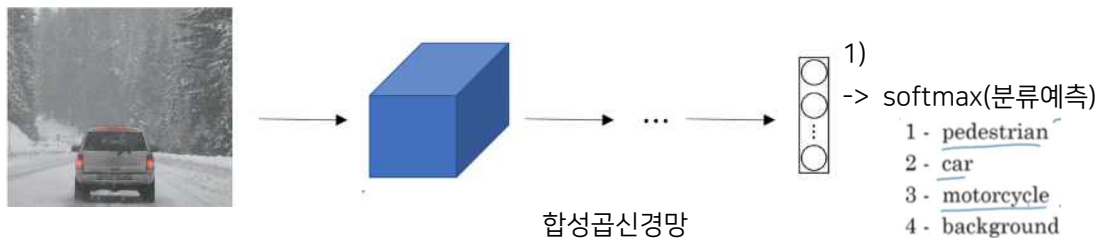


### 3주차 Object Localization

#### 3-1 Object Localization

- Classification: 하나의 물체를 분류하는 것
- Localization: 하나의 물체가 사진의 어느 쪽에 위치하는지 설명하고, 해당 물체 주위에 경계 상자를 그리는 것
- Detection: 여러 개의 물체(이들이 모두 다른 종류일 수 있음)를 감지하고 위치를 알아내고 경계 상자도 그리는 것
- Classification with Localization



2) 이미지에서 자동차의 위치를 알고 싶다면?

- > 몇 개의 출력 유닛을 더 가지도록 신경망 변형 (위치를 알아내고자 하는 bounding box에 관한 출력): 4개의 추가적인 출력을 가지게 됨 ( $b_x$ ,  $b_y$ ,  $b_h$ ,  $b_w$ )
- 이미지의 왼쪽상단의 좌표를 (0,0), 우측하단의 좌표를 (1,1)으로 표시하자
- 감지된 물체의 bounding box의 중앙 좌표( $b_x$ ,  $b_y$ ), 폭( $b_w$ ), 높이( $b_h$ )에 의해서 위치 탐지

지도학습작업에서 어떻게 target label  $y$ 를 정의할 것인가?

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

- $p_c$ : 이미지 내 물체 존재 여부 (ex. 0 or 1,)
- $b_x$ ,  $b_y$ ,  $b_h$ ,  $b_w$ : 감지된 물체의 경계상자. 중앙 좌표( $b_x$ ,  $b_y$ ), 폭( $b_w$ ), 높이( $b_h$ )
- $c_1$ ,  $c_2$ ,  $c_3$ , ...: 각 클래스일 확률(해당 클래스일 경우 1, 아닌 나머지는 0)이미지가 단 하나의 물체만 포함한다고 가정되어있음)

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

- (좌) 이미지 내 물체가 있는 경우,  $p_c$ 는 1이고,  $b_x$ ,  $b_y$ ,  $b_h$ ,  $b_w$ 는 경계 상자를 표현하고, 객체와 동일한 클래스 값만 1을 가짐(자동차일 경우,  $c_2$ 만 1, 나머지는 0)
- 손실함수로 제곱오차를 사용하여, 8개의 요소에 대한 예측값과 실제값에 대한 차이를 나타냄
- (우) 이미지 내 물체가 없는 경우,  $p_c$ 는 0이 되고, 나머지 요소들은 무관함 (don't care; ?)이 됨: 이미지에 물체가 없으므로 신경망의 출력으로 경계상자와  $c_1$ ,  $c_2$ ,  $c_3$ 를 고려할 필요가 없음
- 손실함수로 제곱오차를 사용하면, 2번째부터 8번째 요소가 무관함이므로, 신경망이 물체의 유무를 얼마나 잘 예측하는지( $p_c$ )에 대해서만 신경쓰면 됨

