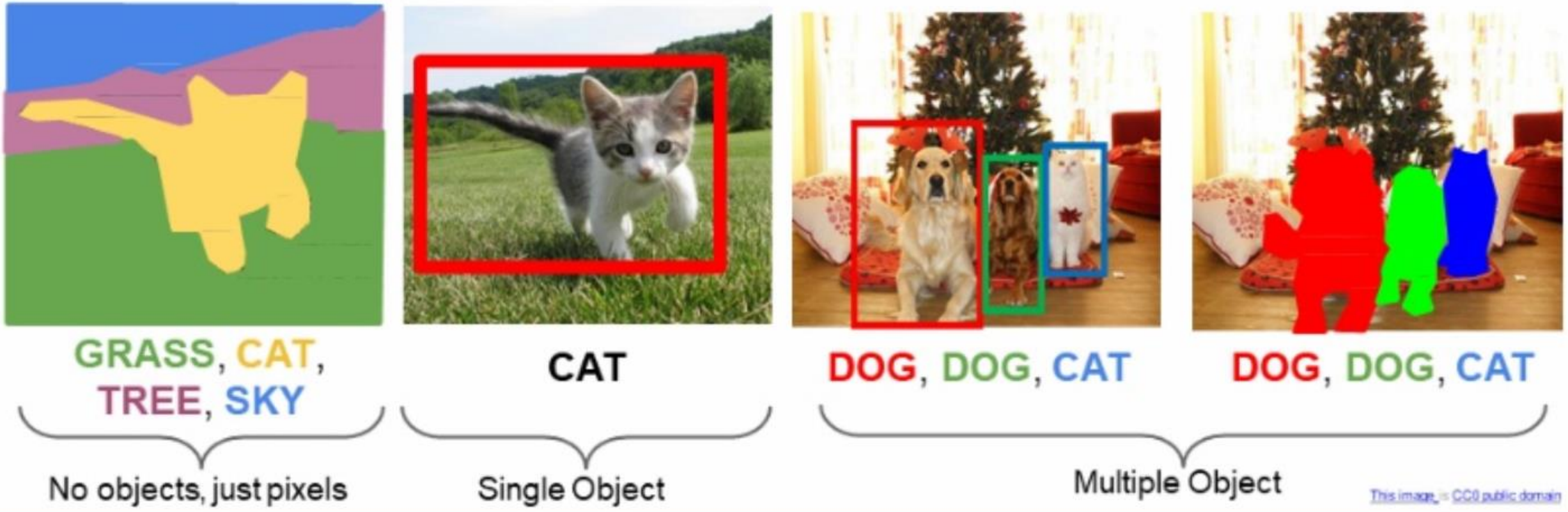
Object Detection
YOLO(You Only Look Once)

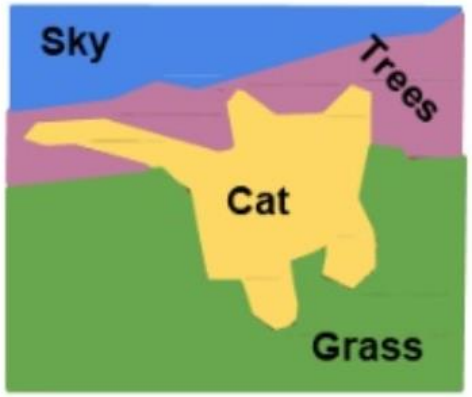
Semantic Segmentation to Instance Segmentation

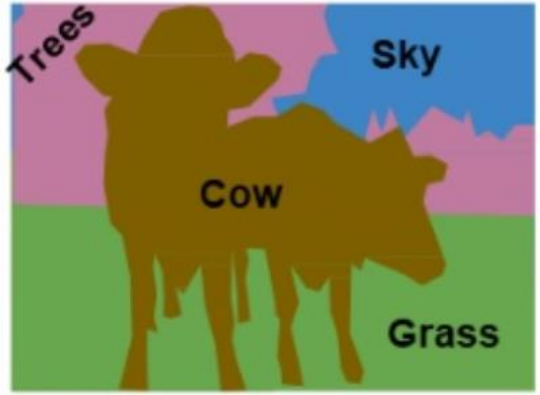


Semantic Segmentation

Label each pixel in the image with a category label

Don't differentiate instanc es, only care about pixels





CNN for Object Recognition

This image is CC0 public domain

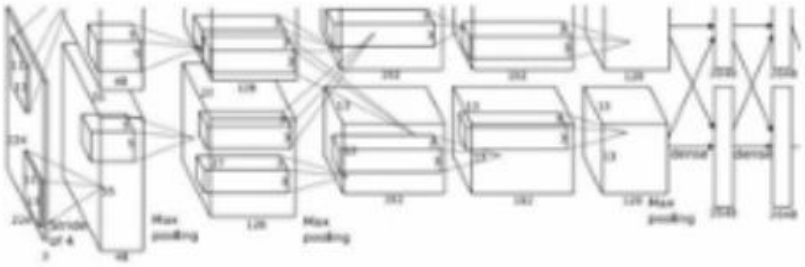


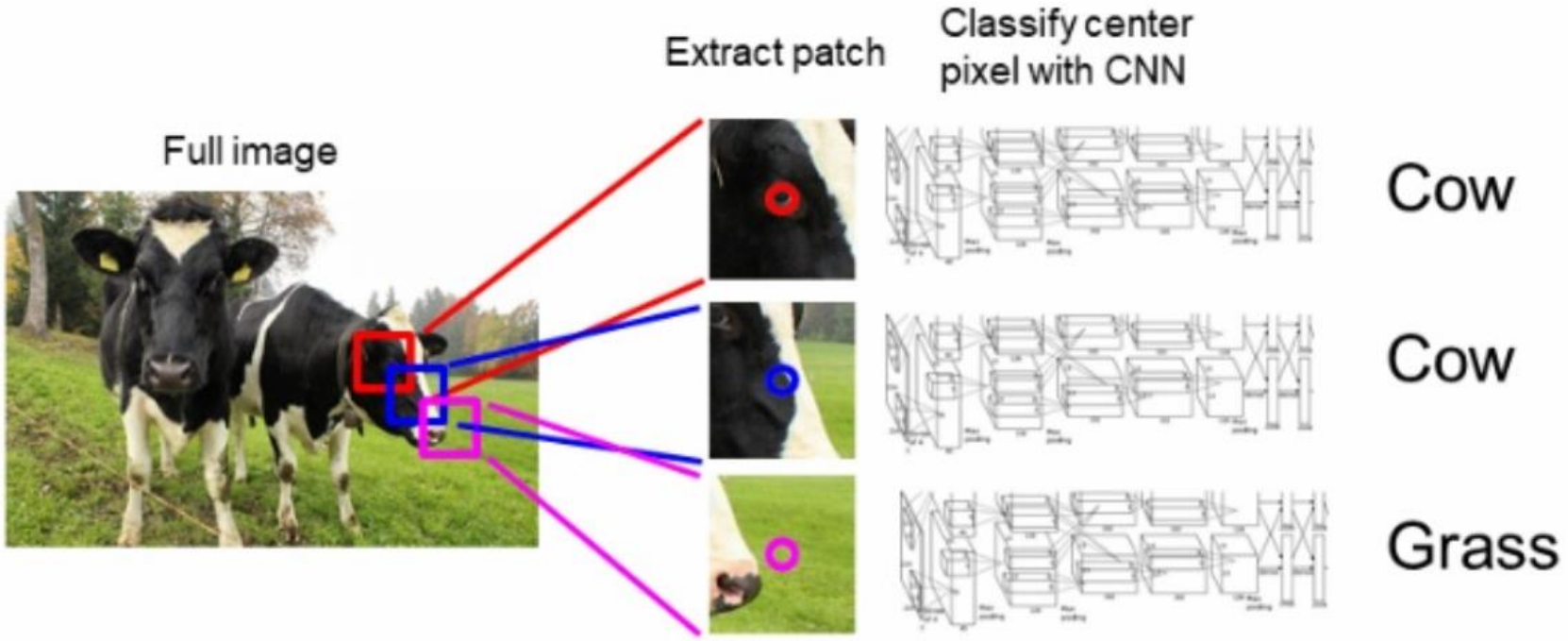
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:
4096

Fully-Connected:
4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Semantic Segmentation Idea: Sliding Window



