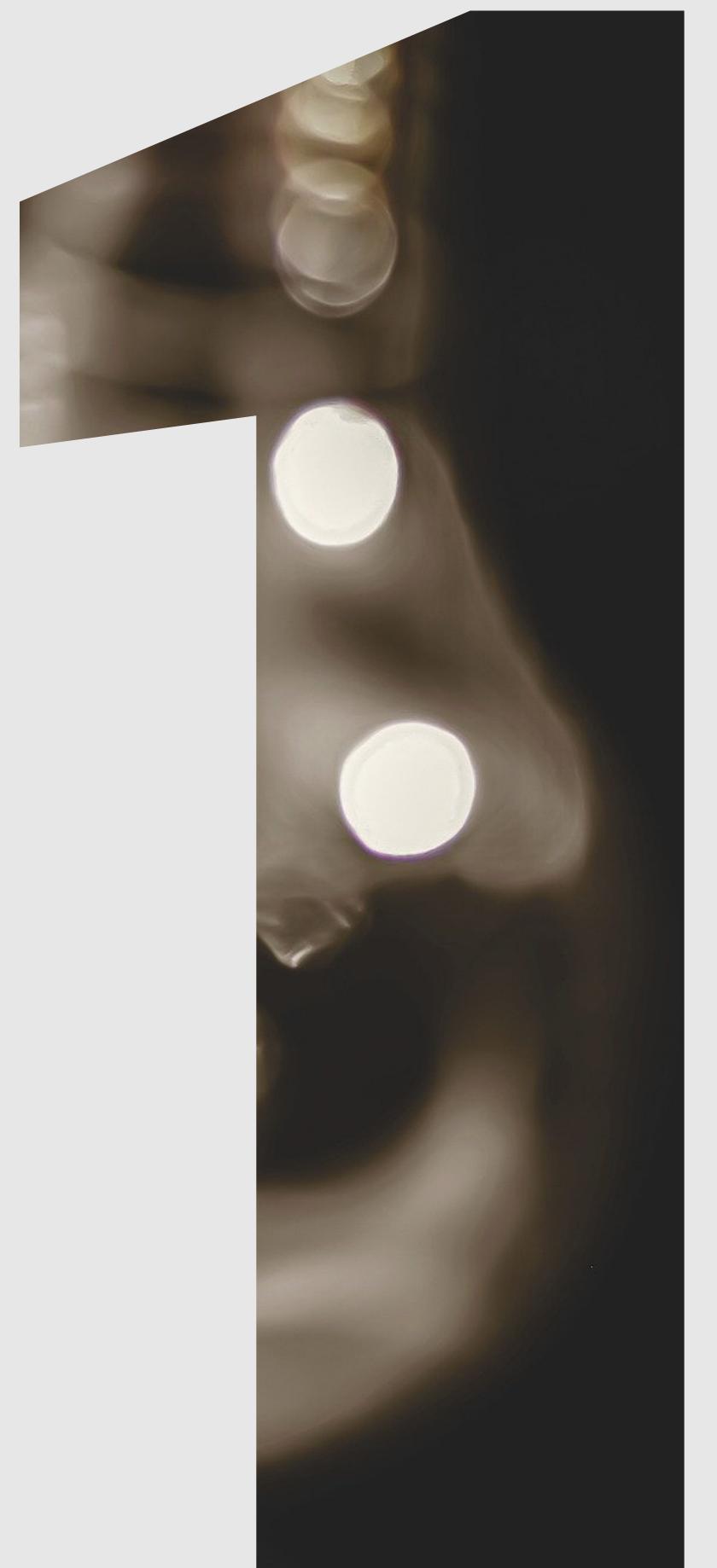
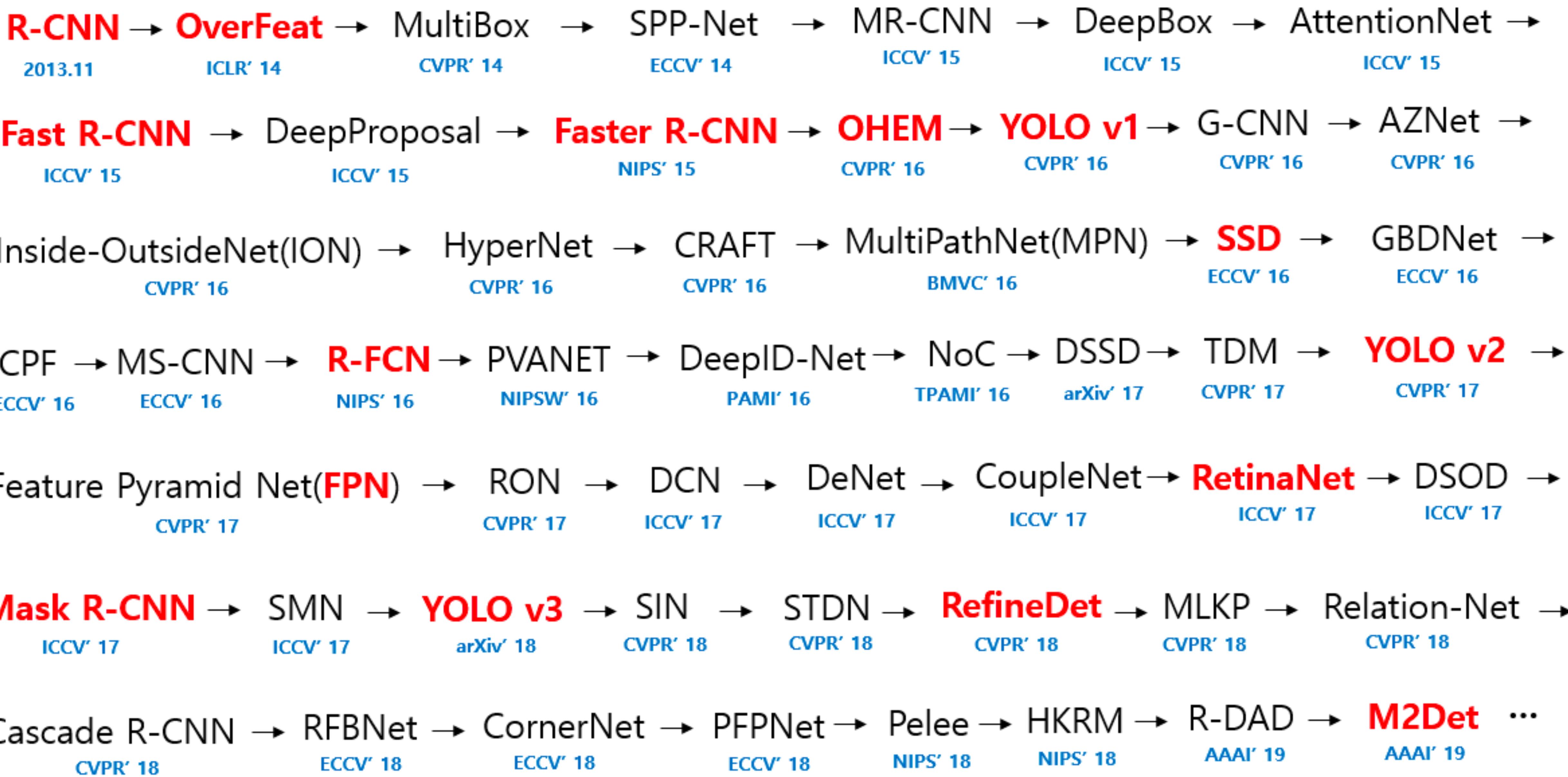


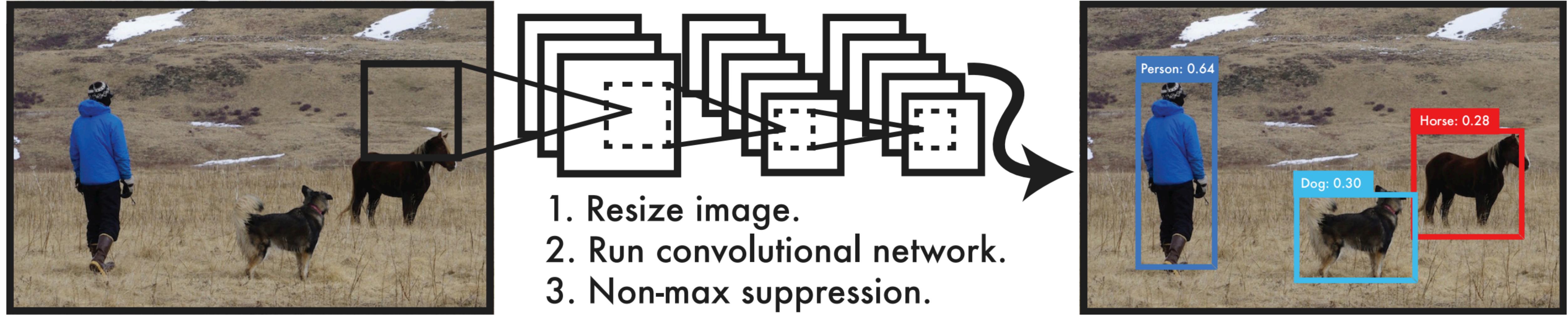
12-15차시. 객체 탐지 및 사례 소개

Detection

객체 탐지



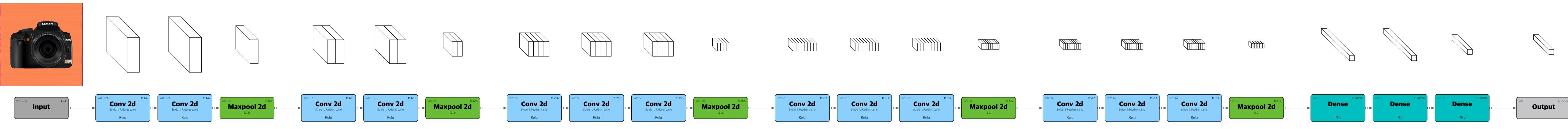




- YOLO로 하는 이미지 처리는 간단하고 직관적
 - (1) 입력 이미지를 448×448 로 크기 조정
 - (2) 이미지에 대해 단일 컨볼루션 네트워크를 실행
 - (3) 모델의 confidence에 의한 결과 탐지

