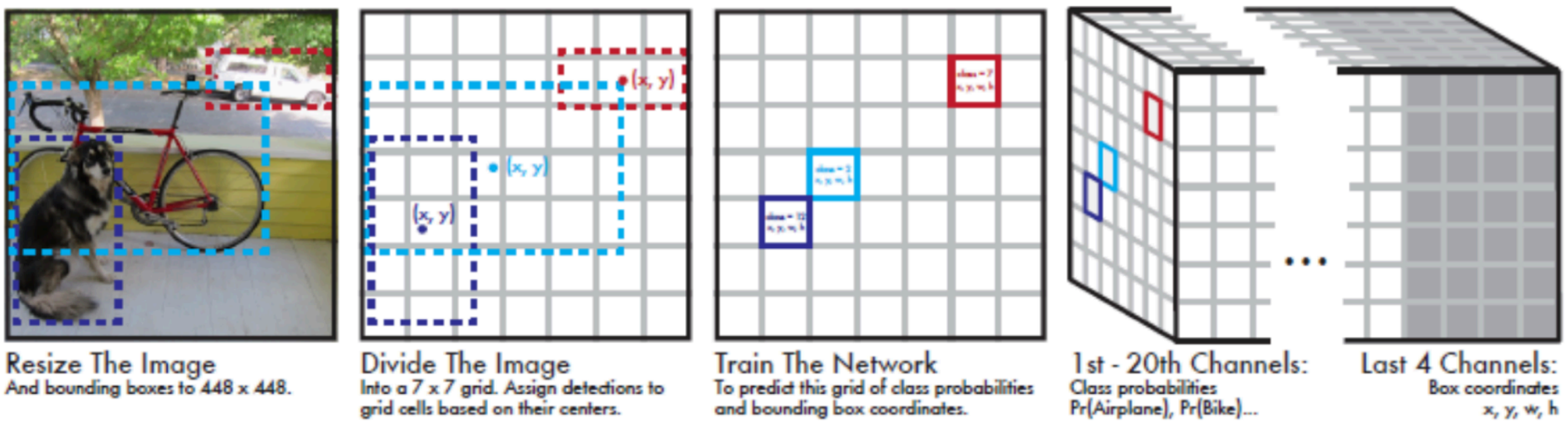
YOLO: 9000

Better, Faster, Stronger

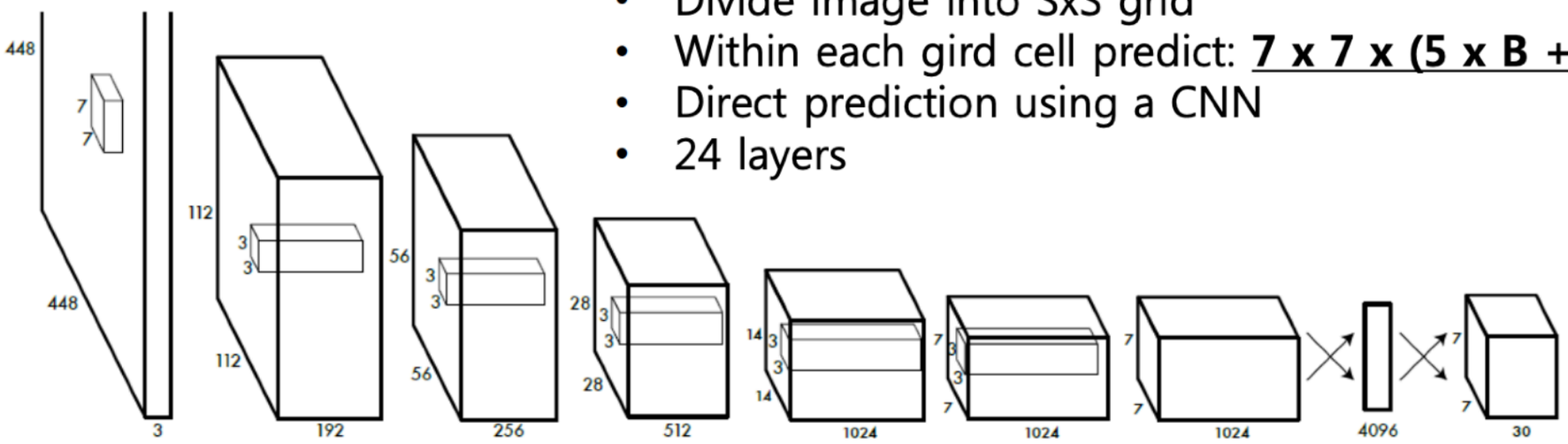
SungMan, Cho

YOLO v1

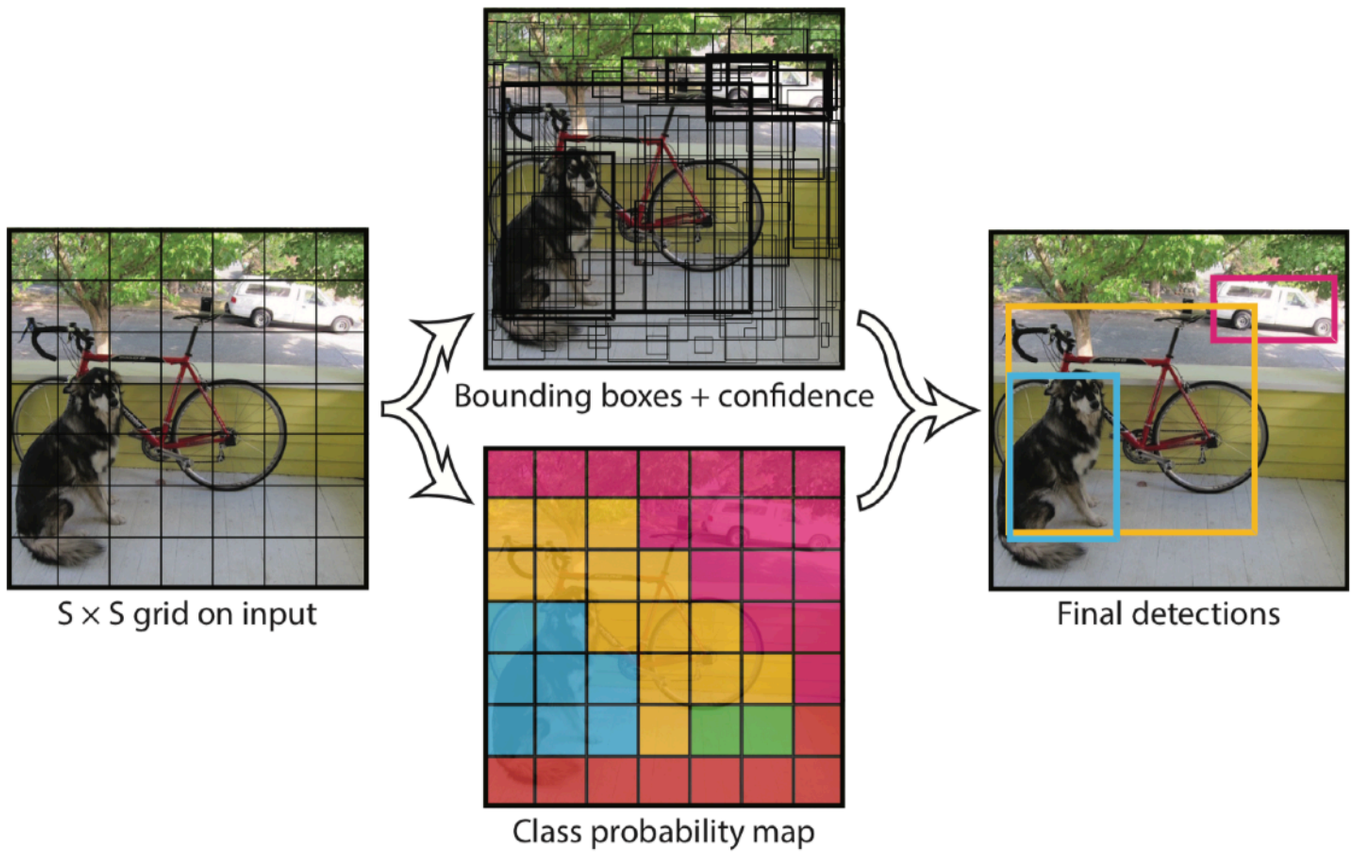
Overall Architecture



- Divide image into SxS grid
- Within each gird cell predict: **7 x 7 x (5 x B + C) tensor**