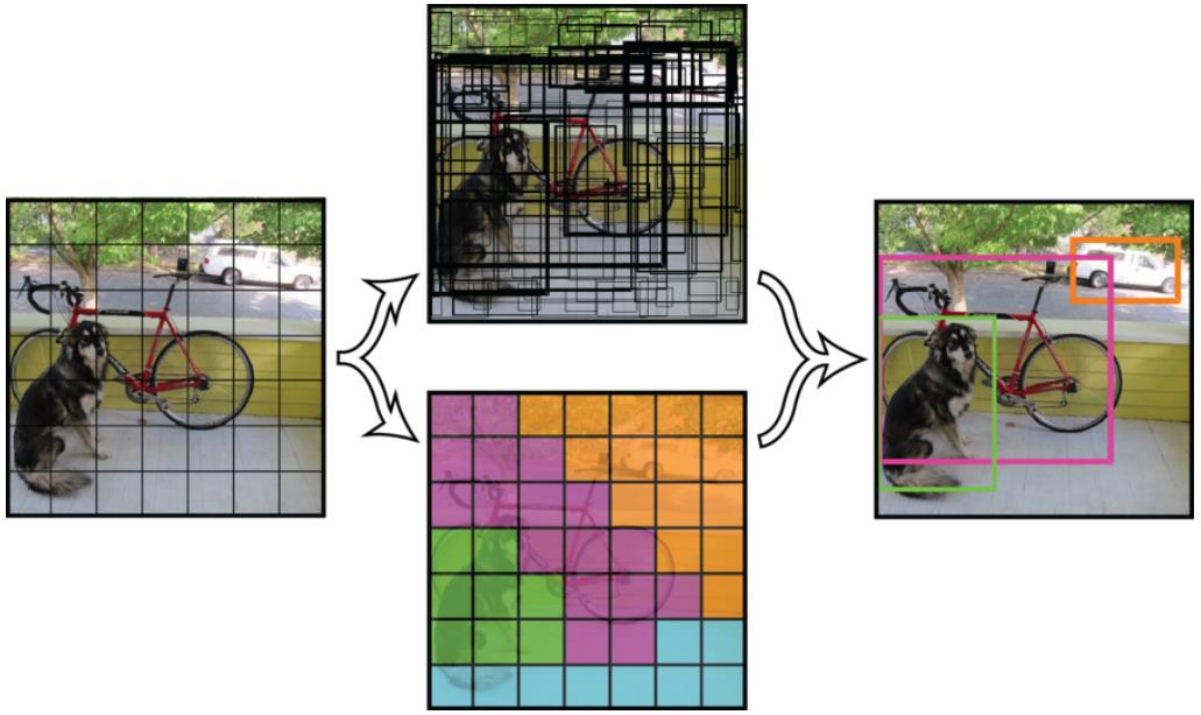
Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

After Segmentation

He et al, "Mask R-CNN", arXiv 2017
Figures copyright Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, 2017.
Reproduced with permission.

https://www.youtube.com/watch?v=MPU2HistivI
https://www.youtube.com/watch?v=OOT3UIXZztE
https://www.youtube.com/watch?v=KYNDzlcQMWA

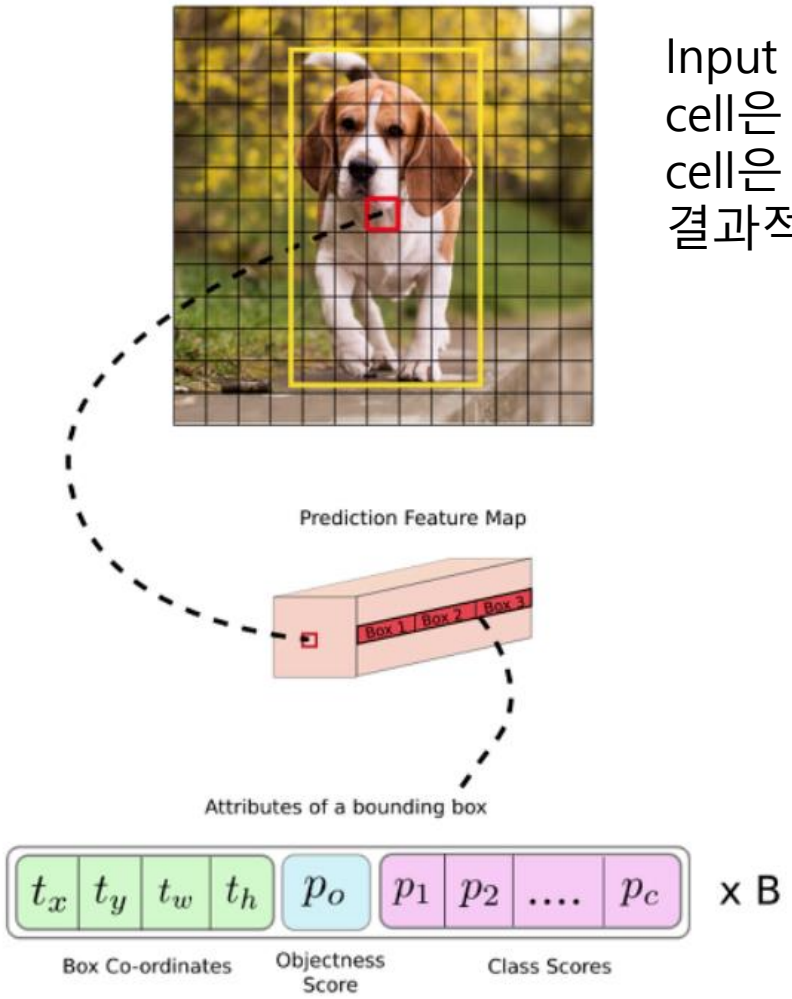
YOLO(You Only Look Once)



R-CNN : 20s
Fast R-CNN : 2s(0.5fps)
Faster R-CNN : 140ms(7~8fps)
YOLO : 45FPS, 155FPS

R-CNN은 region proposal이라는 수백개의 이미지 후보를 생성하고 각각에 대해서 분류
YOLO는 격자 그리드로 나누어 한 번에 클래스를 판단하고 이를 통합해 최종 객체를 구분
기존의 방법들과 달리 Object detection을 **이미지 픽셀 좌표에 대응되는 bounding box**을 찾음
그리고 이에 대한 **class확률**을 구하는 **Single Regression Problem**으로 해결

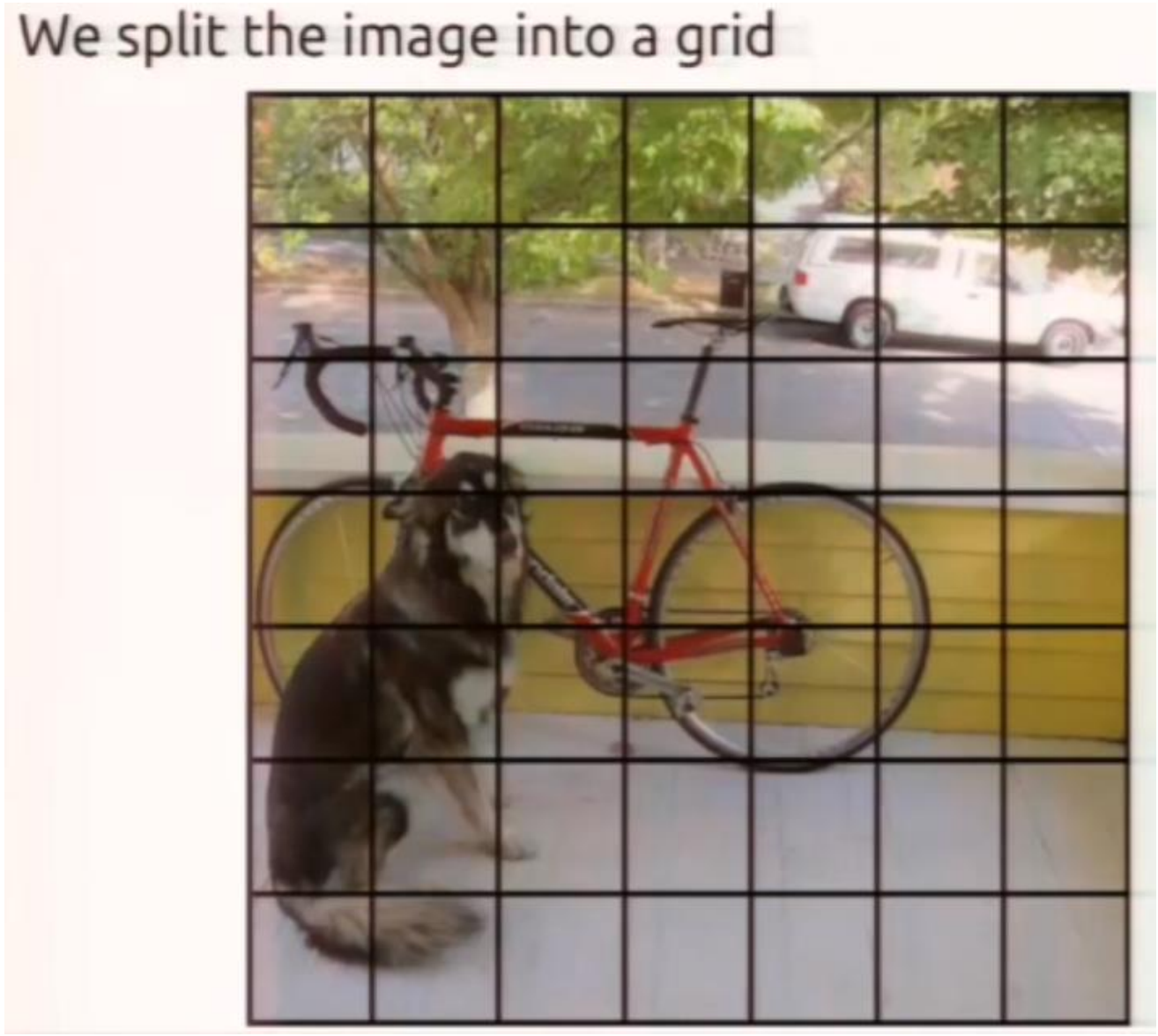
YOLO(You Only Look Once)