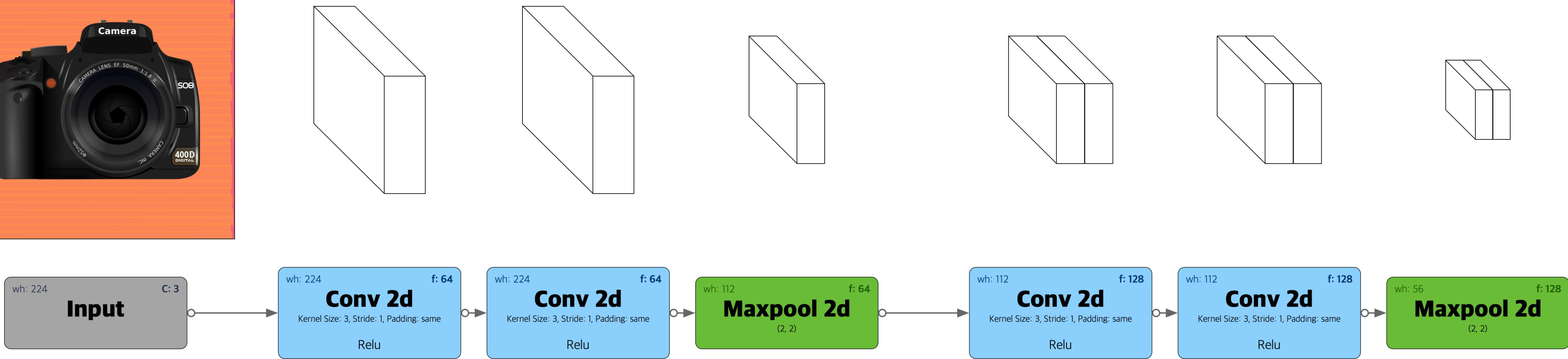
YOLO 객체 탐지 시스템

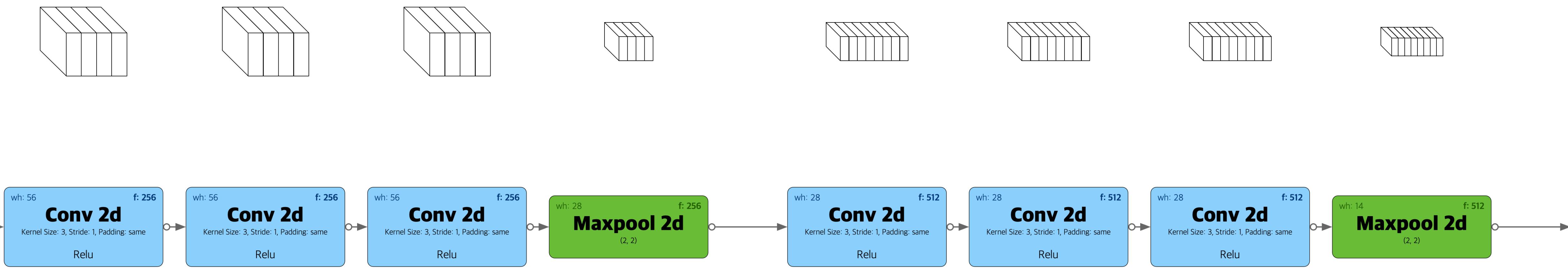
(1) Resize Image. 왜?

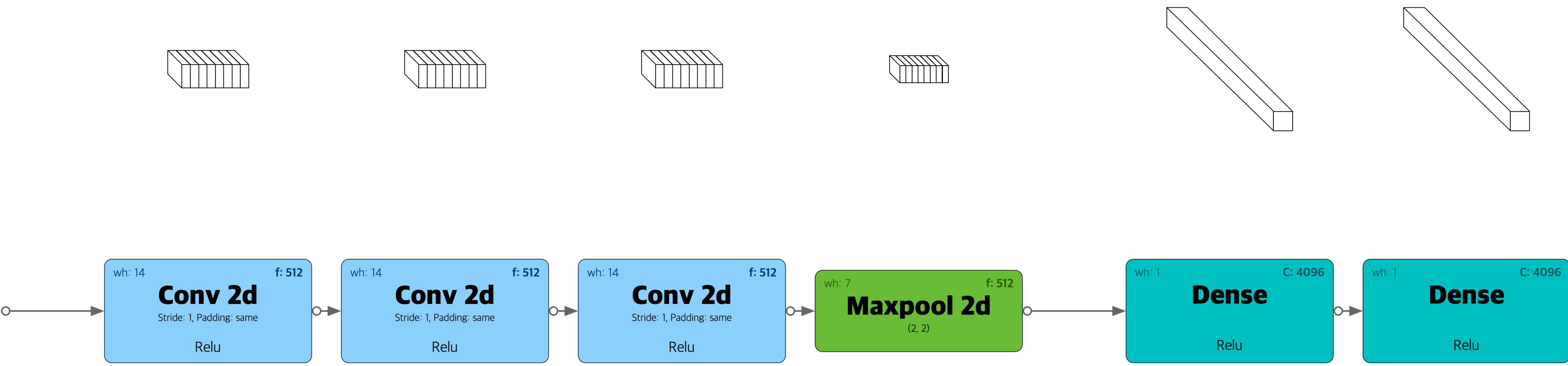


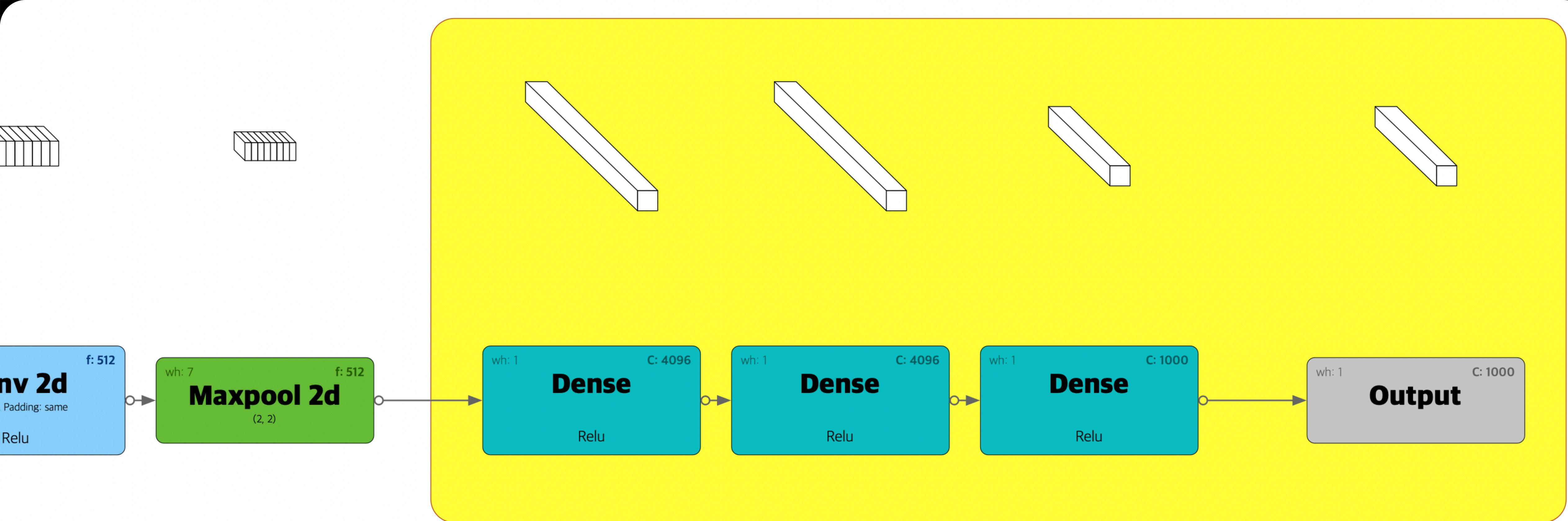
- 일반적인 CNN 네트워크
- VGG16 네트워크
- 224x224x3 입력

