


A solid blue vertical bar is positioned on the left side of the slide, extending from the top to the bottom.

Object Detection

- 
- ✓ **Classification**
 - ✓ **Localization**
 - ✓ **Sliding window**
 - ✓ **Bounding Box Prediction**
 - ✓ **IOU**
 - ✓ **Non Max Suppression**
 - ✓ **Anchor boxes**
-

Classification/localization/detection

- 1) Image classification : CNN 이용해서 물체를 인식, 하나의 물체 대상
- 2) localization : 물체인식 후에 위치인식, 하나의 물체 대상, bounding box로 위치표시
- 3) detection : 여러 물체 대상 물체인식 후 위치인식

Image classification



"Car"

Andrew Ng

Classification with
localization



"Car"

1 object

Detection

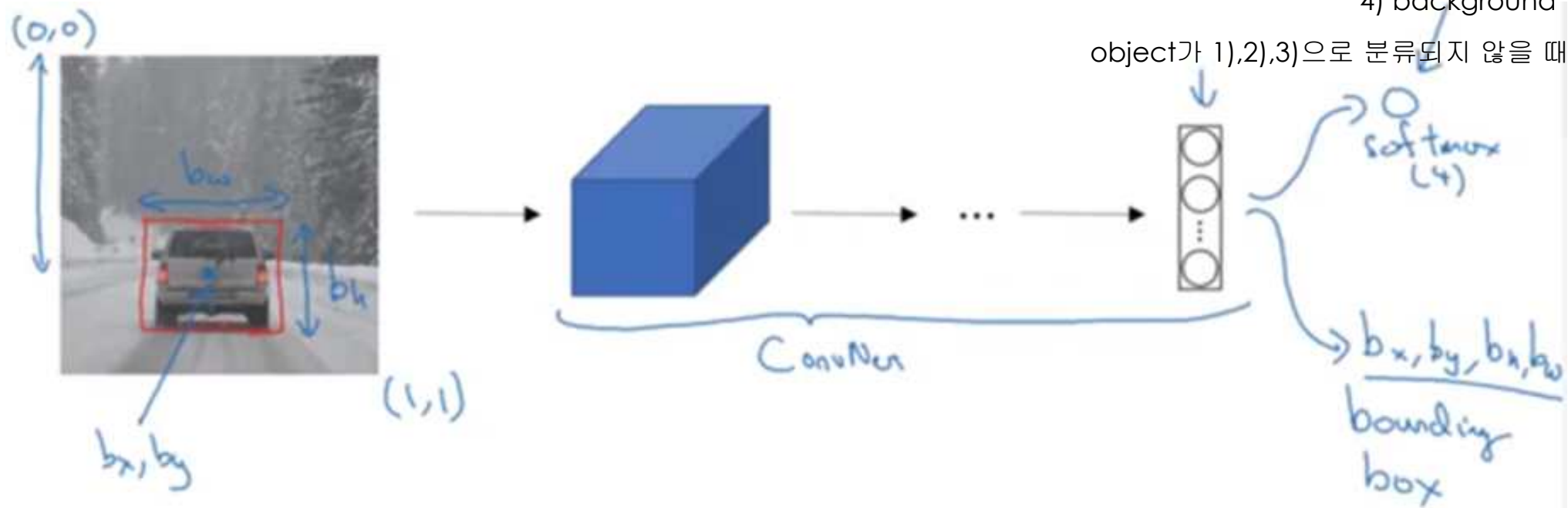


multiple
objects

Classification
Localization
Sliding window
Bounding Box Prediction
IOU
Non Max Suppression
Anchor boxes

Classification with localization

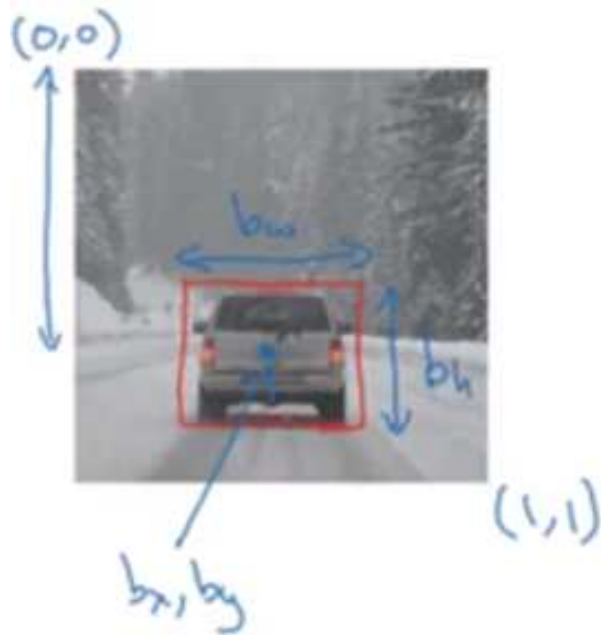
- Classification(인식) : 입력 이미지가 CNN을 거친 후 softmax 통해서 물체인식
- Localization : CNN의 마지막 구조 변경 위치를 알기 위한 기능 추가
- bounding box와 연관된 output 추가 : 좌표 데이터 : b_x, b_y, b_h, b_w



- CNN training 세트
- class label + box의 4개 좌표

- 자동차를 감지고 위치는 $b_x = 0.5$ / $b_y = 0.7$ / $b_h = 0.3$ / $b_w = 0.4$ bounding box로 표현

Bounding Box



- 좌표 데이터 : bx, by, bh, bw
- detect된 물체를 사각형으로 인식 - box 그려서 나타냄
- 이미지의 upper left - $(0,0)$
- lower bottom - $(1,1)$ 로 좌표 계산
- by, bx : bounding box의 중점
- bh, bw : 높이와 너비

$$bx = 0.5 / by = 0.7 / bh = 0.3 / bw = 0.4$$

CNN training 세트 상세

타겟 label y 정의 : class label + bounding box 좌표

$$Y = [Pc, bx, by, bh, bw, c1, c2, c3]$$

$$Y = [1, \underset{bx=0.5}{bx}, \underset{by=0.7}{by}, \underset{bh=0.3}{bh}, \underset{bw=0.4}{bw}, 0, 1, 0]$$

$$Y = [0, \underset{\text{Don't care}}{?}, ?, ?, ?, ?, ?, ?]$$

- Pc (Probability of Class) : Class가 Object인가?
1 : Object 존재
0 : background
- bx, by, bh, bw : bounding box 좌표
- c1, c2, c3 : Class1(Pedestrian), Class2(Car), Class3(Motorcycle)
- 자동차 있는 이미지가 input x 일 때
y = [1, bx, by, bh, bw, 0, 1, 0]
class2(자동차) object 있으니까 Pc = 1, class2 = 1 (나머지 class는 당연히 0)