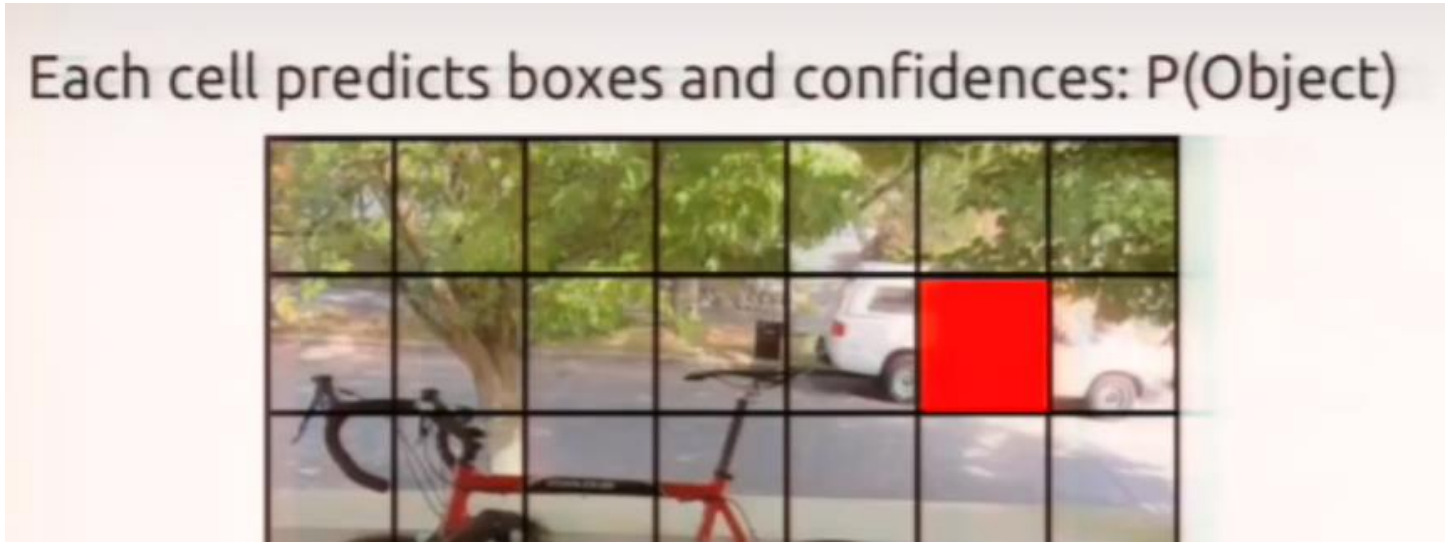
Input image를 S X S grid로 분할(해당 셀에 물체의 중심 위치로 가정)
cell은 B개의 bounding box와 각 객체 존재에 대한 confidence score로 구성
cell은 C개의 클래스 확률로 구성 박스
결과적으로 마지막 prediction layer는S x S x (B * 5 + C) 사이즈가 됨

V1 : 448 x 448 x 3의 이미지가 network를 통과하여 7x7x30의 텐서로 출력
v2는 : 416 x 416 x 3의 이미지가 network를 통과하여 13 x 13 x 25의 텐서로 출력

