You Only Look Once:

Unified, Real-Time Object Detection

- Joseph Redmon, Santosh Divvala, Ross Firshick, Ali Farhadi

Department of Computer Science, Hyeseung Lee.

CONTENTS

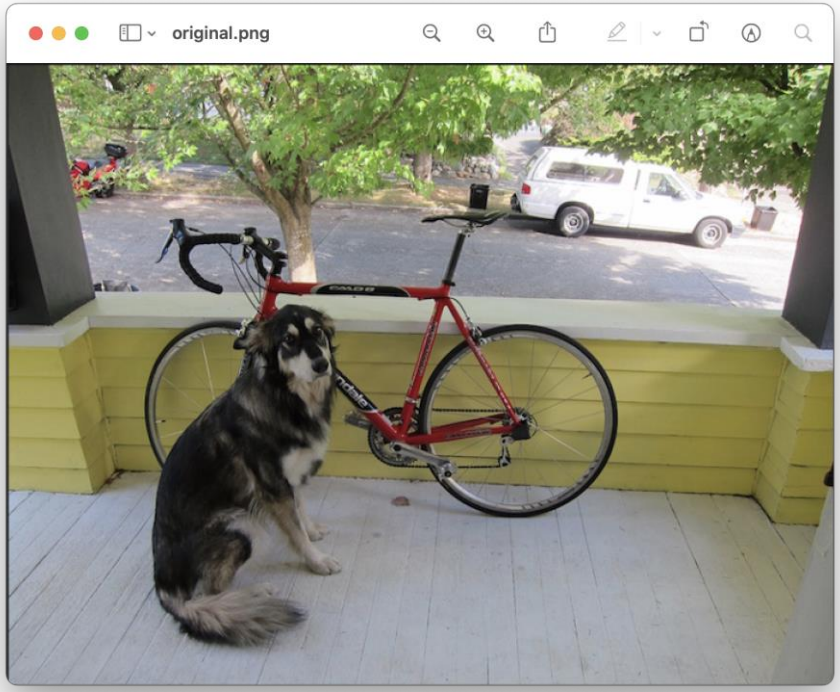
01

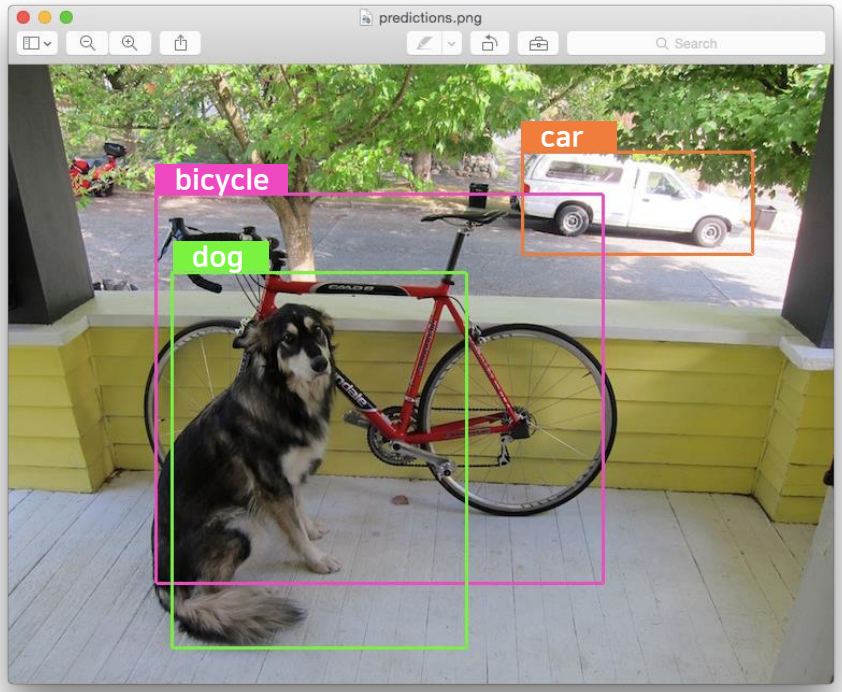
Object Detection

Brief description of object detection task.

Object Detection?

- classification + localization task for multiple objects.
  - multi-class classification
  - localization. i.e, bounding box regression(using (x,y) coordinates)
- 이미지 혹은 영상에서 여러 개의 object가 각각 어떤 것인지 분류하고, 이미지 내의 위치를 찾아내는 두 가지 task를 순차적 혹은 동시에 수행.





02

You Only Look Once

Contents of the YOLO papers.

