

# Pascal VOC 2012 dataset

YOLO = a system for detecting objects on the Pascal VOC 2012 dataset



The [PASCAL](#) Visual Object Classes Homepage

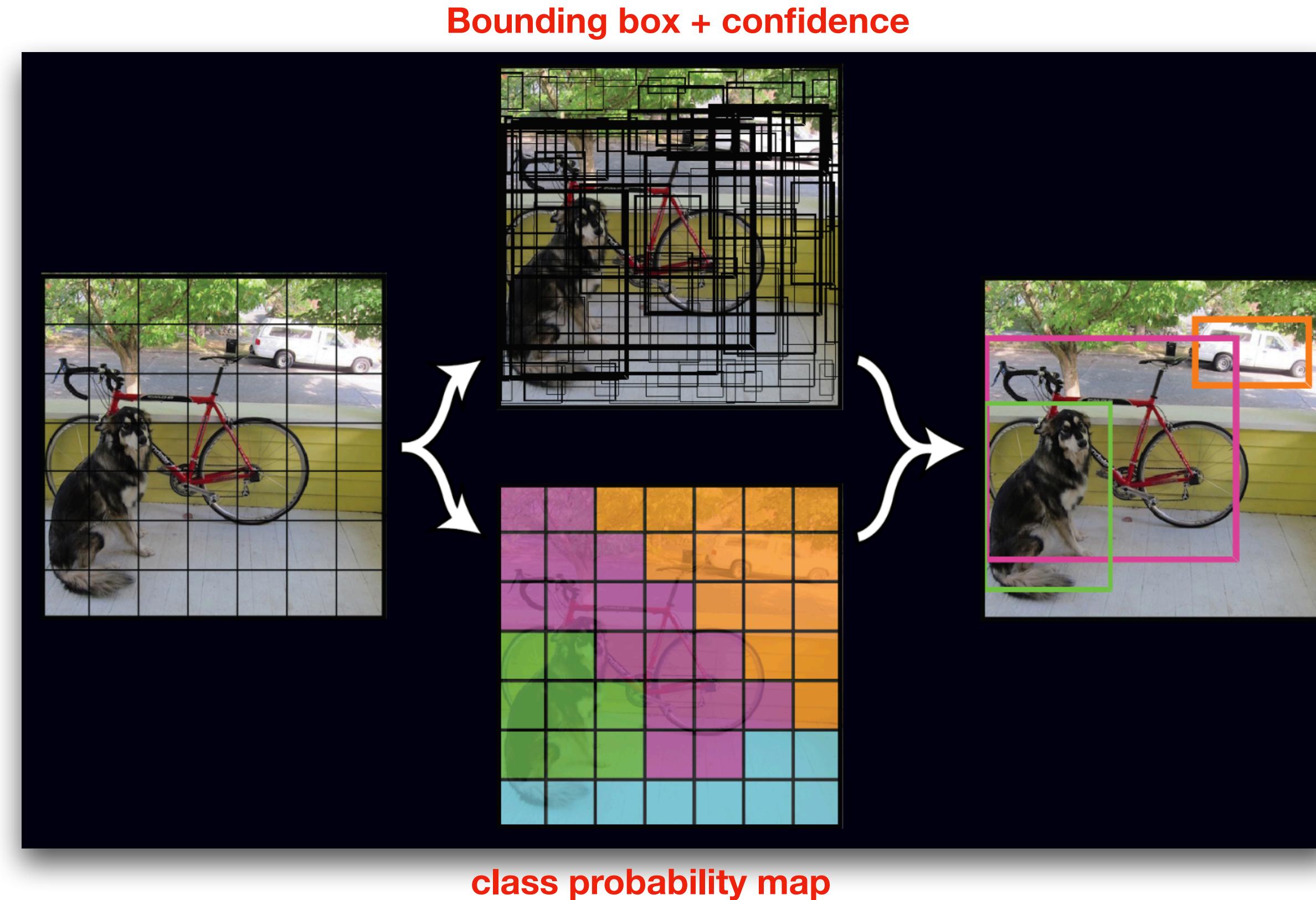


Year	Statistics	New developments	Notes
2012	20 classes. The train/val data has 11,530 images containing 27,450 ROI annotated objects and 6,929 segmentations.	<ul style="list-style-type: none"><li>Size of segmentation dataset substantially increased.</li><li>People in action classification dataset are additionally annotated with a reference point on the body.</li></ul>	<ul style="list-style-type: none"><li>Datasets for classification, detection and person layout are the same as VOC2011.</li></ul>