Step By Step

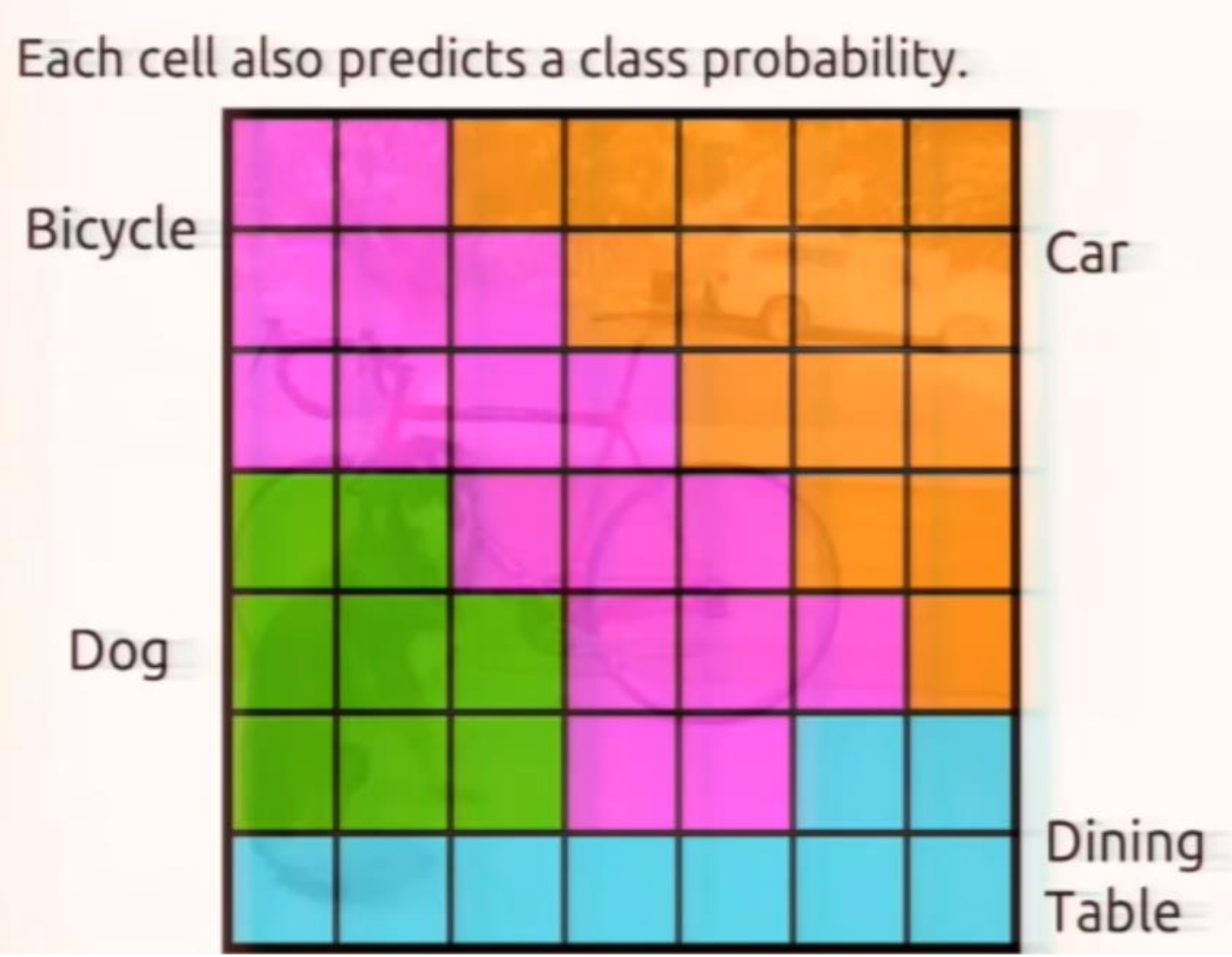


Step By Step

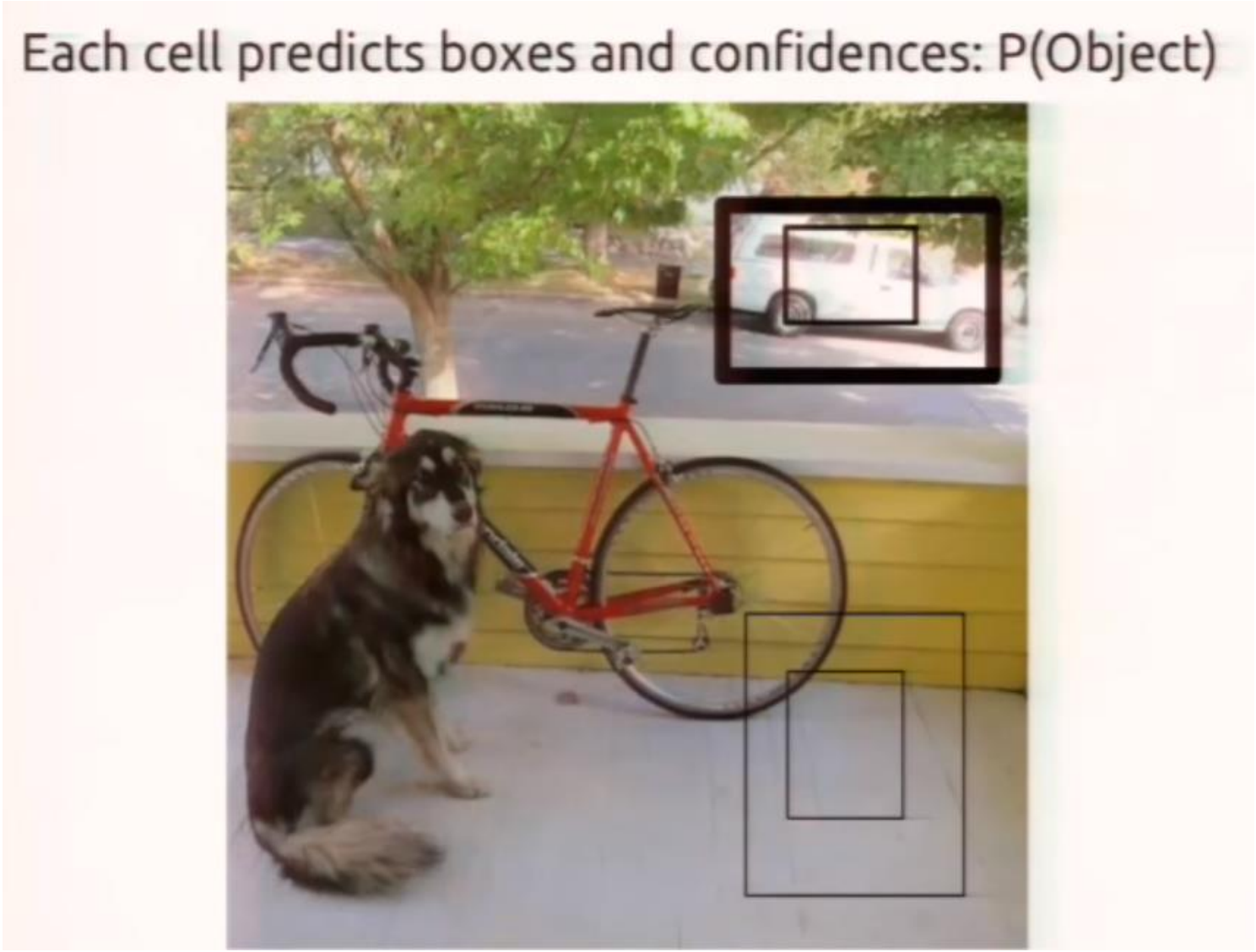

Each cell predicts boxes and confidences: P(Object)


Each cell predicts boxes and confidences: P(Object)