CNN



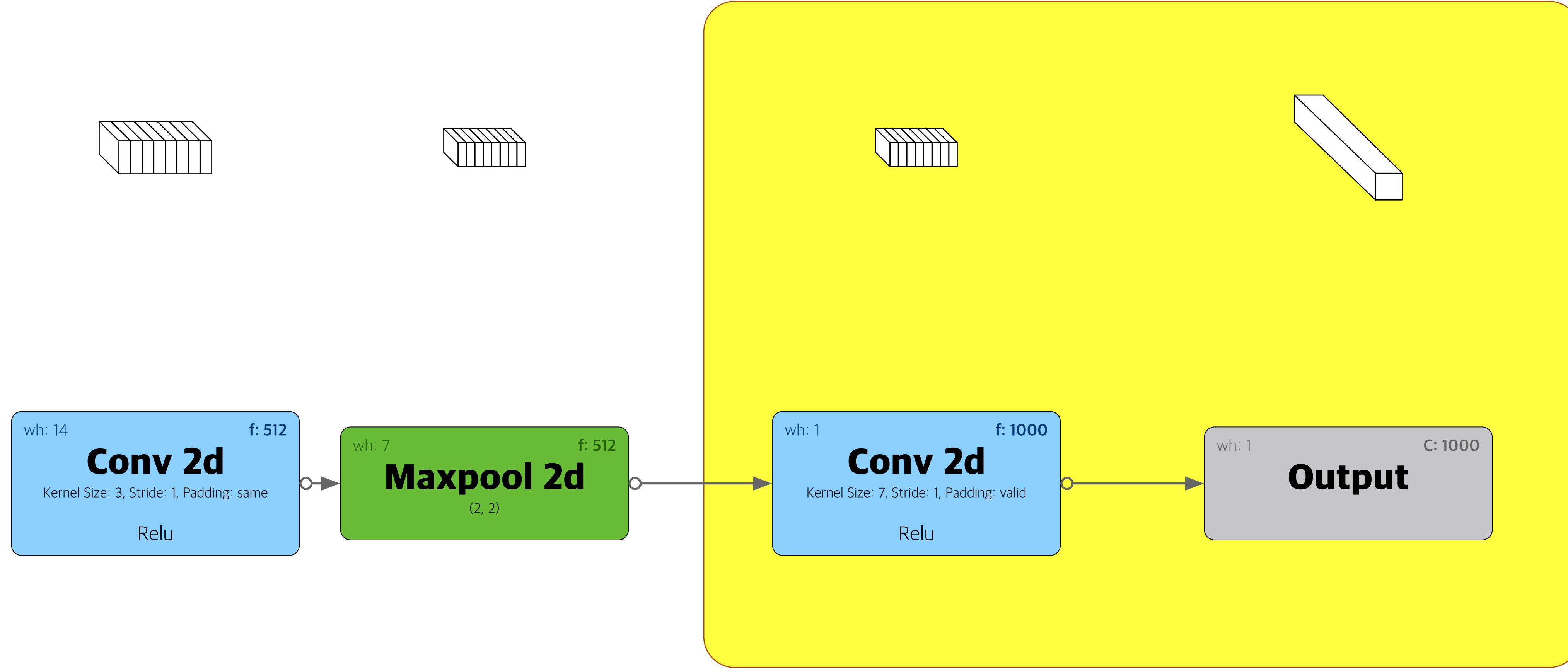






- 뒤쪽의 3개 Dense 레이어를 컨볼루션 레이어로 바꾸자는 아이디어

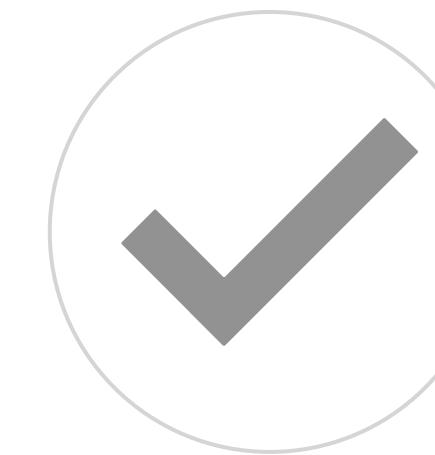
FCN



- FCN의 마지막을 CNN으로 바꾼 예시
- Dense 층의 출력은 [배치크기, 1000] 크기의 텐서이고, 합성곱 층은 [배치크기, 1, 1, 1000] 크기의 텐서이다.

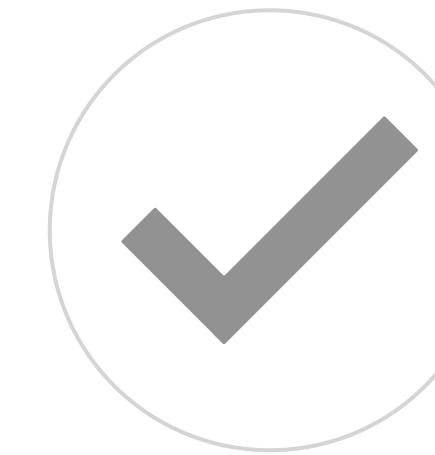
CNN

CNN vs. FCN



크기

밀집 층은 특정 입력 크기를 기대하지만,
합성곱 층은 어떤 크기의 이미지도 처리할 수 있다.



꽃 분류와 위치 추정을 위한 CNN

224x224 이미지에서 트레이닝. 10개의 숫자를 출력.

- [0~4] : 소프트맥스 활성화 함수를 통과, (클래스마다 하나씩) 클래스 확률
- [5] : 로지스틱 활성화 함수를 통과, 존재여부 점수
- [6~9] : 활성화 함수 통과 안함, 바운딩 박스의 좌표와 높이, 너비



CNN to FCN

- CNN을 FCN으로
- 가중치 복사도 가능. (거의 같은 구조)

CNN vs. FCN



FCN 중간 특성맵 크기



FCN 중간 특성맵 크기 2



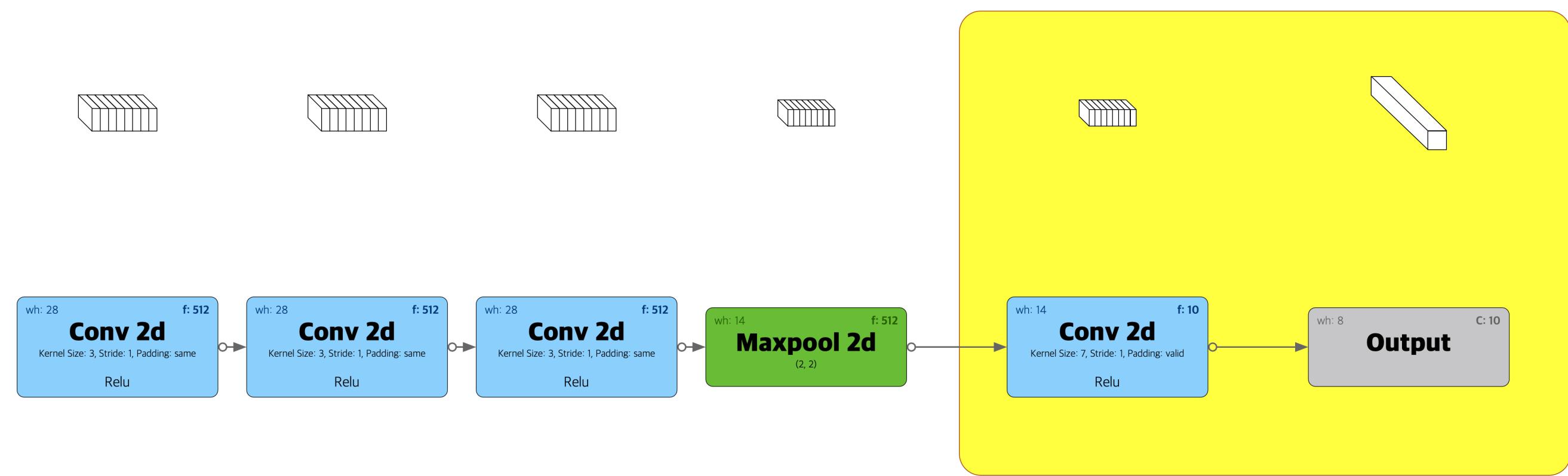
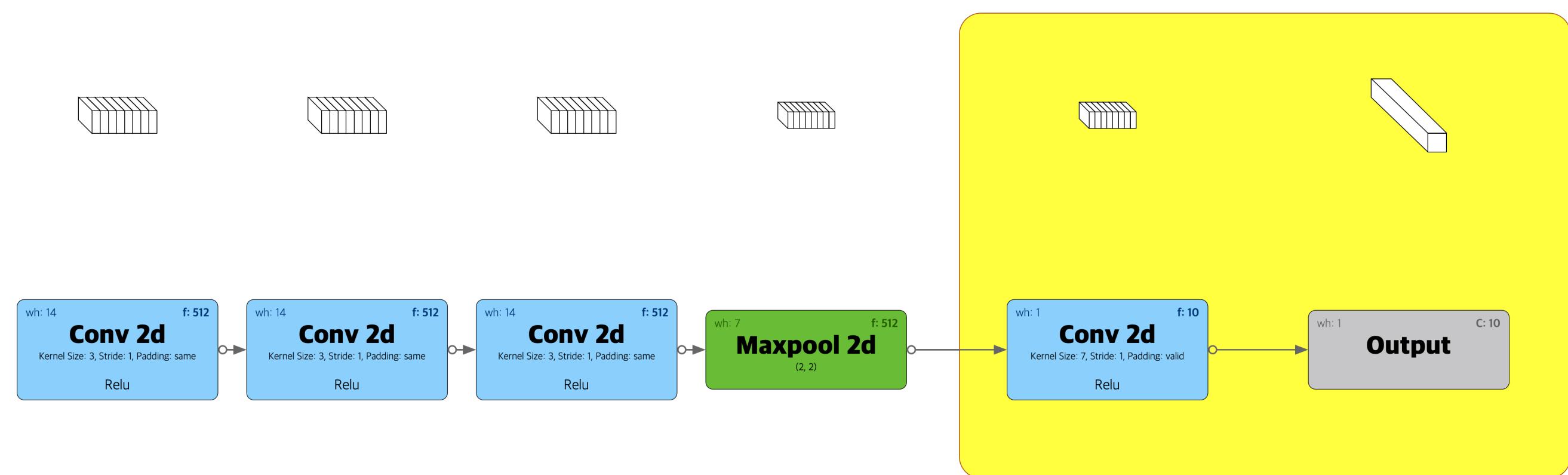
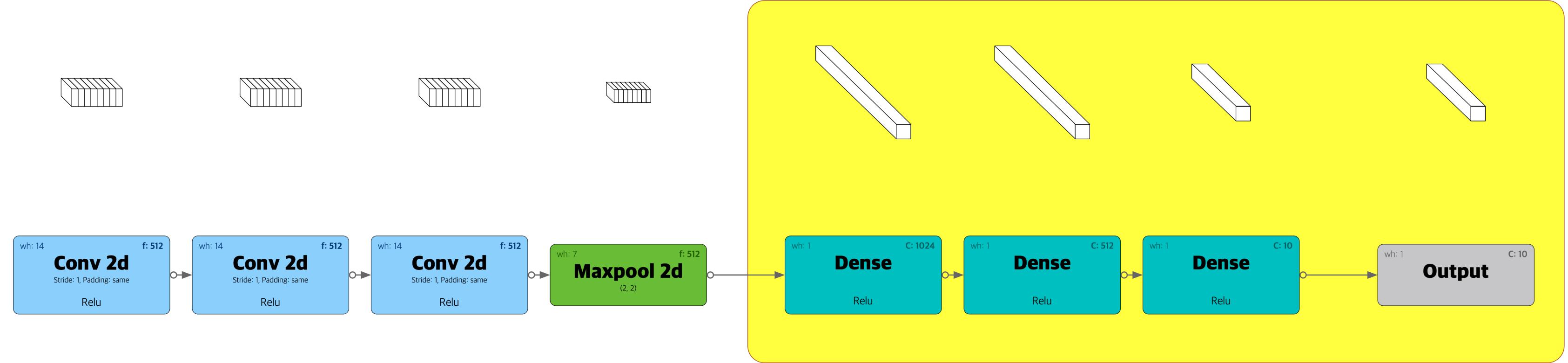
CNN vs. FCN