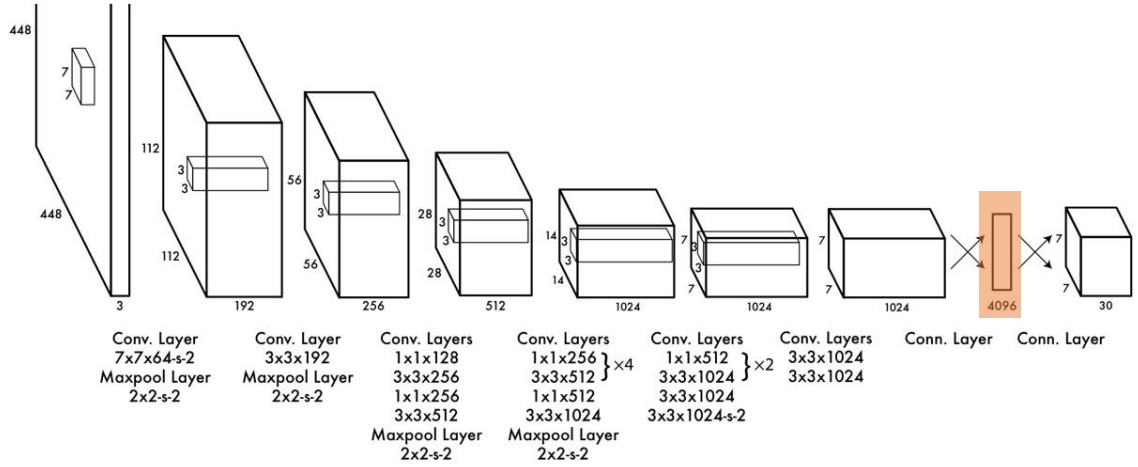
AIM

- 사람처럼 **실시간으로** 객체 탐지를 하기 위한 새로운 네트워크의 개발
- 이전의 object detection: classification, bounding box regression을 따로 진행
  - 느리다.

- YOLO: end-to-end방식으로 한 번에 최적화
- HOW?
  - Regression problem으로 재구성 -> Single neural network으로 multiple 클래스 확률, bounding box 예측

+ 빠르다.
  - 속도: 45FPS, fast yolo – 155FPS
+ context를 담아서 global하게 이미지를 파악
- localization error
  - But False positive ↓
  - Object가 아닌데 object로 예측하는 경우 ↓

Network Design

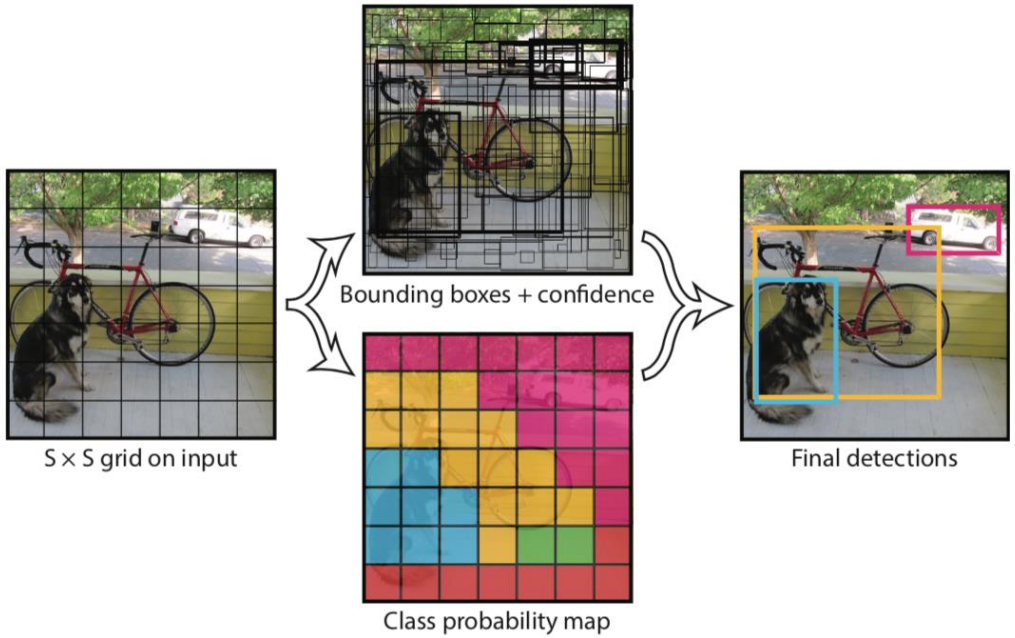


- Convolution neural network로 구현.

- Convolution layers: extract features from image.
- Fully-connected layer: predict probabilities of the output, and coordinates

- 구글넷 기반
- 구글넷의 inception module 대신 1x1 convolution reduction layer, 3x3 convolutional layer 사용.

Unified Detection



- Input image를 S x S grid로 분할
- 각 grid cell에 object의 중심이 있으면, detect 했다고 표기
- 각 grid cell은 B개의 b-box와
  $confidence\ score = \Pr(object) * IOU_{pred}^{truth}$ 를 예측
- 각 b-box는 $x, y, w, h, confidence$로 구성
- 각 grid cell은 $C = Pr(class_i \mid object)$ 를 예측.

S: grid cell 사이즈

B: grid cell마다 갖는/예측하는 bounding box 개수

C: 클래스 확률의 집합

Confidence score: 박스가 object를 포함하고 있는 것 + 예측된 bounding box 위치의 정확도를 반영

S=7, B=2, # of C = 20

Loss Function

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

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

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

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

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

- $\mathbb{I}_i^{obj}$: $cell_i$에 object가 존재하는 지에 대한 indicator
- $\mathbb{I}_{ij}^{obj}$: $cell_i$의 box_j 가 responsible한지의 indicator
- C_i : $cell_i$의 confidence score
- λ_{coord}: coordinates에 대한 loss, 다른 loss들 간의 balancing parameter
- λ_{noobj} : object가 있는 box와 없는 box의 balancing parmeter

1) Localization Loss
   Bounding box 의 위치에 대한 loss.

