C, where $C \subseteq \mathbb{S} \in \mathbb{R}^n$

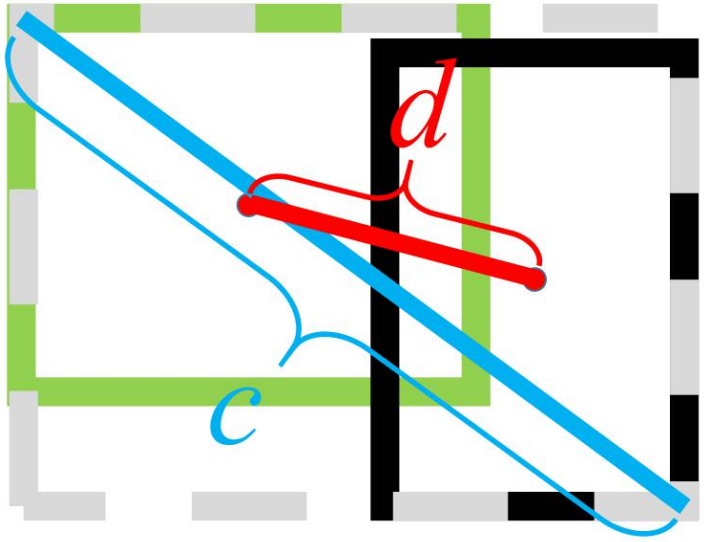
2 $$IoU = \frac{|A \cap B|}{|A \cup B|}$$

3 $$GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|}$$

IoU-based loss

$$\mathcal{L} = 1 - IoU + \mathcal{R}(B, B^{gt}),$$

DIoU

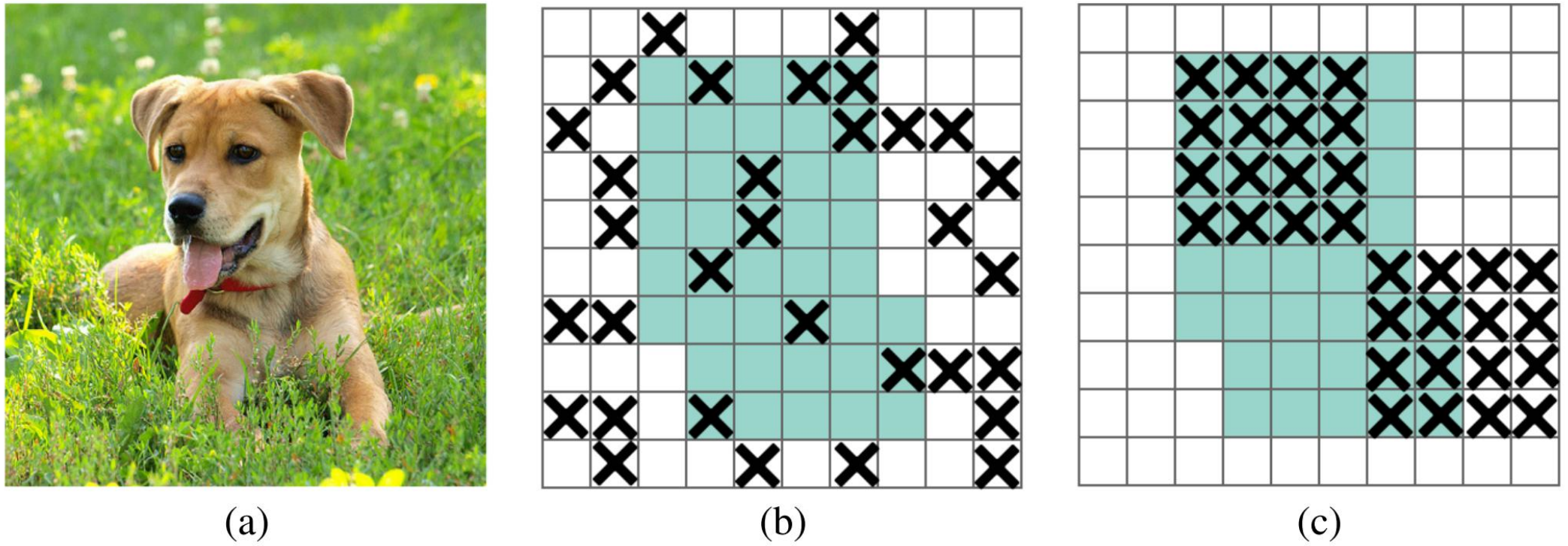


$$DIoU = IoU - \frac{\rho^2(b, b^{gt})}{c^2}$$

CutMix

ResNet-50	Mixup [48]	Cutout [3]	CutMix
			

DropBlock



(a) (b) (c)

课程总结

- YOLOv3算法原理
- YOLOv3源码精讲-讲透Anchor机制
- YOLOv3源码精讲-要点深入理解
- YOLOv4要点

参考资料

- YOLOv4: Optimal Speed and Accuracy of Object Detection
- YOLOv3: An Incremental Improvement
- YOLO9000: Better, Faster, Stronger
- You Only Look Once: Unified, Real-Time Object Detection

后厂理工学院

houchangtech.com

一所专注前沿互联网技术领域的创新实战大学