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

### 3-2 Landmark Detection 특징 점 검출

- 특징 점들을 설정하고 이 특징 점 좌표를 포함하는 레이블 훈련세트를 만들면 신경망이 어디에 특징 점들이 있는지 말할 수 있도록 할 수 있음
- ex) 얼굴인식. 입의 좌표를 출력으로 가지면 입모양을 통해 웃는 or 찡그린 상태 인식 가능
- ex) 특징점 검출을 이용해, 스냅샷에서 증강 현실 필터를 통해 얼굴 위에 왕관을 그려줌
- ex) 자세감지. 몇 개의 중요한 위치(가슴 중앙, 왼쪽 어깨, 왼쪽 팔꿈치, 손목 등)를 정함 -> 신경망은 사람의 자세에서 각 중요 위치에 표시를 함 -> 신경망의 출력으로 이 모든 지점이 표시되어 나오고, 사람의 자세를 출력 값으로 얻음

=> Landmark Detection: 여러 개의 출력 유닛을 추가해서 인식하고자 하는 특징점들의 각 좌표를 출력함.


- 이를 확실하게 하기 위해서 특징점 레이블 순서는 다른 이미지에 대해서 항상 동일해야 함.
- ex) 특징점1은 항상 왼쪽 눈의 가장자리이어야 함, 특징점2는 ...

### 3-3 Object Detection

- 물체 인식 알고리즘을 설계하는 것을 알아보자 - 합성곱 신경망 이용
- 물체인식을 구현하기 위해서는 sliding windows 검출 알고리즘 사용
- ex) 자동차 인식 알고리즘을 만들어보자

Training set:

x



y

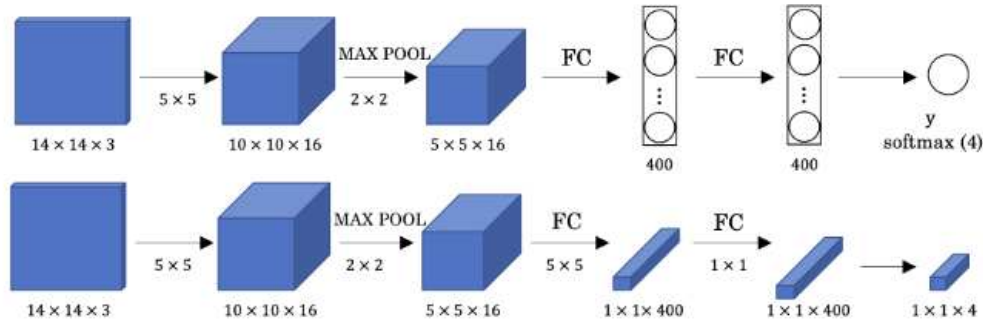
- 가장 먼저 레이블 훈련세트(x,y)를 만들 -> x: 자동차에 근접해서 잘려진 샘플
- 1 - 이 레이블 훈련세트를 이용해 합성곱 신경망을 훈련할 수 있고, 훈련 후 이를 sliding windows detection에 사용할 수 있음
- 1 - 테스트 이미지에서, 특정 윈도우 크기 설정-> 합성곱 신경망에 작은 사각형 영역을 입력-> 합성곱 신경망이 예측값 계산-> 자동차를 포함하지 않는다는 결과 도출(0)-> 상자를 옆으로 옮기면서 이미지

의 모든 위치를 슬라이드 할 때까지 과정 반복-> 각 위치에 대해서 0,1로 분류  
-> 슬라이딩 윈도우 한 번 완료 후, 더 큰 윈도우 크기를 사용해 과정 반복-> 더 큰 윈도우 크기 또 반복-> 수행한 결과로 이미지 어딘가에 자동차가 있다면 그에 해당되는 윈도우도 있음-> 합성곱 신경망에 이 윈도우 통과해서 1의 출력 값 = 자동차 인식