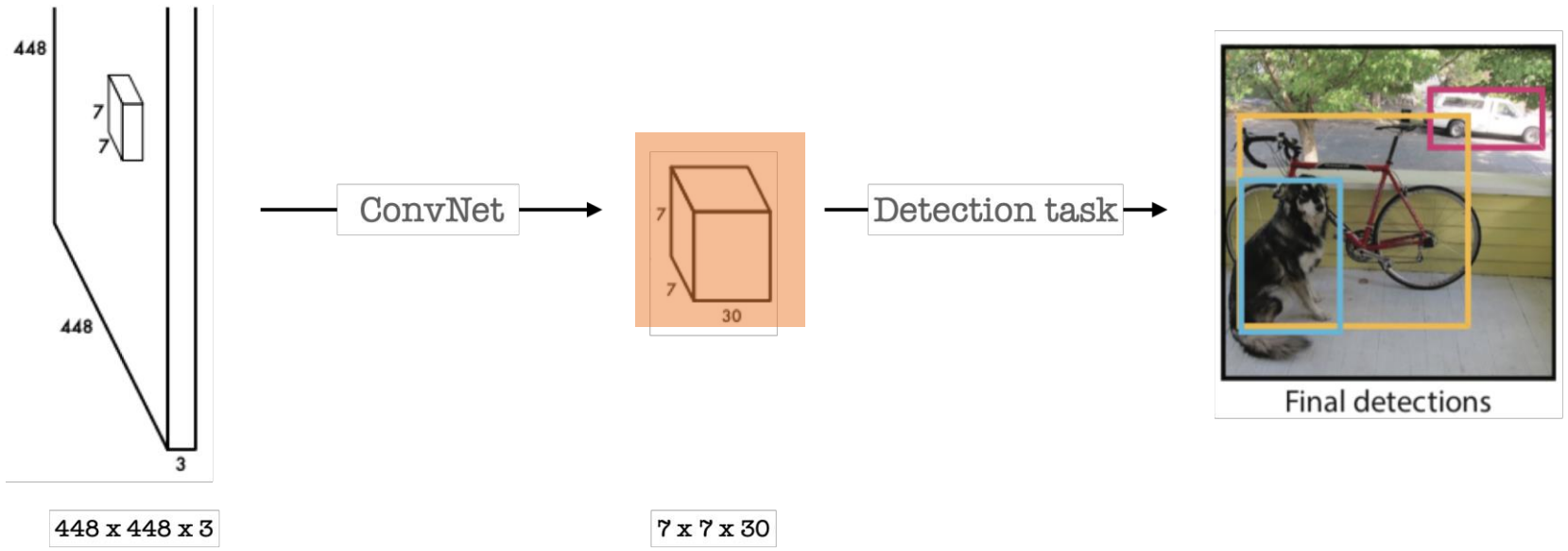
2) Localization Loss
   Bounding box의 크기에 대한 loss. 사이즈가 큰 box의 경우, 상대적으로 iou에 영향을 적게 주기때문에, sum-squared error 를 측정한다.

3) Confidence Loss ($C_i = 1$)
   Grid cell에 object가 있을 때의 confidence 오차.

4) Confidence Loss ($C_i = 0$)
   Grid cell에 object가 없을 때의 confidence 오차.

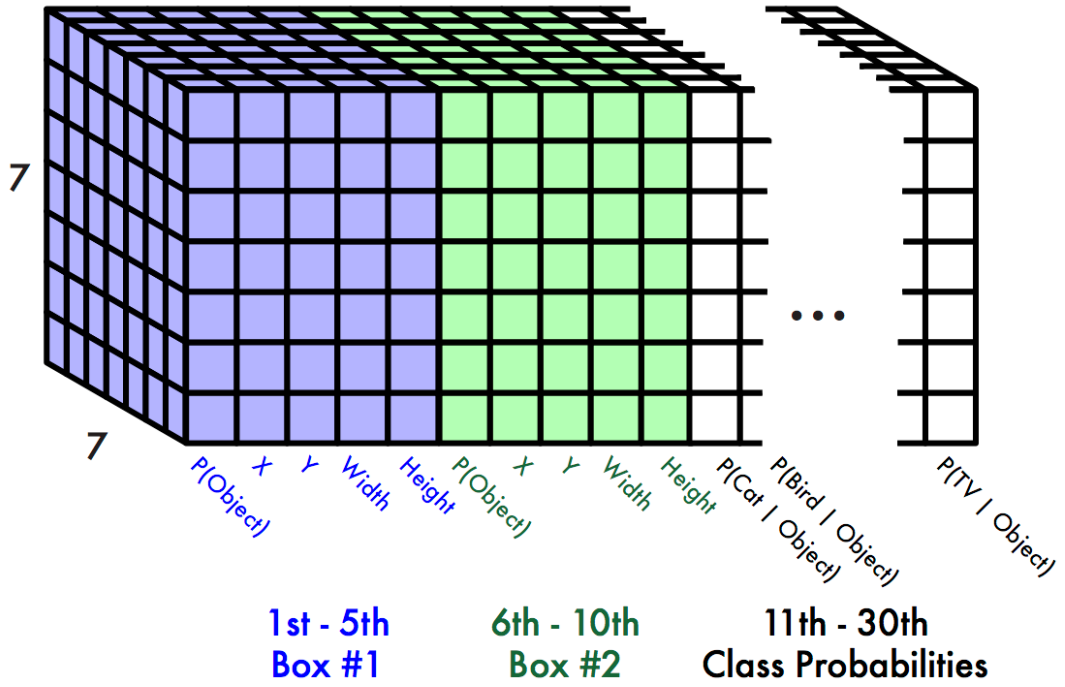
5) Classification Loss
   Conditional class probability의 loss

Inference Algorithm



1. 이미지를 일련의 conv layer에 넣고 특징들을 추출하면, 결론적으로 7 x 7 x 30 이라는 예측 텐서가 나온다.
2. 해당 예측 텐서는 각 그리드 셀의 예측된 클래스 확률과 2(B)개의 bounding box 정보로 구성 되어있다.
3. 낮은 확률의 예측을 삭제하고, 20개의 클래스에 대해 독립적으로 각 클래스로 감지된 object에 대해 NMS를 적용
   → object에 대한 최종 bounding box가 남는다.