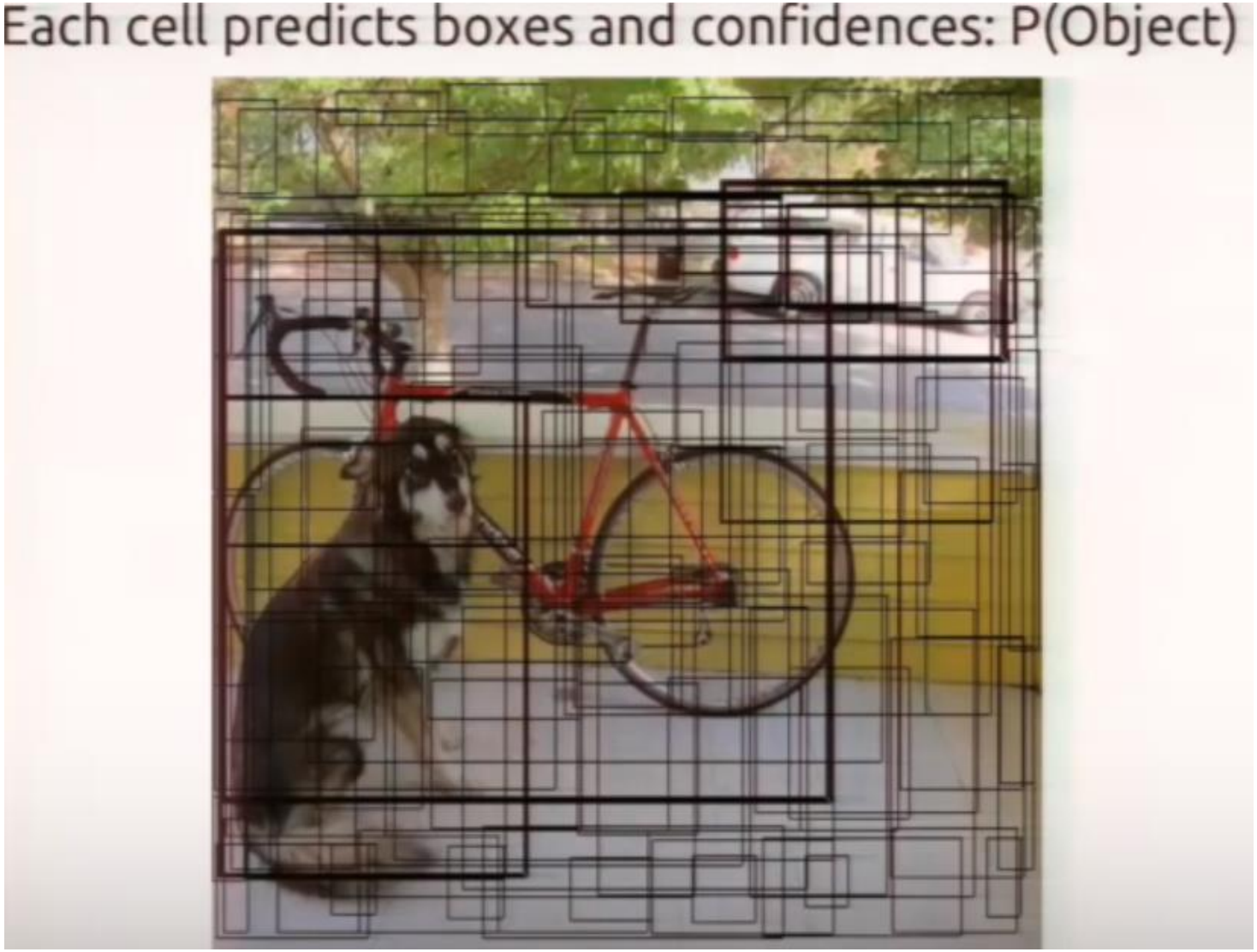
Step By Step


Each cell also predicts a class probability.
Bicycle
Car
Dog
Dining Table

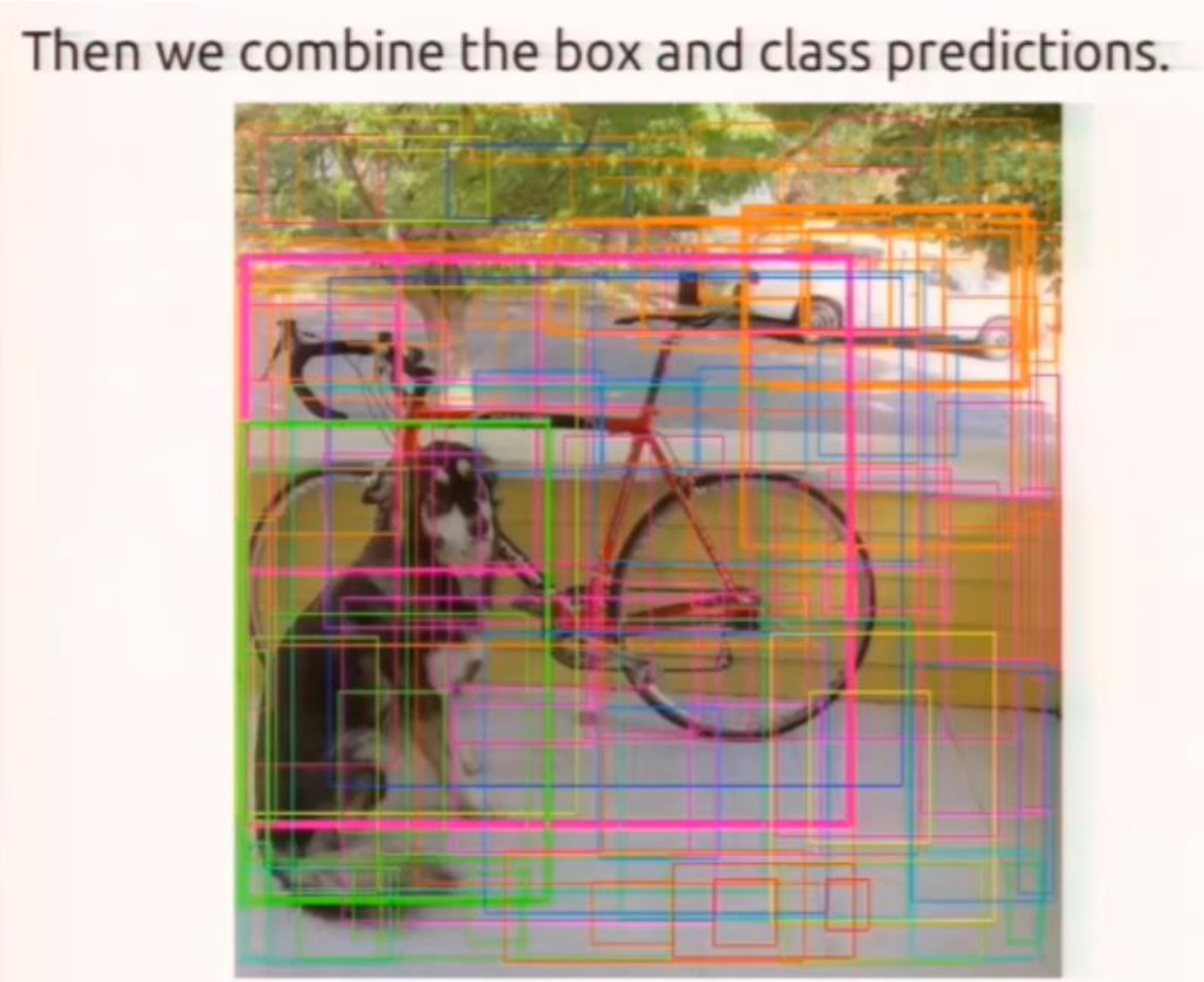
Step By Step



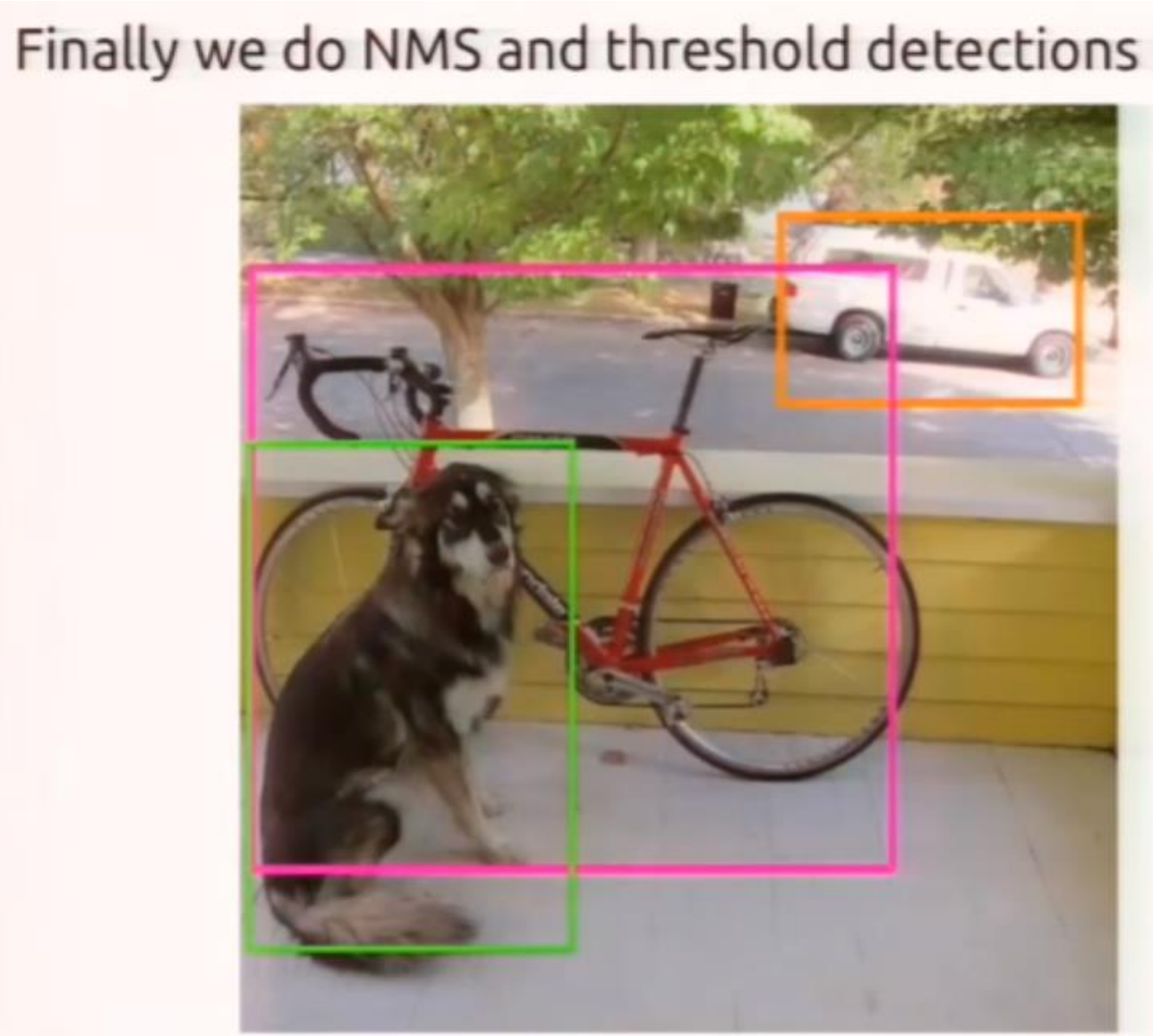
Step By Step



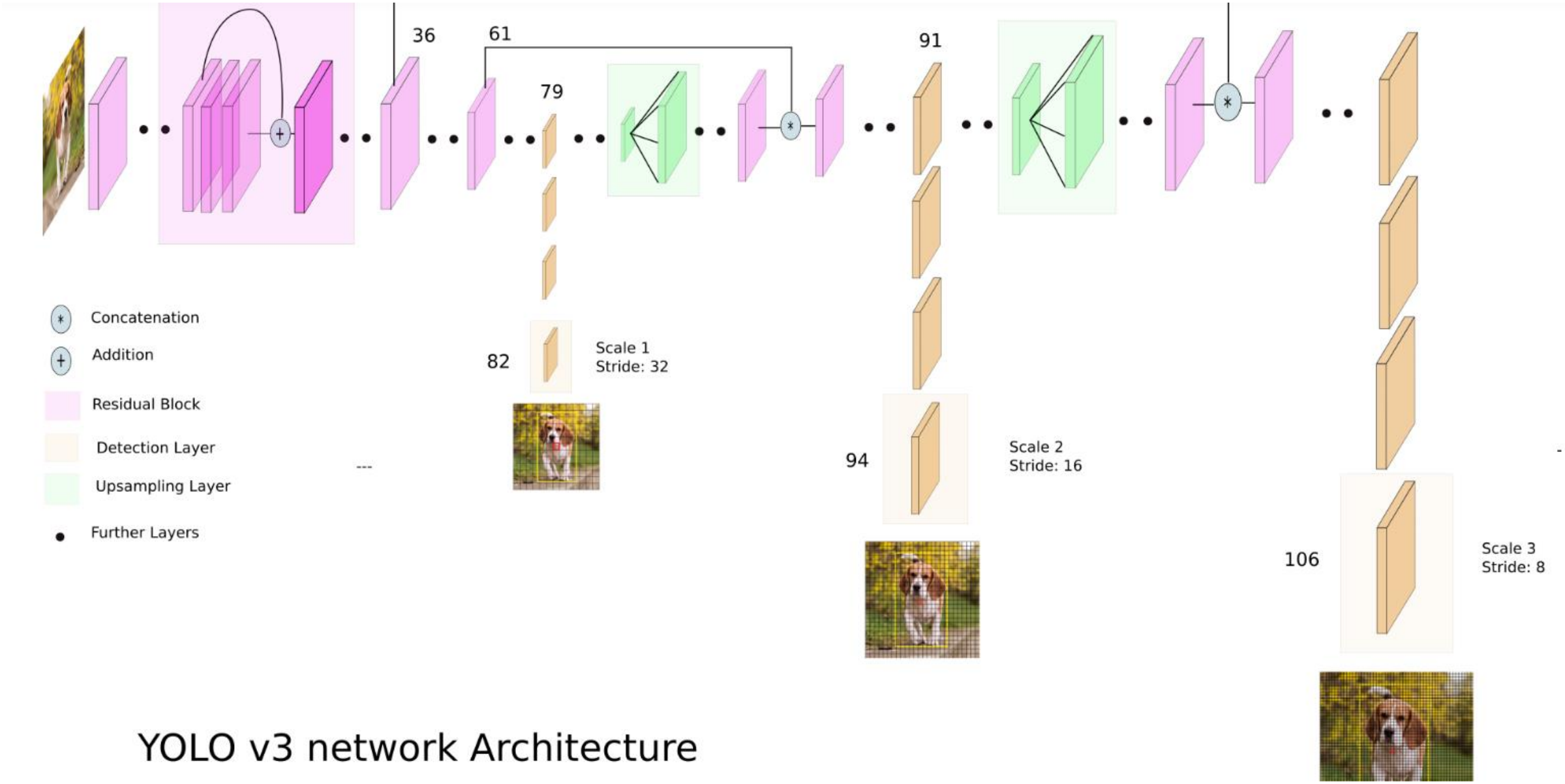
Step By Step



Thresholding by Object Confidence Score
Non-maximum Suppression