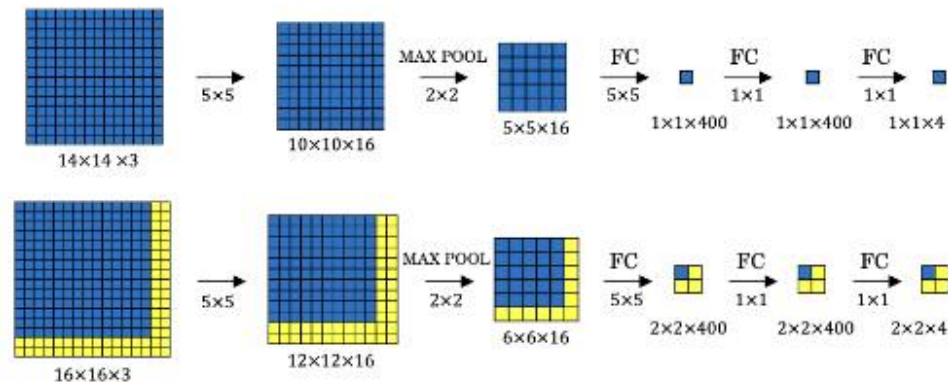
- sliding windows detection: 전체 이미지에 대해서 window를 일정 간격으로 sliding 하면서, 각 위치에 객체가 존재하는지 합성곱 신경망 계산을 통해 감지
- 단점: 계산비용; 이미지의 수많은 영역을 모두 잘라내어 합성곱 신경망을 통해 계산해야하기 때문. 또한 계산 비용을 줄이기 위해 sliding windows의 크기를 늘리면 성능 저하

### 3-4 Convolutional Implementation Sliding Windows

sliding window detection 알고리즘은 계산비용이 크고 매우 느리다는 단점이 있다. 이 알고리즘을 어떻게 합성곱을 이용해 구현하는지 알아보자 - FC layer를 convolutional layer로 바꿈

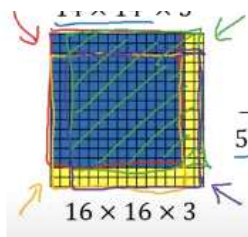


위의 합성곱 신경망에서 초기층은 동일하게 그리고, FC layer을 이용해 합성곱층을 구현함  
400개의 5x5(x16) filter로 연결-> 400개의 1x1 filter->1x1 filter-> 소프트맥스 활성화는 1x1x4

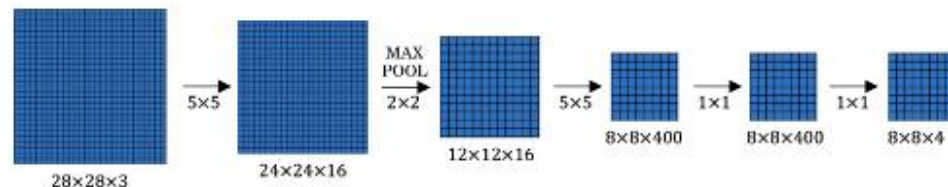


14x14x3 입력을 합성곱 신경망에 통과해 1x1x4 결과를 얻음

이 때, 테스트세트 이미지는 16x16x3임(노란색 가장자리 추가)-> 합성곱신경망을 이용해 항상 동일한 변수에 대해 계산을 함 -> ... -> 2x2x4 결과를 얻음: 여기서 1x1x4 파란색 부분은 왼쪽 상단의 14x14 이미지에 대한 결과임



네 가지 입력 이미지에 대해서 독립적인 계산을 하는 것이 아니라 4가지의 경우를 하나의 계산으로 결합해서 14x14 이미지의 공통되는 영역에 대한 계산을 공유하는 것임



28x28x3 이미지에 sliding window를 수행해보자! 동일한 방법으로 정방향 전파를 수행해 8x8x4 출력을 얻음. 그리고 이것을 입력이미지의 왼쪽 상단 14x14 영역에 대해 슬라이딩 윈도우를 수행->

이 부분(출력의 왼쪽 상단)을 출력함-> 윈도우를 이동시키기 위해 두 픽셀 간격을 사용하면 출력의 첫 번째 행에 대해서 8개의 위치를 가짐-> 아래까지 모두 진행해 8x8x4출력 얻음

3-3 슬라이딩 윈도우를 구현할 때는 상자를 옆으로 옮기면서 이미지에서 자동차를 인식할 수 있을 때까지 과정을 반복했었음 -> 하지만 이를 순차적으로 하는 대신 3-4에서 설명한 방식의 합성곱 신경망을 이용하면 동시에 모든 예측값을 계산할 수 있음 : 큰 합성곱 신경망에 한 번만 통과시켜 자동차의 위치를 인식할 수 있음