- **20 classes:**
  - (person), (bird, cat, cow, dog, horse, sheep), (airplane, bicycle, boat, bus, car, motorbike, train) 등
- **train/val data with annotation & test data without annotation**
- **ROI = Region of Interest Annotation (= Bounding Box)**
- `cat 2007_* 2012_train.txt > train.txt`
- **5 main tasks (총 7개)**
  - Classification, Detection, Segmentation, Action Classification, Large Scale Recognition 등
    - \* Action Classification 란
      - a specified person's (bounding box로 표시) 10 action classes
      - action: jumping; phoning; running; playing a musical instrument 등
  - 각 task에 따른 Submission of Results의 format이나 Evaluation 등 각각 다르므로 메뉴얼 확인 필요

# YOLO v1 (저번주)

## How It Works



- **network**
  - a single neural network to the full image
  - divides the image into regions & predicts bounding boxes and probabilities for each region
  - NMS: only see high scoring detections
- **evaluation**
  - trained with the IOU prediction (*threshold*) which gives slightly better mAP scores
  - ! global context, extremely fast

# Darknet

## YOLO v1

- **Darknet (Original)** [pjreddie / darknet](https://pjreddie.com/darknet/)
  - <https://github.com/pjreddie/darknet/>
  - Joseph Redmon이 YOLO 학습을 위해 개발한 프레임워크
  - open source neural network framework written in C and CUDA (supports CPU & GPU)



- **Alexey** [AlexeyAB / darknet](https://github.com/AlexeyAB/darknet)  
forked from pjreddie/darknet
  - <https://github.com/AlexeyAB/yolo-windows>
  - YOLOv4의 저자인 Alexey Bochkovskiy의 버전
  - 특징: Darknet을 fork함. 정리가 잘 되어 있음

# prepare Data set

## \*\*Generate Labels for VOC

- Get The Pascal VOC Data
- Generate Labels for VOC: 다크넷의 라벨 형식 (직접 레이블링 시, 그대로 적용 가능)
  - .xml to .txt <object-class> <x> <y> <width> <height>
  - x, y, width, and height are relative to the image's width and height

```
sets=[('2012', 'train'), ('2012', 'val'), ('2007', 'train'), ('2007', 'val'), ('2007', 'test')]

classes = ["aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow"]

def convert(size, box):
    dw = 1./size[0]
    dh = 1./size[1]
    x = (box[0] + box[1])/2.0
    y = (box[2] + box[3])/2.0
    w = box[1] - box[0]
    h = box[3] - box[2]
    x = x*dw
    w = w*dw
    y = y*dh
    h = h*dh
    return (x,y,w,h)
```

# Before Train

## Detection Using A Pre-Trained Model

- **Pre-Trained Model = pre-trained weight** (정해진 weight 사용 + 학습시 변환..?)
- **YOLO Model Comparison** (둘 다 weight file 존재 = trained on 2007 train/val+ 2012 train/val)
  - yolo.cfg: extraction network 기반
    - yolo.cfg = extraction.cfg + 4 conv layer + 1 fc layer
  - tiny-yolo.cfg: much smaller than yolo.cfg & Darknet reference network 기반
    - .cfg: 네트워크 구조와 하이퍼파라미터 결정 (자신의 데이터에 맞게 조정 필요)
    - **weight file**: pre-trained weight

**Extraction**

I developed this model as an offshoot of the GoogleNet model. It doesn't use the "inception" modules, only 1x1 and 3x3 convolutional layers.

- Top-1 Accuracy: 72.5%
- Top-5 Accuracy: 90.8%
- Forward Timing: 4.8 ms/img
- CPU Forward Timing: 0.97 s/img
- cfg file
- weight file (90 MB)

**Darknet Reference Model**

This model is designed to be small but powerful. It attains the same top-1 and top-5 performance as AlexNet but with 1/10th the parameters. It uses mostly convolutional layers without the large fully connected layers at the end. It is about twice as fast as AlexNet on CPU making it more suitable for some vision applications.