- Direct prediction using a CNN
- 24 layers

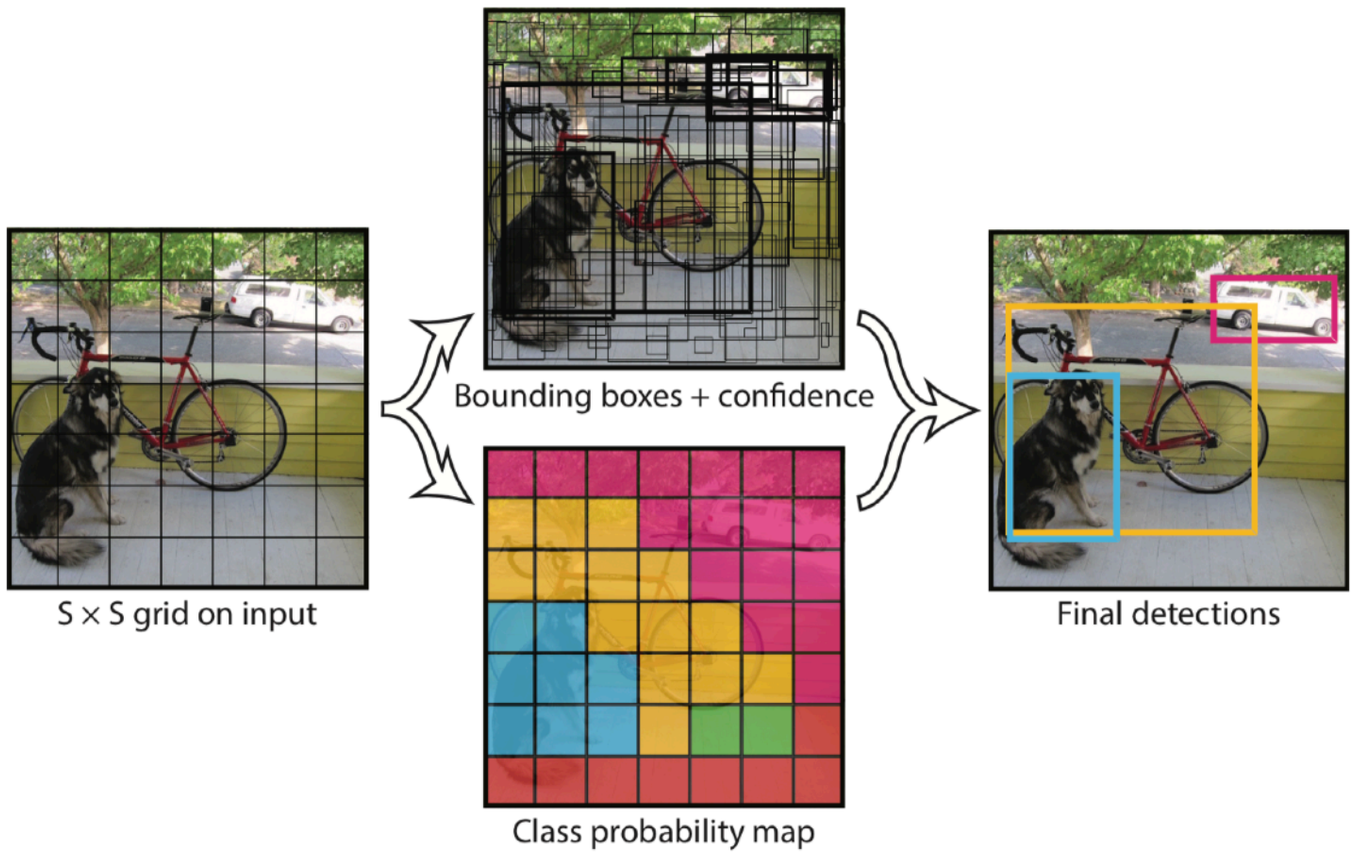


Tensor Shape



- YOLO divides the image into an **S x S grid.**
- Each grid cell predicts **B bounding boxes**, confidence for those boxes, and **C class probabilities.**
- These predictions are encoded as an **S x S x (B * 5 + C) tensor**