Finally we do NMS and threshold detections

YOLO3



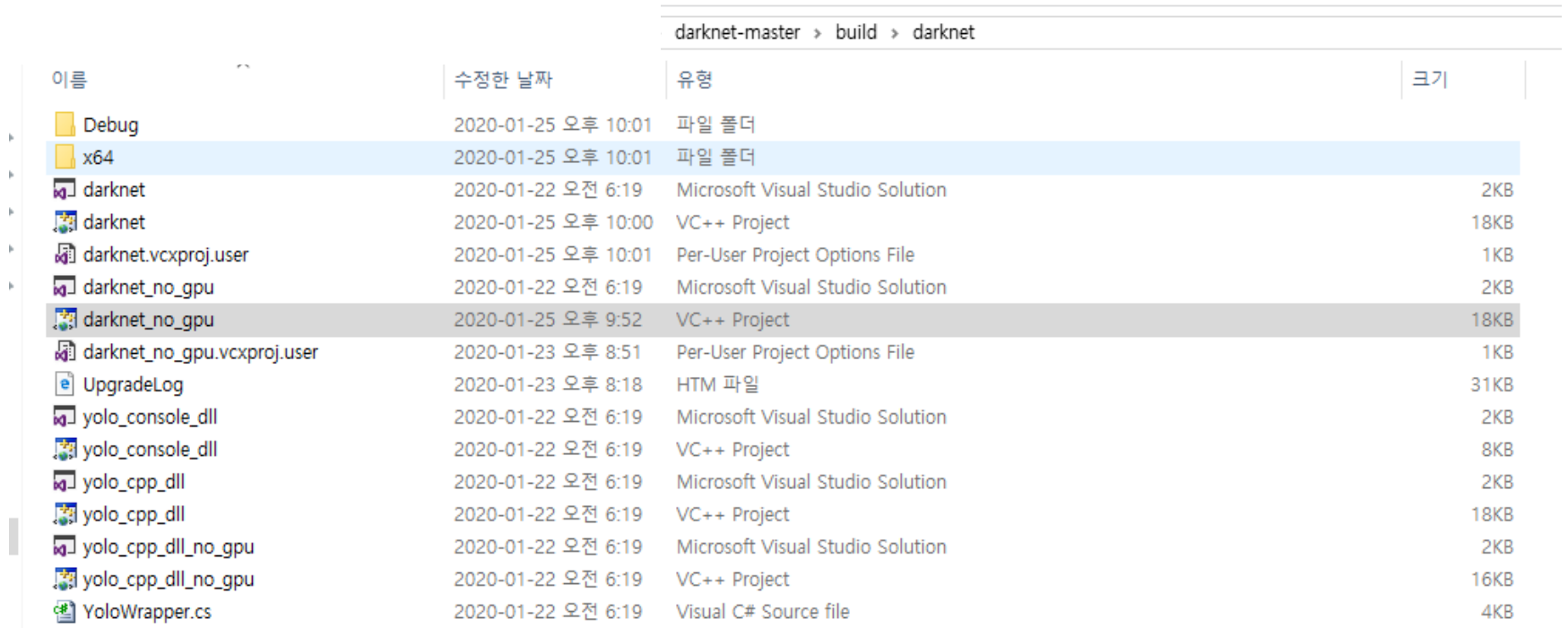
YOLO v3 network Architecture

Windows 에서 YOLO 설치

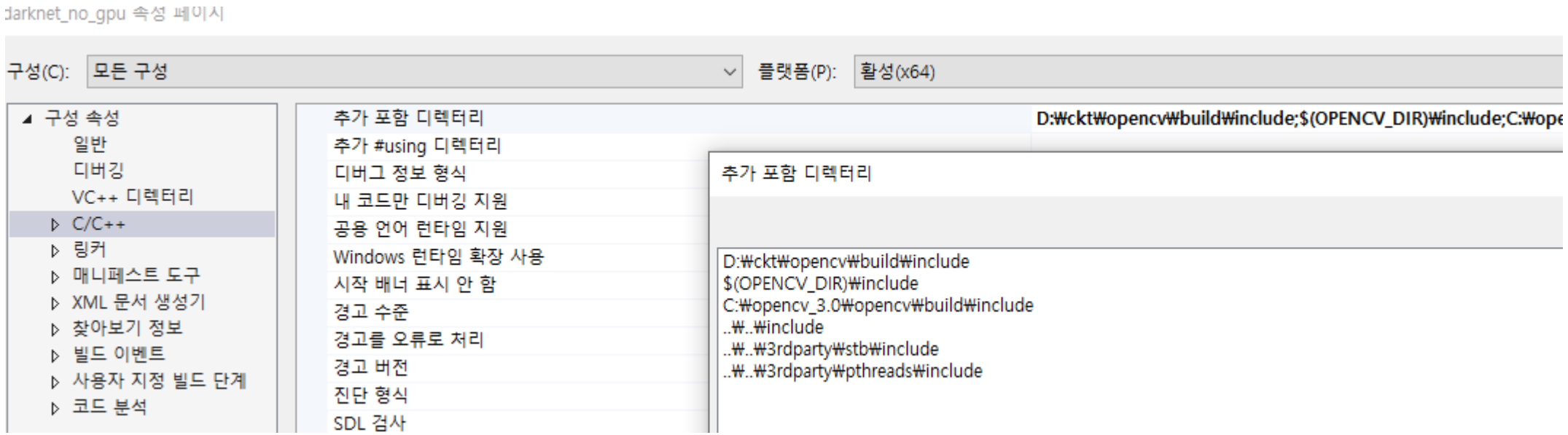
- Visual C++ 설치
- OpenCV 설치
- YOLO 설치
  - https://github.com/AlexeyAB/darknet
  - git clone https://github.com/AlexeyAB/darknet
- 학습된 가중치
  - https://pjreddie.com/darknet/yolo/