- Top-1 Accuracy: 61.1%
- Top-5 Accuracy: 83.0%
- Forward Timing: 2.9 ms/img
- CPU Forward Timing: 0.14 s/img
- cfg file
- weight file (28 MB)

# Train

## Detection Using A Pre-Trained Model

- **Crop Layer:  $448 \times 448 \rightarrow 448 \times 448 \times 3$  image**
  - Crop Layer 이란 ?
    - 이미지 잘라내기 위한 단계 (일정 이상 크기의 이미지 생성 막음)
- **test's output: confidence & how long it took to find them**

```
./darknet yolo test cfg/yolov1/yolo.cfg yolov1.weights data/dog.jpg
0: Crop Layer: 448 x 448 -> 448 x 448 x 3 image
1: Convolutional Layer: 448 x 448 x 3 image, 64 filters -> 224 x 224 x 64 image
...
27: Connected Layer: 4096 inputs, 1225 outputs
28: Detection Layer
Loading weights from yolov1.weights...Done!
data/dog.jpg: Predicted in 8.012962 seconds.
0.941620 car
0.397087 bicycle
0.220952 dog
Not compiled with OpenCV, saving to predictions.png instead
```

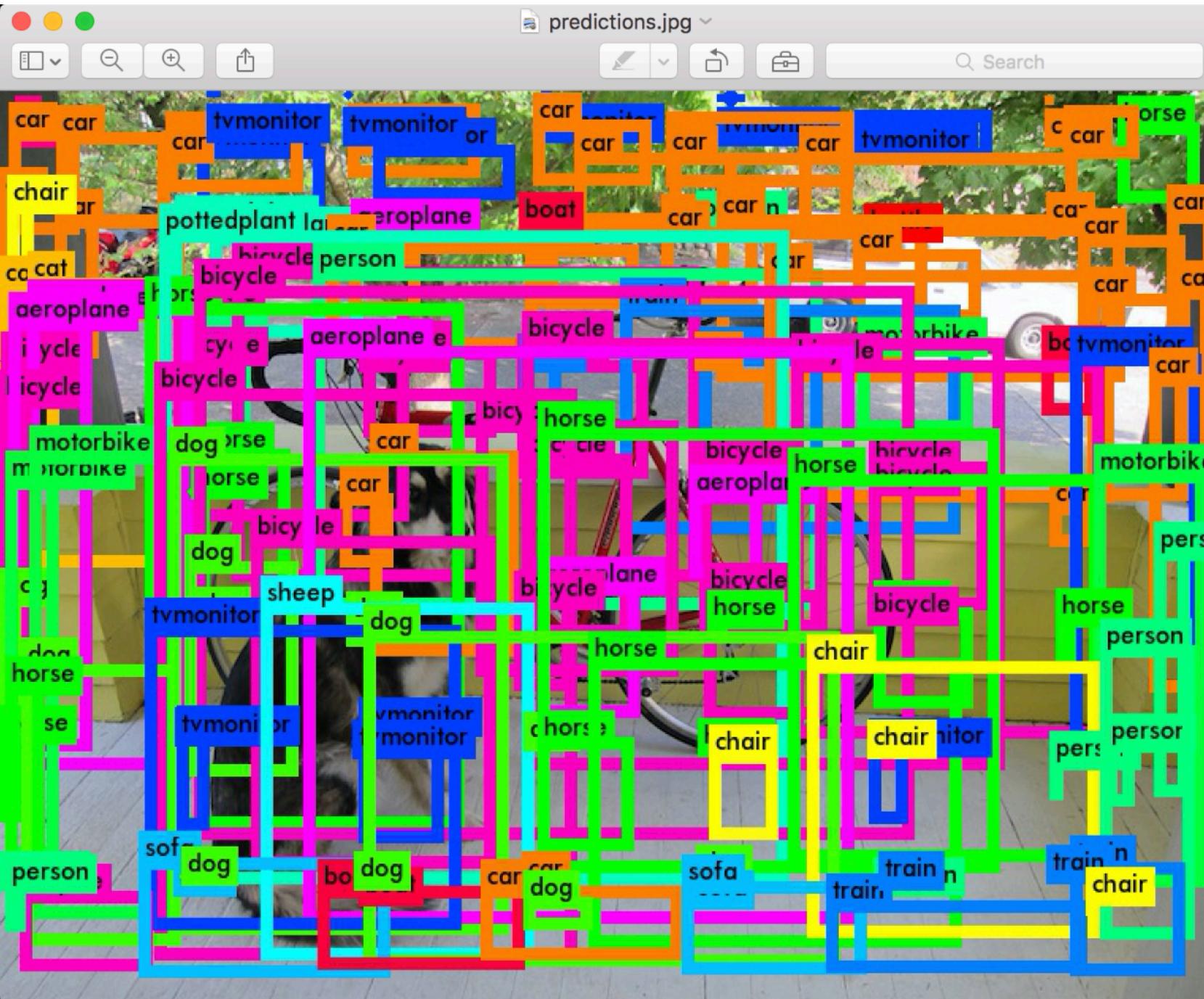
```
1 [net]
2 # Testing
3 batch=1
4 subdivisions=1
5 # Training
6 # batch=64
7 # subdivisions=8
8 height=448
9 width=448
10 channels=3
11 momentum=0.9
12 decay=0.0005
13 saturation=1.5
14 exposure=1.5
15 hue=.1
16
17 learning_rate=0.0005
18 policy=steps
19 steps=200,400,600,20000,30000
20 scales=2.5,2,2,.1,.1
21 max_batches = 40000
22
23 [convolutional]
24 batch_normalize=1
25 filters=64
26 size=7
27 stride=2
28 pad=1
29 activation=leaky
30
31 [maxpool]
```