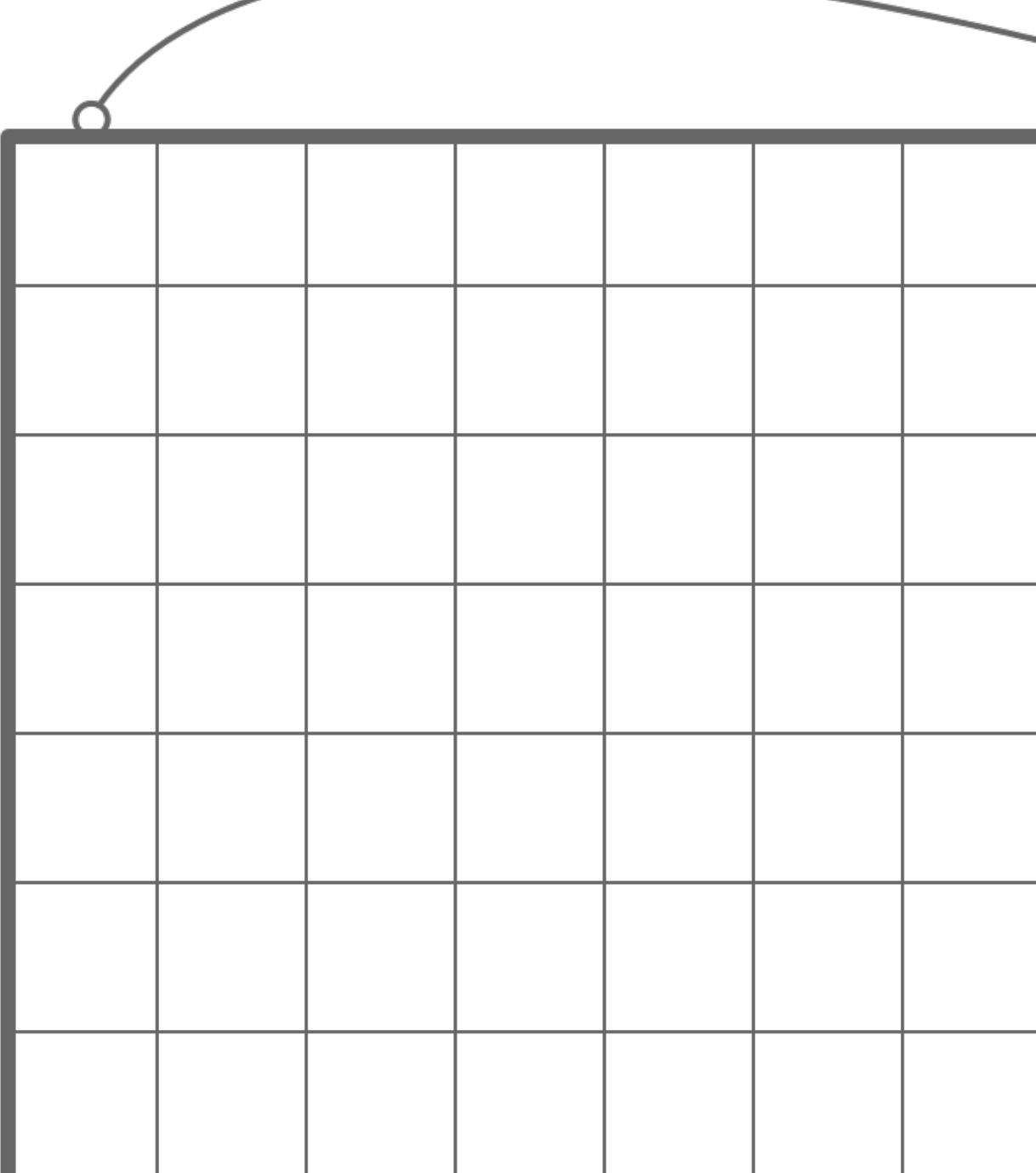
FCN 네트워크에서,
224x224 이미지를 주입하면,
(앞 장의) 노란색 부분 이전은 7x7 특성맵을 출력

