

---

# YOLO(You Only Look Once)

---

## ▶ Object Detection에서 2가지 방식

### 2-Stage Object Detection

Object Detection을 두 단계로 나누어 수행  
Region proposal + Classification/Regression

대표적으로 R-CNN 계열

### 1-Stage Object Detection

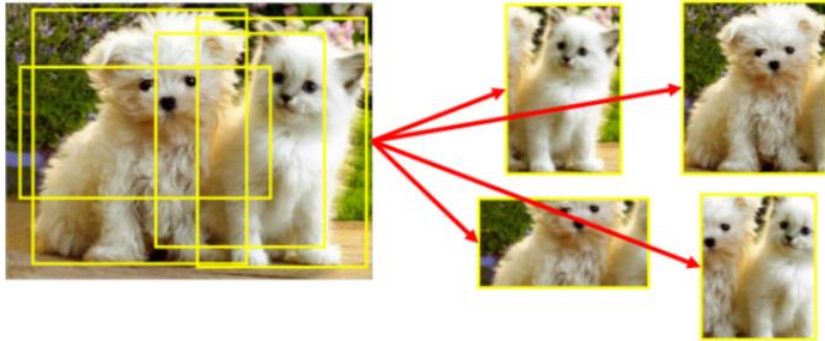
Object Detection을 한 번에 수행

대표적으로 YOLO 계열

## ► 2-Stage Object Detection

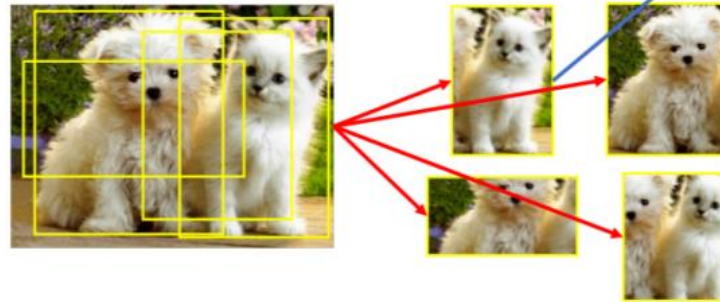
### Stage 1: Region proposal

이미지에서 물체가 있을 만한 영역을 bounding box로 제안



### Stage 2: classification & Regression

Bounding box에 있는 물체를 분류  
Bounding box의 위치와 크기를 재조정



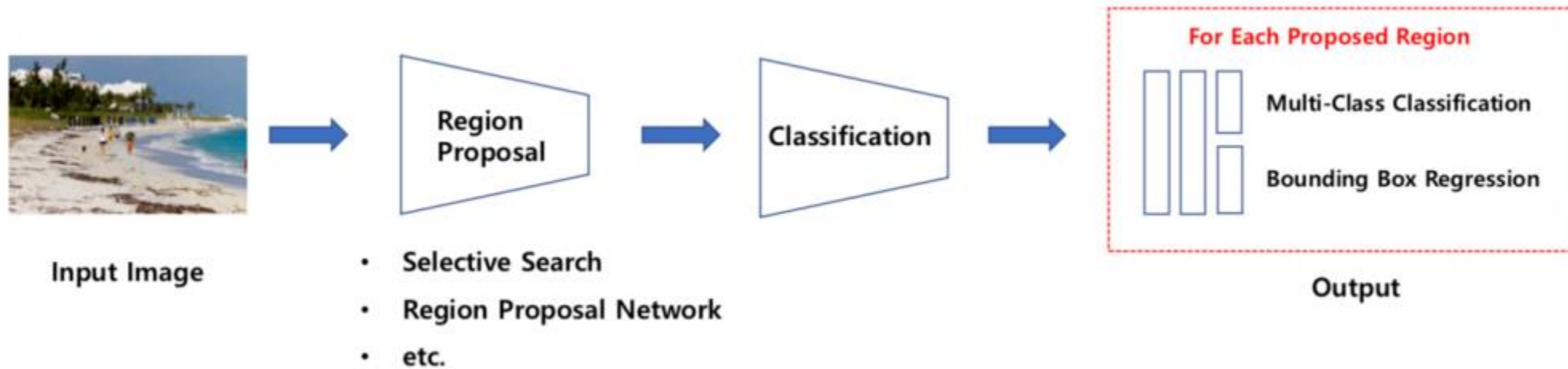
Class probability distribution		Bounding box parameters	
Cat	0.88	$t_x$	0.01
Dog	0.05	$t_y$	-0.01
...	...	$t_w$	0.09
		$t_h$	-0.11

Class probability distribution		Bounding box parameters	
Cat	0.10	$t_x$	0.02
Dog	0.85	$t_y$	0.00
...	...	$t_w$	-0.1
		$t_h$	-0.11

⋮

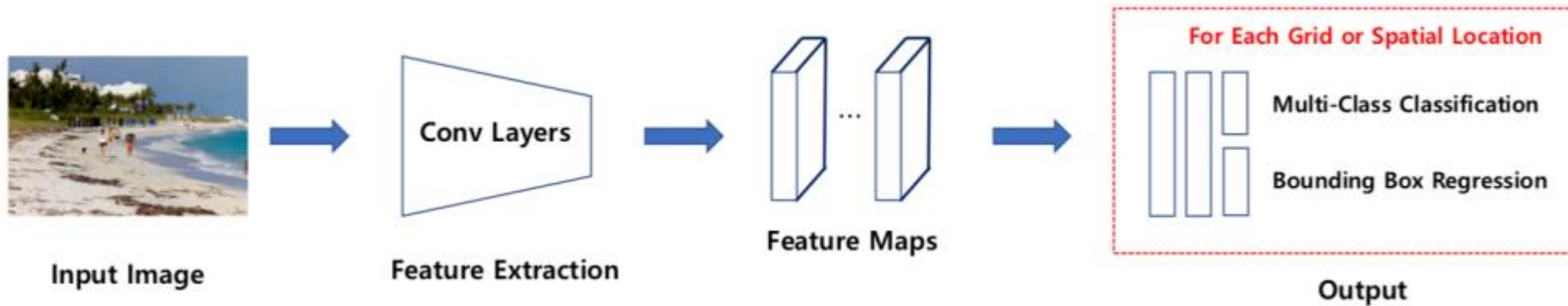
## ▶ 2-Stage Object Detection

- 대표적으로 R-CNN 계열의 접근 방식
- 탐색 영역을 찾는 과정(Region Proposal), 해당 영역을 분류(Classification)하는 과정을 **순차적으로 수행**
- **느리지만 정확한** 방법 → 높은 정확도를 요구하는 task에 사용 (ex, challenge, high-end PC environment)

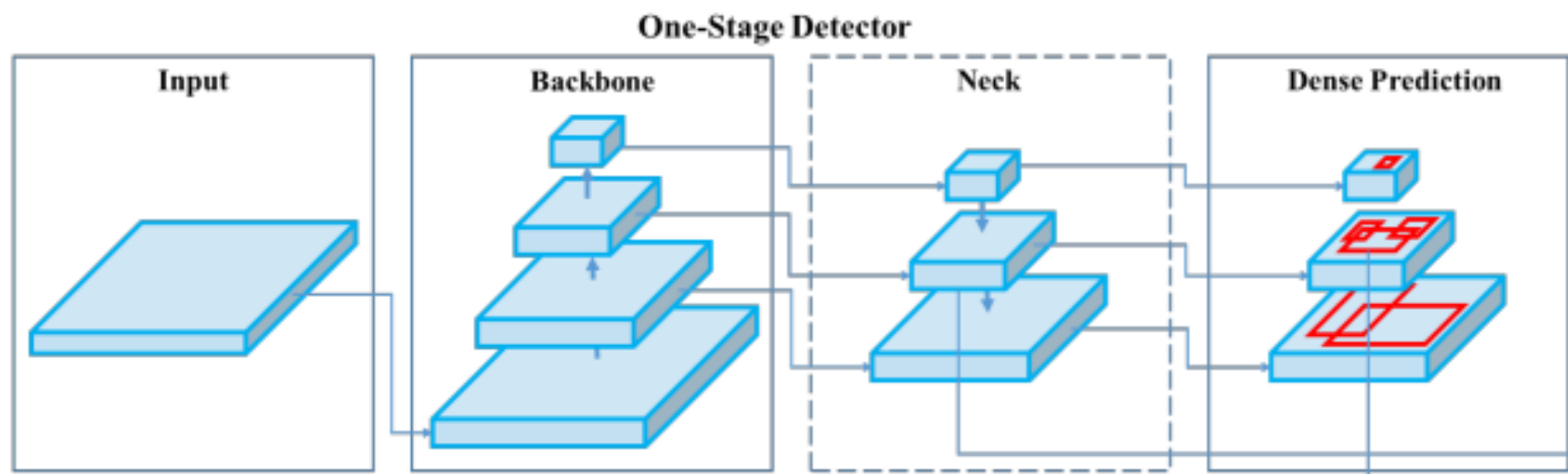


# ▶ 1-Stage Object Detection

- 대표적으로 SSD, YOLO 계열의 접근 방식
- 탐색 영역을 찾는 과정(Region Proposal) + 해당 영역을 분류(Classification)하는 과정 **동시에 수행**
- 2-stage에 비해 부정확지만 **빠른** 방법 → 빠른 연산속도를 요구하는 task에 사용 (ex, embedded device)

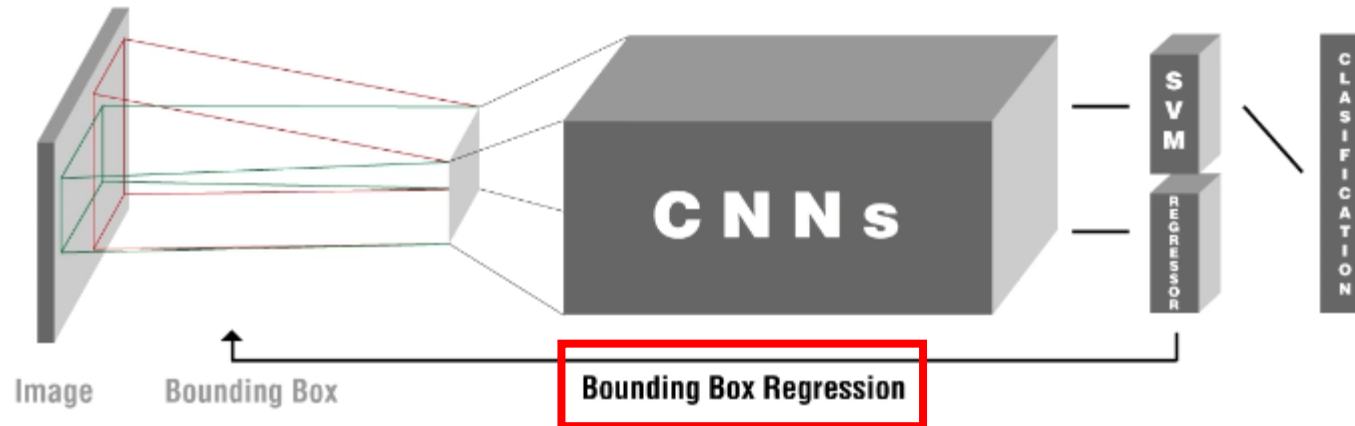


## ▶ 1-Stage Object Detection



1. Backbone: 이미지에서 영상의 국소적 혹은 전역적 특성을 나타내는 feature map 추출
2. Neck: backbone feature에서 직접 head로 연결되는 경우도 있지만 최근 연구들은 다양한 스케일의 backbone feature들을 재조합하여 객체 검출에 더 적합한 feature map을 만든다. 이 단계를 'neck'이라 한다.
3. Head: neck feature에서 conv 연산을 반복적용하여 출력을 만들어낸다.
4. Decoder: head feature은 conv 연산의 직접적인 출력이기 때문에 값의 범위가 한정되지 않아서 이를 직접 박스의 좌표나 클래스 확률 등으로 사용하기는 어렵다. Decoder는 학습 파라미터 없이 head feature에 정해진 연산을 적용하여 객체 출력 형식으로 변환한다.