Tensor Shape



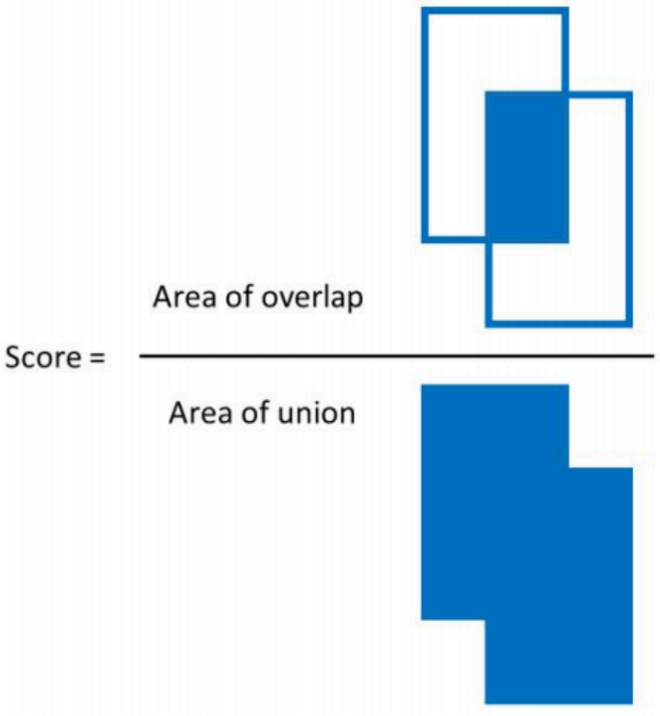
- These predictions are encoded as an **S x S x (B * 5 + C) tensor**

- For evaluating YOLO on PASCAL VOC, S = 7, B = 2, C = 20.
  -> final prediction 7 x 7 x 30 tensor.

Bounding Boxes

- Each bounding box consist of 5 predictions: x, y, w, h and confidence.

$$\Pr(\text{Object}) * \hat{\text{IOU}}_{\text{pred}}^{\text{truth}}$$



Loss function

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

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

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

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

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \qquad (3)$$

Loss function

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

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

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

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

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (3)$$

- SSE **weights localization error equally with classification error.**
- Also, in every image **many grid cells do not contain object.**

Loss function

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

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

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

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

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (3)$$

- SSE error equally **weights errors in large boxes and small boxes.**
- **Small deviation in large boxes matter less than in small boxes.**

YOLO v2

Better, Faster, Stronger

- **Better**
  - Batch normalization
  - High resolution classifier
  - Convolutional with Anchor boxes
  - Dimension clusters
  - Direct location prediction
  - Fine-grained features
  - Multi-scale training
- **Faster**
  - Darknet-19
  - Training for classification
  - Training for detection
- **Stronger**
  - Hierarchical classification
  - Dataset combination with Word-tree
  - Joint classification and detection

Better

Better

Convolutional with Anchor Boxes

- **Remove the fully connected layers from YOLO** and **use anchor boxes** to predict bounding boxes.

- Shrink the network to operate on 416 input images instead of 448 x 448.
  -> we want an **odd number of locations** in our feature map

- YOLO : 98 boxes per image -> YOLO v2 : 9000 boxes

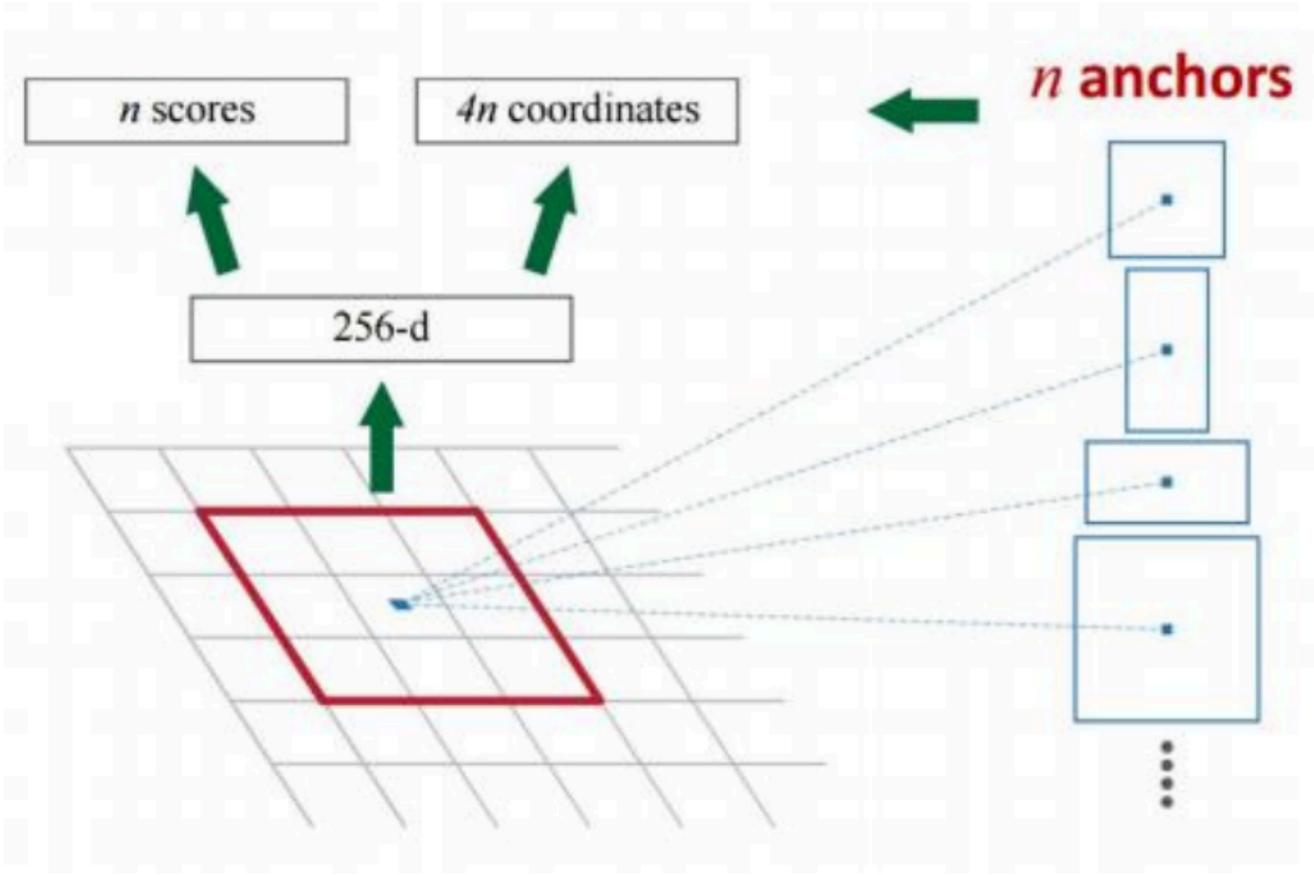
- With anchor boxes : 69.2mAP with a recall of 88%
  (Without anchor boxes : 69.5mAP with a recall of 81%)