이 FCN에 448x448 이미지를 주입하면,
14x14 특성맵을 출력할 것이고,
최종 출력은 8x8 크기의 특성맵 10개

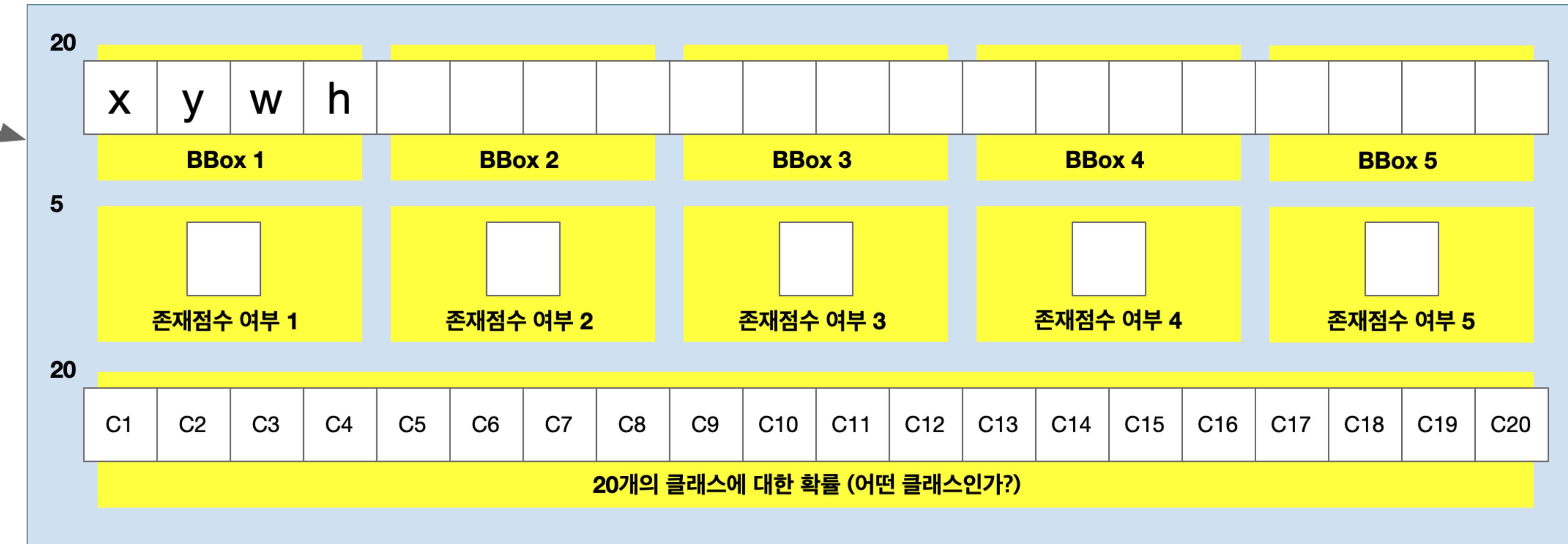
CNN에 448x448 이미지를 주입하면, 같은 결과

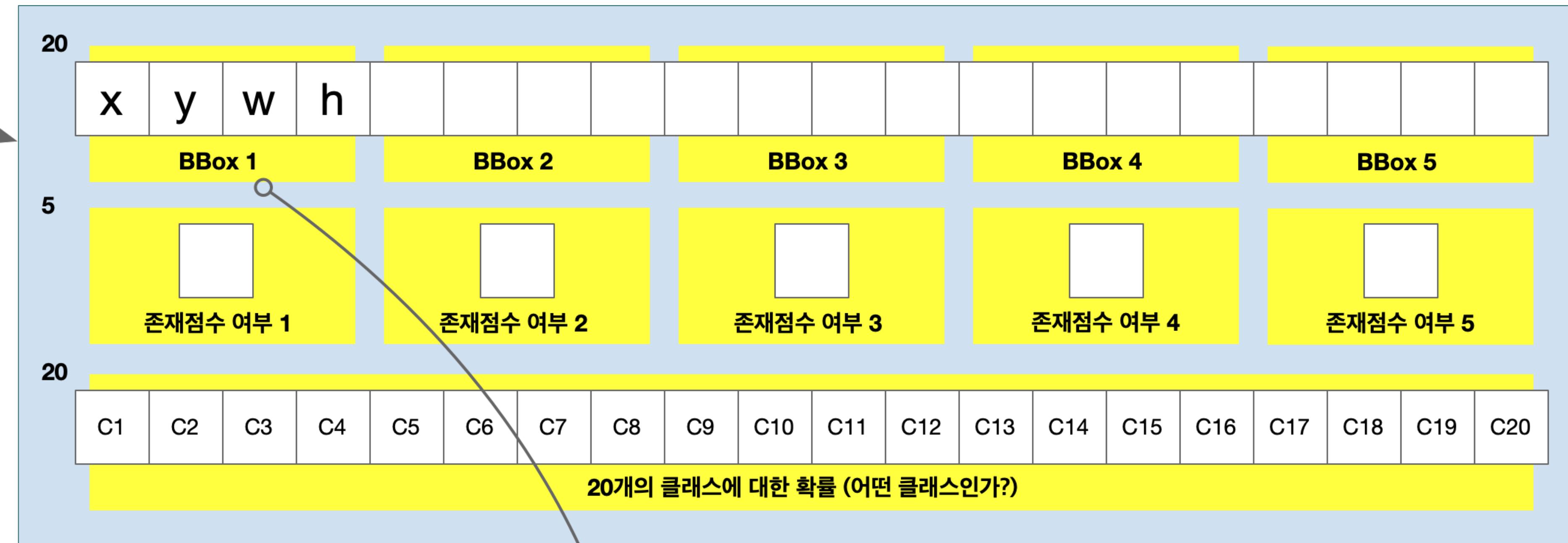
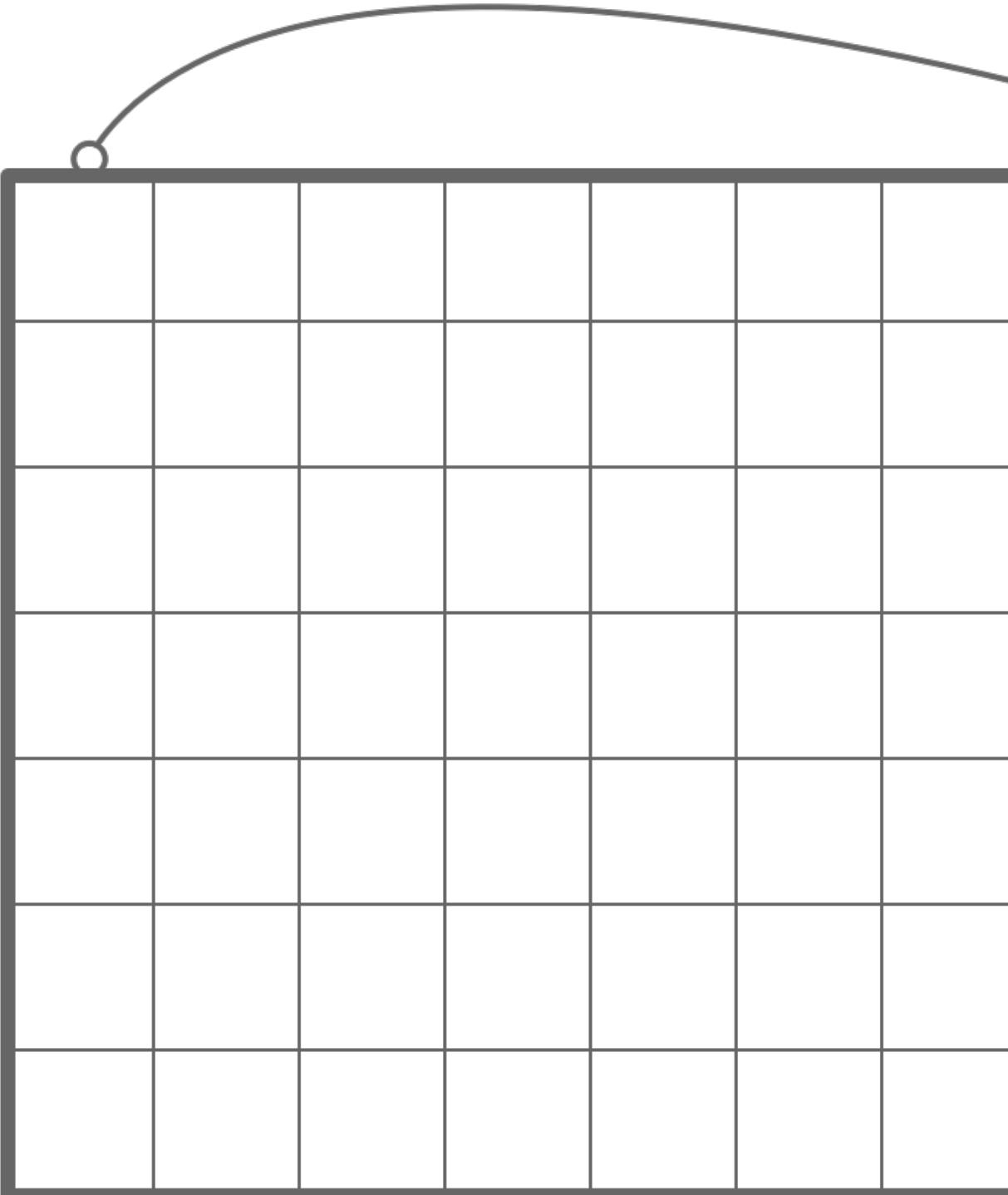
즉, 전체 이미지를 한 번만 처리해 8x8 크기의 배열을 출력.
이는 원래 CNN으로 하려면
행방향으로 8스텝, 열방향으로 8스텝 슬라이딩 하는 것과 동일





45





바운딩 박스 중심의 절대 좌표를 예측하는 대신, **격자 셀에 대한 상대 좌표를 예측**

- (0, 0) : 셀의 왼쪽 위
- (1, 1) : 오른쪽 아래

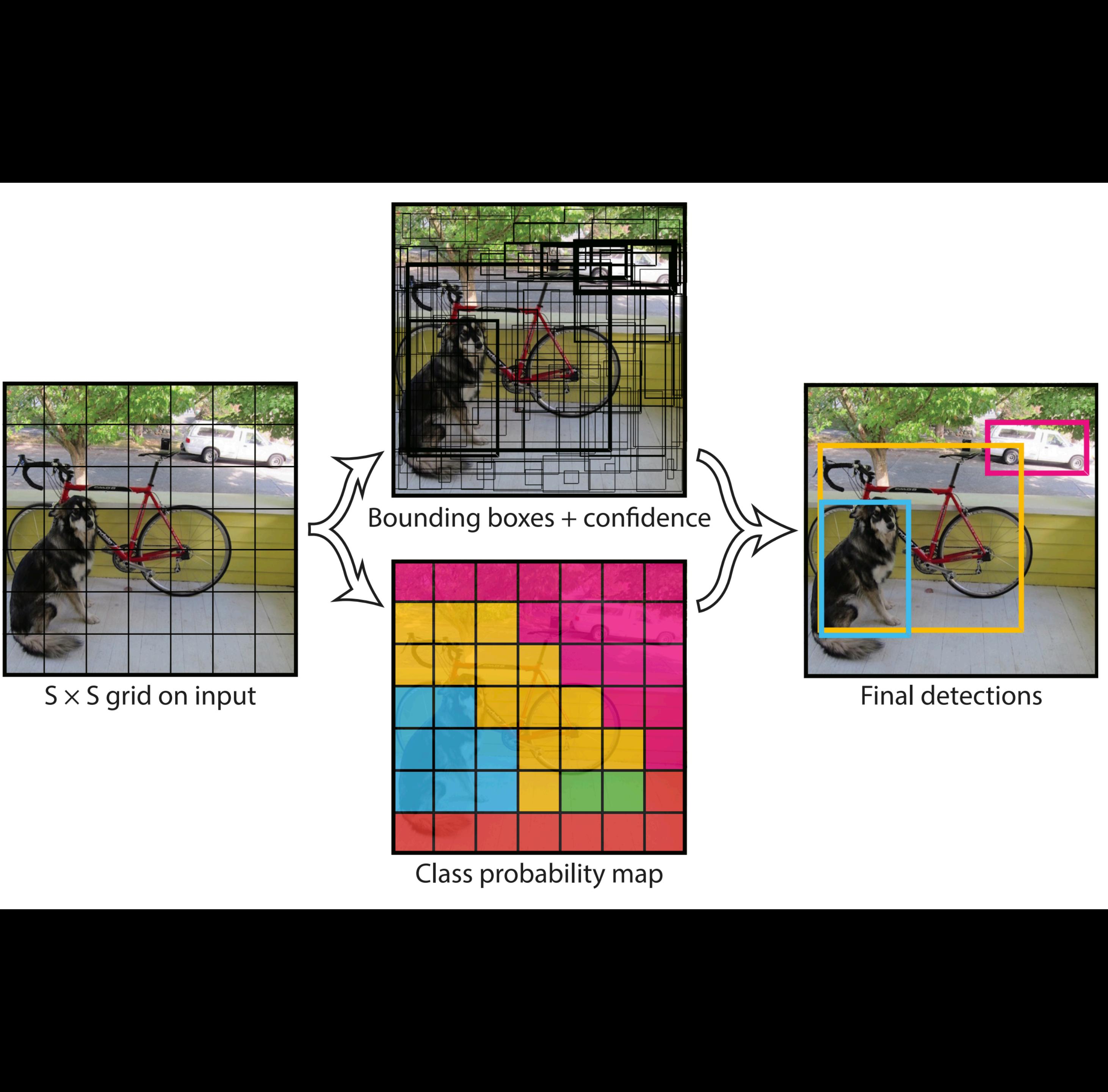
각 격자 셀에 대해 바운딩 박스의 **중심이 격자 셀 안에 놓인 것만을 예측**하도록 훈련된다.