## ▶ R-CNN 계열(2-Stage Object Detection)



**Step1.** 데이터와 레이블을 투입.

**Step2.** 영역 제안(Region Proposal).

**Step3.** 제안된 영역을 CNNs에 넣음.

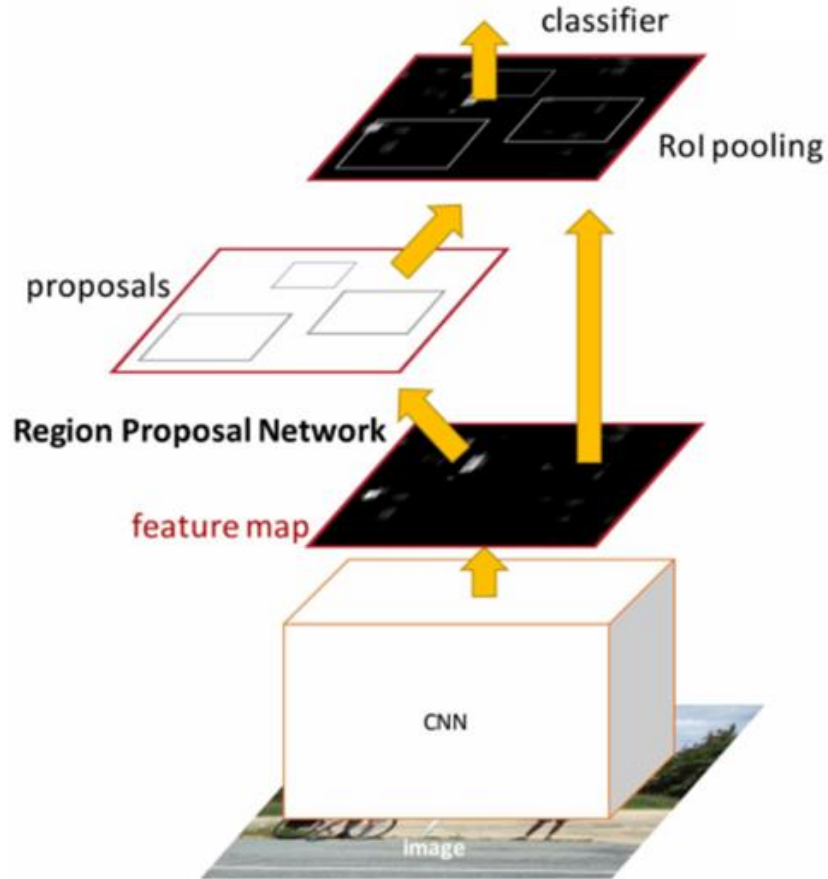
**Step4.** 분류 및 Bounding Box 조정

Input으로 주어진 레이블(label)과 CNN에서 형성한 Bounding Box의 차이를 구해서 그만큼 조정을 할 수 있도록 하는 절차.

이를 통해 신경망 초기로 위의 절차 결과가 전달되어서 Region proposal이 좀 더 의미 있는 영역이 되도록 만듦.

## ▶ R-CNN 계열(2-Stage Object Detection)

많은 **Proposals**, 그 과정에서의 **오버헤드**



1. 영상에서 오브젝트가 있을 것 같은 후보(ROI – Region Of Interest : 관심 영역)를 먼저 뽑는다.
  - Region Proposal Network가 영상에서 오브젝트가 있을 것 같은 영역(Region)들을 뽑아서 (여기서 어떤 클래스가 있는지 확인해보라고) 제안(proposal)한다.
2. 후보로 뽑힌 ROI들은 분류기(Classification network)에 의해 클래스 분류가 이뤄지고 Bound Box를 찾는다.
  - 앞의 CNN이 특징추출기의 역할을 하며, RPN(Region Proposal Network)는 오브젝트가 있을 법 한 경계 박스를 추천하며, classifier는 앞의 두 네트워크가 전달하는 특징데이터와 경계 박스 위치를 받아서 분류를 한다.

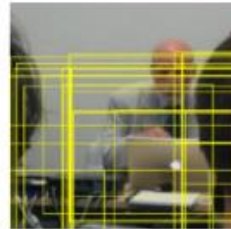


# ▶ YOLO (You Only Look One)

Example: find the father of the internet

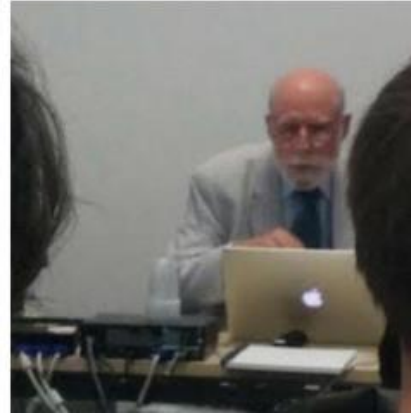


Selective Search  
2.24 seconds



EdgeBoxes  
0.38 seconds

Example: find the father of the internet

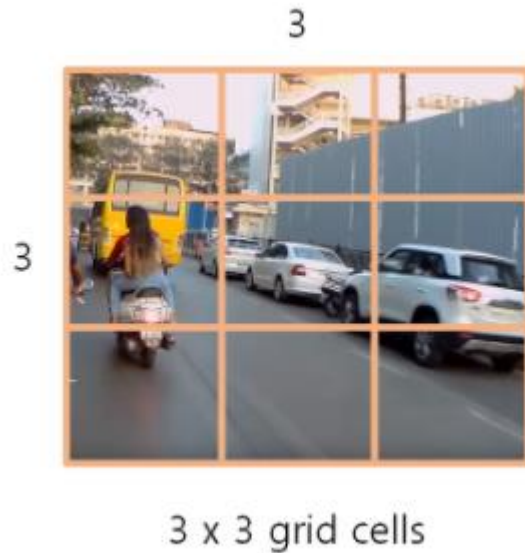


Cheaper Alternative: **grids**

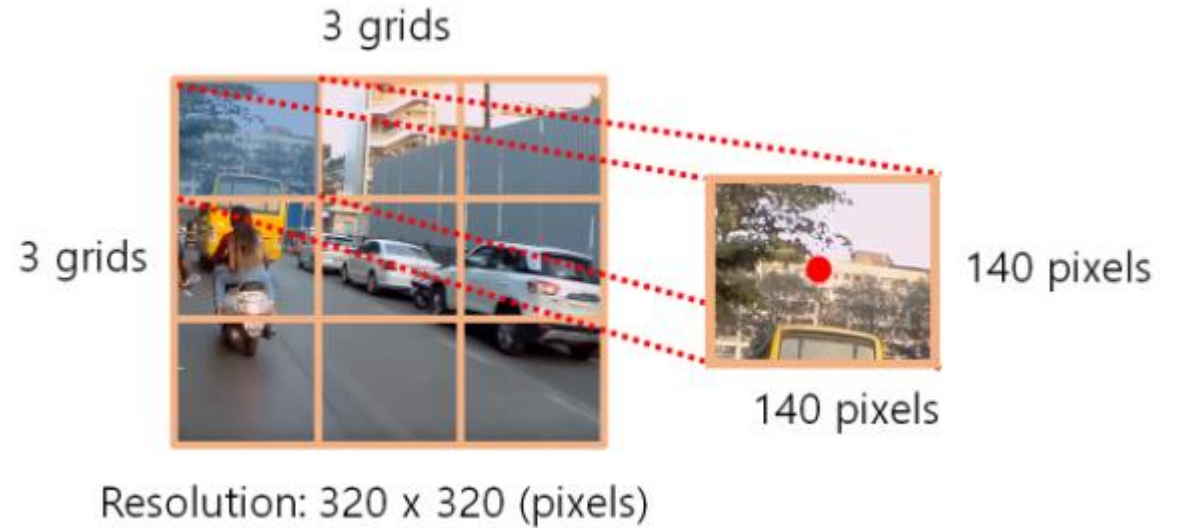
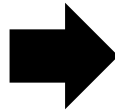


- R-CNN 계열에서 proposal의 수가 무한해 지는 것은 결국 오버헤드 문제를 발생시키므로, YOLO에서는 proposal을 제안하는 방식을 Grid 방식으로 바꾸었다.
- Grid cell의 개수가 곧 proposal의 수로 proposal을 구하는 과정에서 오버헤드가 전혀 없다.
- Grid cell안에 오브젝트가 있다는 보장은 전혀 없지만 그 것은 다른 방식도 마찬가지다.
- YOLO는 실시간성을 확보하기 위해 Proposal의 수가 적은 grid 방식을 더욱 발전시켜서 사용함.