Construction of a prediction tensor



Construction of a prediction tensor

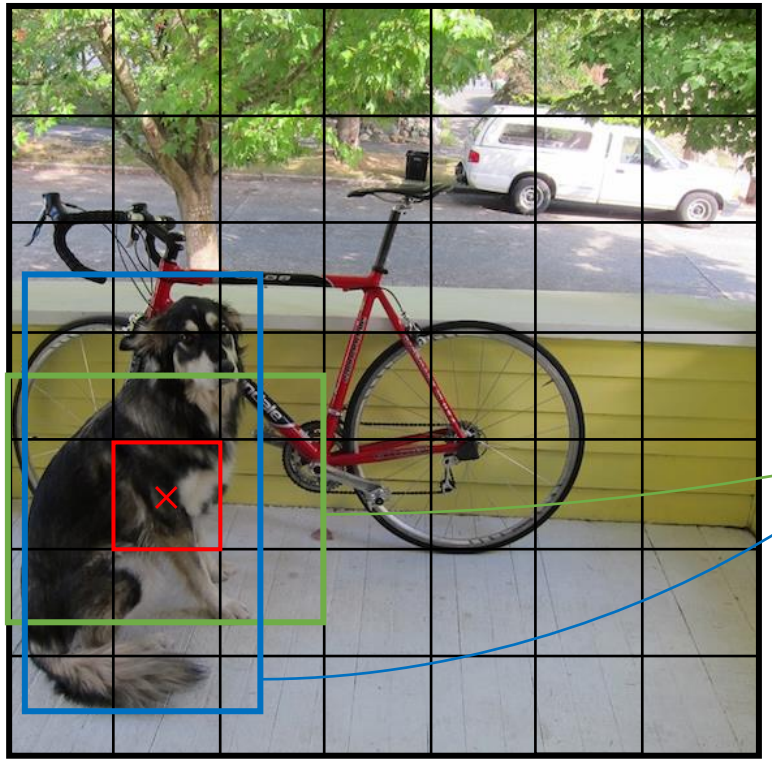
Prediction tensor: **1 x 1 x 30**

x	y	w	h	c	x	y	w	h	c	C_1	.	.	.	C_{20}

Bounding box 1

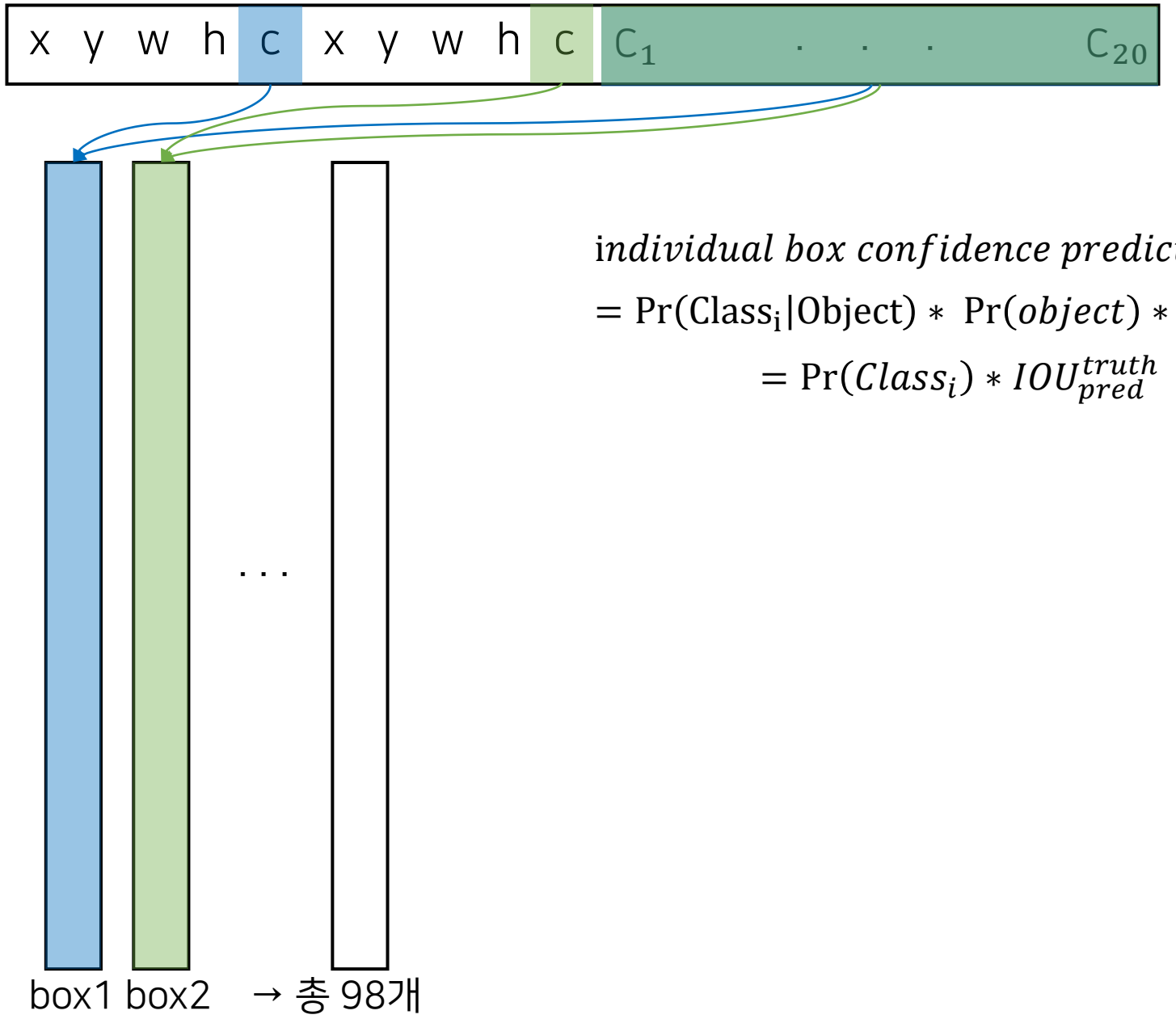
Bounding box 2