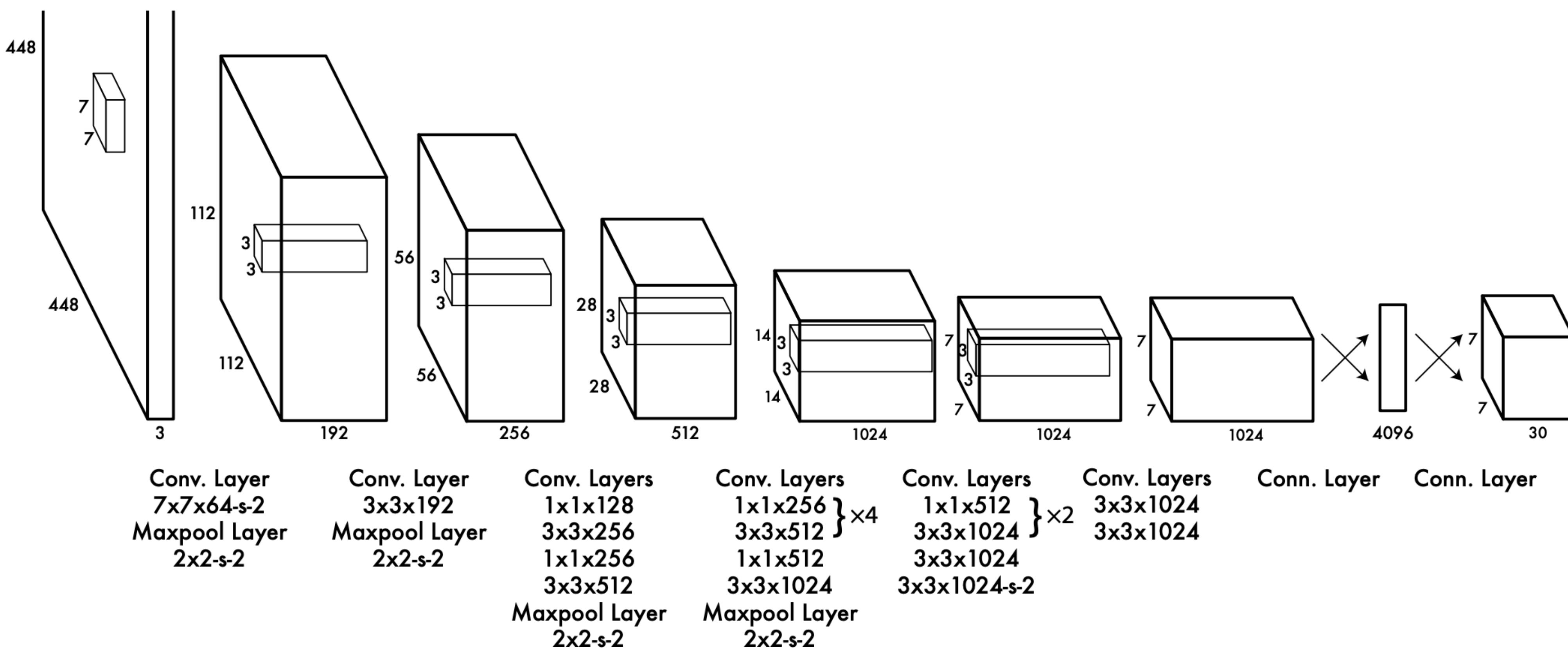
- 물론, 바운딩 박스 자체는 격자 셀 밖으로 넘어갈 수 있다.
- 로지스틱 활성화 함수를 적용하여, 바운딩 박스 좌표가 0~1 사이가 되도록 한다.

(2) Convolution 네트워크

모델

- 시스템은 탐지를 회귀 문제로 모델링
- 이미지를 $S \times S$ 그리드로 나눔.
 - 각 그리드 셀에 대해 B 경계 상자, 해당 상자에 대한 신뢰도
 - C 클래스 확률을 예측
- 이러한 예측은 $S \times S \times (B \cdot 5 + C)$ 텐서로 인코딩





- 탐지 네트워크는 24개의 컨볼루션 레이어와 2개의 완전 연결 레이어로 구성
- 1x1 컨볼루션 레이어를 번갈아 사용하면 이전 레이어의 특징 공간이 줄어든다.
- 절반 해상도(224×224 입력 이미지)의 ImageNet 분류에 대해 사전 트레이닝된 컨벌루션 레이어를 사용
- 감지를 위해 해상도를 두 배로 늘린다.