*yolov1.cfg*

# Train (+option)

## Changing The Detection Threshold

- default: confidence threshold  $\geq 0.2$
- if confidence threshold == 0,



- threshold 설정

```
./darknet yolo test cfg/yolov1/yolo.cfg yolov1.weights data/dog.jpg -thresh 0
```

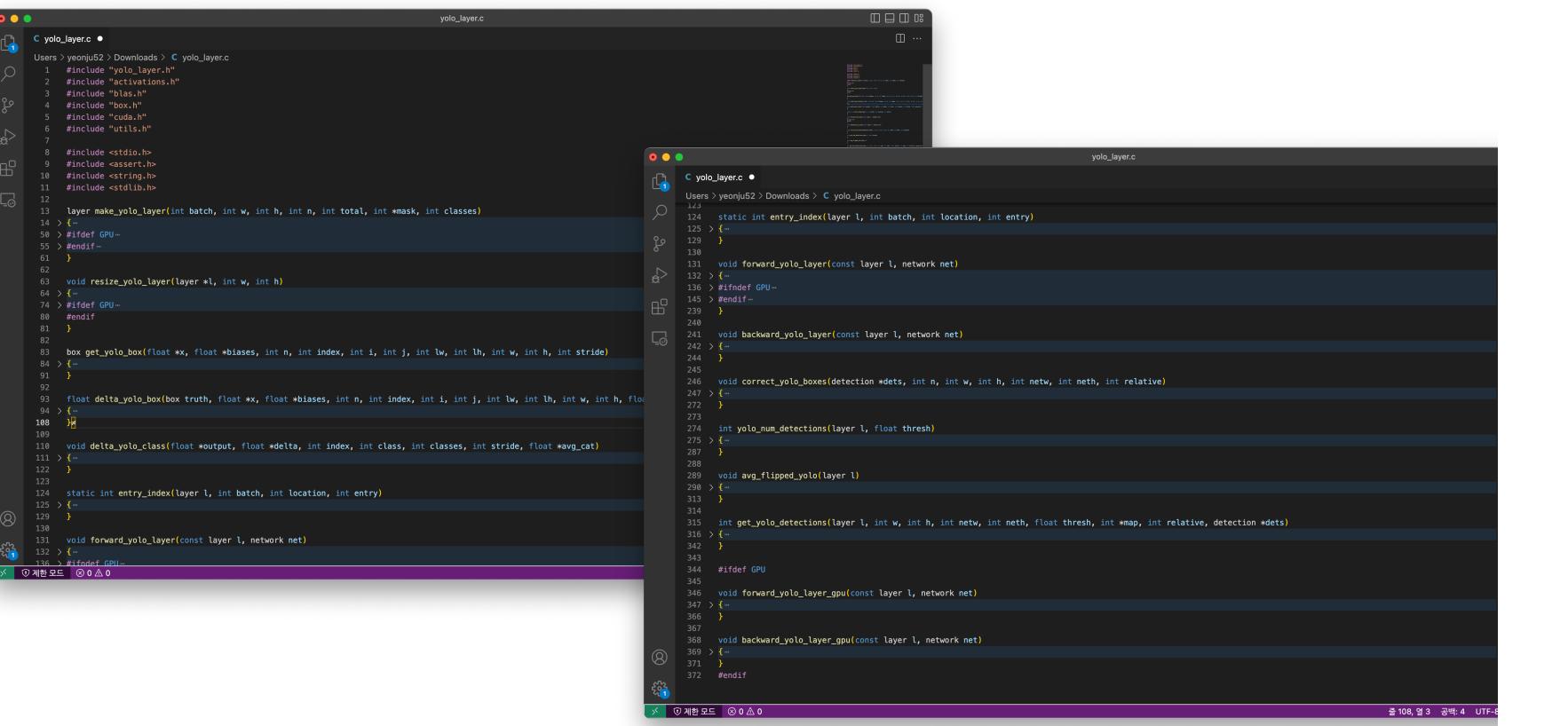
or (추정)

```
247 [detection]
248 classes=20
249 coords=4
250 rescore=1
251 side=7
252 num=3
253 softmax=0
254 sqrt=1
255 jitter=.2
256
257 object_scale=1
258 noobject_scale=.5
259 class_scale=1
260 coord_scale=5
```