Loss function

$$Y = [P_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3]$$

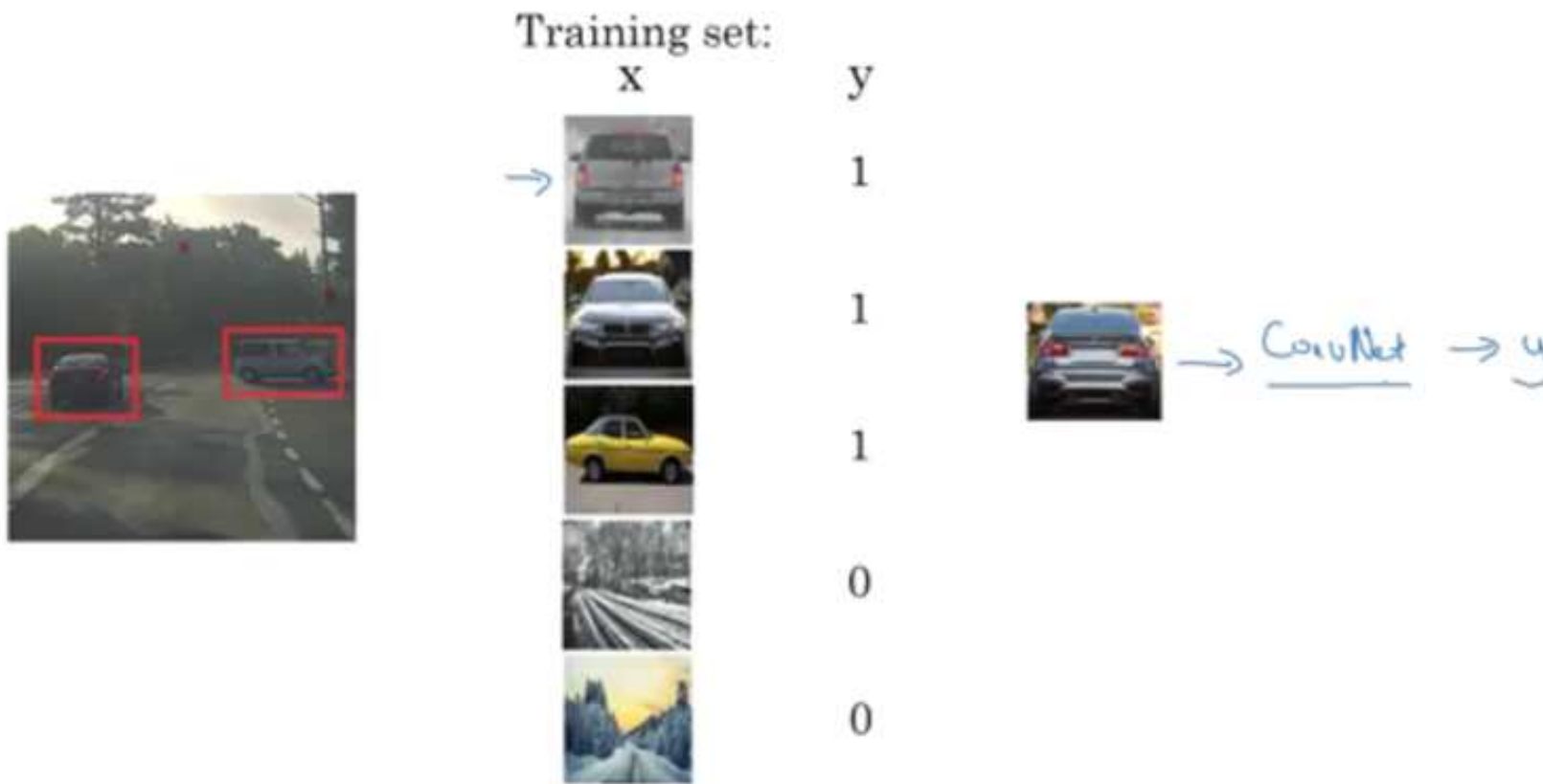
y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8

Mean square error

$$\text{Loss} = (y_1^{\wedge} - y_1)^2 + (y_2^{\wedge} - y_2)^2 + (y_3^{\wedge} - y_3)^2 + (y_4^{\wedge} - y_4)^2 + (y_5^{\wedge} - y_5)^2 + (y_6^{\wedge} - y_6)^2 + (y_7^{\wedge} - y_7)^2 + (y_8^{\wedge} - y_8)^2$$

- If $y_1 = 1$
 $y_1 \sim y_8$ 에 대해 Mean square error
- If $y_1 = 0$
 y_1 만 Mean square error
- Loss function은 cross entropy 같은 log feature 사용해도 됨
 P_c 는 0/1 중 하나니까 logistic regression 가능

Object Detection – Sliding Window algo



Andrew Ng

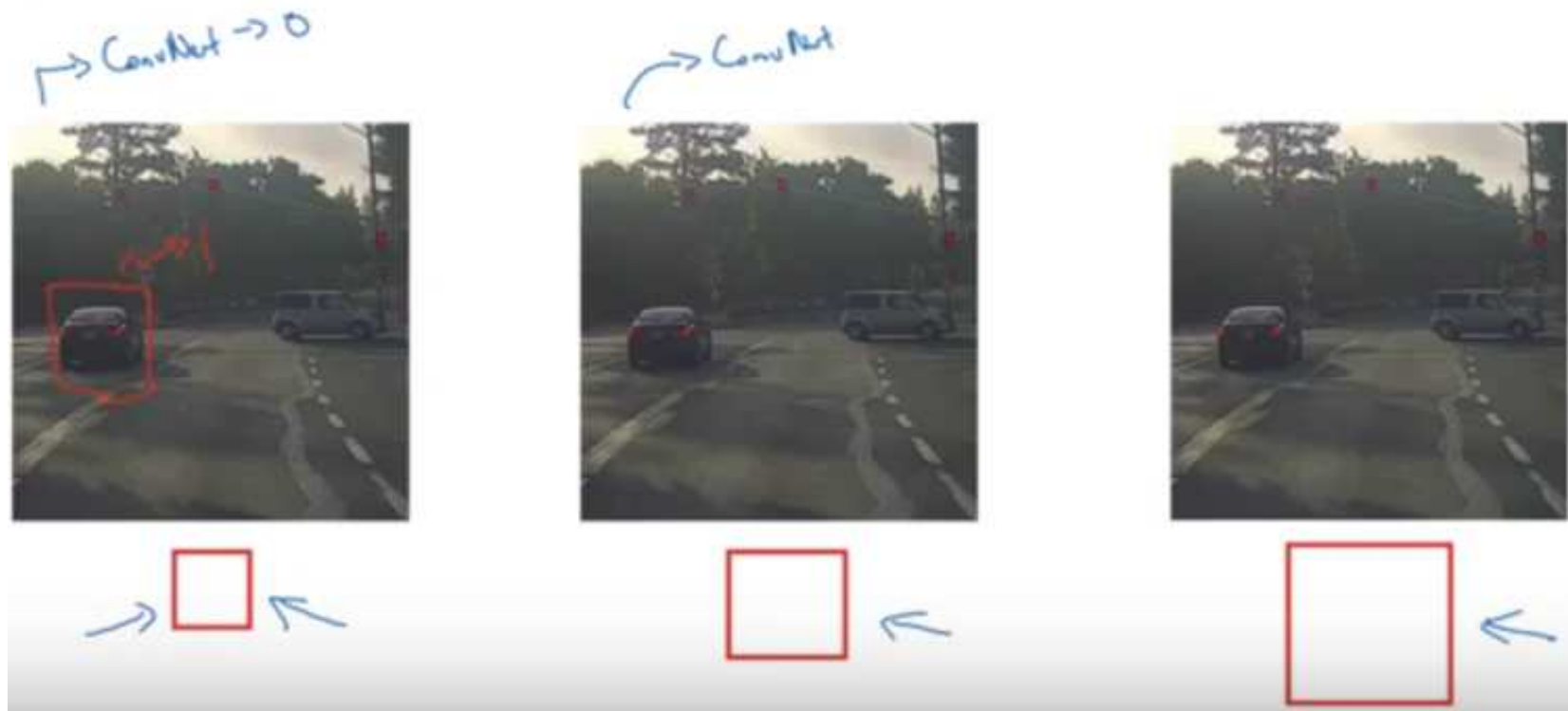
- Car detection 예제
- Training set에 대한 정답 label을 작성
- x : training set
object와 배경 다 같이 있는 게 아니라 object의 명확성을 위해 crop한 이미지
object인 자동차가 이미지 중간에 위치하며 이미지 전체가 됨
- y : 0 or 1
학습한 CNN에 이미지를 넣으면 1 or 0 나옴

Sliding Window detection



- CNN으로 학습하면, Sliding window detection도 사용 가능하다.
- Sliding Window
window size 선택 (빨간색 네모 크기)
 - > 이미지 upper-left부터 window 크기 만큼의 이미지를 학습된 Conv에 feed
- 첫번째 그림의 좌상단 window에 자동차 object 없다 -> conv는 background(0)
- sliding window의 방향은 좌상단에서 오른쪽 & 아래쪽 으로 움직임
(최종 목적지는 그림처럼 우하단)
window size만큼의 이미지는 계속 conv에 feed