Better

Dimension Clusters

- **Anchor boxes** dimensions are **hand picked.**

  (If we **pick better priors** for the network to start with we can make it **easier** for the network **to learn to predict good detections.**)

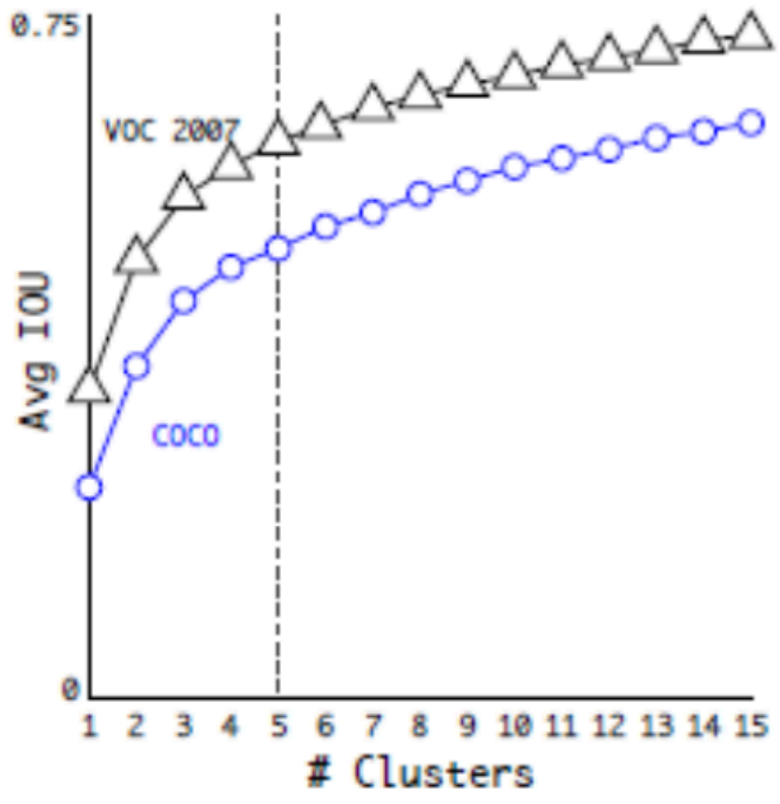


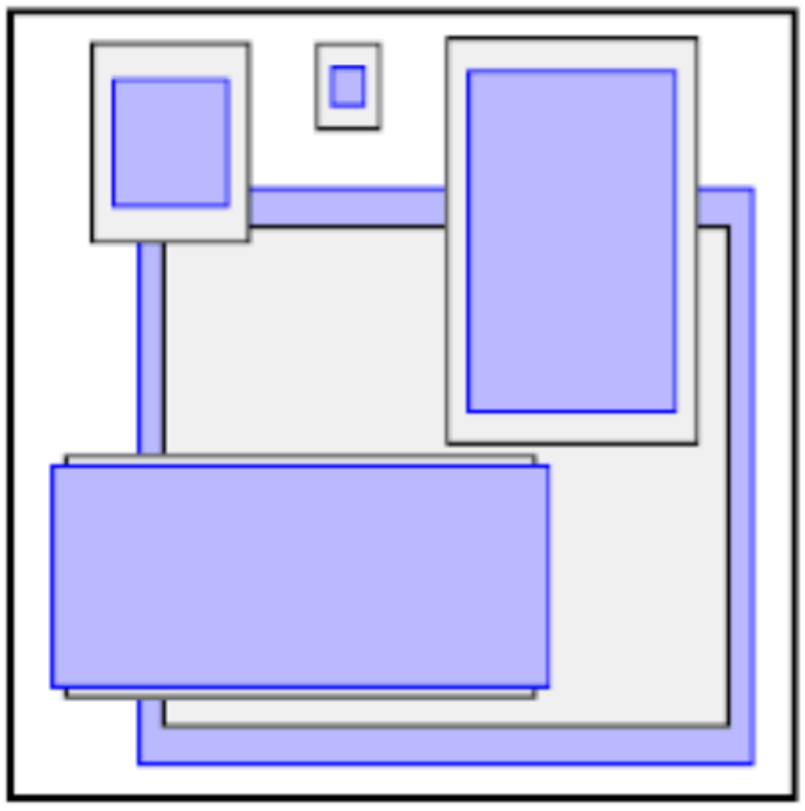
Better

Dimension Clusters

- Instead of choosing priors by hand, we run **k-means clustering**

  (The cluster centroids are **significantly different than hand-picked anchor boxes.**)