YOLO

(3) Non-max suppression

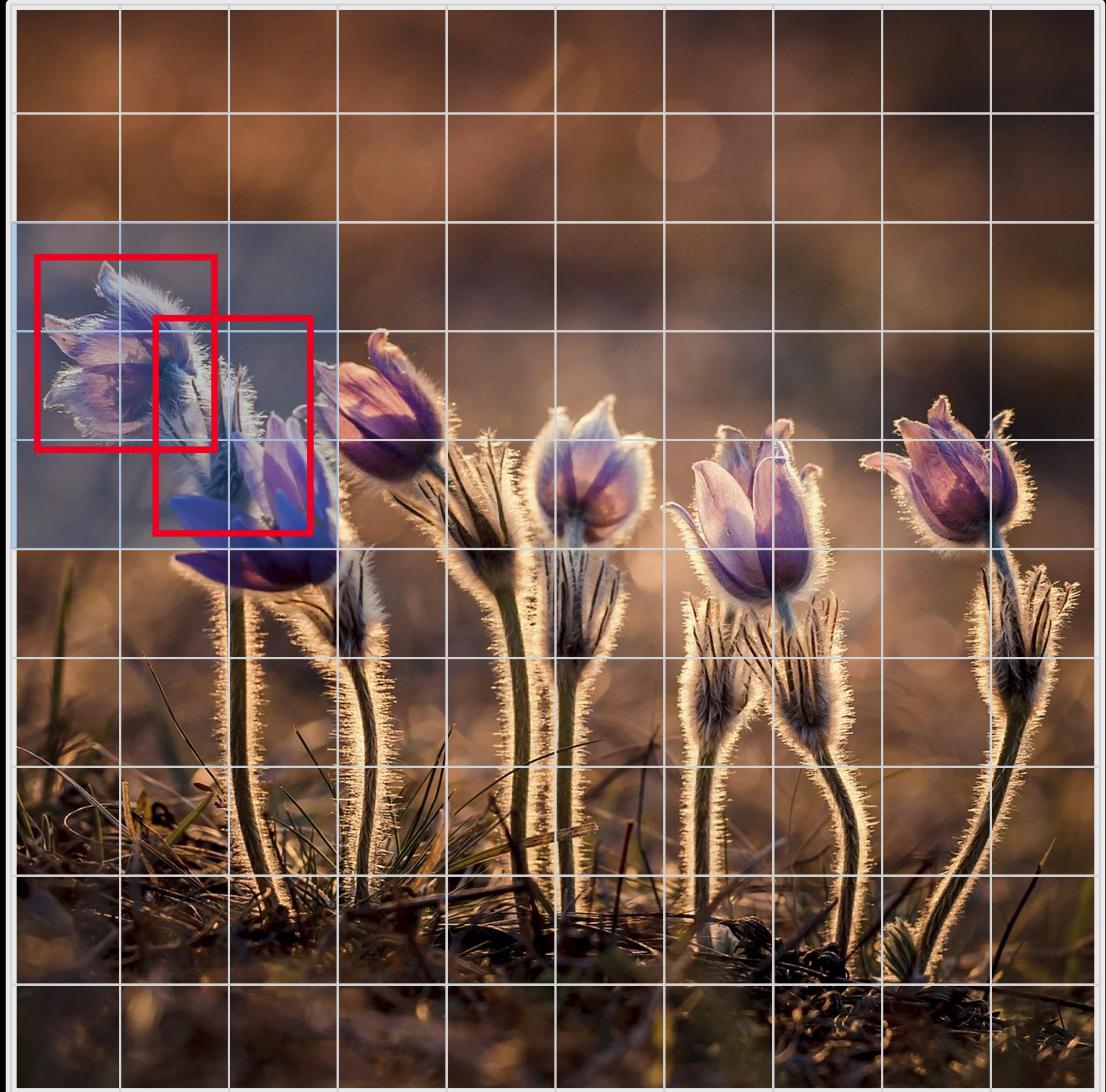
간단한 객체 탐지

- 하나의 물체를 분류하고,
위치를 찾는 분류기를 훈련
- CNN으로 이미지를 모두 훑기
- 이미지를 10x10 격자로 나누고 슬라이딩



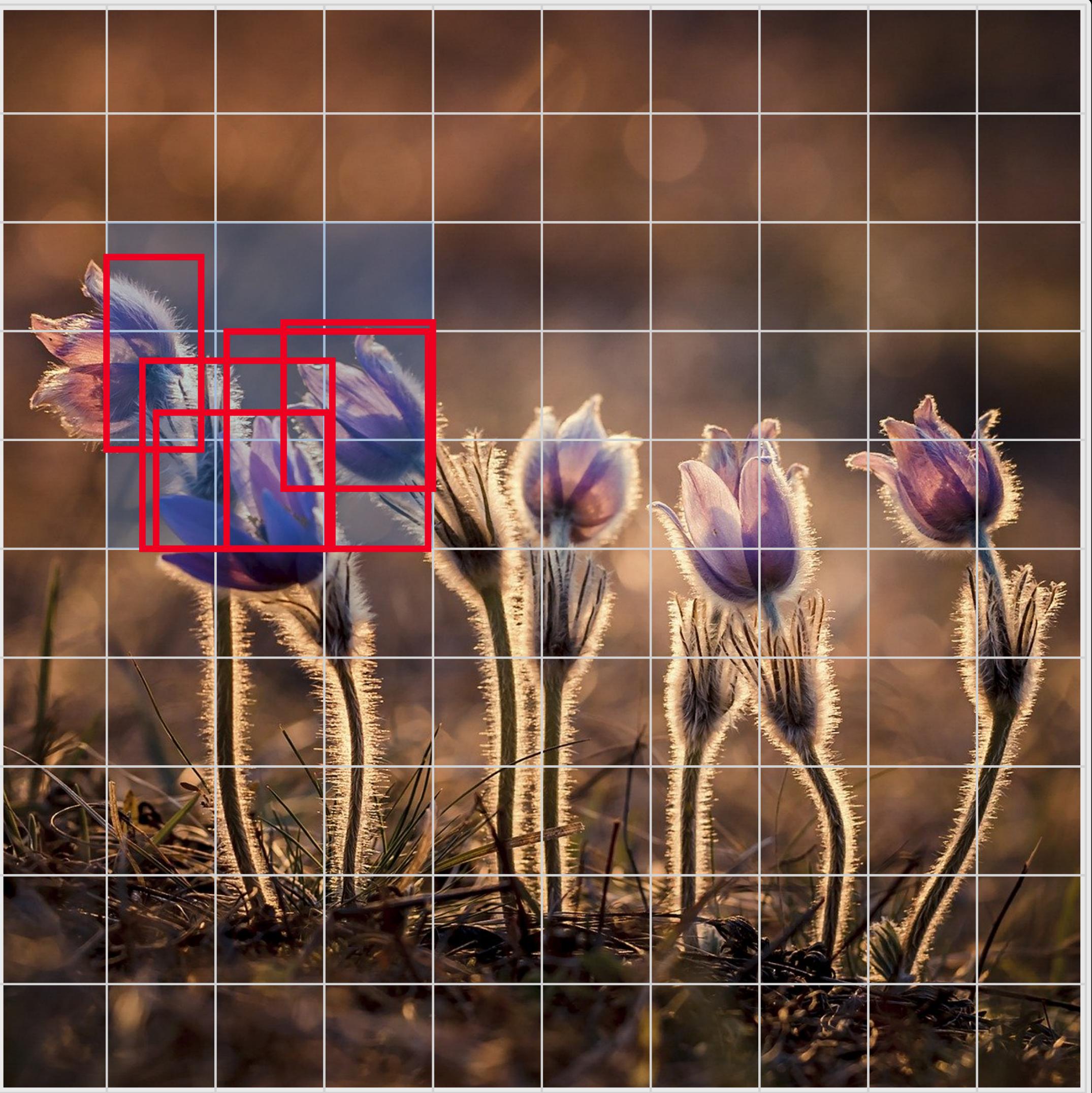
슬라이딩 훈기

- 3x3 격자에서 슬라이딩으로 객체 감지 (내부는 CNN)
- 빨간 상자가 감지한 영역



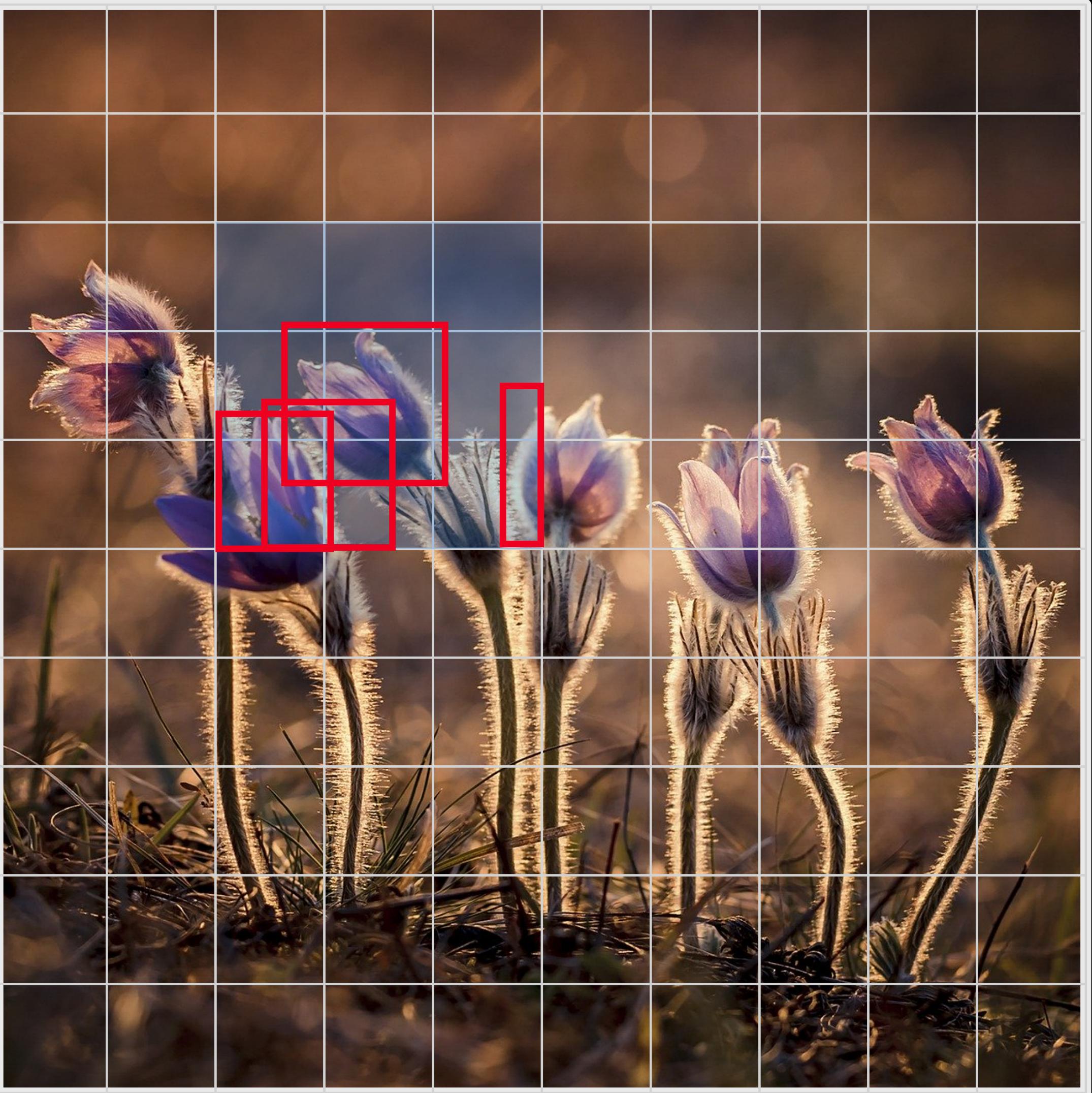
슬라이딩 훈기

- 3x3 격자에서 슬라이딩으로 객체 감지
- 빨간 상자가 감지한 영역
 - 잘못 감지한 영역도 존재한다.