Sliding Window detection



window 크기를 키워가며 반복함

- object 있을 때 CNN은 1이 되고, 그 위치에서 bounding box를 그림

sliding window 알고리즘 단점

- computational cost 증가

다양한 window size를 CNN에 feed하여 많은 연산 필요

- 연산량 줄이려고 stride 크게 해서 window를 한 번에 여러 칸 옮길 순 있지만 정밀도가 낮아짐

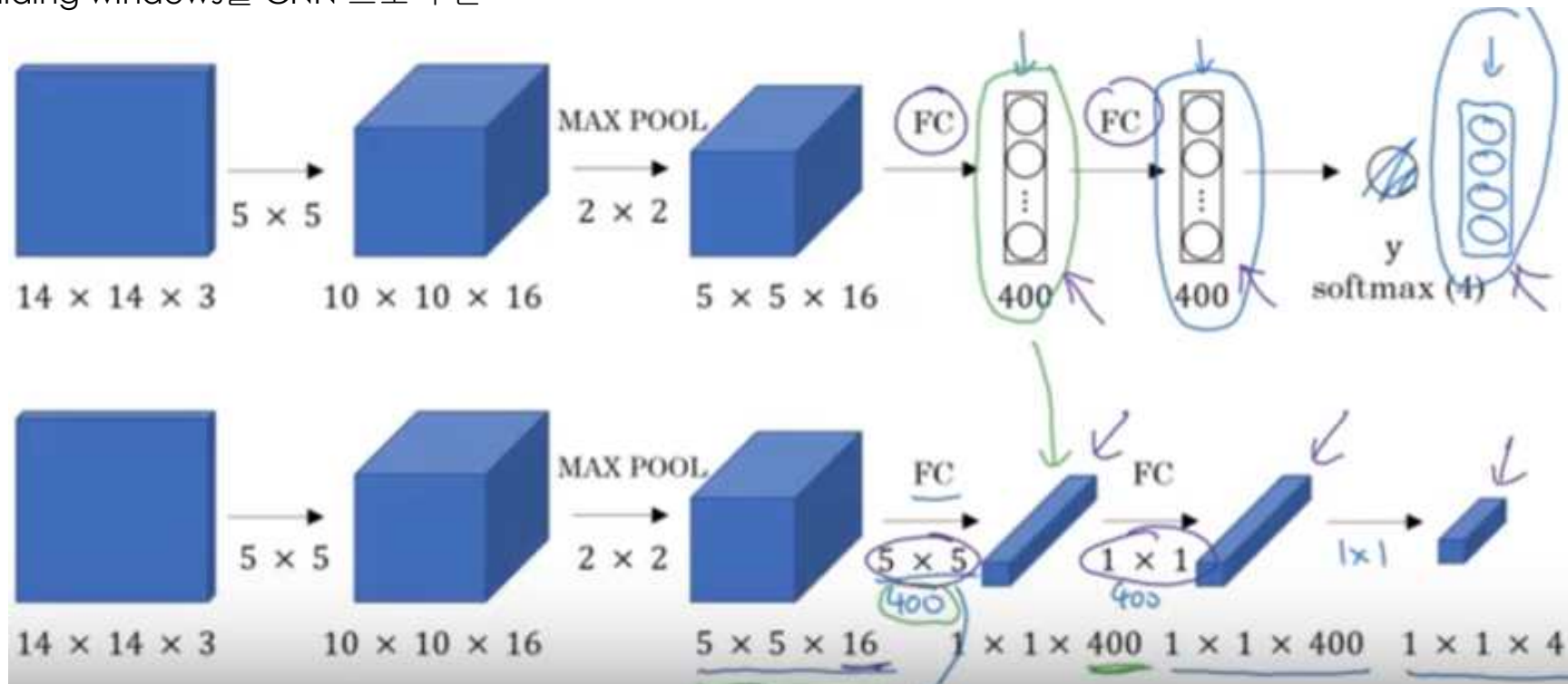
sliding window object detection을 CNN 사용으로 해결

CNN Implementation of Sliding Windows

Turning FC layer into convolutional layer

Sliding windows를 CNN 으로 구현

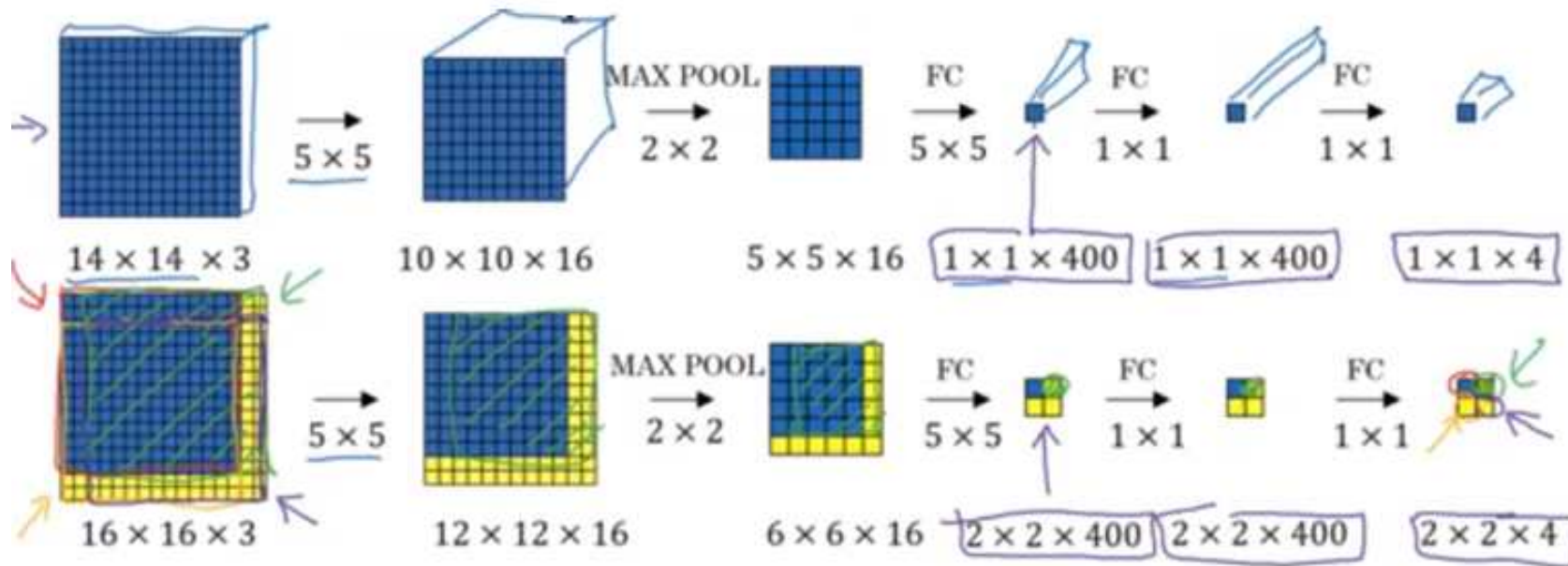
4개 output (3개 class + background)
(Class는 1 Pedestrian, 2 Car, 3 Motorcycle)



- 뉴럴넷의 열벡터 FC를 매트릭스로 구현.
-> $5 \times 5 \times 16$ 필터 400개를 통해 $1 \times 1 \times 400$ 매트릭스를 만든다.
- 위 : FC를 통해 output layer 만들
아래 : convolution 연산으로 output layer 만들
-> output이 벡터에서 volume 형태가 됨
-> sliding window의 convolution 실행이 가능해짐!