Better

Direct location prediction

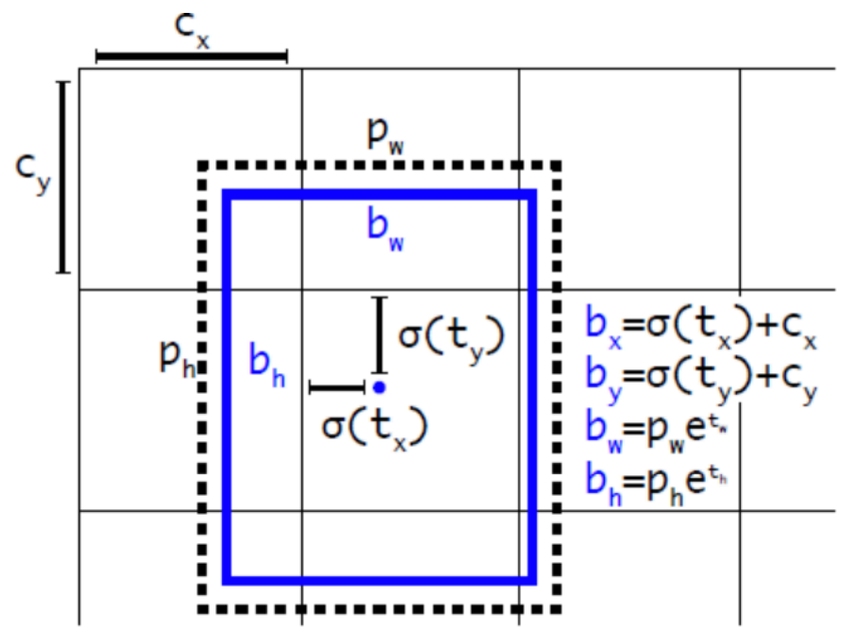
- Model instability comes from predicting the (x,y) locations for the box.
  (especially, during early iterations).

$$x = (t_x * w_a) - x_a$$

$$y = (t_y * h_a) - y_a$$

- YOLO v2 : predict location coordinates **relative to the location of grid cell** (offset X)

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$



Better

Direct location prediction (in RPN)

$$x = (t_x * w_a) - x_a$$

$$y = (t_y * h_a) - y_a$$

For example, a prediction of $t_x = 1$ would shift the box to the right by the width of the anchor box, a prediction of $t_x = -1$ would shift it to the left by the same amount.

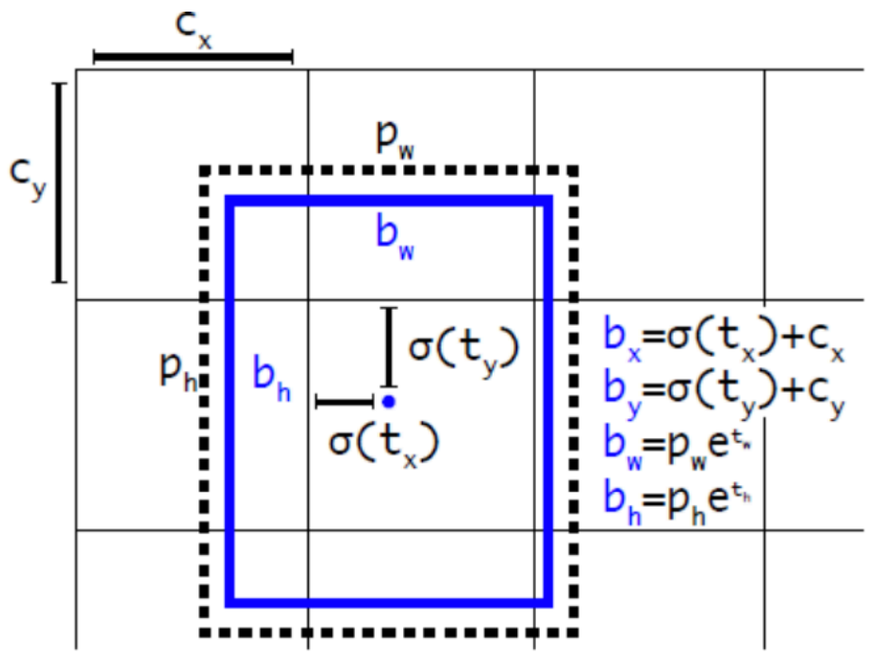
- This formulation is **unconstrained** so any anchor box can end up at any point in the image, regardless of what location predicted the box.
- With random initialization the **model takes a long time to stabilize** to predicting sensible offsets.

Better

Direct location prediction (in YOLO v2)

- Instead of predicting offsets we follow the approach of YOLO and predict location coordinates relative to the **location of the grid cell.**

- This bounds the **ground truth to fall between 0 and 1.**
-> use a **logistic activation** to constrain the network's prediction to fall in this range.