해당 grid cell에 나타나는 object의 클래스 확률



Input : **448 x 448 x 3**

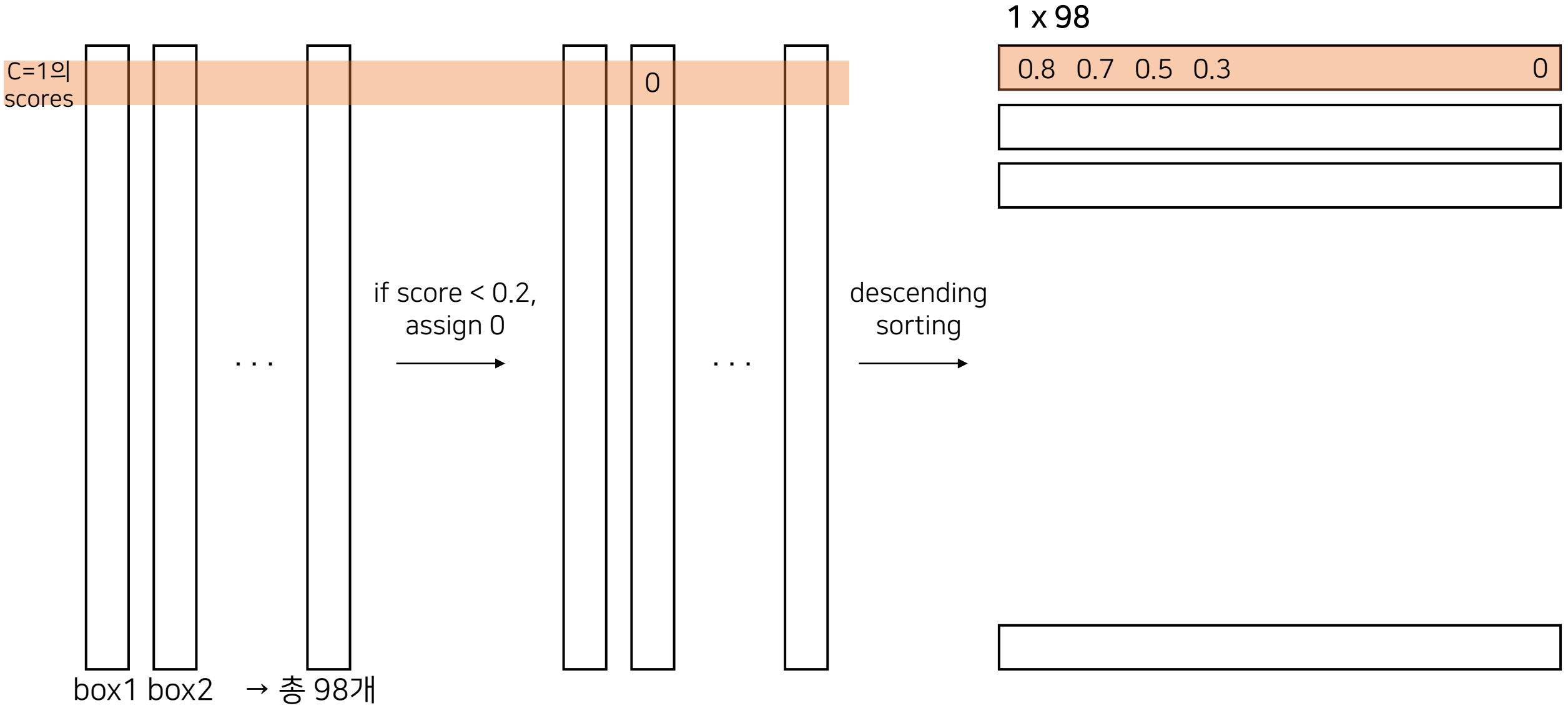
Prediction tensor: 1 x 1 x 30



$$individual\ box\ confidence\ predictions$$
$$= \Pr(\text{Class}_i|\text{Object}) * \Pr(object) * IOU_{pred}^{truth}$$
$$= \Pr(Class_i) * IOU_{pred}^{truth}$$