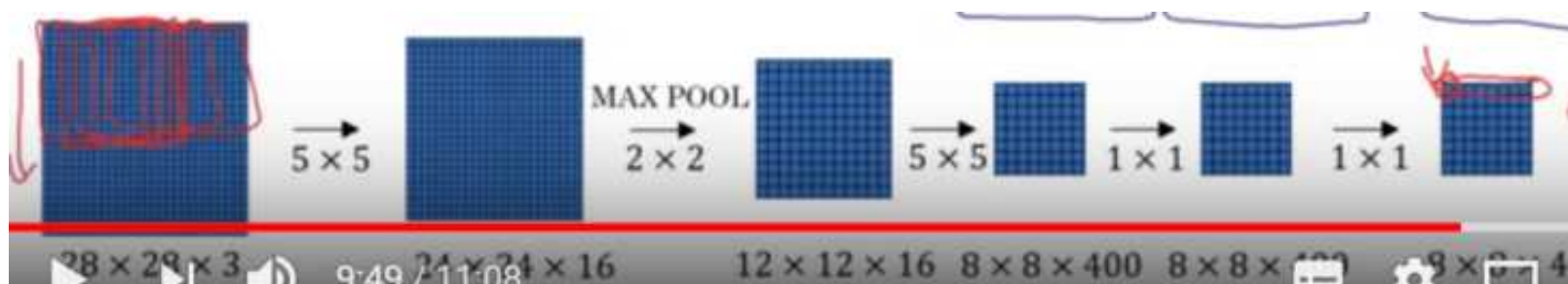
CNN Implementation of Sliding Windows

- training시 input이 14x14x3 이미지로 1x1x4 volume output 얻고 softmax로 예측
-> CNN 학습!



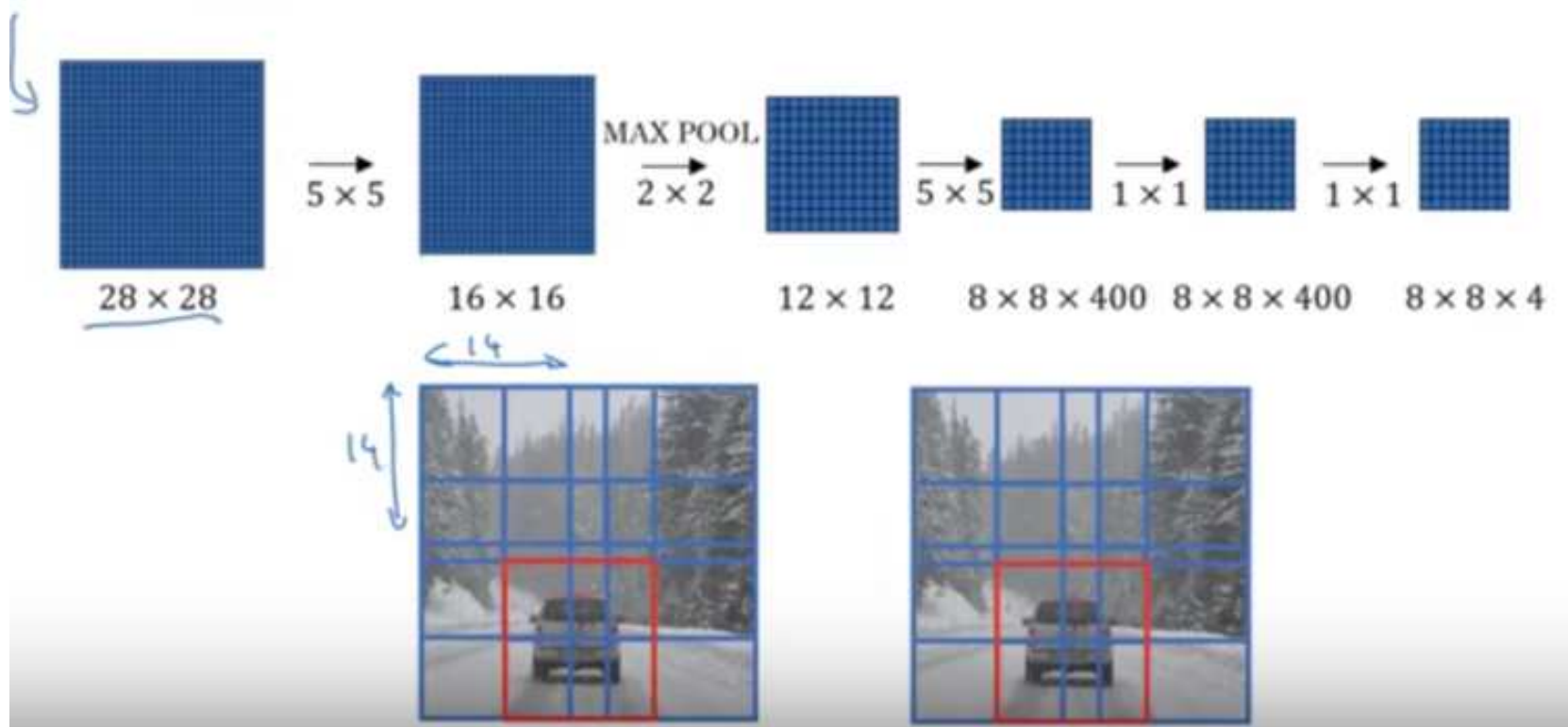
- 학습한 CNN을 Test image에 적용해보자
- test input 이미지 : $16 \times 16 \times 3$
test의 $14 \times 14 \times 3$ 를 똑같이 CNN 적용 -> output : $2 \times 2 \times 4$
- 여기서 2×2 는 CNN으로 부터 얻은 거다
- $16 \times 16 \times 3$ input 이미지에서 4번의 sliding window를 가져 4개의 label을 얻음.

2칸씩 이동
14x14
4개의 윈도우 효과



2칸씩 이동
14x14
64개의 윈도우 효과

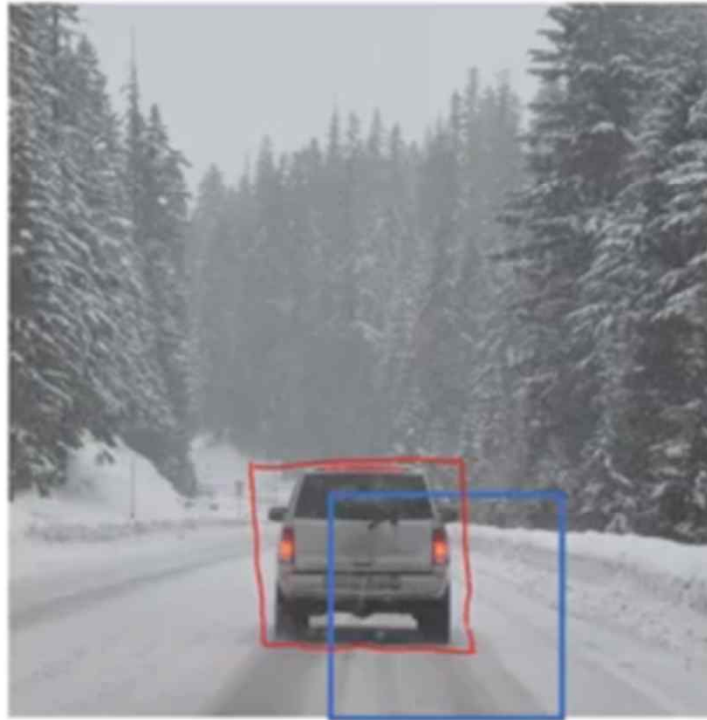
CNN Implementation of Sliding Windows



- Car Object Detection
test input 이미지 : $28 \times 28 \times 3$
window 사이즈 : $14 \times 14 \times 3$
좌상단 부터 sliding window를 통해 이미지 전체로 진행 (파란색 네모로 움직임)
- 이 때 Window에 맞는 Object가 있다면(빨간색 네모)