yolov1.cfg

# 다음주 예정

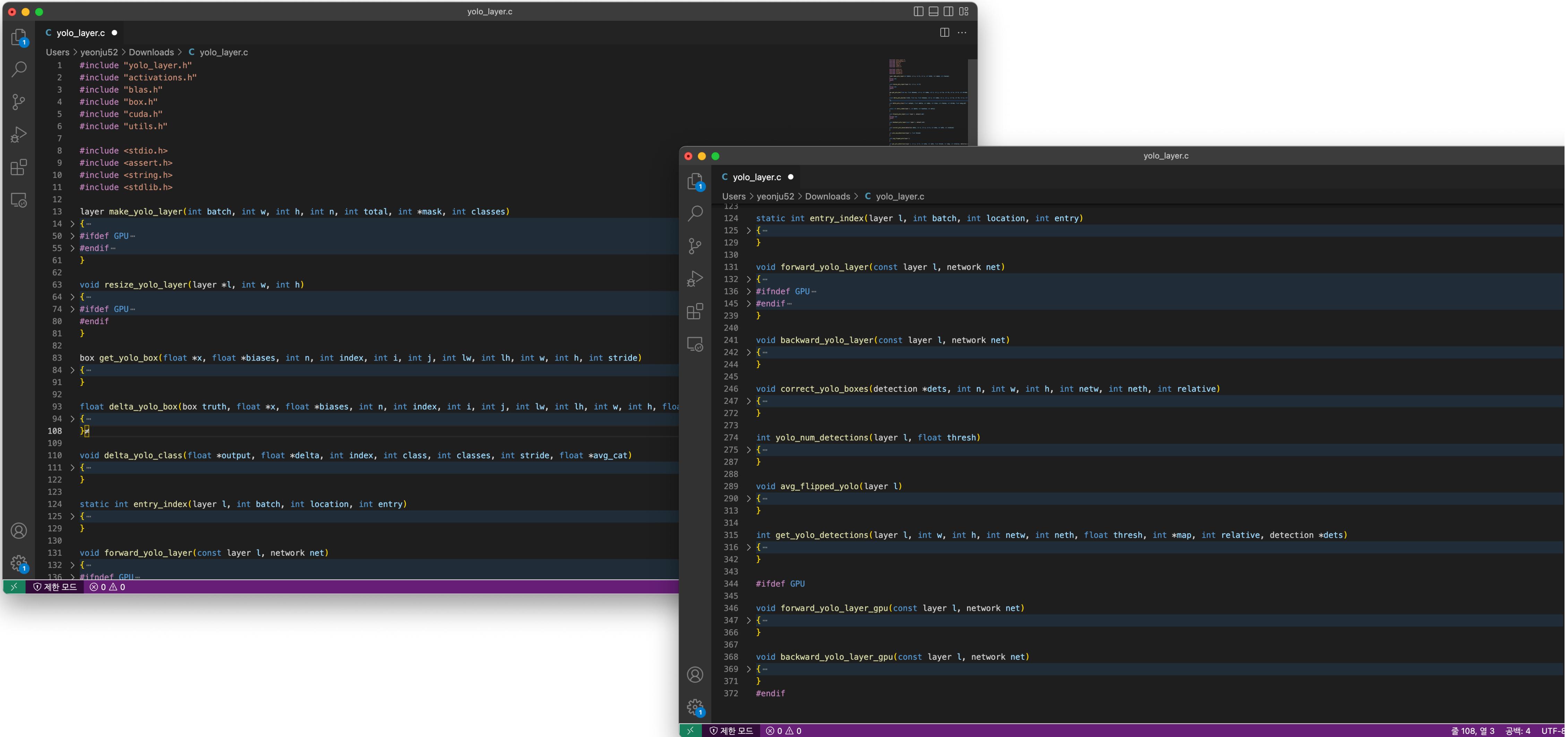
- 서버컴 - 가상환경 세팅 후 코드 실행
- 코드 상세 리뷰 (논문 비교)
  - (예시1) 코드 내 loss 함수 -> 논문과 비교
  - (예시2) yolo\_layer.c 함수 기능 분석