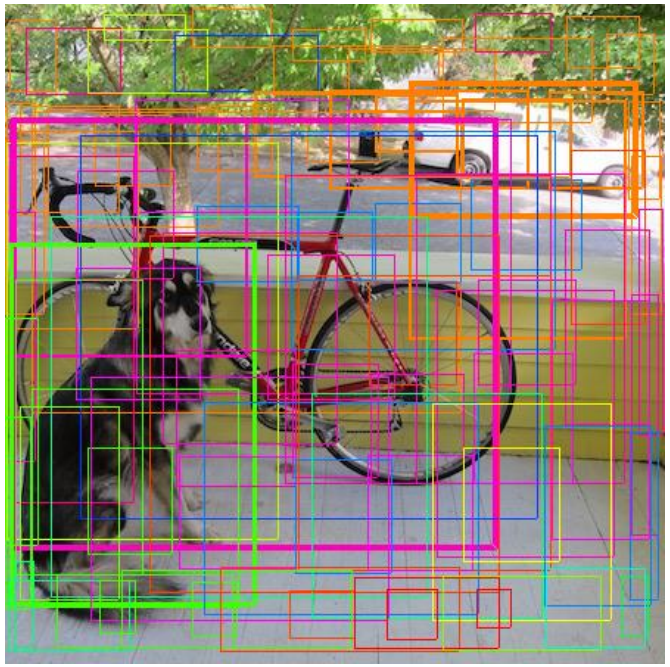
Prediction tensor: 1 x 1 x 30

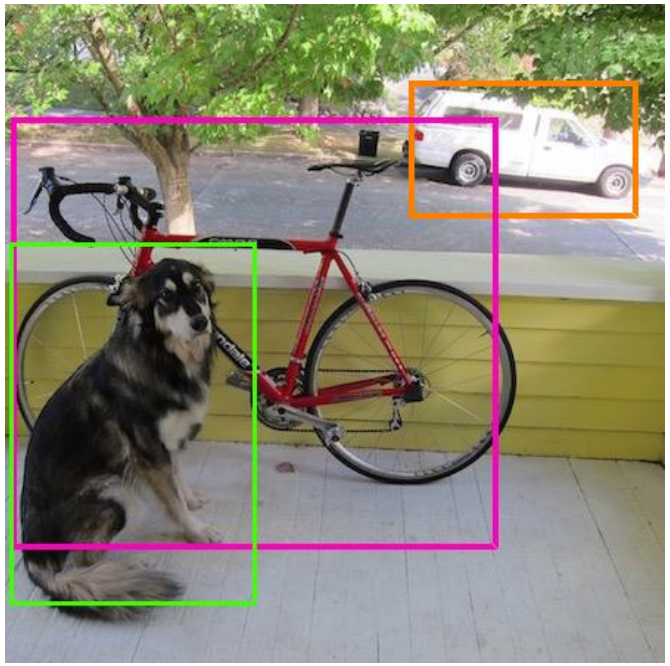
x	y	w	h	c	x	y	w	h	c	C_1	.	.	.	C_{20}



Non max suppression

```
def non_max_suppression(prediction, conf_thres=0.1, iou_thres=0.6, merge=False,
classes=None, agnostic=False):
    nc = prediction[0].shape[1] - 5  # number of classes
    xc = prediction[..., 4] > conf_thres

    output = [None] * prediction.shape[0]
    for xi, x in enumerate(prediction):  # image index, image inference
        x = x[xc[xi]]  # confidence

        x[:, 5:] *= x[:, 4:5]  # conf = obj_conf * cls_conf

        # Box (center x, center y, width, height) to (x1, y1, x2, y2)
        box = xywh2xyxy(x[:, :4])


        conf, j = x[:, 5:].max(1, keepdim = True)
        x = torch.cat( (box, conf, j.float()), 1 )[conf.view(-1) > conf_thres]

        # NMS
        c = x[:, 5:6] * (0 if agnostic else max_wh)  # classes
        boxes, scores = x[:, :4] + c, x[:, 4]  # boxes (offset by class), scores
        i = torchvision.ops.boxes.nms(boxes, scores, iou_thres)

        output[xi] = x[i]
return output
```





Limitation

- 각각의 grid cell에서 2개의 예측한 bounding boxes를 얻고, 하나의 class만 가질 수 있으므로 겹치는 **물체에** 대해 잘 감지 못 한다.
    - Pedestrain detection에서 극복해야 할 issue
- YOLO 자체가 작은 물체를 잘 못 잡는다.

- NMS시에 사용하는 threshold 2개 값에 따라 mAP성능이 크게 바뀐다.