Windows 에서 YOLO 설치

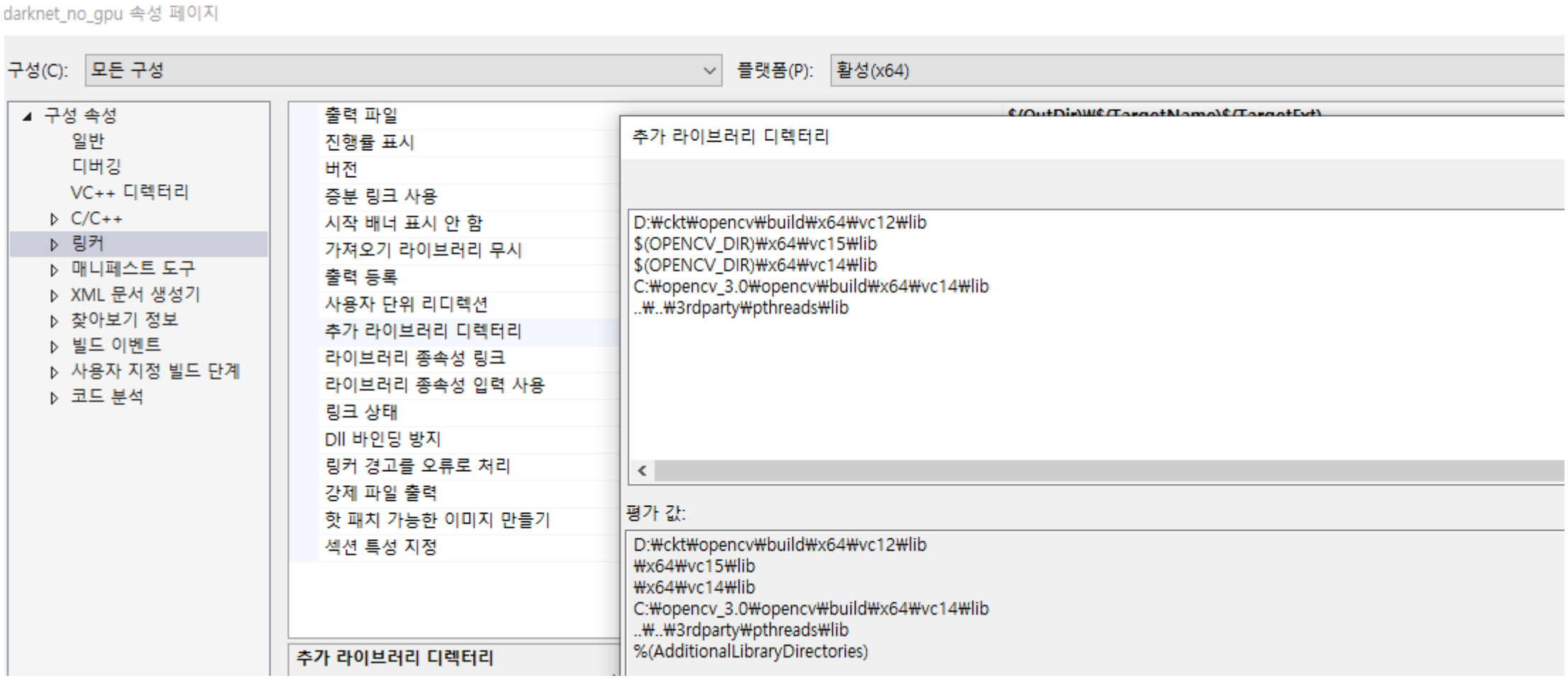
- darknet_no_gpu 더블클릭



Windows 에서 YOLO 설치



Windows 에서 YOLO 설치



Windows 에서 YOLO 설치

- 실행 파일 생성 위치
  - darknet-master₩build₩darknet₩x64
- opencv_world300.dll, pthreadVC2.dll, opencv_ffmpeg300_64.dll 필요
- Image에서 detection
  - darknet_no_gpu detector test coco.data yolov3.cfg yolov3.weights dog.jpg
  - prediction.jpg 생성
- 동영상에서 detection
- darknet_no_gpu detector demo coco.data yolov3.cfg yolov3.weights video.mp4

coco.data / train file

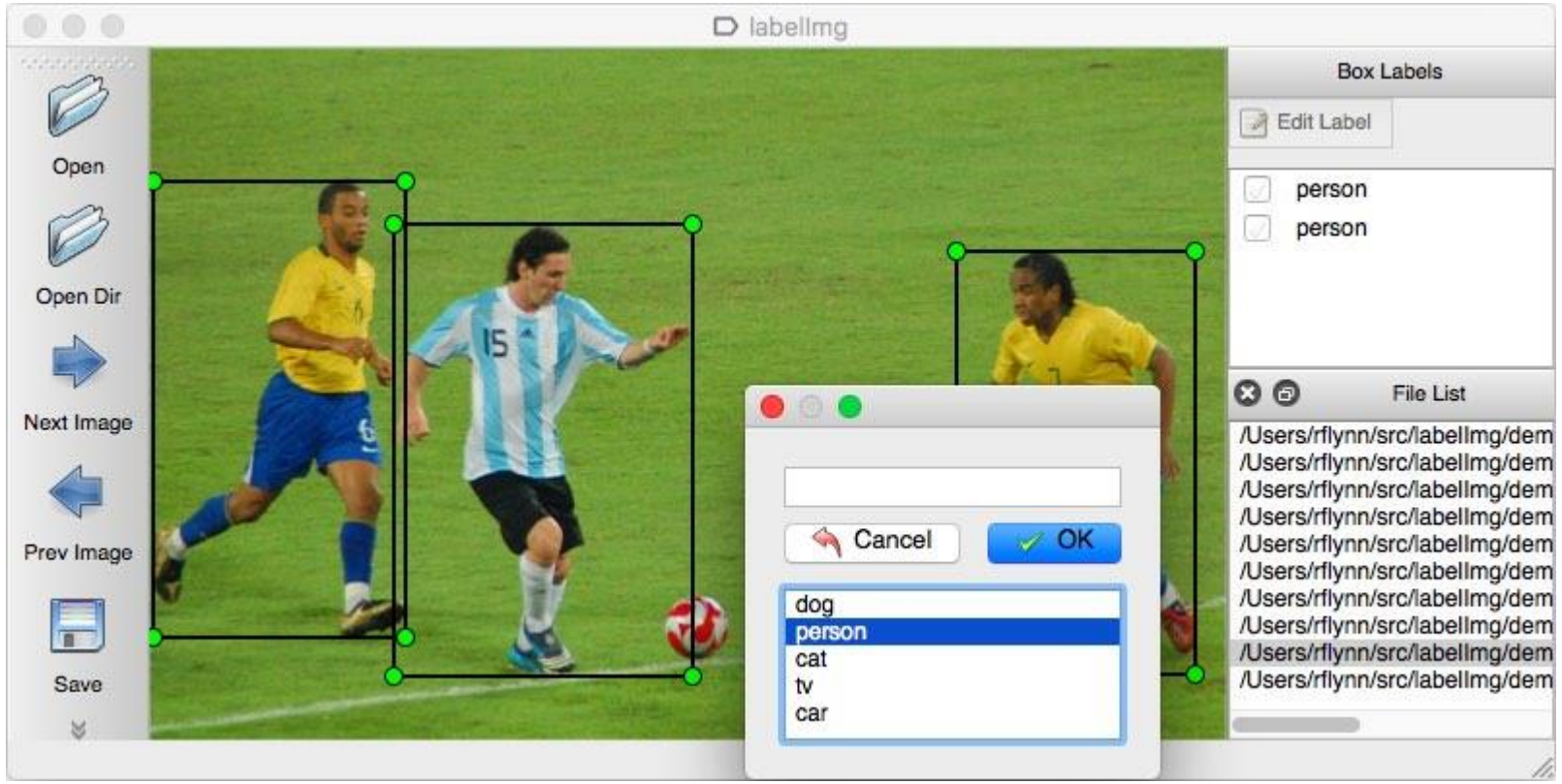
```
classes= 80
train  = data/coco/trainvalno5k.txt
valid  = data/coco_testdev
names = coco.names
backup = backup/
eval=coco
```

```
1  D:/Detection/dark/data/obj/JPEGImages/006e5b0bb2cafc7c.jpg
2  D:/Detection/dark/data/obj/JPEGImages/01e5b1521c4952c2.jpg
3  D:/Detection/dark/data/obj/JPEGImages/022cdc857c47d623.jpg
4  D:/Detection/dark/data/obj/JPEGImages/02a1bb6036a6a99c.jpg
5  D:/Detection/dark/data/obj/JPEGImages/04f5fedf892c10df.jpg
6  D:/Detection/dark/data/obj/JPEGImages/0510b027c407c4a1.jpg
7  D:/Detection/dark/data/obj/JPEGImages/06501fe25d767e63.jpg
8  D:/Detection/dark/data/obj/JPEGImages/0705db91ba1a1319.jpg
9  D:/Detection/dark/data/obj/JPEGImages/07ad8718cba67492.jpg
10 D:/Detection/dark/data/obj/JPEGImages/09de93fe81a30887.jpg
11 D:/Detection/dark/data/obj/JPEGImages/0afab325c3e4bc7e.jpg
12 D:/Detection/dark/data/obj/JPEGImages/13311a9cd1b822d9.jpg
13 D:/Detection/dark/data/obj/JPEGImages/15926b84d96517e0.jpg
```

```
1  person
2  bicycle
3  car
4  motorbike
5  aeroplane
6  bus
7  train
8  truck
9  boat
10 traffic light
11 fire hydrant
12 stop sign
13 parking meter
14 bench
15 bird
16 cat
```