## ▶ YOLO (You Only Look One) \_ Grid cells

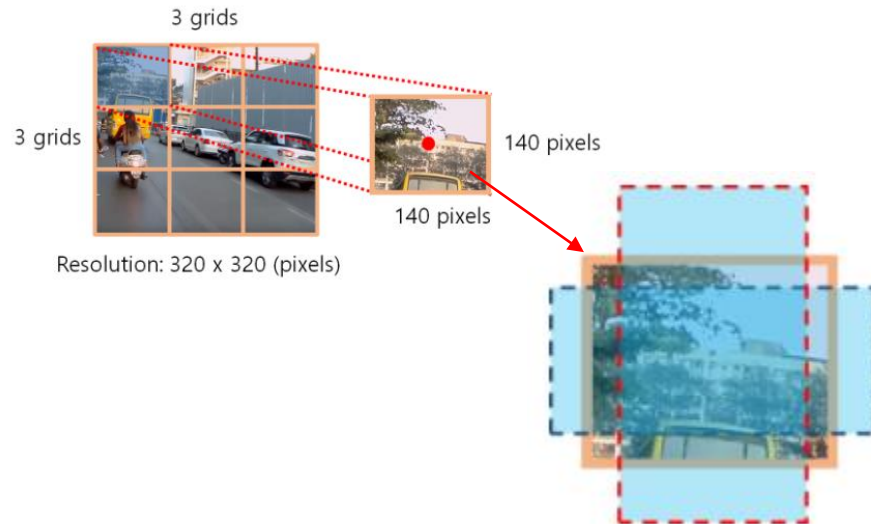


YOLO는 처음에 여러 격자(grid)로 그림을  $S \times S$ 로 나눕니다.  
(논문에선 7 x 7의 격자로 분할 합니다.)  
앤드류응 교수님의 강의에선 3 x 3의 grid cells로 나눕니다.



Grid cell로 나누면 위와 같이 한 셀의 정보를 갖게 됩니다.  
각 cell도 각각의 pixel size와 중앙점을 갖게 됩니다.  
오른쪽은 첫번째 cell을 예시로 떼어내어 봤습니다.

## ▶ YOLO (You Only Look One) \_ Anchor Box



각각의 cell에서는 Anchor Box (cell과 Anchor는 다름)라는 것은 가집니다.

Anchor Box의 수는 사용자가 설정할 수 있으며, 본 자료에서는 2개의 Anchor Box를 갖고 있다고 가정하고 설명합니다.

Anchor Box는 보통 1:1, 1:2, 2:1등의 비율이 정해져 있고 크기가 다른 박스들이 있습니다. 정하기는 개발자 나름이겠죠.

`anchor[] = {10, 10, 10, 20}`

위와 같이 배열을 만들어주면아래와 같이 비율적으로 정할 수 있겠죠. 픽셀 사이즈로 정해주면 됩니다.

첫번째 Anchor 넓이, 높이

`anchor[] = {10, 10, 10, 20}`

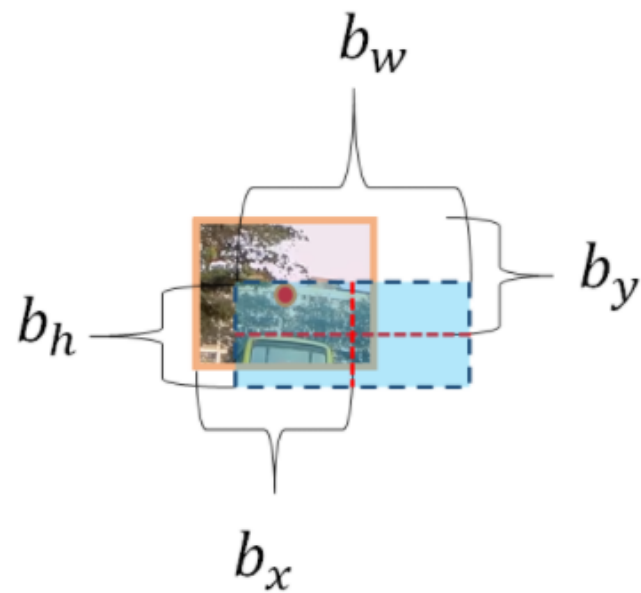
두번째 Anchor 넓이, 높이

## ▶ YOLO (You Only Look One) \_ Anchor Box

이 Anchor Box의 정보인 아래 변수(parameter)이 정말 중요합니다.

<Bounding Box 정보>

$$P_o, b_x, b_y, b_w, b_h$$



$P_o$ : 해당 cell의 Objectness (물체가 있을 확률)

$b_x, b_y$ : 해당 cell의 x, y 값

$b_w, b_h$ : 해당 cell에 있는 Anchor Box의 w, h 값

(위 모든 값은 0~1의 값으로 normalize 됨)

140 x 140 (pixels)

- $b_x$ 와  $b_y$ 는 셀의 왼쪽 위 모서리에서부터 거리입니다.
- $b_w, b_h$ 는 Anchor(Bounding Box)의 w, h값입니다.
- $P_o$ 는 해당 Box의 Confidence Score(=objectness)이라는 물체가 있을 확률입니다.

## ▶ YOLO (You Only Look One) \_ Grid cells

각 셀에선 Class Probability + Bounding Box Parameter ( $B_x, B_y, B_w, B_h, P_o$ )로 구성 됩니다. Bounding Box가 많다면 더 추가가 되겠죠.

Bounding Box 파라미터는 위에서 설명을 했고, Class Probability에 대해서 설명 해드리겠습니다.

클래스가 3개라고 가정 합시다.

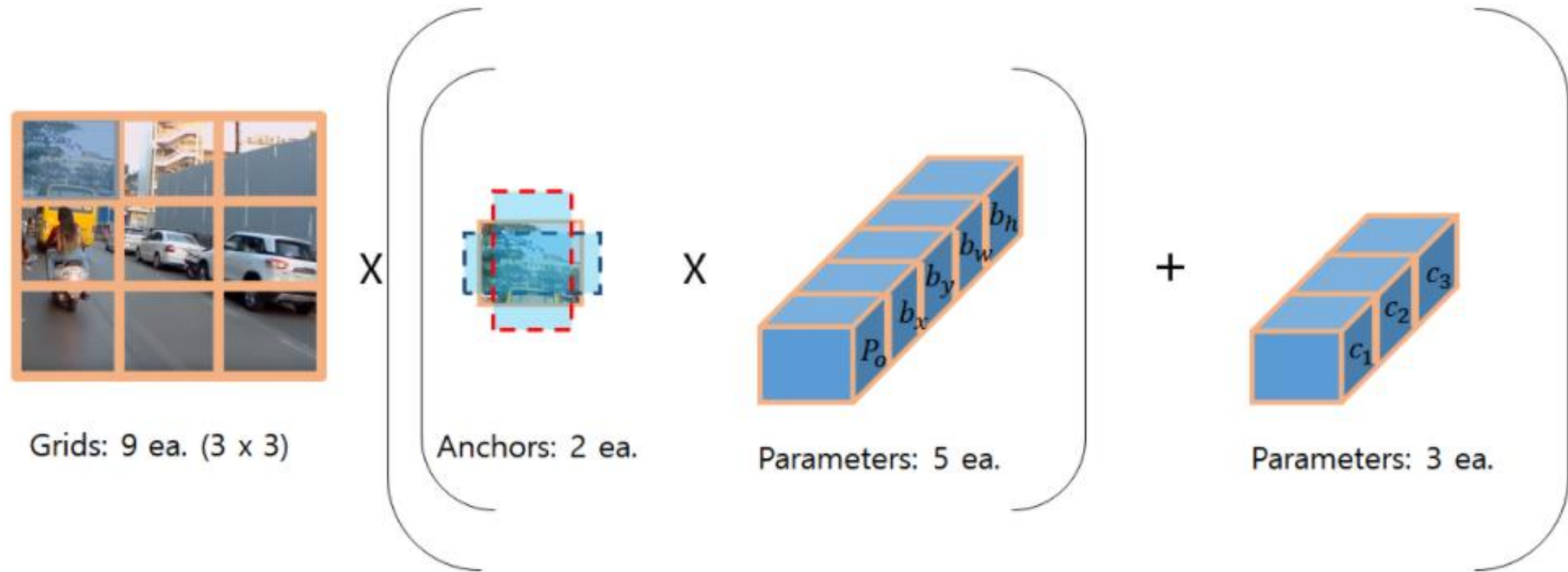
**c1, c2, c3** : 해당 Bounding box가 클래스별로 몇퍼센트를 갖는지에 대한 파라미터 입니다.

(차: 0.3(30%), 동물 0.2(20%), 사람 0.5(50%) 이런식으로)

그러면 3개가 될테고 Bounding box가 2개가 아래처럼 있으면 총  $3 + 5 + 5$ 가 되는 것 입니다.



## ▶ YOLO (You Only Look One)



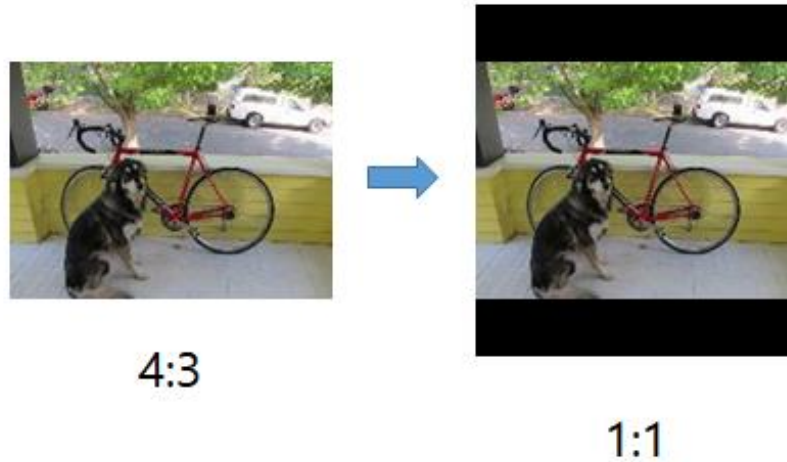
최종 파라미터 개수 = 117 ea. (9 x ((2 x 5) + 3)).