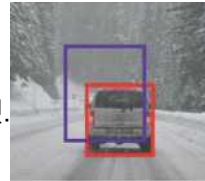
- 단점: 경계 상자의 위치가 정확하지 않을 수 있음

### 3-6 Intersection Over Union

- Intersection over Union(IoU; 합집합 위의 교집합): 물체 감지(Object Detection) 알고리즘의 평가와 다른 컴포넌트를 추가해 물체 감지 알고리즘을 향상시키는 것에 사용됨

- IoU를 구하는 방법: 두 경계 상자의 교집합 / 합집합

- IoU값이 0.5보다 크면 맞다고 판단 (IoU가 높을수록 올바르게 localization한 것. 둘이 완전히 겹치면  $\text{IoU} = 1$ )



### 3-7 Nonmax Suppression

- Non-max Suppression: 동일한 객체에 대해 중복되거나 겹치는 경계 상자를 제거하여 최적의 경계 상자를 선택하는 알고리즘(각 물체를 한 번씩만 감지하도록 보장함)

- Non-max Suppression algorithm 과정

1. 이미지 내에서 객체의 경계 상자와 그에 대한 예측 확률을 출력
2. 확률의 최댓값을 가진 경계 상자를 선택하고 해당 상자와 나머지 상자들의 IoU를 계산
3. IoU 값이 임계값(ex: 0.5)을 초과하면 중복 객체로 간주하고 제거
4. 나머지 상자들 중에서 다음 예측 확률이 높은 상자를 선택하고 위의 과정을 반복
5. 탐지하고자 하는 객체가 여러 개인 경우 여러 개에 대해서 독립적으로 시행

### 2-8 Anchor Boxes

Anchor Boxes: 한 격자 셀에 여러 개의 물체를 감지하고 싶거나 겹쳐 있는 객체를 모두 감지하고 싶을 때 사용하는 것으로, 다양한 크기와 비율의 객체를 효과적으로 탐지 가능



Anchor Boxes algorithm 과정

- 입력 이미지를 일정한 크기의 그리드로 분할
- 각 그리드 셀의 중심에 여러 개의 앵커 박스를 배치
- 각 앵커 박스마다 이와 같은 예측 값 출력
- 각 앵커 박스와 실제 객체의 경계상자 간 IoU 계산
- IoU가 가장 높은 앵커 박스를 해당 객체와 매칭

Anchor box 1:



Anchor box 2:



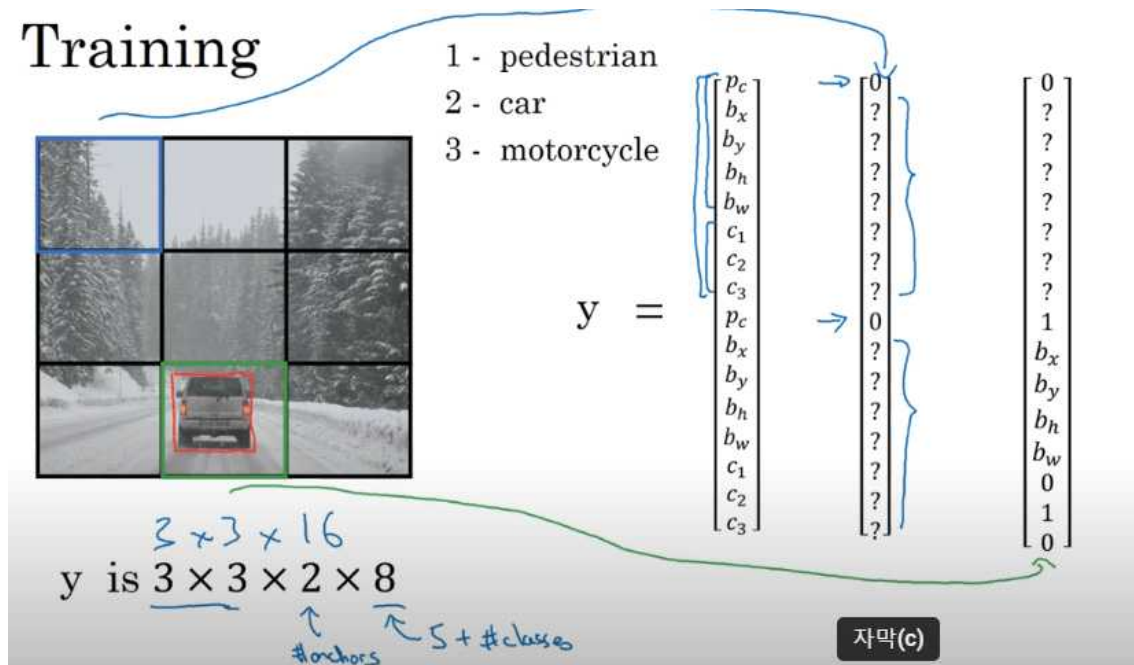
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ c_2 \\ p_c \\ b_x \\ \vdots \\ c_3 \end{bmatrix} \begin{cases} \text{Anchor box 1} \\ \text{Anchor box 2} \end{cases}$$