The image shows two terminal windows side-by-side, both displaying the same C code for the yolo\_layer.c file. The code is a deep learning layer implementation for YOLO, containing functions like `make_yolo_layer`, `forward_yolo_layer`, `backward_yolo_layer`, and various utility functions for boxes and detections. The code is heavily annotated with comments explaining its purpose and parameters.

```
1 //include "yolo_layer.h"
2 //include "activations.h"
3 //include "box.h"
4 //include "code.h"
5 //include "cblas.h"
6 //include "cpu.h"
7 //include "vulkan.h"
8 //include "cassette.h"
9 //include "string.h"
10 //include "vector.h"
11 //include "math.h"
12
13 layer make_yolo_layer(int batch, int w, int h, int n, int total, int mask, int classes)
14 {
15     #ifndef GPU
16     #ifdef CPU
17         void resize_yolo_layer(layer l, int w, int h)
18     #endif
19     #endif
20 }
21
22 void get_yolo_box(float **x, float *biases, int n, int index, int i, int j, int lw, int lh, int h, int stride)
23 {
24     float delta_yolo_boxbox_truth, float **x, float *biases, int n, int index, int i, int j, int lw, int lh, int w, int h, int stride
25 }
26
27 void delta_yolo_classify(float *output, float *delta, int index, int class, int classes, int stride, float **avg_cat)
28 {
29 }
30
31 static int entry_index(layer l, int batch, int location, int entry)
32 {
33     int e = entry;
34     int b = batch;
35     int l_ = location;
36     int e_ = entry;
37 }
38
39 void forward_yolo_layer(const layer l, network net)
40 {
41     #ifndef GPU
42         forward_yolo_layer_gpu(l, net);
43     #endif
44 }
```

# (확대) yolo\_layer.c: train/valid/test와 관련 함수 (추정)



```
yolo_layer.c
```

```
1 #include "yolo_layer.h"
2 #include "activations.h"
3 #include "blas.h"
4 #include "box.h"
5 #include "cuda.h"
6 #include "utils.h"
7
8 #include <stdio.h>
9 #include <assert.h>
10 #include <string.h>
11 #include <stdlib.h>
12
13 layer make_yolo_layer(int batch, int w, int h, int n, int total, int *mask, int classes)
14 > { ...
50 > #ifdef GPU...
55 > #endif...
61 }
62
63 void resize_yolo_layer(layer *l, int w, int h)
64 > { ...
74 > #ifdef GPU...
80 #endif
81 }
82
83 box get_yolo_box(float *x, float *biases, int n, int index, int i, int j, int lw, int lh, int w, int h, int stride)
84 > { ...
91 }
92
93 float delta_yolo_box(box truth, float *x, float *biases, int n, int index, int i, int j, int lw, int lh, int w, int h, float)
94 > { ...
108 }
109
110 void delta_yolo_class(float *output, float *delta, int index, int class, int classes, int stride, float *avg_cat)
111 > { ...
122 }
123
124 static int entry_index(layer l, int batch, int location, int entry)
125 > { ...
129 }
130
131 void forward_yolo_layer(const layer l, network net)
132 > { ...
136 > #ifndef GPU...

```

```
yolo_layer.c
```

```
125
124 static int entry_index(layer l, int batch, int location, int entry)
125 > { ...
129 }
130
131 void forward_yolo_layer(const layer l, network net)
132 > { ...
136 > #ifndef GPU...
145 > #endif...
239 }
240
241 void backward_yolo_layer(const layer l, network net)
242 > { ...
244 }
245
246 void correct_yolo_boxes(detection *dets, int n, int w, int h, int netw, int neth, int relative)
247 > { ...
272 }
273
274 int yolo_num_detections(layer l, float thresh)
275 > { ...
287 }
288
289 void avg_flipped_yolo(layer l)
290 > { ...
313 }
314
315 int get_yolo_detections(layer l, int w, int h, int netw, int neth, float thresh, int *map, int relative, detection *dets)
316 > { ...
342 }
343
344 #ifdef GPU
345
346 void forward_yolo_layer_gpu(const layer l, network net)
347 > { ...
366 }
367
368 void backward_yolo_layer_gpu(const layer l, network net)
369 > { ...
371 }
372 #endif

```

줄 108, 열 3 공백: 4 UTF-8