# ▶ YOLO (You Only Look One) \_ Algorithm Flow

## 1. Letter Box Image 생성

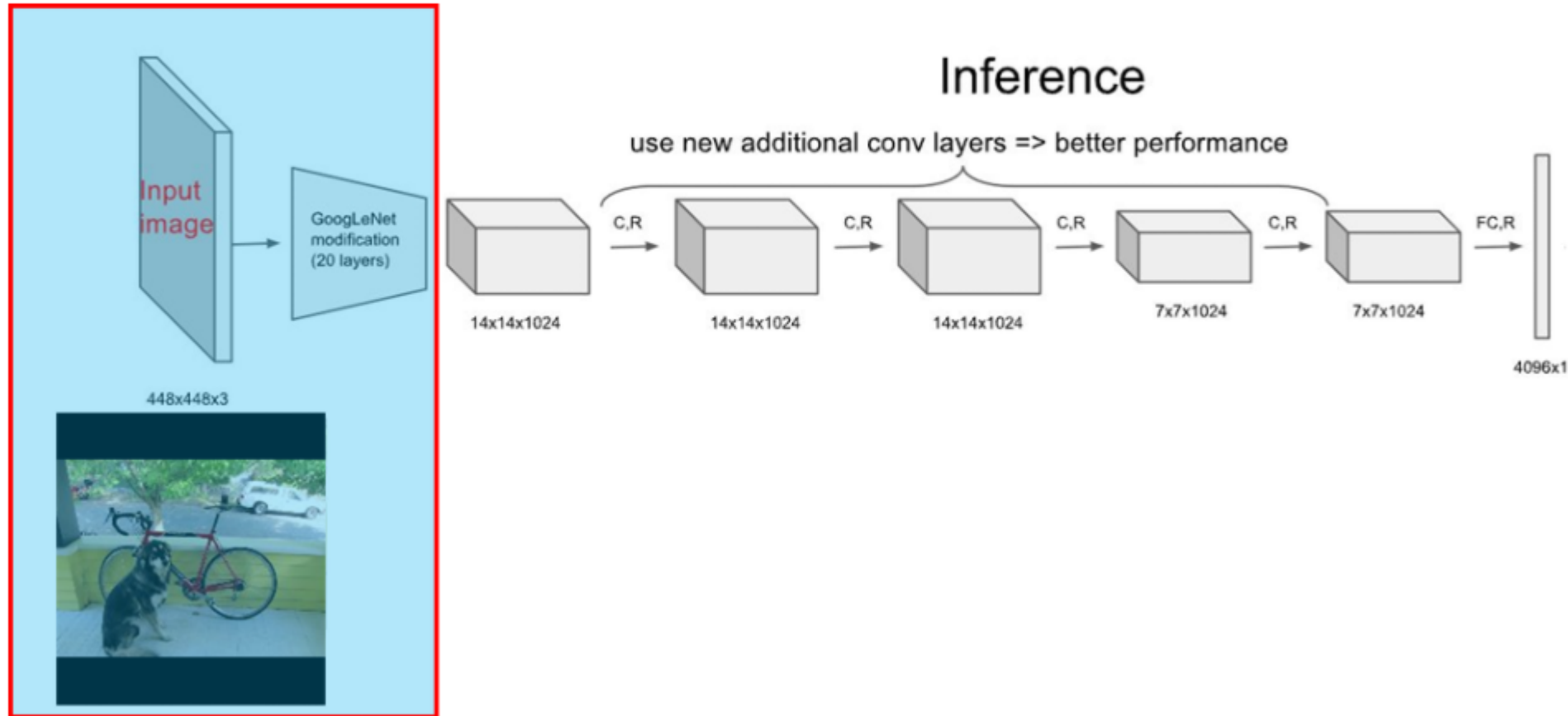
원본 이미지는 보통 4:3 또는 16:9를 갖습니다. 하지만 네트워크 input은 608x608 이런식으로 사각형을 갖게 되므로, 비율이 다르게 된 이미지를 그대로 넣을 경우 모양이 찌그러질수 있습니다. 따라서 여분을 매꿔주는 이미지를 새로 생성 하는 작업을 Letter Box Image라고 합니다.



# ▶ YOLO (You Only Look One) \_ Algorithm Flow

## 2. GoogLeNet에 입력 이미지를 넣어 줍니다.

이때 Trained Data(Weight) + Network Model + image가 Input으로 들어가게 됩니다.





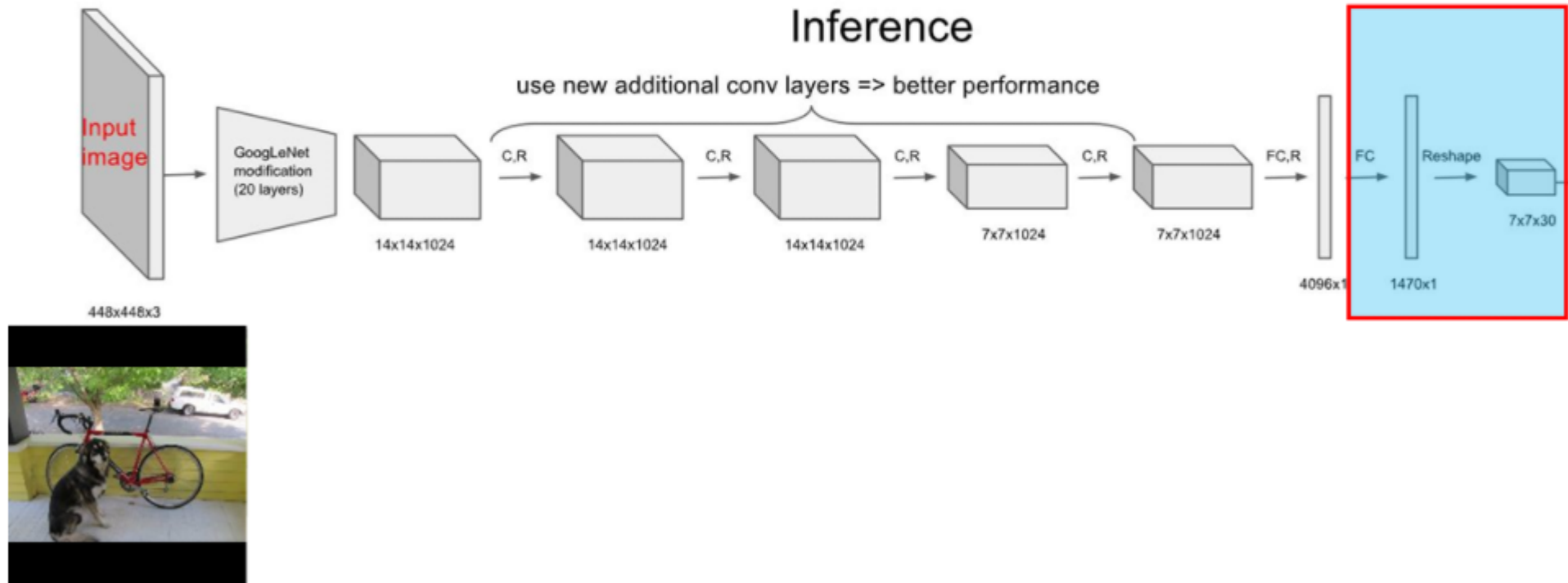
# ▶ YOLO (You Only Look One) \_ Algorithm Flow

## 3. Fully Connected Layer를 아래의 순서로 설정한 Grid Cell크기에 맞게 변형하여 줍니다.

1470의 Tensor를 갖는 FCL(Fully Connected Layer)로 변경 한 이유는  $7 \times 7 \times ((2 \times 5) + 20) = 1,470$ 개 이기 때문입니다. 저 계산식은  $(S \times S \times ((B \times \text{parameters}) + \text{class numbers}))$ 에서 나옵니다. (여기서 parameters는 Po, Bx, By, Bw, Bh)

따라서,  $7 \times 7 \times ((2 \times 5) + 20) = 1,470$ 개의 텐서 생성

(위에 대해선 추후 자세히 설명 합니다.)



## ▶ YOLO (You Only Look One) \_ Algorithm Flow

### 4. 각 Bounding Box에 대한 Class Confidence Score를 계산 한다.

4-1) 인풋을  $S \times S$ 로 나눕니다. ( $S$ 는 Grid Cell)

본 글에서  $S = 7$  (따라서  $7 \times 7$  그리드 셀)

4-2) Bounding Box Parameter 구하기

- 각 '그리드(셀)'는  $B$  개의 Bounding Box가 있습니다. ( $B$ 는 Bounding Box = Anchor = Window)

- 본 글에서  $B = 2$

- 각 'Bounding Box'는 5개의 파라미터( $B_x, B_y, B_w, B_h, P_o$ )를 갖습니다.

\* $B_x, B_y$  : 각 그리드의 왼쪽위 모서리에서부터 Bounding Box의 중앙점 까지의 거리 ( $0 \sim 1$  사이의 값으로 정규화)

\* $B_w, B_h$  : 각 Bounding Box의 크기. 본 크기는 화면 전체 이미지를 기준으로 함. ( $0 \sim 1$  사이로 정규화)

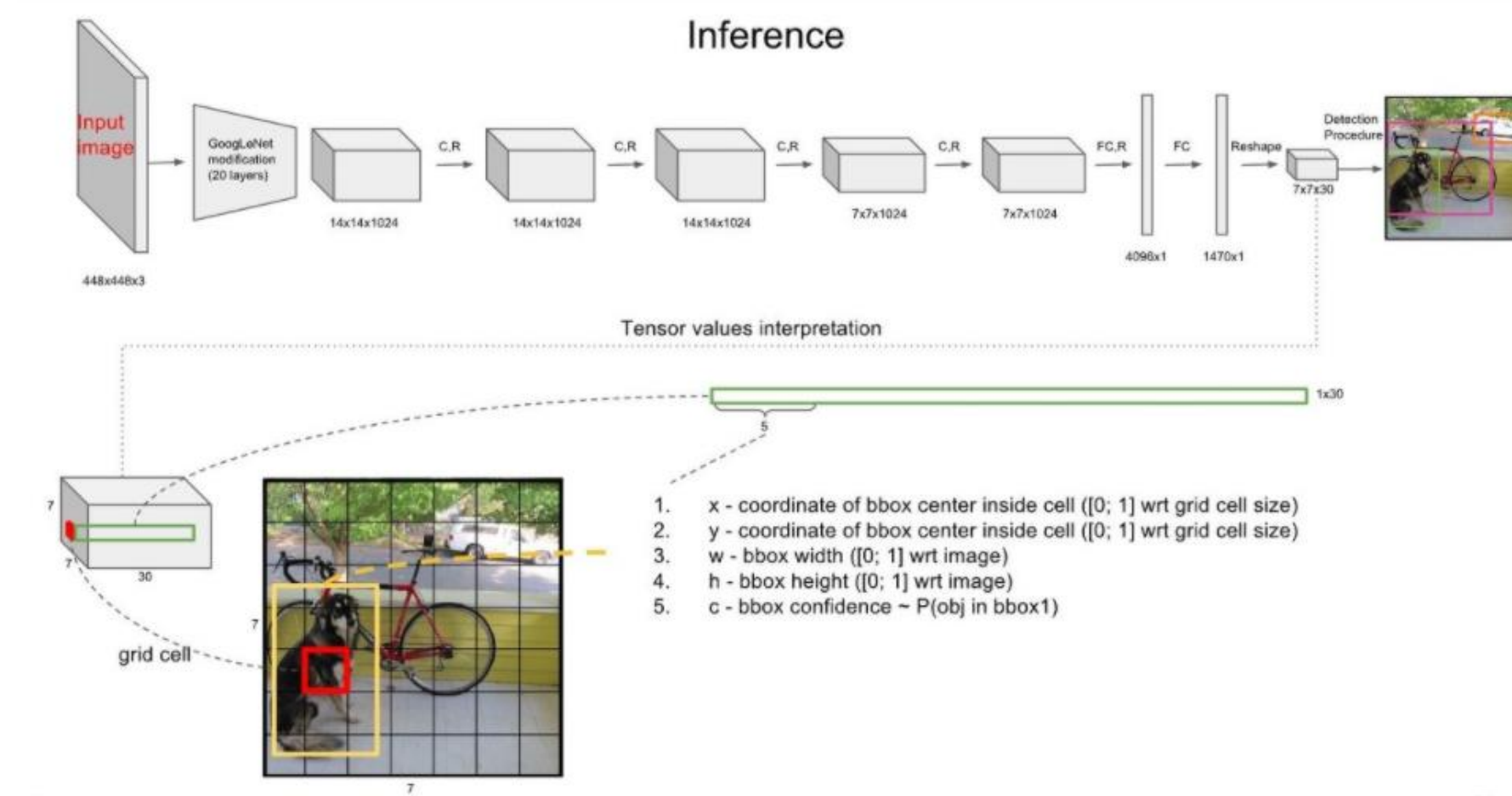
\* $P_o$  : confidence score (=objectness)