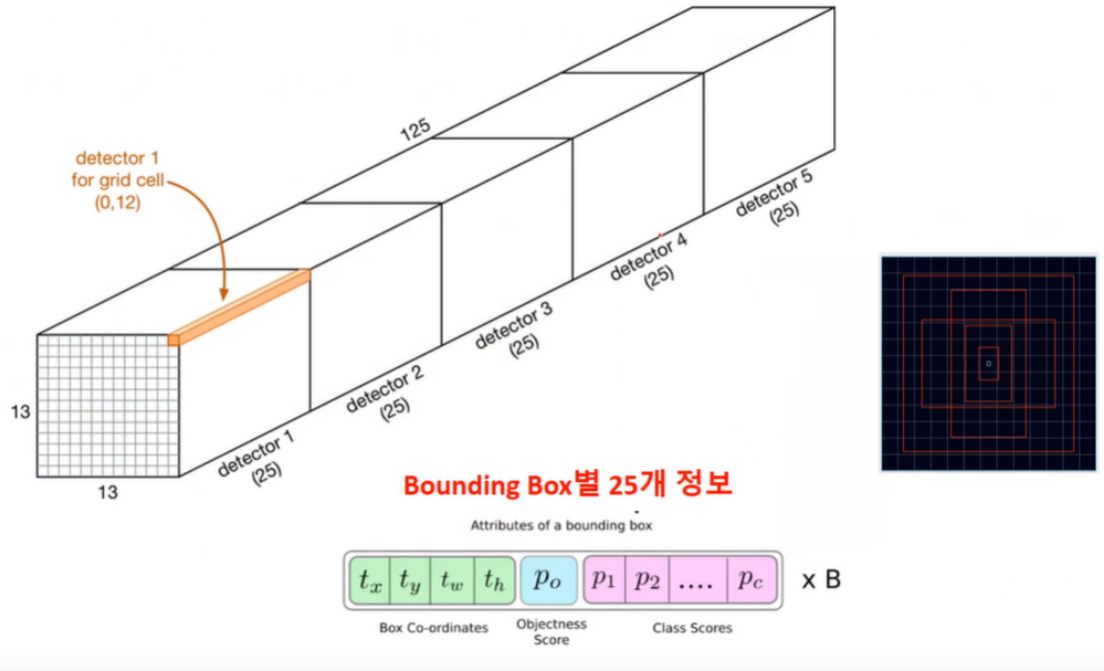
YOLO v2, v3

- Batch normalization 레이어 추가 -> 학습 속도를 향상
- 이후, yolov2, yolov3에서는 Darknet-19, 53라는 새로운 네트워크를 제안.
    - FC-layer대신 convolution layer를 사용함

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

YOLO v2, v3

- 각 bounding box마다 class probabilities, confidence score를 예측하도록 함.
  - 겹쳐진 object들을 잡을 수 있는 구조가 됨.

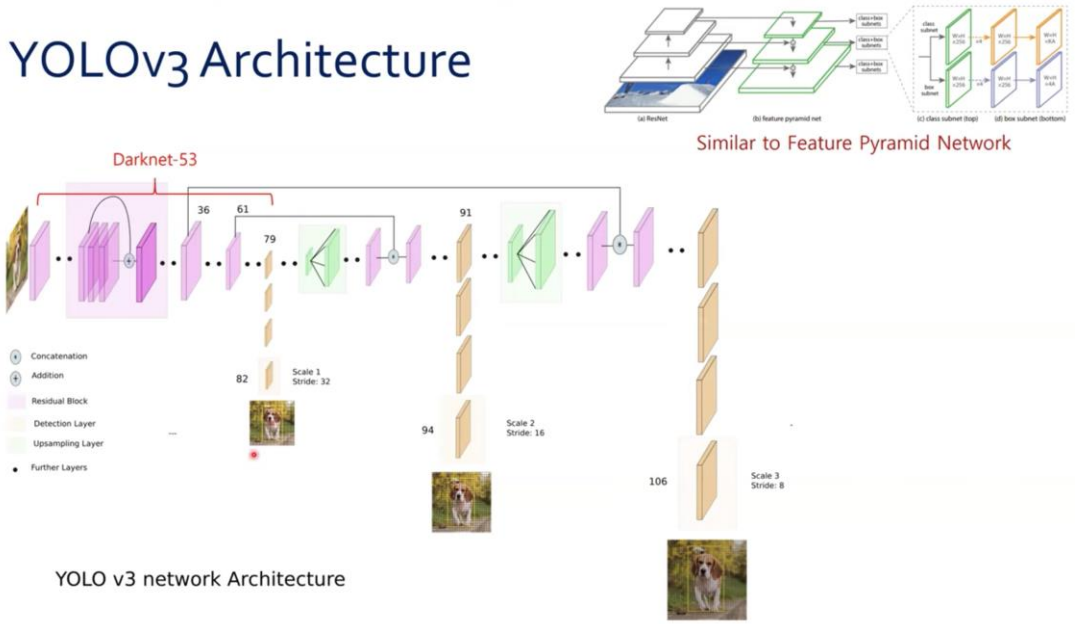


YOLO v2, v3

- 사전에 anchor사이즈를 정의 (with k-means clustering)
  - 트레이닝 데이터 셋의 ground truth bounding box들을 k-means clustering
  - if the task is pedestrian detection, it's more efficient when the default box is vertically long.

- multi-label classification을 할 수 있도록,
  binary cross-entropy 로 변경 (v1: softmax)

- Network architecture의 변화
  - head – Darknet backbone,
  - neck – FPN과 유사한 형태. 작은 object의 feature들을 잃지 않게 하는 효과.



03

w/ YOLO ?