Bounding Box Predictions

Output accurate bounding boxes

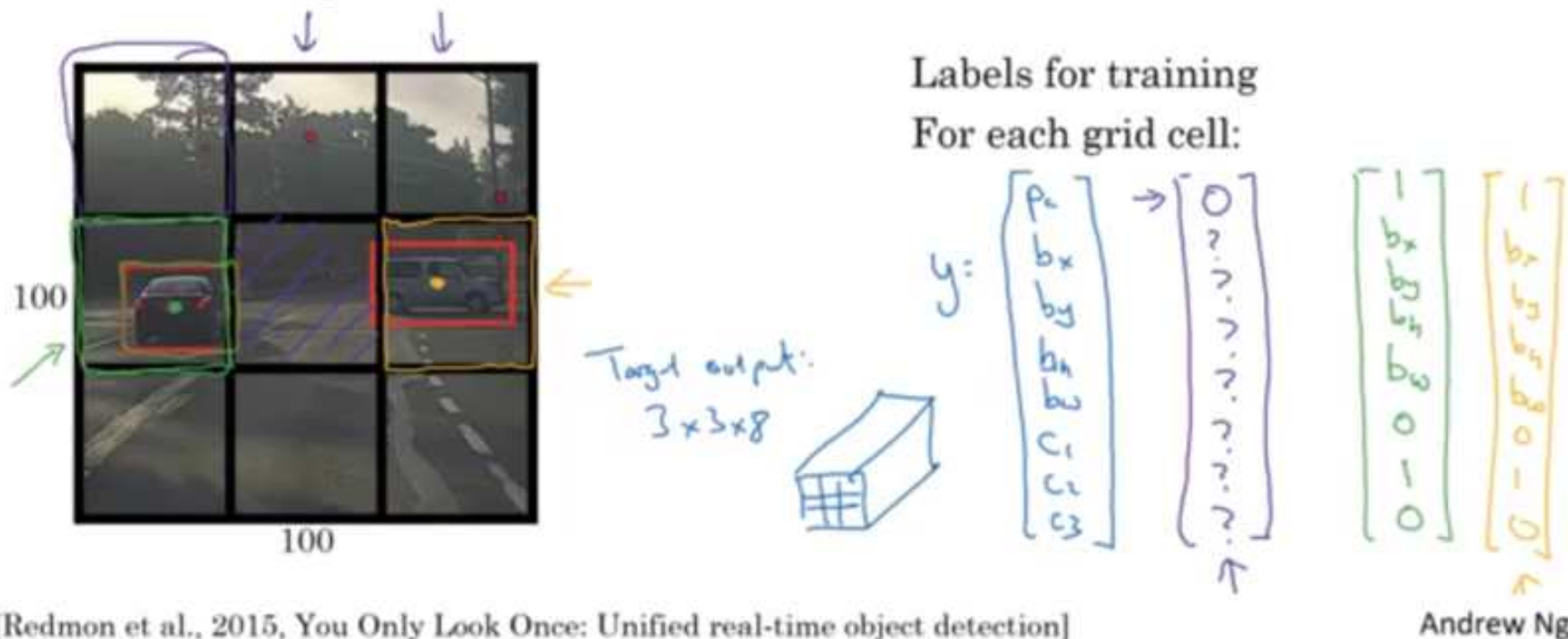


Andrew Ng

- input이미지에서 car를 detection 하기 위해, 파란색 bounding box를 좌상단부터 우하단까지 Sliding
- 빨간색 박스와 정확하게 만나는 파란색 박스는 없으나 현재 위치(파란색 박스) bounding box가 object와 가장 많이 겹침
- 또한, 실제 object에 필요한 박스는 정사각형이 아닐 때가 많다
물체들의 가로세로 비율이 다른 경우가 더 많음

Bounding Box Predictions

YOLO algorithm

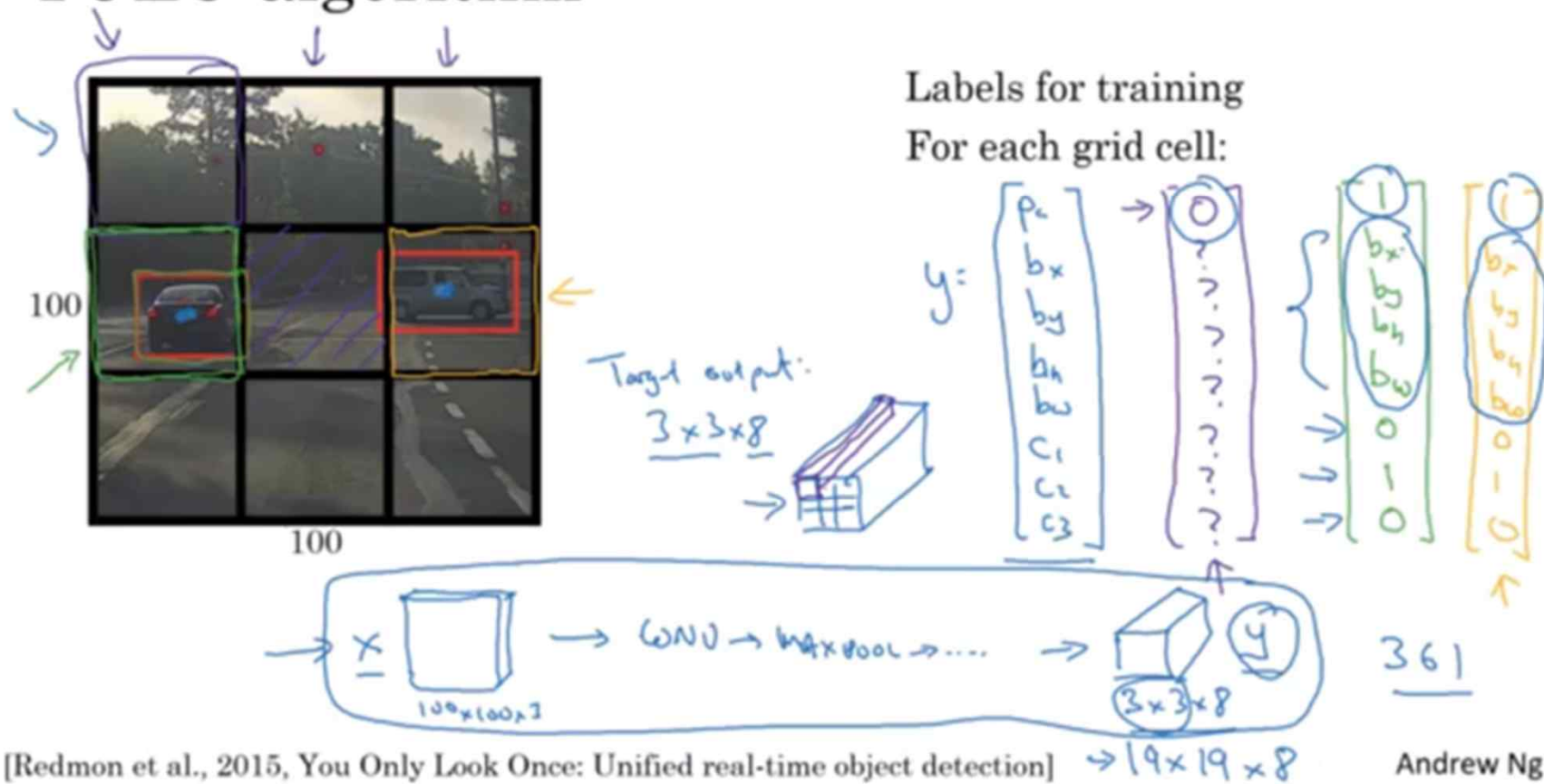


[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

- input 이미지 :100x100
- grid : 3x3 (YOLO는 7x7 grid 사용)
- 정답 label : p_c , bounding box 좌표값, class 값
 - > grid의 벡터값 참조(보라색, 초록색, 노란색)
- 보라색 : object가 없음 -> $p_c = 0$, 나머지 "?"(의미없는 값)
- 초록색, 노란색 : grid 내 중앙점 표시(b_x, b_y 좌표값)
- 3x3 grid YOLO : $3 \times 3 \times 8$ output volume (grid당 8개 원소 벡터)