### 3-9 YOLO Algorithm

YOLO Algorithm: 이미지를 한 번에 분석해 객체의 위치와 클래스(종류)를 동시에 빠르게 예측하는 객체 탐지 알고리즘 - 속도가 빠르고 실시간 처리에 적합함

1)

## Training



- 훈련세트를 구성하기 위해서, 9개의 격자 셀에 대한 target vector인 Y는  $3 \times 3 \times 2 \times 8$  : 1)  $3 \times 3$  격자를 사용 2) 앵커의 수: 2 3)그의 차원인 8( $p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3$ )
- 두 개의 앵커박스를 사용하므로 9개의 그리드 셀 각각에 대해 2개의 bounding box가 도출됨
- 훈련세트가 자동차에 대해 가진 빨간 경계 상자는 세로보다 가로가 약간 더 김 -> 앵커박스2와 더 높은 IOU를 가지기 때문에 벡터의 아래 부분과 연관됨
- 이 과정을  $3 \times 3$ 의 격자 셀에 대해 적용해서 16차원 벡터를 구함=> 최종 결과의 부피:  $3 \times 3 \times 16$  (일반적으로 더 많은 격자가 사용됨 ex.  $19 \times 19 \times 16$ / 더 많은 앵커박스를 사용하면  $19 \times 19 \times 40$ )



요약)

신경망이 이미지를 입력 값으로 받아서 결과부피 도출

### 2) Making prediction

- 격자 셀에 아무 물체가 없다면, 신경망은  $p_c$ 에 0 도출, 신경망은 ?을 도출할 수 없기 때문에  $b_x, b_y, b_h, b_w, c_1, c_2, c_3$ 에 어떤 숫자들을 도출하지만, 이는 무시됨(노이즈)
- 격자 셀에 자동차가 있다면, 자동차에 대해 꽤 정확한 경계상자를 구체화하는 숫자 도출 = 신경망이 예측하는 방식
- 여기에 각 클래스(보행자, 자동차, 오토바이)에 대해 독립적으로 Non-Max Suppression을 적용해 중복된 bounding box를 제거하면서 최종 객체의 위치와 클래스를 검출

### 3-10 Region Proposals (optional)

- R-CNN(Region-CNN): 슬라이딩 윈도우를 모든 윈도우에 적용하지 않고 component classifier를 실행할 몇 개의 지역을 고르는 것 (명확하게 물체가 없는 많은 지역까지 분류하는(알고리즘이 실행)하는 단점 극복)
- 어떤 게 물체가 될 수 있는지 알아내기 위해 여러 파란색 영역에 실행해 보행자를 찾고, 하늘색 영역에 실행해 자동차를 찾음 (여러 개의 bounding box를 만들고 그 영역에 대해서만 분류기 실행)
- Fast R-CNN: R-CNN의 느린 처리 속도를 개선한 알고리즘으로 기본적인 R-CNN 알고리즘에 합성곱 슬라이딩 윈도우 구현을 더한 것. (이미지 전체를 한 번만 CNN을 적용하여 feature map을 추출하고 해당 feature map에서 region proposal을 추출한다.)
- Fast R-CNN은 Region proposal을 위한 clustering 단계가 여전히 느리다는 단점 존재
- Faster R-CNN: Fast R-CNN을 개선한 알고리즘으로 분할 알고리즘 대신 CNN을 사용해 이미지 특징 추출하고 객체가 있을 가능성이 높은 영역을 제안. YOLO보다는 느림