What I did to upgrade the network

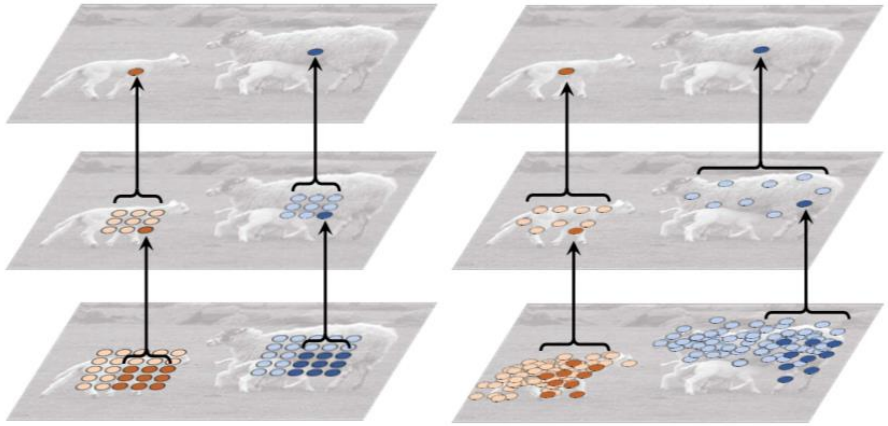
What I did more?

```
def non_max_suppression(prediction, conf_thres=0.1, iou_thres=0.6, merge=False,
classes=None, agnostic=False):
    nc = prediction[0].shape[1] - 5  # number of classes
    xc = prediction[..., 5] > conf_thres  # head candidates

    multi_label = nc > 1  # multiple labels per box (adds 0.5ms/img)

    output = [None] * prediction.shape[0]
    for xi, x in enumerate(prediction):  # image index, image inference
        x = x[xc[xi]]  # head confidence

        x[:, 5:] *= x[:, 4:5]  # conf = obj_conf * cls_conf

        # Box (center x, center y, width, height) to (x1, y1, x2, y2)
        box = xywh2xyxy(x[:, :4])

        if multi_label:
            j, _ = (x[:, 6:] > conf_thres).nonzero(as_tuple=False).T
            j = torch.unique(j)
            if j.nelement() == 0:
                continue
            original_x = x[j, :]
            original_box = box[j, :]
            x = x[j, :6]
            box = box[j]

        # Batched NMS
        c = x[:, 5:6] * (0 if agnostic else max_wh)  # classes
        boxes, scores = box, x[:, 5]  # boxes (offset by class), scores
        i = torchvision.ops.boxes.nms(boxes, scores, iou_thres)

        value, indices = original_x[i, 6:].max(1) # 1 ~ 7 헤드방향 중 베스트 저장
        indices += 1
        headpose = torch.cat( (original_box[i], value.view(-1, 1), indices.view(-1,
1)), 1)
        head = headpose.clone().detach()
        head[:, 5:6] = float(0)
        predictions_after_nms = torch.cat((headpose, head), 0)
        output[xi] = predictions_after_nms
return output
```

- Variation인 Scaled yolo v4를 활용
  - multi-label이 가능하지만, 독립적으로 class 확률들이 계산되기 때문에 hierarchical 혹은 포함되는 관계의 클래스들을 동시에 예측할 때는 바로 적용이 불가
  - ex) 사람으로 예측되었을 때, 사람의 attribute를 예측

- (해결) head/person인 것을 먼저 예측하고, head로 예측된 b-box에 대해서만 head 방향을 multi-class형태로 예측하도록 함.

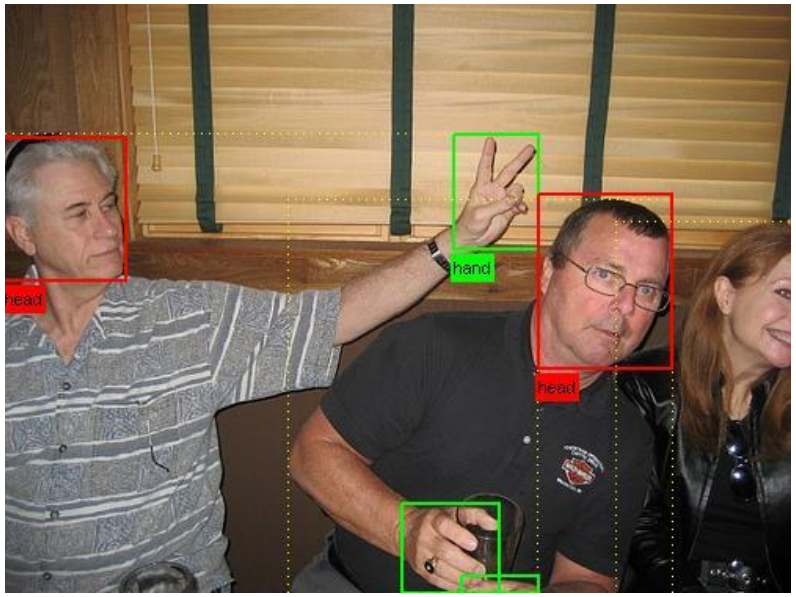
What I did more?



(a) standard convolution (b) deformable convolution

- pedestrian은 카메라에 가깝게 잡히지는 않기 때문에, crowd 데이터를 기준으로 학습
- 회사에서 캠으로 시연을 할 때, 큰 사람이 아예 detect 되지 않거나 눈 코 입을 다 따로 잡는 문제가 발생
- 다양한 크기를 잡기 위해서 conv 레이어를 **deformable conv layer**로 한 개 변경함 (pp-yolo참조)
  - DeConv layer: 작은 object에는 작은 filter, 큰 object에는 큰 filter를 사용하는 flexible filter를 적용
  - Input feature map이 어디로 이동할 지에 대해 학습하는 offset field있기때문에, 속도가 느려지는 단점.

What I did more?



- Augmentation crop방법으로는 큰 얼굴을 학습하기에 무리
- 얼굴이 뚜렷하고 크게 보이고, head 방향에 대한 annotation을 갖는 open 이미지 데이터 셋을 탐색
- But 이미지 annotation이 우리 팀의 기준과 다르고 엉망!

- (해결) VOC2011 데이터 셋에서 얼굴과 상반신까지만 나온 이미지를 약 800개를 골라서, 다시 annotation을 진행하여 추가 학습

reference

- Darknet wiki: https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detection
- YOLO CVPR 2016: https://goo.gl/Xj2Eik
- PR-002 Deformable Convolutional Networks(2017): https://www.youtube.com/watch?v=RRwaz0fBQ0Y
- Scaled-YOLOv4 paper: https://arxiv.org/abs/2011.08036
- Scaled-YOLOv4 github: https://github.com/WongKinYiu/ScaledYOLOv4/tree/yolov4-csp