Bounding Box Predictions

YOLO algorithm



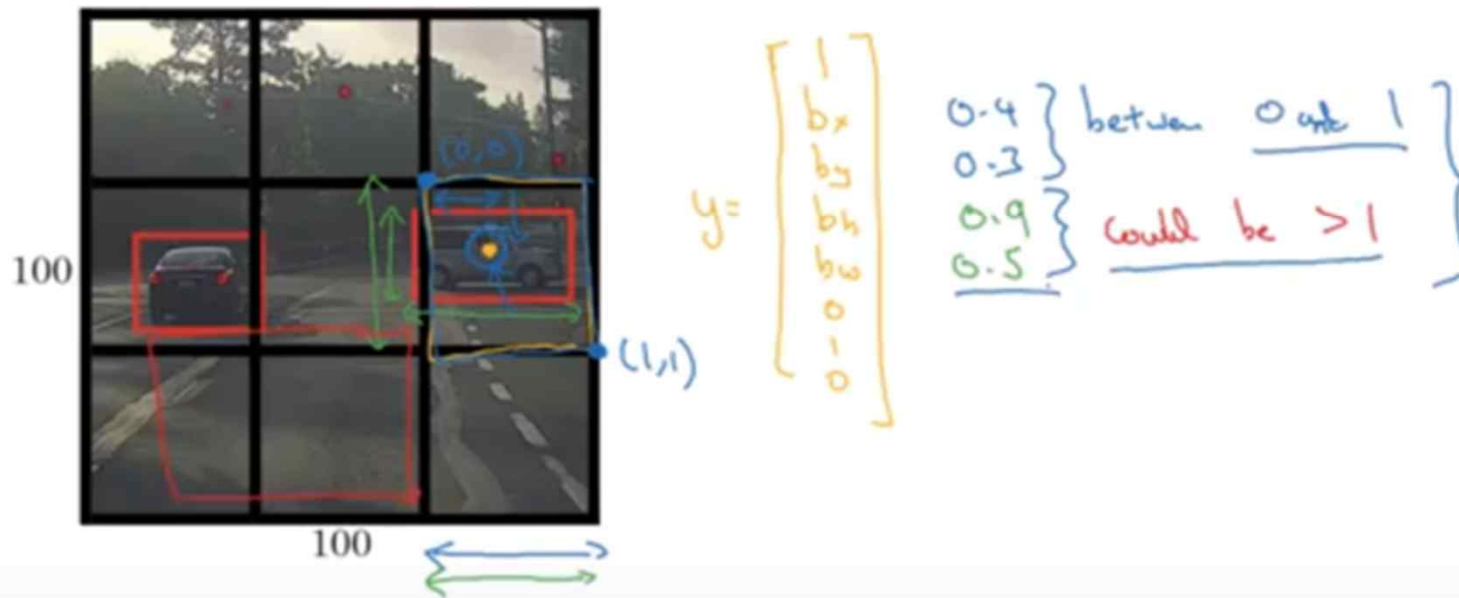
[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Andrew Ng

- input을 Conv로 처리 후 output으로 3x3x8 네트워크 구성해 training
실제로는 19x19x8 처럼 만드는 것이 더 정교함
- 한 grid에 여러 object 가 있는 경우
- grid 수가 많아 질 수록 한 grid에서 여러 object가 detection 될 확률은 줄어든다.
- YOLO알고리즘 : 각 window 별 sliding 하면서 CNN을 적용하는 것이 아니다.
input 하나를 conv 적용하고 output을 grid로 나누어 한 번에 object에 대한 classification과 localization을 하는 것
-> 처리속도 빨라짐 -> 실시간 object 적용가능

Bounding Box Predictions

Specify the bounding boxes



- Bounding box : grid내 중앙점(노란점)을 기준으로 height와 width로 그려진다
- 각 grid의 좌상단(0,0), 우하단(1,1)로 정의
bx/by : 0~1 범위
bh/bw : 1보다 커질 수 있음 -> object가 커서 여러 grid에 걸쳐 있을 수도 있음

Object Detection: Intersection Over Union

Evaluation object localization



Intersection over Union (IoU)

$$= \frac{\text{Size of } \text{Intersection}}{\text{Size of } \text{Union}}$$

"Correct" if $\text{IoU} \geq 0.5$ ←

0.6 ←

More generally, IoU is a measure of the overlap between two bounding boxes.

- 빨간색 박스 정답 / 보라색 박스가 예측값
- Localization이 잘 되지 않음
- IOU(Intersection Over Union)
 - 초록색 영역 : 빨간색과 보라색의 Union(합집합)
 - 노란색 영역 : 빨간색과 보라색의 Intersection(교집합)
- $\text{IOU} = \text{노란색 size} / \text{초록색 size} = \text{Intersection} / \text{Union}$
 - $\text{IOU} > 0.5$: 예측이 Correct 하다고 여김
 - 정확한 판정을 위해 0.5보다 더 큰 값을 설정하면 됨(0.6 같은)
 - IOU는 예측값과 Label의 Bounding Box가 얼마나 유사한지 확인하는 기준

Object Detection: Non-max Suppression

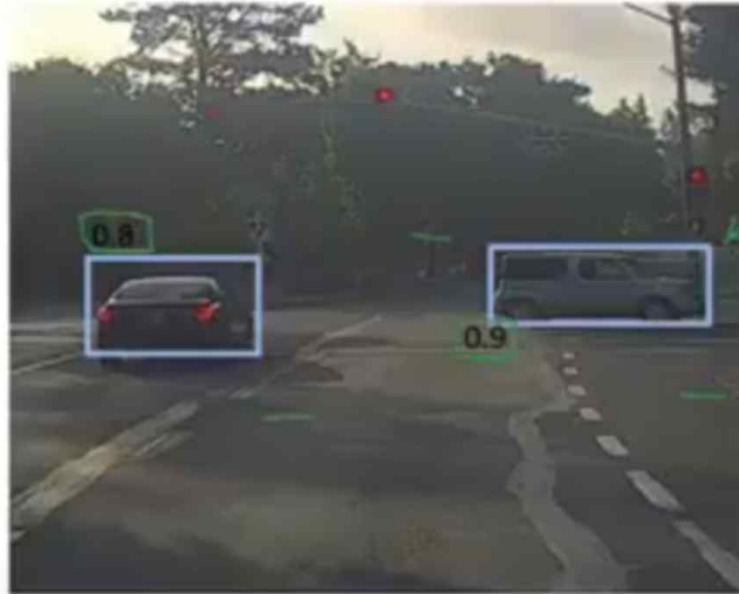
- Object Detection의 문제점 : 동일한 object를 여러번 detection 함
- 하나의 object 한 번 detection



- 이상적인 case : 19x19 grid에서, grid 내 하나의 중앙점이 있는 것
- 예의 case : 초록색, 노란색 처럼 중앙점이 여러 개인 경우 하나 선택
 - > Non-max suppression 알고리즘을 사용
- YOLO 알고리즘을 사용 -> 각 grid들에서 detection 하여 여러 개가 detection 됨
- class가 car인 grid에서 PC와 bounding box(파란색 네모)가 그려진다.

Object Detection: Non-max Suppression

Non-max suppression example



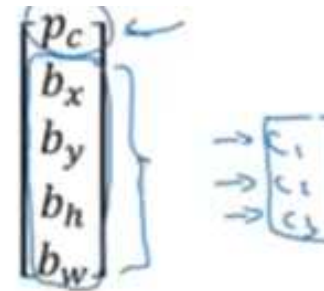
- 선택되지 않은 박스들은 제거해준다. (하늘색 박스만 남음)

Object Detection: Non-max Suppression



19x 19

Each output prediction is:



Discard all boxes with $p_c \leq 0.6$

→ While there are any remaining boxes:

- Pick the box with the largest p_c
Output that as a prediction.
- Discard any remaining box with $\text{IoU} \geq 0.5$ with the box output
in the previous step