\*여기서  $P_o$ 는 개발자가 Threshold를 정해서 일정수준 이하의 정확도를 가지면 제외 시키고 다시 찾는 방식을 가집니다. 계속 찾다 없으면 그 Anchor(=Bounding Box)는 0의 값을 갖겠죠.

# ▶ YOLO (You Only Look One) \_ Algorithm Flow

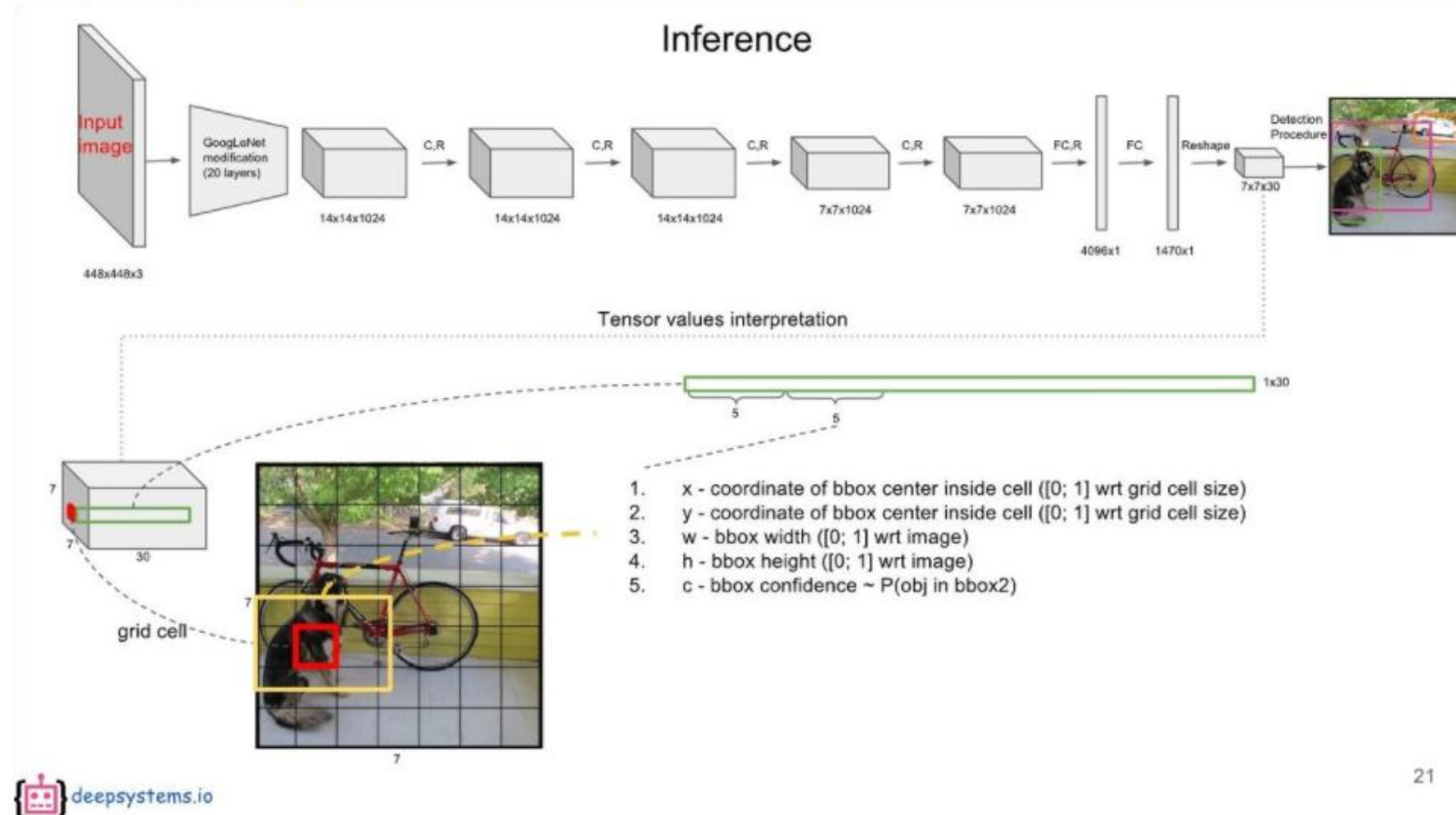
## 4-2-1) 첫번째 Bounding Box

각 그리드에 위에서 설명한 5개의 파라미터가 나열 됩니다.



# ▶ YOLO (You Only Look One) \_ Algorithm Flow

## 4-2-2) 두번째 Bounding Box



21

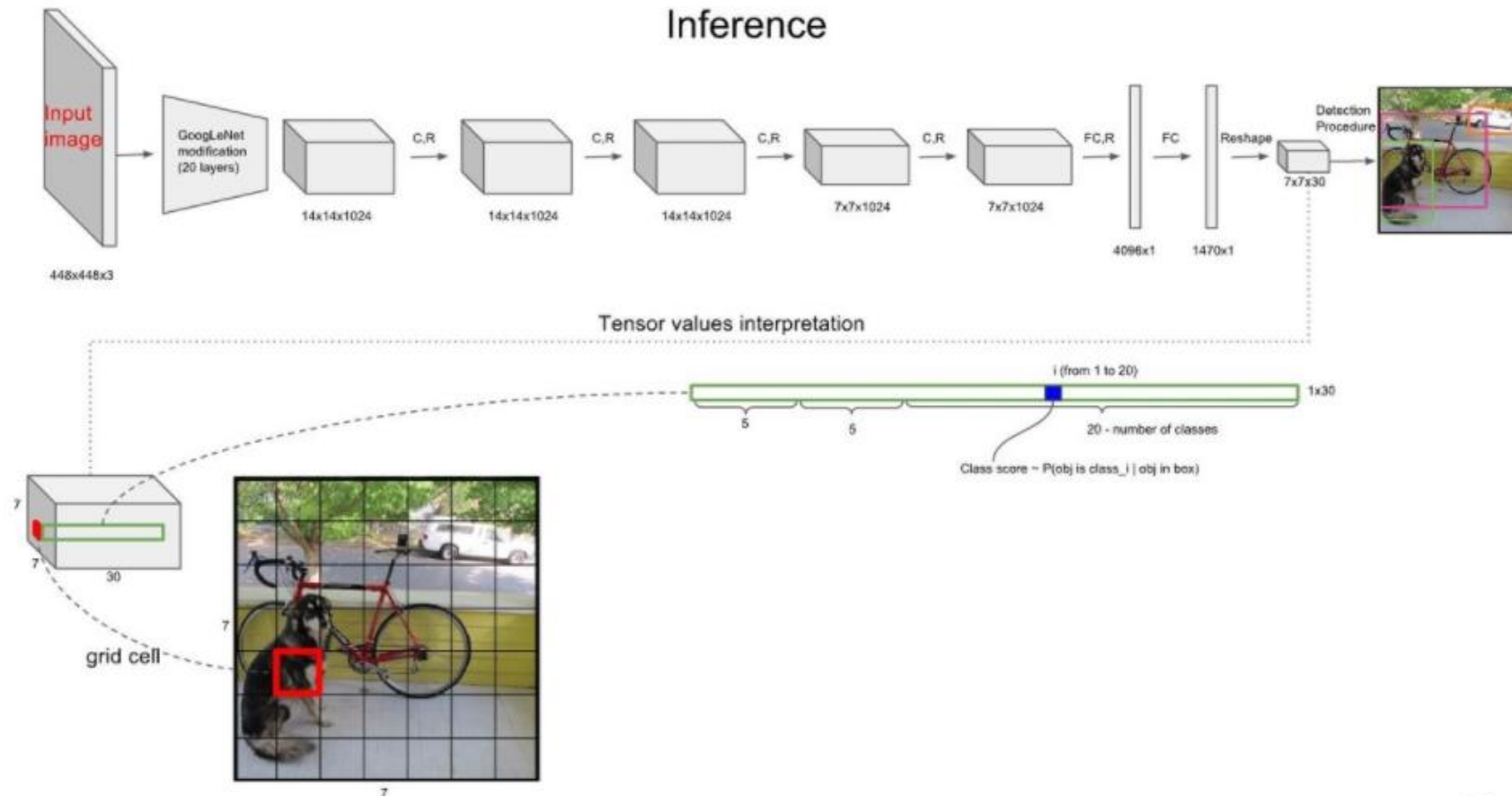
# ▶ YOLO (You Only Look One) \_ Algorithm Flow

4-4) 각 그리드 '셀'은 클래스 개수에 대해 C를 갖는다.

- 본 글에서 C(클래스 개수)는 20으로 지정. ((예시)C1: 사람, C2: 차, C3: 자전거, 등등)

\*C : conditional class probability (=probability)

\*여기서 C도 개발자가 Threshold를 정합니다. 따라서 기준을 충족하지 못할시엔 박스를 그리지 않습니다.