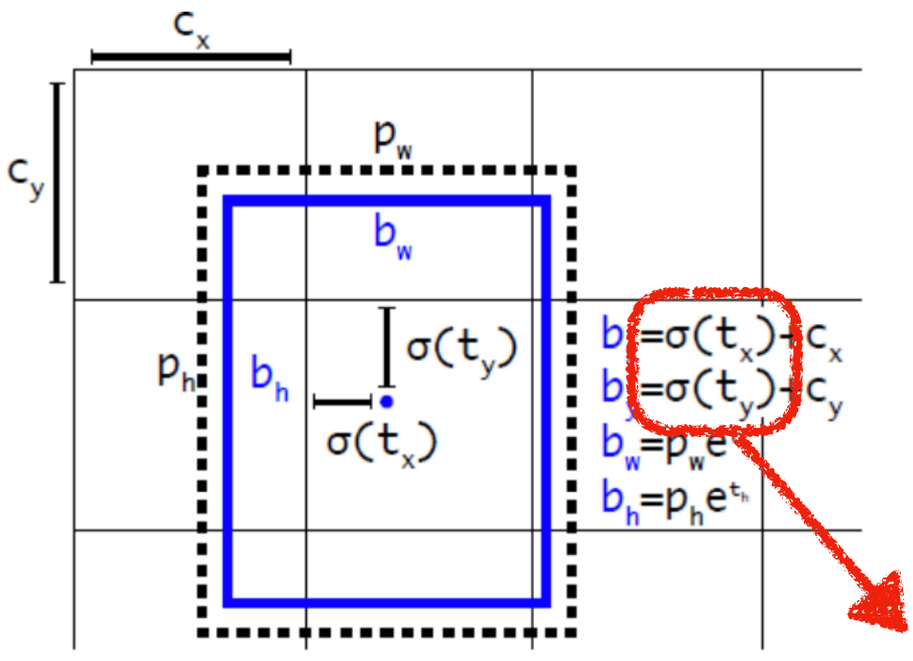


Better

Direct location prediction (in YOLO v2)

- Instead of predicting offsets we follow the approach of YOLO and predict location coordinates relative to the **location of the grid cell.**

- This bounds the **ground truth to fall between 0 and 1.**
  -> use a **logistic activation** to constrain the network's prediction to fall in this range.



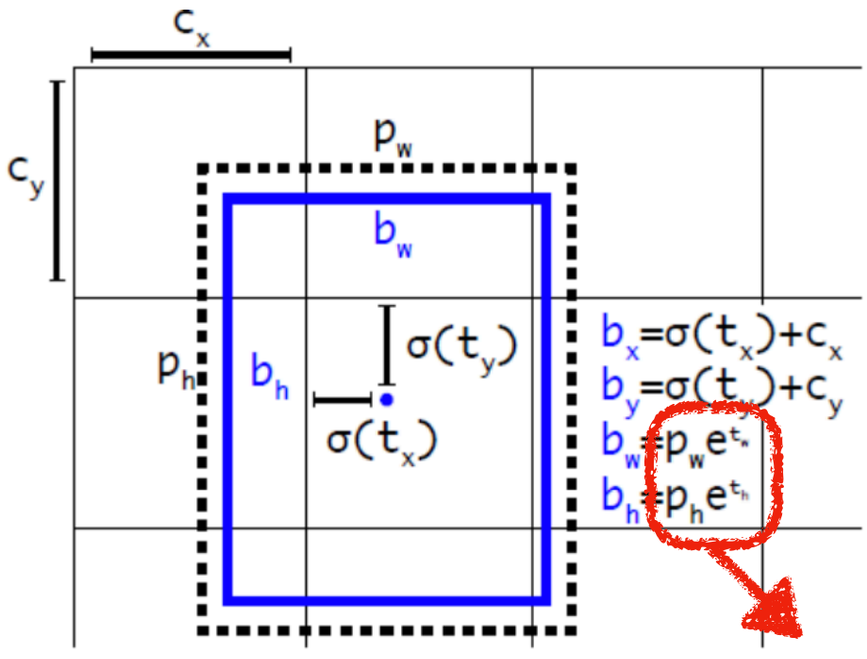
- **because of sigmoid(0) = 0.5, want to make tx=0, ty = 0.**
- **If sigmoid(tx) = 0.5, it is center of grid cell**

Better

Direct location prediction (in YOLO v2)

- Instead of predicting offsets we follow the approach of YOLO and predict location coordinates relative to the **location of the grid cell.**

- This bounds the **ground truth to fall between 0 and 1.**
  -> use a **logistic activation** to constrain the network's prediction to fall in this range.



- **In earlier step we calculated prior width, height.**
- **tw=0, th=0 -> bw = pw, bh = ph.**

Faster

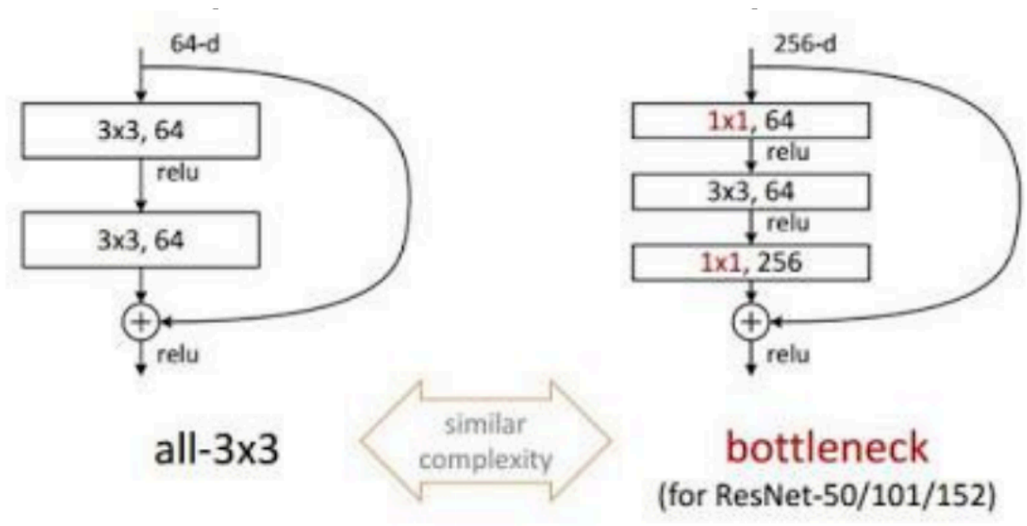
Faster

Darknet-19

- Most detection frameworks rely on VGG-16 as the base feature extractor. (ex. SSD) -> 30.69 billion floating point ops. at 224 x 224