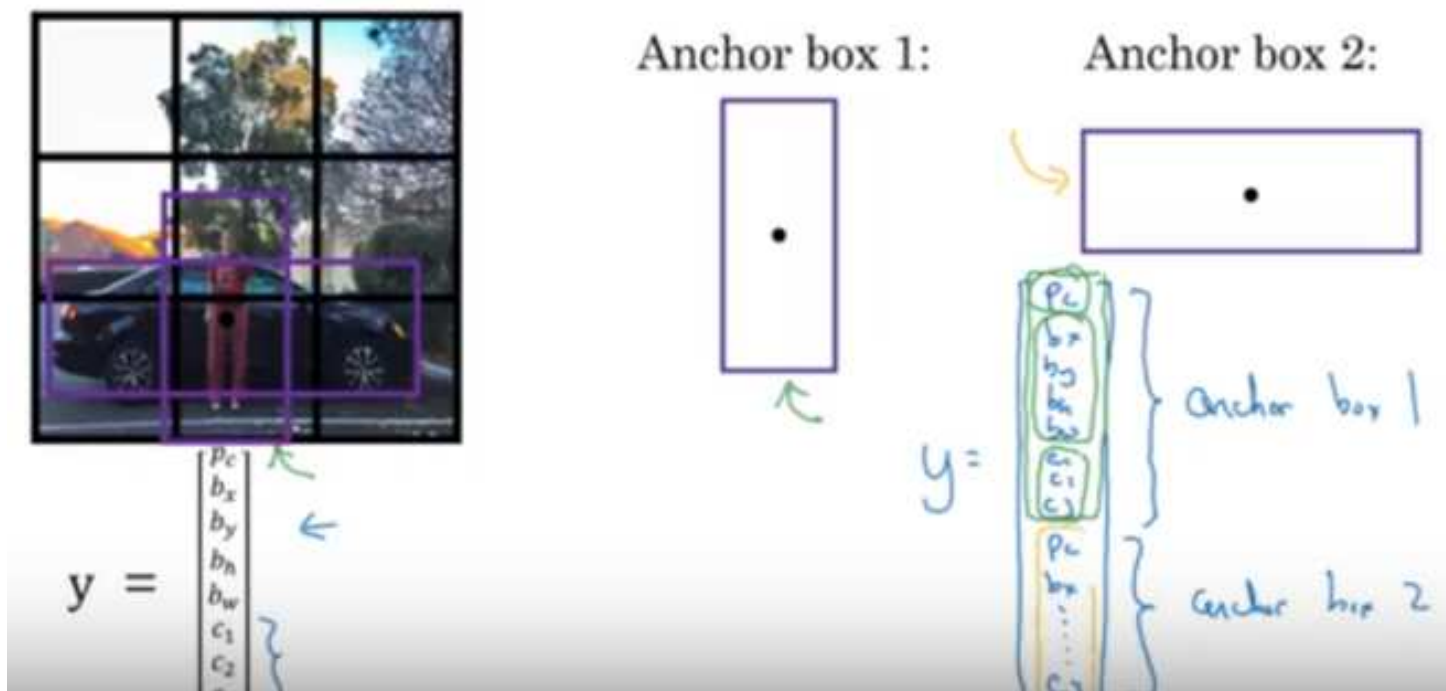
Andrew Ng

- 1) CNN을 이용해 Output 출력 [p_c b_x b_y b_h b_w]
- 2) $p_c \leq 0.6$ (threshold)인 grid의 박스 버림
- 3) 남아있는 박스가 있으면,
 - 3-1) 가장 큰 p_c output의 박스 선택
 - 3-2) 3-1의 output 박스와 $\text{IoU} \geq 0.5$ 박스 버림
- > 각 object당 1개의 bounding box만 남음

Anchor Boxes

Overlapping objects

- 각 grid에서 한 개 object만 detection
- 각 grid에서 여러개 object detection -> anchor box 사용



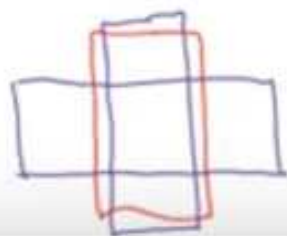
- 이미지에 object 두 개 있음 - Pedestrian, car
- 두 object가 겹쳐있고 중앙점 역시 한 grid에 있다. (검정색 점)
- 기존 벡터(y)를 이용하면 한 grid 내 한 개의 object만 검출
-> Anchor box로 여러 개 object를 검출
- Anchor box 1, Anchor box 2 처럼 height, width가 각각 큰 사각형을 두고,
 y 에 2개 object를 모두 저장한다.
 $p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3$ 의 set가 anchor box가 되고, 이게 두 set 필요

Anchor Box algorithm

Previously:

Each object in training image is assigned to grid cell that contains that object's midpoint.

Output y:
 $3 \times 2 \times 8$



With two anchor boxes:

Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

Output y:
 $3 \times 3 \times 16$
 $3 \times 3 \times 2 \times 8$

(grid cell, anchor box)

- 이전 : training 이미지 각 Object는 Object의 중앙점이 포함된 grid에 할당됨
- 2개 anchor box : training 이미지인 각 object는 중앙점이 포함된 grid 중 anchor box와 가장 큰 IoU를 가지는 grid에 할당됨
- 빨간색 박스가 object일 때, height가 더 긴 anchor box가 1, width가 더 긴 게 2이면 1의 IoU가 더 커서 1이 선택-
- 이전의 output은 $3 \times 3 \times 8$ 이지만,
- anchor box 두 개 일 때 output은 $3 \times 3 \times 16$ ($3 \times 3 \times 8 \times 2$)가 됨

Anchor Box example

Anchor box example



Anchor box 1: Anchor box 2:



$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Handwritten annotations for the vector y :

- For the first 16 elements (two sets of 8):
 - Yellow: b_x, b_y, b_h, b_w for both sets.
 - Green: c_1, c_2, c_3 for both sets.
 - Blue: p_c for both sets.
- For the second set of 8 elements (Anchor box 2):
 - Green: $b_x, b_y, b_h, b_w, c_1, c_2, c_3$.
 - Blue: p_c .
- For the first set of 8 elements (Anchor box 1):
 - Blue: p_c .
 - Green: $b_x, b_y, b_h, b_w, c_1, c_2, c_3$.

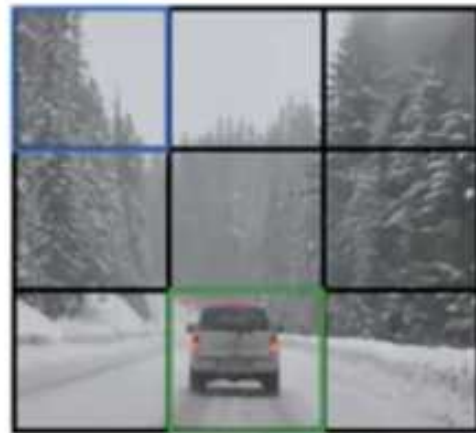
Additional handwritten notes:

- On the right, a bracket groups the last 8 elements and is labeled "anchor box 2".
- On the left, a bracket groups the first 8 elements and is labeled "anchor box 1".
- At the top right, "car only?" is written above the first 8 elements.

- Pedestrian은 Anchor box 1과 더 가깝고, car는 2와 더 가깝음
- 중앙점이 있는(검정색 점) grid cell의 label y 를 보면 anchor box 1,2의 detection
- car만 있으면, anchor box1의 $P_c = 0$, 나머지 = "?" (돈케어)
anchor box2의 $P_c = 1$, 나머지 값들 label에 저장