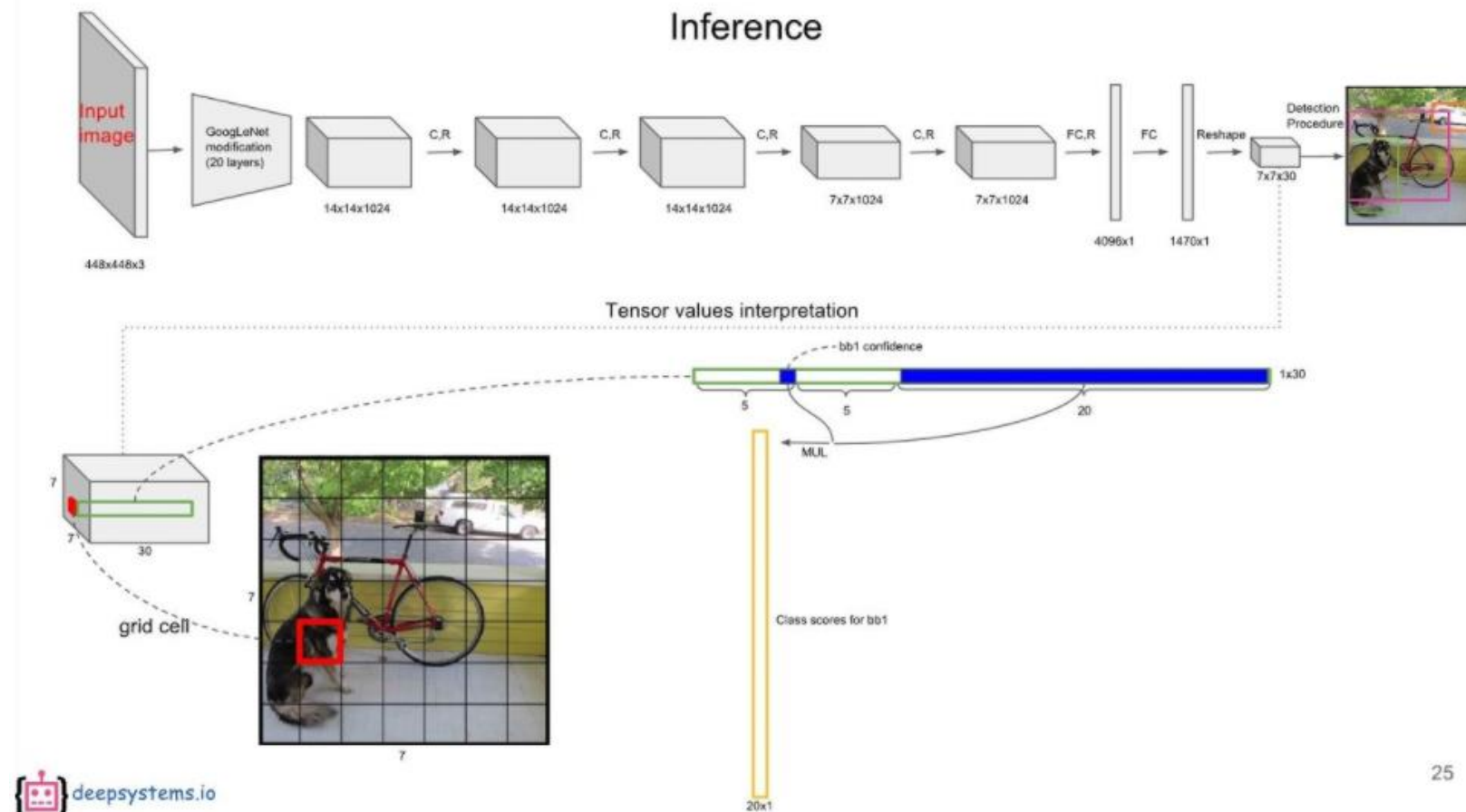


# ▶ YOLO (You Only Look One) \_ Algorithm Flow

4-5) 각 셀마다 2개(Anchors)의 class-specific confidence score값을 찾습니다.

class-specific confidence score는 그 셀의 C(conditional class probability)와 각 박스의 Po(confidence score)를 곱하여 한 박스에 대해 구합니다.

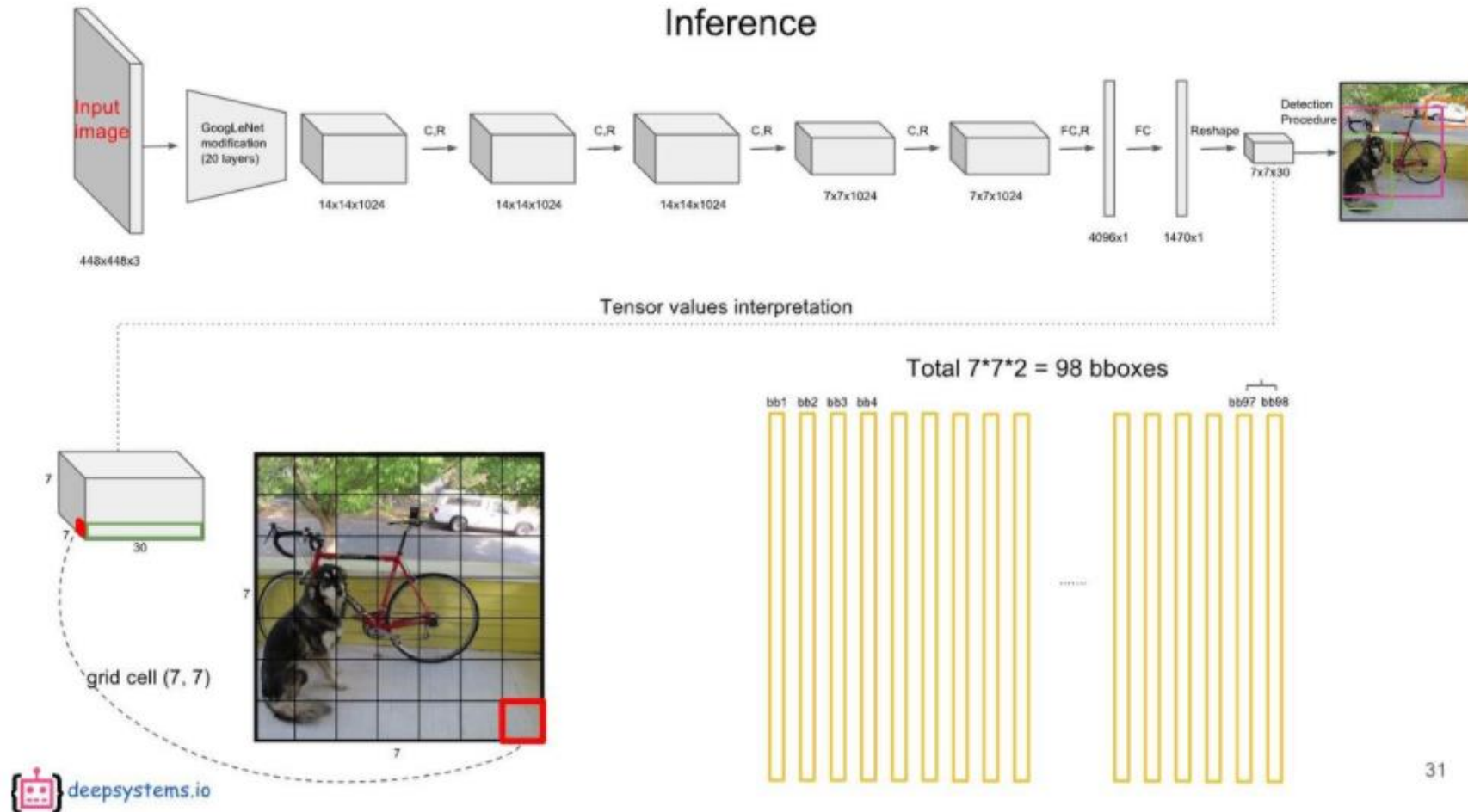
$$\text{class-specific confidence score} = C(\text{conditional class probability}) * Po(\text{confidence score})$$





# ▶ YOLO (You Only Look One) \_ Algorithm Flow

다른 박스에 대해서도 또 해야겠죠.



즉,  $7 \times 7 \times 2 = 98$ 개의 셀은 모두 class-specific confidence score를 갖게 됩니다.

# ▶ YOLO (You Only Look One) \_ Algorithm Flow

## 5. NMS 계산

이제 98개의 Bounding Box가 나왔으므로 이걸 NMS(Non-Maximum Suppression)을 하여 중복되는 박스들을 없애줍니다.

**NMS란** 하나 박스를 기준을 잡고 일정 Threshold를 정한 후, 다른 박스를 비교 해 보았을때 교집합이 되는 부분이 정해진 Threshold보다 적을시 제거 하는 방법 입니다.

NMS전에 기준 BOX를 잡기 위해 Class Confidence Score를 기준으로 박스를 정렬 합니다. (Quick Sort 알고리즘 사용)

그 다음 Score가 제일 높은 박스를 기준으로 다른 박스를 순서대로 비교 하며 NMS를 진행 합니다.

YOLOv3에선 0.4 또는 0.5값으로 한다.

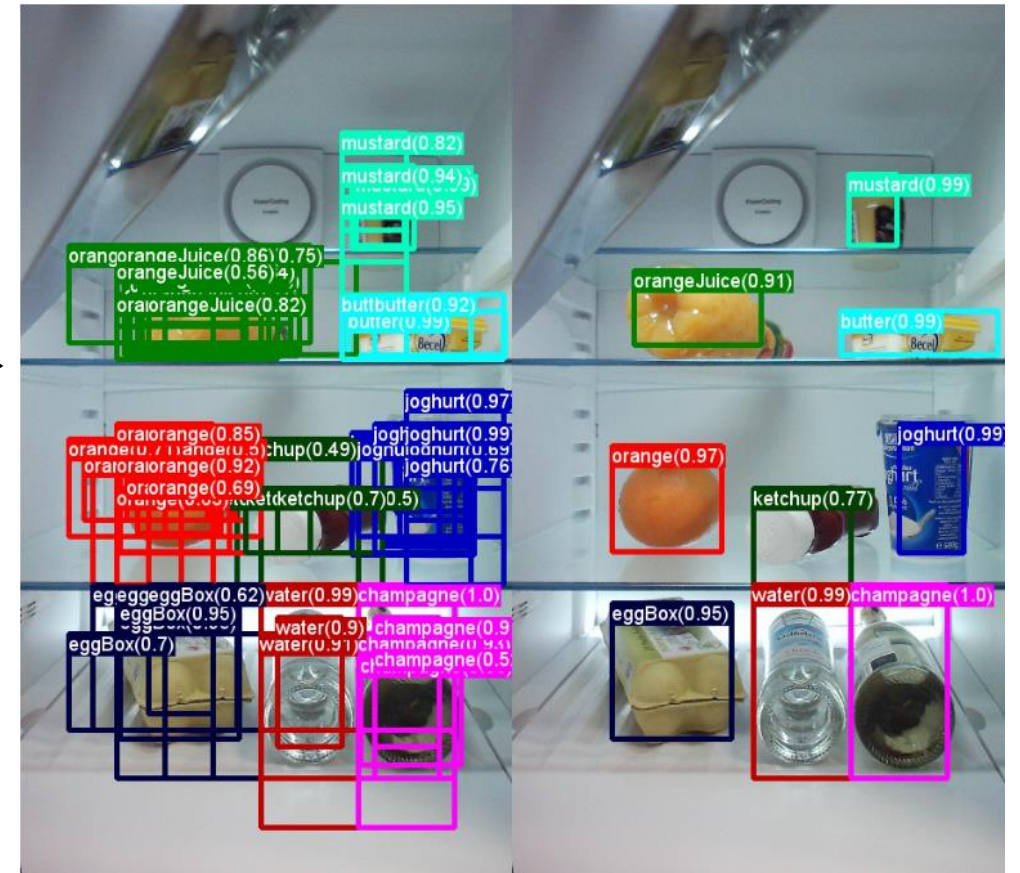
하나의 오브젝트에 대해 경계박스가 겹치는 경우

그 중에서 신뢰도가 가장 높은 것 하나만 남기고 나머지는 모두 지우는 기법.