- **YOLO v2**

A. Mostly 3x3 filters
B. Following the work on NIN, use GAP



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			

Stronger

Stronger

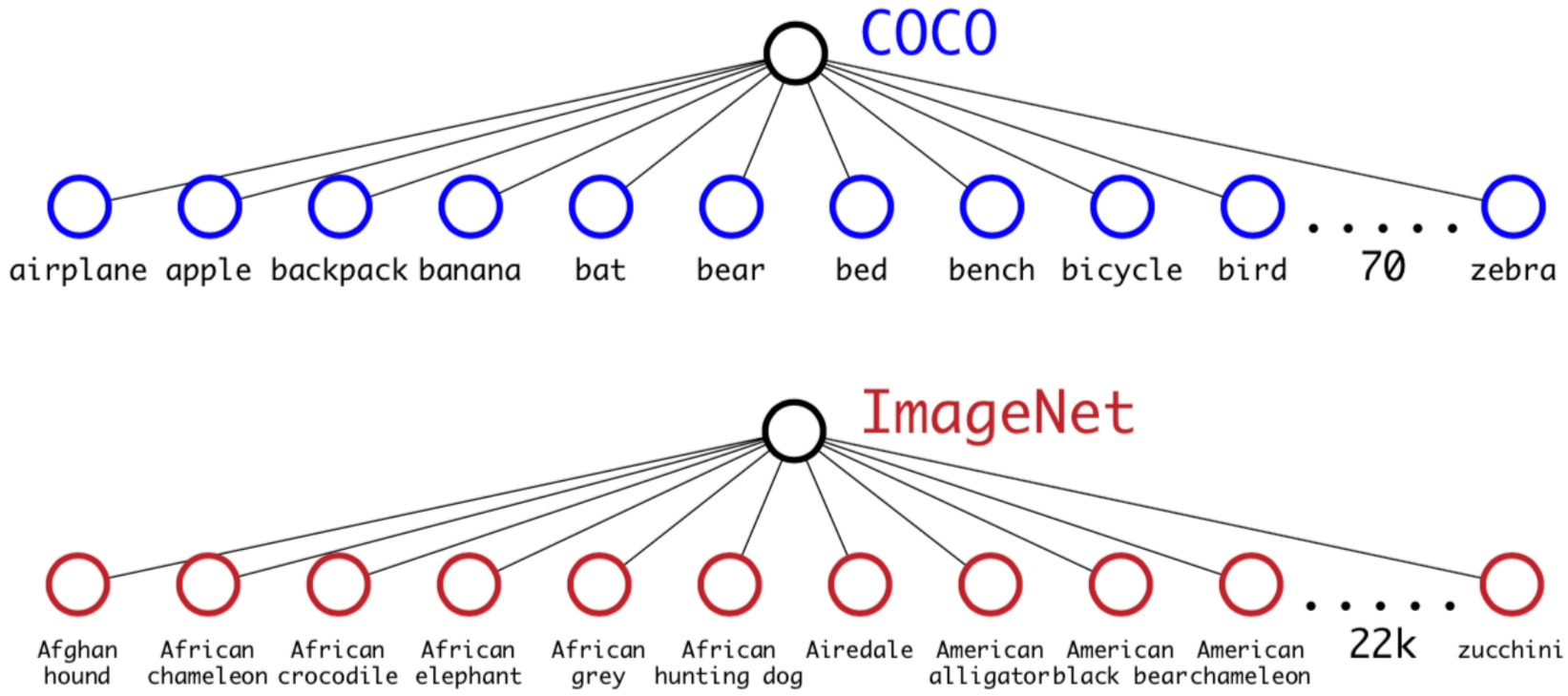
Jointly training on classification and detection data

- The final result is WordTree, a hierarchical model of visual concepts.

$$
\begin{aligned}
&Pr(\text{Norfolk terrier}|\text{terrier}) \\
&Pr(\text{Yorkshire terrier}|\text{terrier}) \\
&Pr(\text{Bedlington terrier}|\text{terrier}) \\
&\cdots
\end{aligned}
$$

$$
\begin{aligned}
Pr(\text{Norfolk terrier}) = {} & Pr(\text{Norfolk terrier}|\text{terrier}) \\
& * Pr(\text{terrier}|\text{hunting dog}) \\
& * \ldots * \\
& * Pr(\text{mammal}|Pr(\text{animal}) \\
& * Pr(\text{animal}|\text{physical object})
\end{aligned}
$$

- Using the joint training, YOLO 9000 learns to find objects in images using the detection data in COCO and it learns to classify a wide variety of these objects using data from ImageNet


COCO
airplane
apple
backpack
banana
bat
bear
bed
bench
bicycle
bird
70
zebra
ImageNet
Afghan hound
African chameleon
African crocodile
African elephant
African grey
African hunting dog
Airedale
American alligator
American black bear
American chameleon
22k
zucchini
WordTree
physical object
animal
artifact
natural object
phenomenon
cat
dog
fish
vehicle
equipment
plant
fungus
ground
water
air
tabby
Persian
car
airplane
houseplant
vascular plant
biplane
jet
airbus
stealth fighter
golden fern
potato fern
felt fern
sea lavender
American twinflower