YOLO Algorithm

Training



- 1 - pedestrian
- 2 - car
- 3 - motorcycle

$y =$

$$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

y is $3 \times 3 \times 2 \times 8$

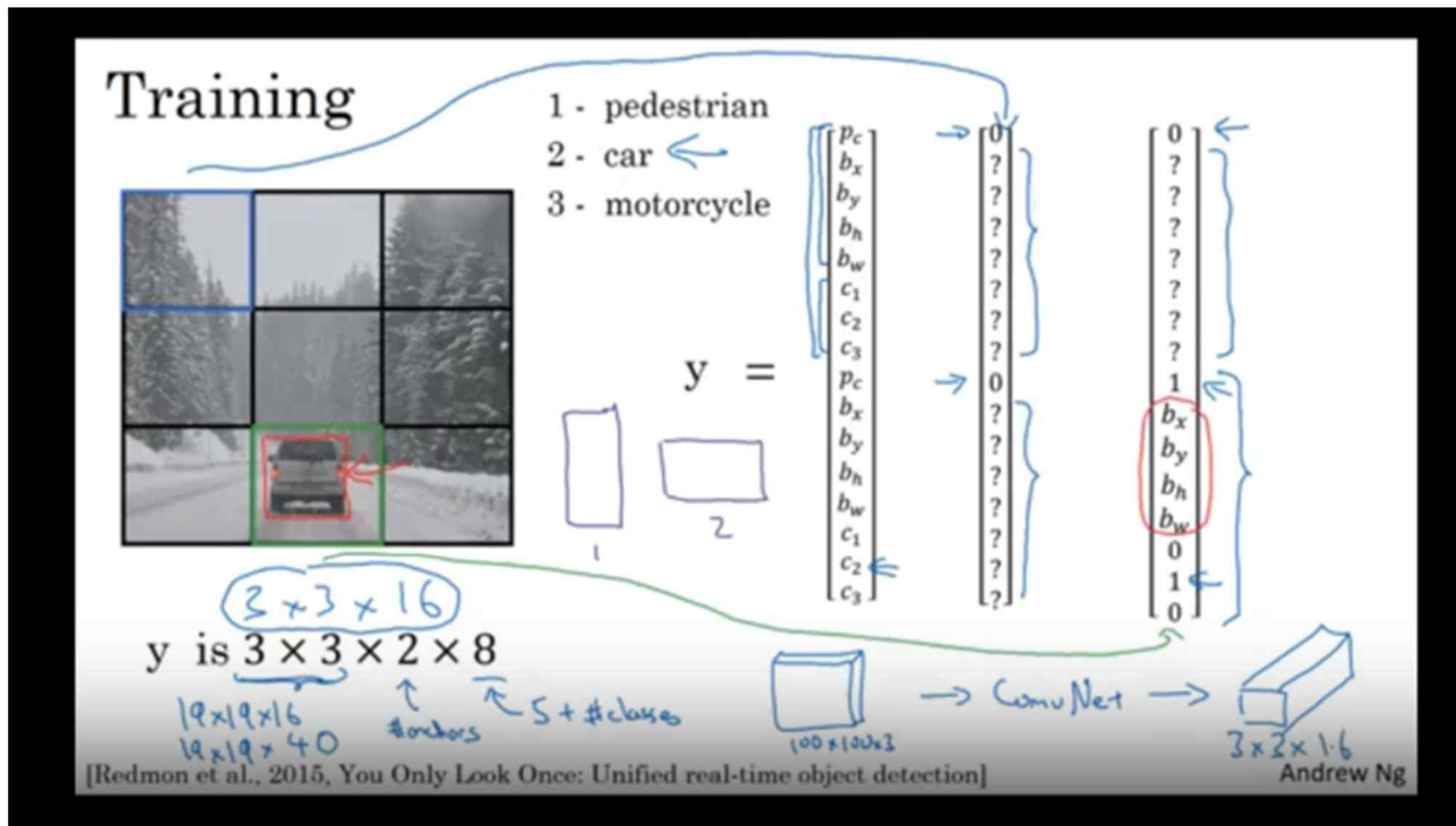
\uparrow #anchors \uparrow 5 + #classes

[Redmon et al., 2015, You Only Look Once: Unified real-time object detection]

Andrew Ng

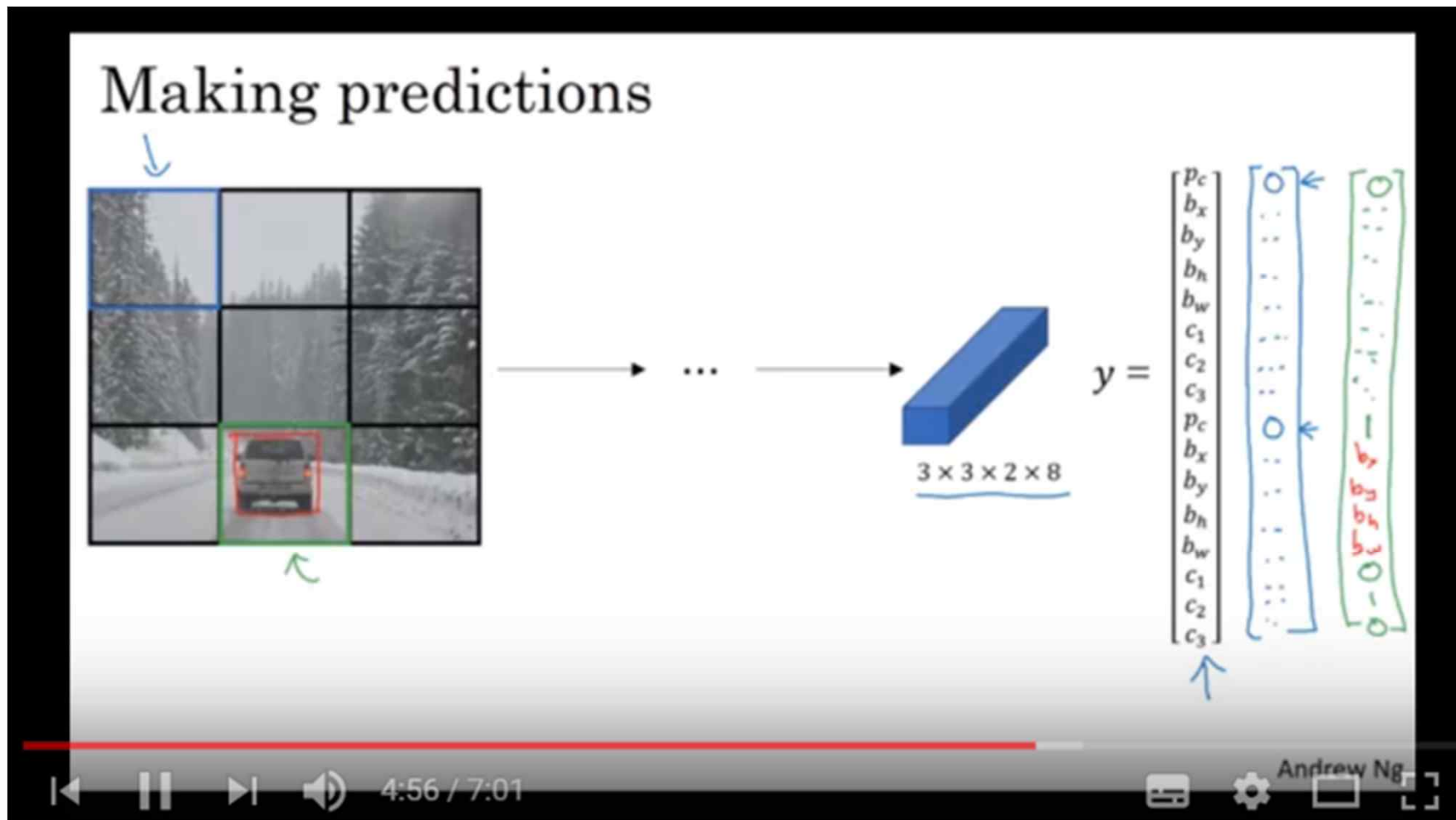
- 3x3 grid, 2개의 anchor box, 3개의 Class
- y : 3x3(grid), 2(anchor box 수), 8(5(p_c , bounding box 좌표($b_x/b_y/b_h/b_w$)) + class수) -> 3x3x16 이라고 써도 됨
- 좌상단(파란색네모) object는 class에 해당되지 않는다.
 - > p_c 들이 0이 되고 label 값들은 "?"(don't care)
- 초록색네모(자동차있음) grid cell을 보면
 - 자동차를 detect하는 anchor box가 있음

YOLO Algorithm



- anchor box(보란색네모) 1(height 큼), 2(width가 큼)가 있으면
자동차는 width가 큰 것이 IoU가 더 커서 2를 선택할 가능성이 높다.
그럼 anchor box1이 돈케어 된다.
- 이 output 벡터를 이용해 detection 하면
100x100x3 input 이미지가 Conv 통과한 후 output 벡터가 3x3x16이 된다.
- 실제로는 좀 더 세밀한 19x19 grid로 detection 할 거고,
19x19x16(anchor box 2개), 19x19x40(anchor box 5개) 등의 output 벡터 만들 수 있음.

YOLO Algorithm



- 이렇게 네트워크를 구성하고 output y를 예측 가능하다.

Outputting the non-max suppressed output



- For each grid cell, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

- output을 바로 이용하기 전에 Non-max suppression이용해 중복 detection 제거
- Anchor box 2개 있다고 가정
 - > 각 grid 마다 2개의 bounding box들이 나온다.
- Non-max suppression 전에 low probability 예측값을 가진 것들을 제거한다.
(car, pedestrian 없는 것들)
- 각 class에 Non-max Suppression 적용 -> 1 object, 1 box!