NMS 적용 예 ▶

## 6. 박스 그리기

최종 박스를 그린다.

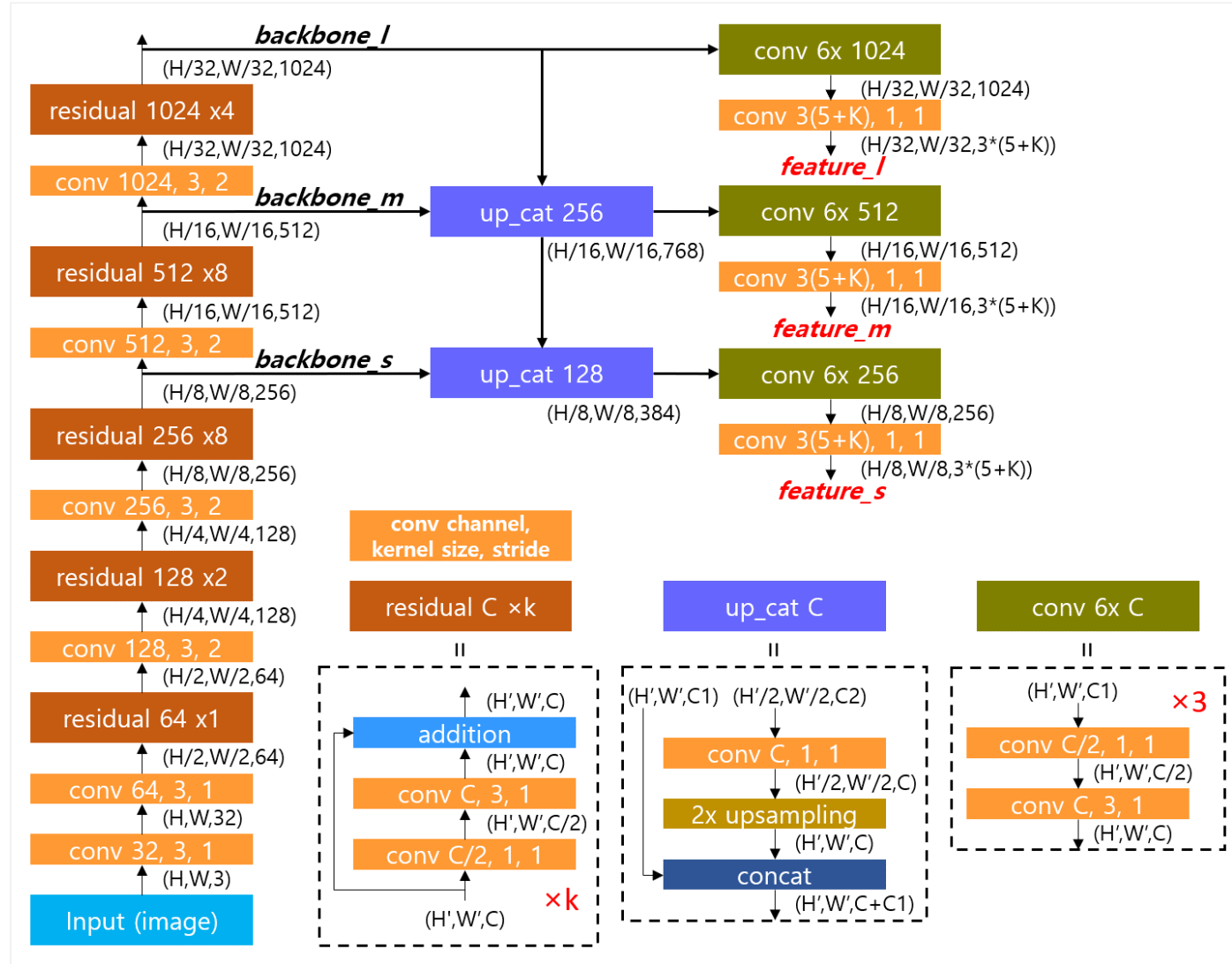




# ▶ YOLOv3

- YOLO의 Loss function과 관련된 부분은 아래 링크를 참조.  
<https://goodgodgd.github.io/ian-lecture/archivers/det-loss-and-log>

YOLOv3 세부 구조 ▶



# ▶ YOLOv3 \_ DarkNet 실행

## [1] Darknet 설치

① Darknet 소스코드를 Github에서 다운로드한다.

```
$ mkdir ~/Draknet
$ cd ~/Draknet
$ git clone https://github.com/pjreddie/darknet.git
$ cd darknet
```

② Makefile을 gedit으로 열어서 아래와 같이 수정 \_ Jetson TX2의 CUDA 아키텍처는 "62" 이다.

```
$ gedit Makefile
```

▶ (참조: <https://developer.nvidia.com/cuda-gpus>)



```
GPU=1
CUDNN=1
OPENCV=1
OPENMP=0
DEBUG=0

ARCH= -gencode arch=compute_62,code=[sm_62,compute_62]
#ARCH= -gencode arch=compute_30,code=sm_30 \
#      -gencode arch=compute_35,code=sm_35 \
#      -gencode arch=compute_50,code=[sm_50,compute_50] \
#      -gencode arch=compute_52,code=[sm_52,compute_52]
#      -gencode arch=compute_20,code=[sm_20,sm_21] \ This one is deprecated?
```

# ▶ YOLOv3 \_ DarkNet 실행

## [1] Darknet 설치

③ Make 명령어로 컴파일하면, darknet 파일이 생성된다.

```
$ make -j4  
$ sudo ldconfig
```

# ▶ YOLOv3 \_ DarkNet 실행

## [2] YOLOv3 실행 준비

- 일반적으로 물체 인식을 위해서는 두개의 파일이 필요하다
  - 1) 신경망 레이어 정보가 담긴 .cfg 파일
  - 2) 가중치 파라미터 정보가 담긴 .weight 파일
- darknet 소스 코드에서 cfg 파일이 존재하나, weights 파일이 없다.  
따라서, weights 이름의 경로를 만들어 yolov3-tiny.weights 파일을 저장한다.

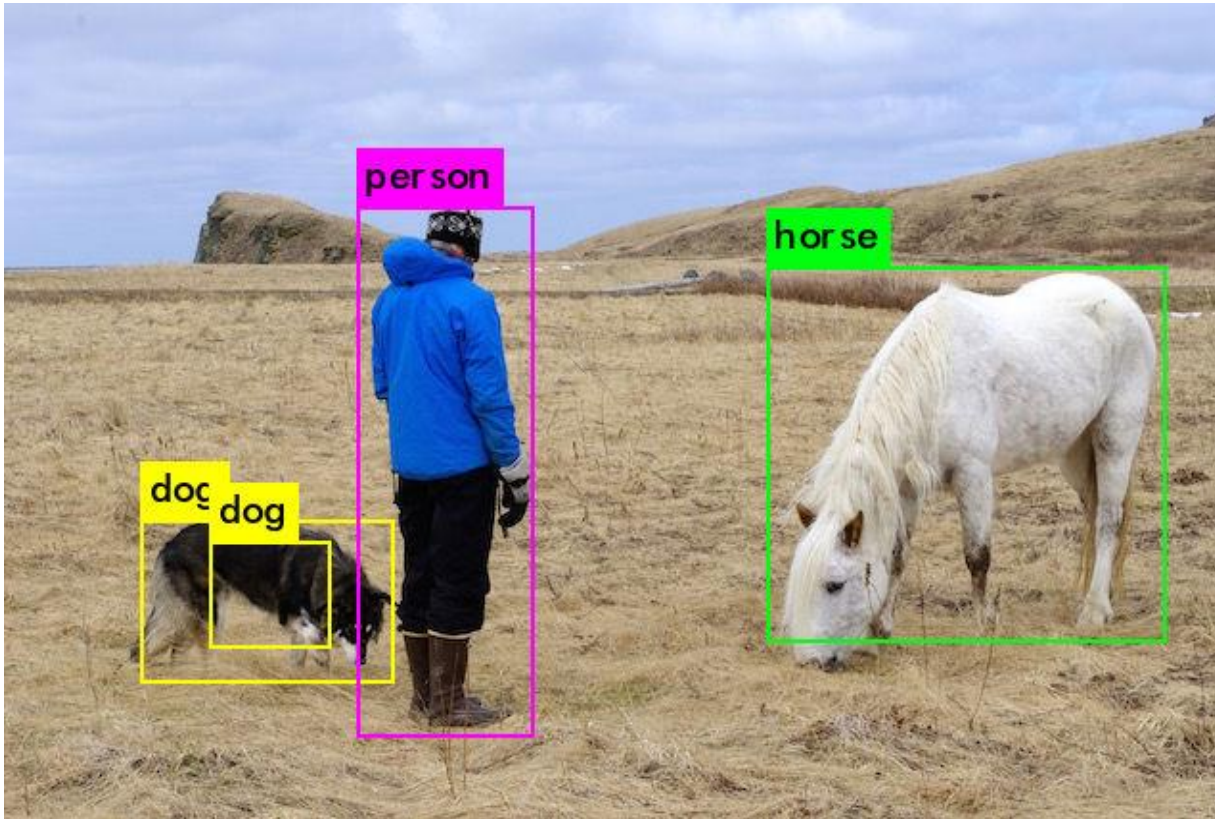
```
$ mkdir ~/Darknet/darknet/weights  
$ wget https://pjreddie.com/media/files/yolov3-tiny.weights -P./weights
```