Conclusion

Conclusion

The path from YOLO to YOLOv2

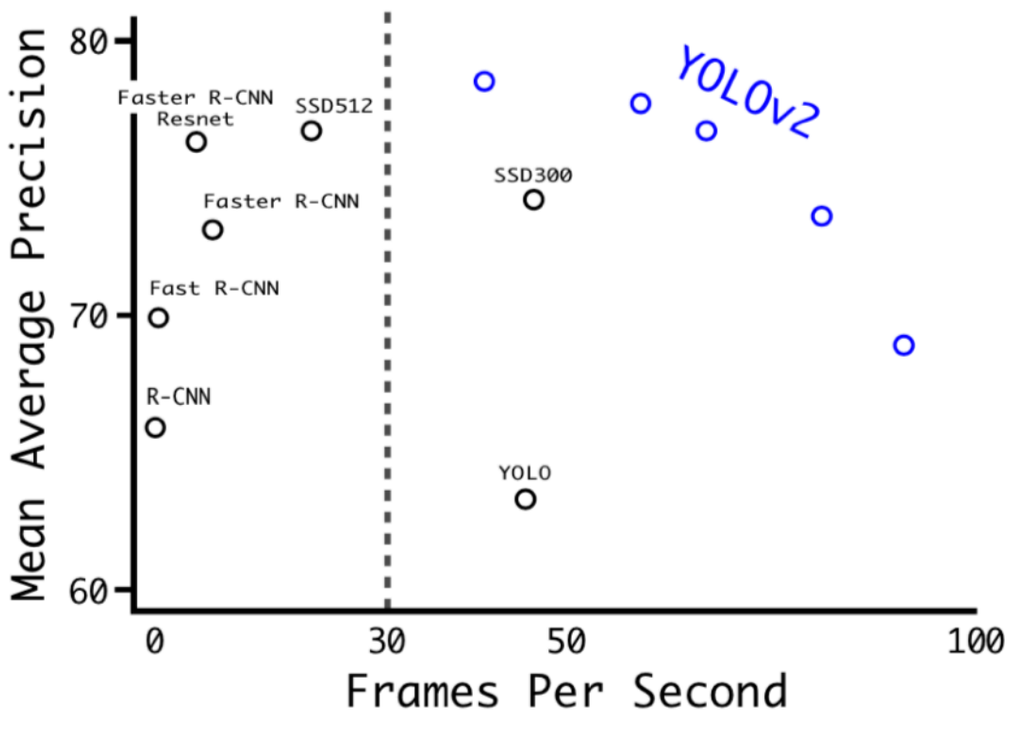
	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?					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

Conclusion

PASCAL VOC2012 test detection results.

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [5]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster R-CNN [15]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [14]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300 [11]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [11]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [6]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

Accuracy and speed on PASCAL VOC 2007.



Code Implementation

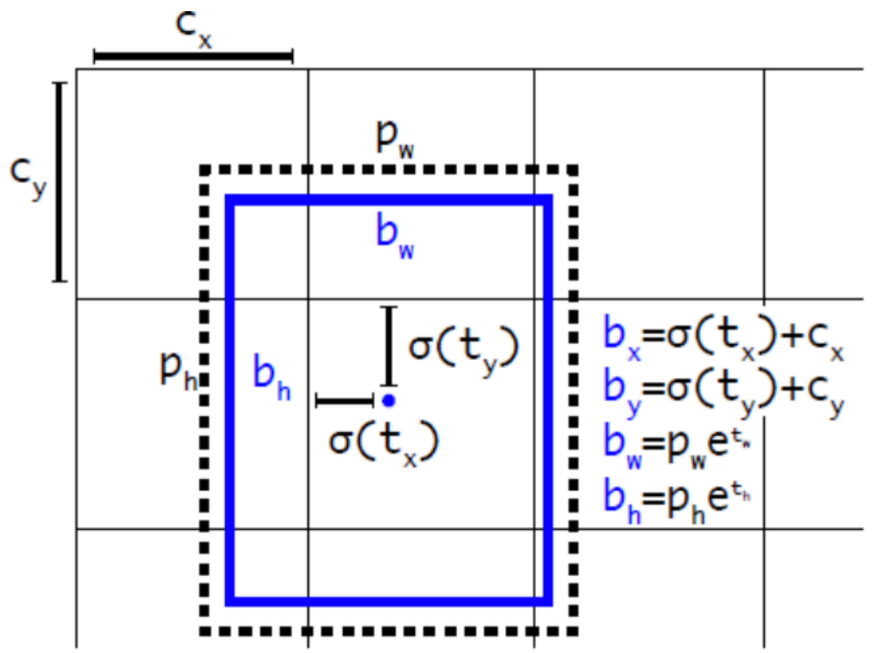
Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	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			

```
net_cfgs = [
    # conv1s
    [(32, 3)],
    ['M', (64, 3)],
    ['M', (128, 3), (64, 1), (128, 3)],
    ['M', (256, 3), (128, 1), (256, 3)],
    ['M', (512, 3), (256, 1), (512, 3), (256, 1), (512, 3)],
    # conv2
    ['M', (1024, 3), (512, 1), (1024, 3), (512, 1), (1024, 3)],
    # ------------
    # conv3
    [(1024, 3), (1024, 3)],
    # conv4
    [(1024, 3)]
]

# darknet
self.conv1s, c1 = _make_layers(3, net_cfgs[0:5])
self.conv2, c2 = _make_layers(c1, net_cfgs[5])
# ---
self.conv3, c3 = _make_layers(c2, net_cfgs[6])

stride = 2
# stride*stride times the channels of conv1s
self.reorg = ReorgLayer(stride=2)
# cat [conv1s, conv3]
self.conv4, c4 = _make_layers((c1*(stride*stride) + c3), net_cfgs[7])

# linear
out_channels = cfg.num_anchors * (cfg.num_classes + 5)
self.conv5 = net_utils.Conv2d(c4, out_channels, 1, 1, relu=False)
self.global_average_pool = nn.AvgPool2d((1, 1))
```




```
# tx, ty, tw, th, to -> sig(tx), sig(ty), exp(tw), exp(th), sig(to)
xy_pred = F.sigmoid(global_average_pool_reshaped[:, :, :, 0:2])
wh_pred = torch.exp(global_average_pool_reshaped[:, :, :, 2:4])
bbox_pred = torch.cat([xy_pred, wh_pred], 3)
iou_pred = F.sigmoid(global_average_pool_reshaped[:, :, :, 4:5])

score_pred = global_average_pool_reshaped[:, :, :, 5:].contiguous()
prob_pred = F.softmax(score_pred.view(-1, score_pred.size()[-1])).view_as(score_pred)  # noqa
```

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(x_i - \hat{x}_i \right)^2 + \left(y_i - \hat{y}_i \right)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \qquad (3)
\end{aligned}
$$

```
self.bbox_loss = nn.MSELoss(size_average=False)(bbox_pred * box_mask, _boxes * box_mask) / num_boxes  # noqa
self.iou_loss = nn.MSELoss(size_average=False)(iou_pred * iou_mask, _ious * iou_mask) / num_boxes  # noqa

class_mask = class_mask.expand_as(prob_pred)
self.cls_loss = nn.MSELoss(size_average=False)(prob_pred * class_mask, _classes * class_mask) / num_boxes  # noqa
```