이미지 라벨링 툴

http://tzutalin.github.io/labelImg/



Id cx, cy, width, height
0 0.651389 0.5898145 0.416666 0.816667

Configuration

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=416
height=416
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

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
```

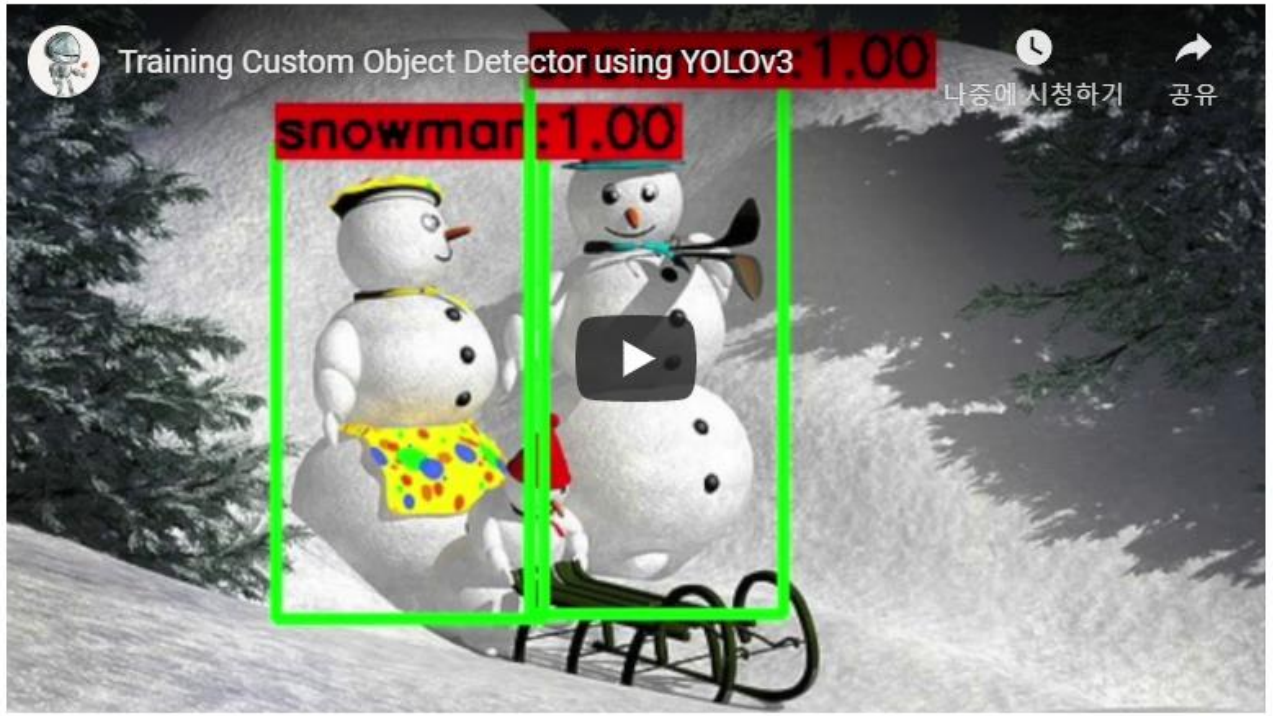
```
[convolutional]
size=1
stride=1
pad=1
filters=255
activation=linear


[yolo]
mask = 6,7,8
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=80
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

(classes+5)*3

Snowman 검출

https://www.learnopencv.com/training-yolov3-deep-learning-based-custom-object-detector/



- ./darknet detector train ~/snowman/darknet.data ~/snowman/darknet-yolov3.cfg ~/snowman/darknet53.conv.74 > ~/snowman/snowman.log
- grep "avg" snowman.log
- python plotTrainLoss.py snowman.log
  - training_loss_plot.png로 저장

Snowman 학습

```
1: 6981.572266, 6981.572266 avg, 0.000000 rate, 7.150312 seconds, 64 images
2: 7006.680176, 6984.083008 avg, 0.000000 rate, 7.272166 seconds, 128 images
3: 6999.226562, 6985.597168 avg, 0.000000 rate, 7.255994 seconds, 192 images
4: 7027.143555, 6989.751953 avg, 0.000000 rate, 7.294813 seconds, 256 images
5: 7006.223633, 6991.398926 avg, 0.000000 rate, 7.334759 seconds, 320 images
6: 7026.386230, 6994.897461 avg, 0.000000 rate, 7.397123 seconds, 384 images
7: 6980.981934, 6993.505859 avg, 0.000000 rate, 7.472622 seconds, 448 images
8: 6999.113281, 6994.066406 avg, 0.000000 rate, 7.491609 seconds, 512 images
9: 6988.436523, 6993.503418 avg, 0.000000 rate, 7.511878 seconds, 576 images
10: 6974.817383, 6991.634766 avg, 0.000000 rate, 7.586254 seconds, 640 images
11: 8805.033203, 7172.974609 avg, 0.000000 rate, 10.290684 seconds, 704 images
12: 8817.729492, 7337.450195 avg, 0.000000 rate, 10.707979 seconds, 768 images
13: 8845.060547, 7488.211426 avg, 0.000000 rate, 10.778275 seconds, 832 images
14: 8797.466797, 7619.136719 avg, 0.000000 rate, 10.914041 seconds, 896 images
15: 8776.360352, 7734.858887 avg, 0.000000 rate, 11.084632 seconds, 960 images
16: 8757.814453, 7837.154297 avg, 0.000000 rate, 11.199432 seconds, 1024 images
17: 8744.082031, 7927.847168 avg, 0.000000 rate, 11.279750 seconds, 1088 images
18: 8653.07324
19: 8659.71582
20: 8538.56054
21: 3164.88476
22: 3130.65502
23: 3082.30688
24: 3026.98901
    102 conv    256  3 x 3 / 1   104 x 104 x 128   ->   104 x 104 x 256  6.380 BFLOPs
    103 conv    128  1 x 1 / 1   104 x 104 x 256   ->   104 x 104 x 128  0.709 BFLOPs
    104 conv    256  3 x 3 / 1   104 x 104 x 128   ->   104 x 104 x 256  6.380 BFLOPs
    105 conv     18  1 x 1 / 1   104 x 104 x 256   ->   104 x 104 x  18  0.100 BFLOPs
    106 yolo
Loading weights from /home/ubuntu/snowman/darknet53.conv.74...Done!
Saving weights to /home/ubuntu/snowman/weights/darknet-yolov3.backup
./darknet detector train ~/snowman/darknet.data ~/snowman/darknet-yolov3.cfg ~/snowman/da
man.logSaving weights to /home/ubuntu/snowman/weights/darknet-yolov3.backup
Saving weights to /home/ubuntu/snowman/weights/darknet-yolov3_200.weights
```