## ▶ YOLOv3 \_ DarkNet 실행

### [3] YOLOv3 실행

- .cfg 파일과 .weight 파일을 불러와 YOLOv3로 물체 인식을 한다.
- 맨 뒤에는 원하는 사진을 인터넷에서 다운로드하여 다양하게 직접 시험해 볼 수 있다.

```
$ ./darknet detect cfg/yolov3-tiny.cfg weights/yolov3-tiny.weights data/person.jpg
```

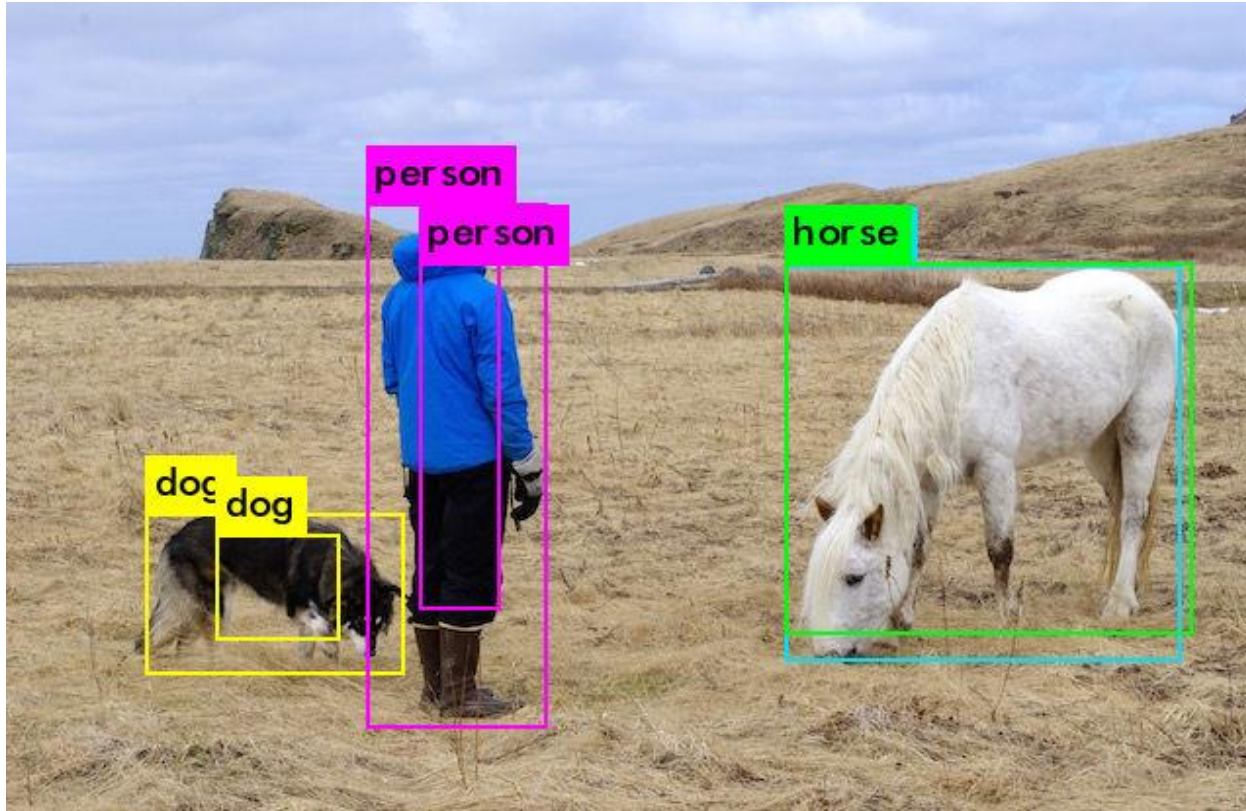


## ▶ YOLOv3 \_ DarkNet 실행

### [3] YOLOv3 실행 \_ 임계치 설정

- 임계치 기본값은 " -thresh 0.5 " 이다. 이를 " -thresh 0.1"로 수정 후 실행 해본다.
- thresh 0.1 : 인식률이 10% 이상인 것을 바운딩 박스로 보여준다.

```
$ ./darknet detect cfg/yolov3-tiny.cfg weights/yolov3-tiny.weights data/person.jpg -thresh 0.1
```



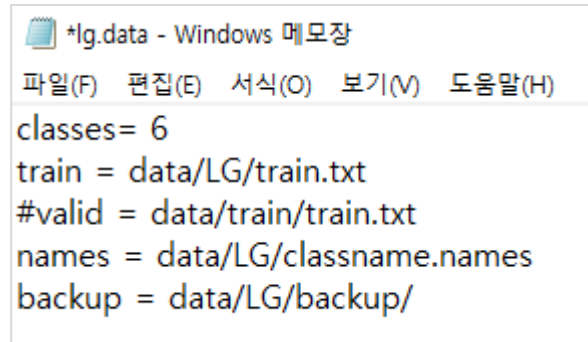
## ▶ YOLOv3 \_ DarkNet Finetuning

- \* DarkNet 학습에 필요한 파일 3가지 ➡
- ① .data 파일
  - ② .txt 파일
  - ③ .names 파일

## ▶ YOLOv3 \_ DarkNet Finetuning

\* DarkNet 학습에 필요한 파일 3가지 ➡ ① .data 파일

: class 개수, txt 파일 경로, names 파일경로, weight 파일 저장할 경로 등 저장.



```
*lg.data - Windows 메모장
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)
classes= 6
train = data/LG/train.txt
#valid = data/train/train.txt
names = data/LG/classname.names
backup = data/LG/backup/
```



## ▶ YOLOv3 \_ DarkNet Finetuning

\* DarkNet 학습에 필요한 파일 3가지 ➡ ② .txt 파일

: 이미지파일이 저장된 경로를 기록한 파일

(실제 이미지가 저장된 폴더 내에는 각 이미지에 대한 라벨링 정보가 들어간 txt 파일도 있어야함)



```
train - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
/home/juyeon/Darknet/darknet/data/LG/img/0.jpg
/home/juyeon/Darknet/darknet/data/LG/img/1.jpg
/home/juyeon/Darknet/darknet/data/LG/img/2.jpg
/home/juyeon/Darknet/darknet/data/LG/img/3.jpg
/home/juyeon/Darknet/darknet/data/LG/img/4.jpg
/home/juyeon/Darknet/darknet/data/LG/img/5.jpg
/home/juyeon/Darknet/darknet/data/LG/img/6.jpg
/home/juyeon/Darknet/darknet/data/LG/img/7.jpg
/home/juyeon/Darknet/darknet/data/LG/img/8.jpg
/home/juyeon/Darknet/darknet/data/LG/img/9.jpg
/home/juyeon/Darknet/darknet/data/LG/img/10.jpg
```

[ train.txt 파일 ]



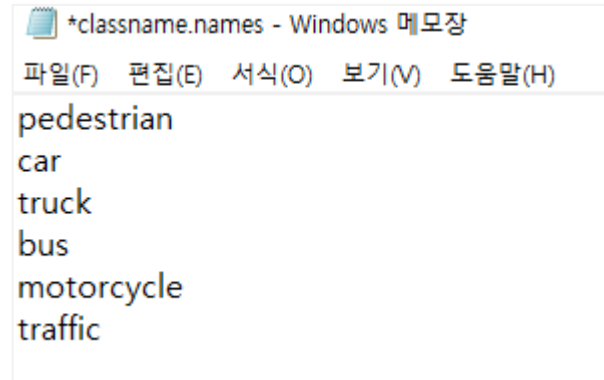
JUYEON (E:) > Darknet > Darknet > darknet > data > LG > img		
	이름	유형
	0	JPG 파일
	0	텍스트 문서
	1	JPG 파일
	1	텍스트 문서
	2	JPG 파일
	2	텍스트 문서

[ 이미지가 저장된 폴더 내부 ]

## ▶ YOLOv3 \_ DarkNet Finetuning

\* DarkNet 학습에 필요한 파일 3가지 ➡ ③ .names 파일

: 사용하는 데이터셋의 class명을 기입.



# ▶ YOLOv3 \_ DarkNet Finetuning

- class를 수정하였다면 YOLO 모델 구조도 변경해야 함 (train 및 test에 사용하는 .cfg파일을 수정해야함).

## ① class 수 변경

```
yolov3.cfg - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
activation=leaky

[convolutional]
size=1
stride=1
pad=1
filters=33
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=6
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

※ 수정해야하는 부분(classes)이 여러 개 일 수 있으니 ctrl + F 로 classes 검색하여 수정하기.

## ▶ YOLOv3 \_ DarkNet Finetuning

- class를 수정하였다면 YOLO 모델 구조도 변경해야 함 (train 및 test에 사용하는 .cfg파일을 수정해야함).

### ② [Convolutional]의 filters 수 변경

$$\text{Filters} = (\text{classes} + 5) * 3$$

```
yolov3.cfg - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
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
activation=leaky
```

※ 수정해야하는 부분(filters)이 여러 개 일 수 있으니 ctrl + F 로 filters 검색하여 수정하기.

## ▶ YOLOv3 \_ DarkNet Finetuning

- .data / .txt / .names 파일 수정을 마쳤다면 make 명령어 실행.

```
$ make
```

- Train 명령어

```
$ ./darknet detector train .data파일위치 .cfg파일위치 (weigh파일위치)
```

- Test 명령어

```
$ ./darknet detector test .data파일위치 .cfg파일위치 weigh파일위치 test이미지
```

## ▶ YOLOv3 \_ DarkNet Finetuning

- 에러나면 .cfg 파일에서 batch size 낮춰서 진행해보기.
- weight 파일 저장하는 횟수는 examples 폴더에서 detector.c의 138 line 수정

```
#ifdef GPU
    if(ngpus != 1) sync_nets(nets, ngpus, 0);
#endif
    char buff[256];
    sprintf(buff, "%s/%s.backup", backup_directory, base);
    save_weights(net, buff);
}
if(i%10000==0 || (i < 1000 && i%100 == 0)){
#ifdef GPU
    if(ngpus != 1) sync_nets(nets, ngpus, 0);
#endif
    char buff[256];
    sprintf(buff, "%s/%s_%d.weights", backup_directory, base, i);
    save_weights(net, buff);
}
free_data(train);
}
```

(i%10000==0)을 (i%2==0)로 바꿔주면 darknet/backup 폴더에 weights 데이터(파일)가 2단위 씩 생성됨.