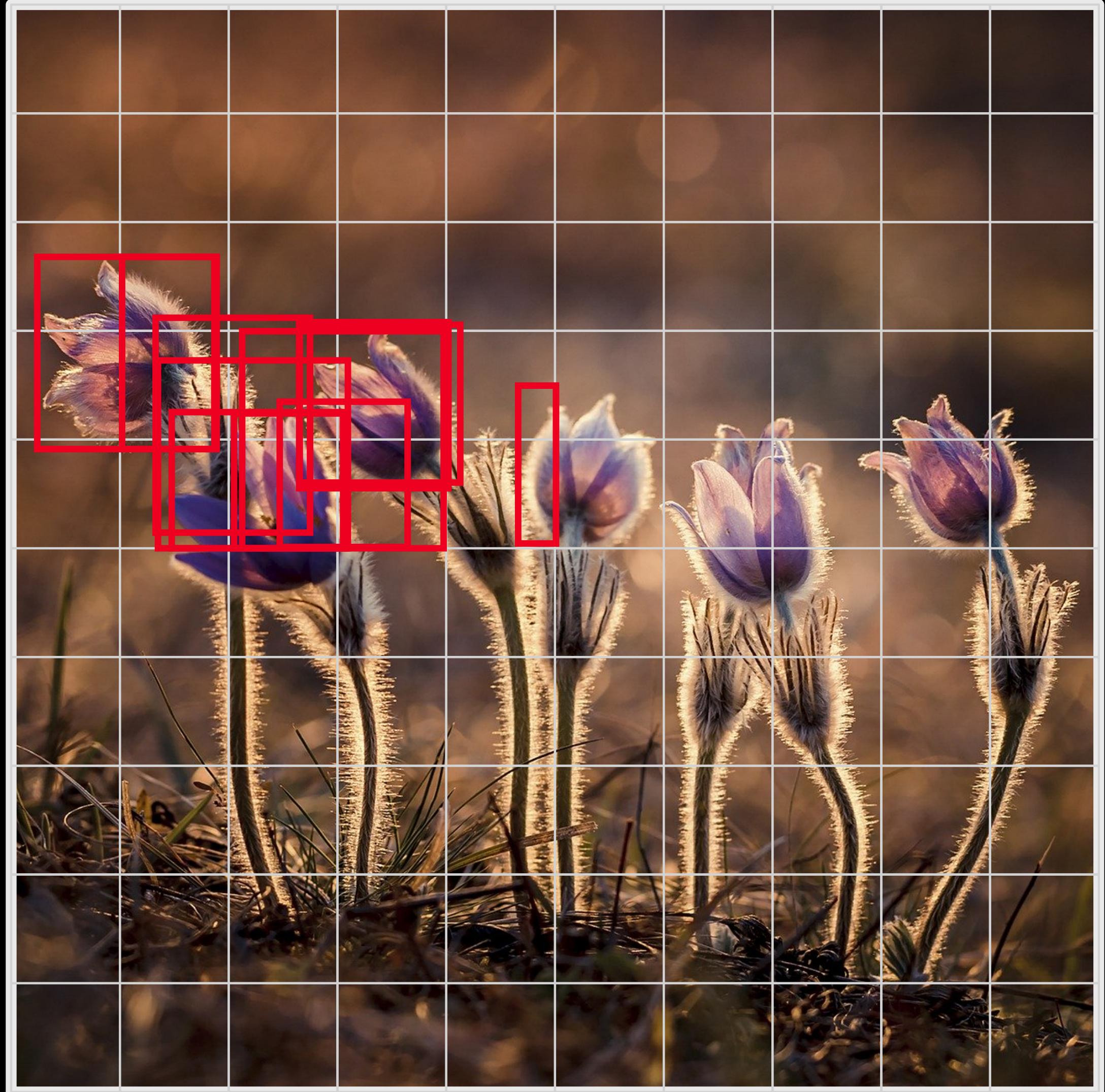
슬라이딩 훈기

- 3x3 격자에서 슬라이딩으로 객체 감지
- 빨간 상자가 감지한 영역
 - 잘못 감지한 영역도 존재한다.



슬라이딩 훠기 단점

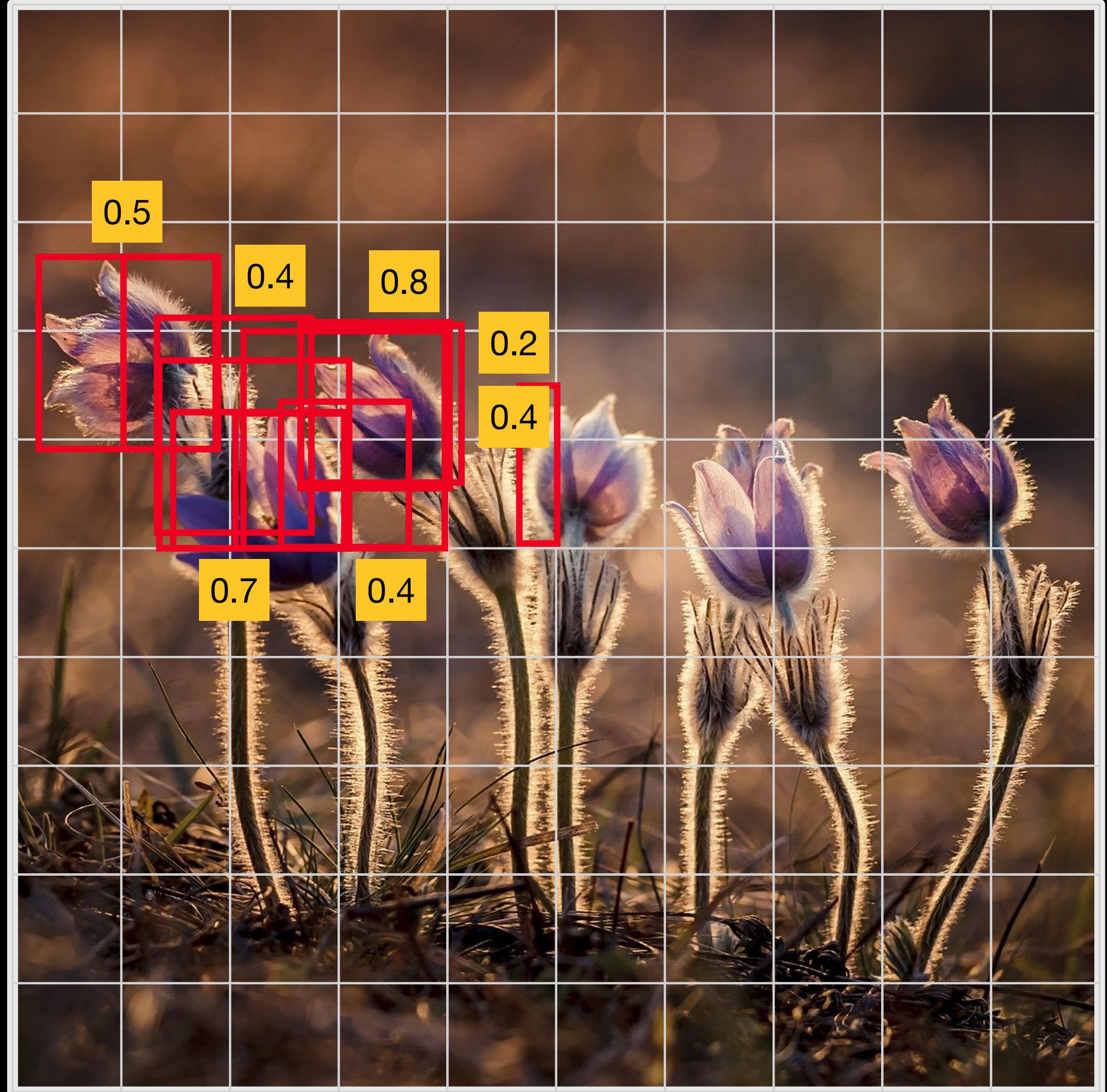
- 조금씩 다른 위치에서 동일한 물체를 여러 번 감지
- 불필요한 바운딩 박스를 제거하기 위해 사후 처리 필요



사후처리

NMS(non-max suppression)

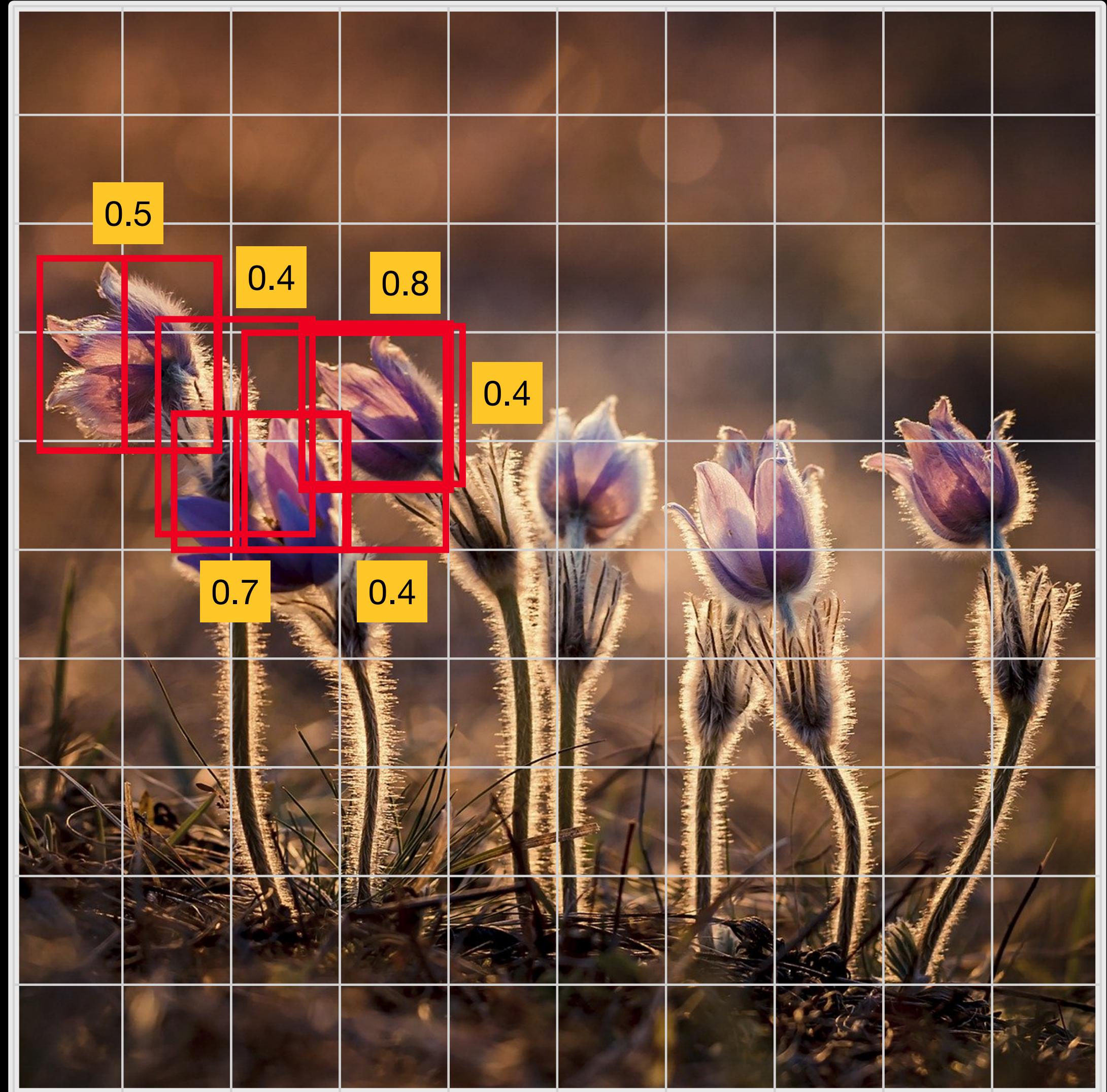
- 각각 바운딩 박스에 대해, 객체의 존재여부 출력의 추가
- 시그모이드 활성화 함수
- 이진 크로스 엔트로피 손실을 사용한 트레이닝



사후처리

NMS(non-max suppression)

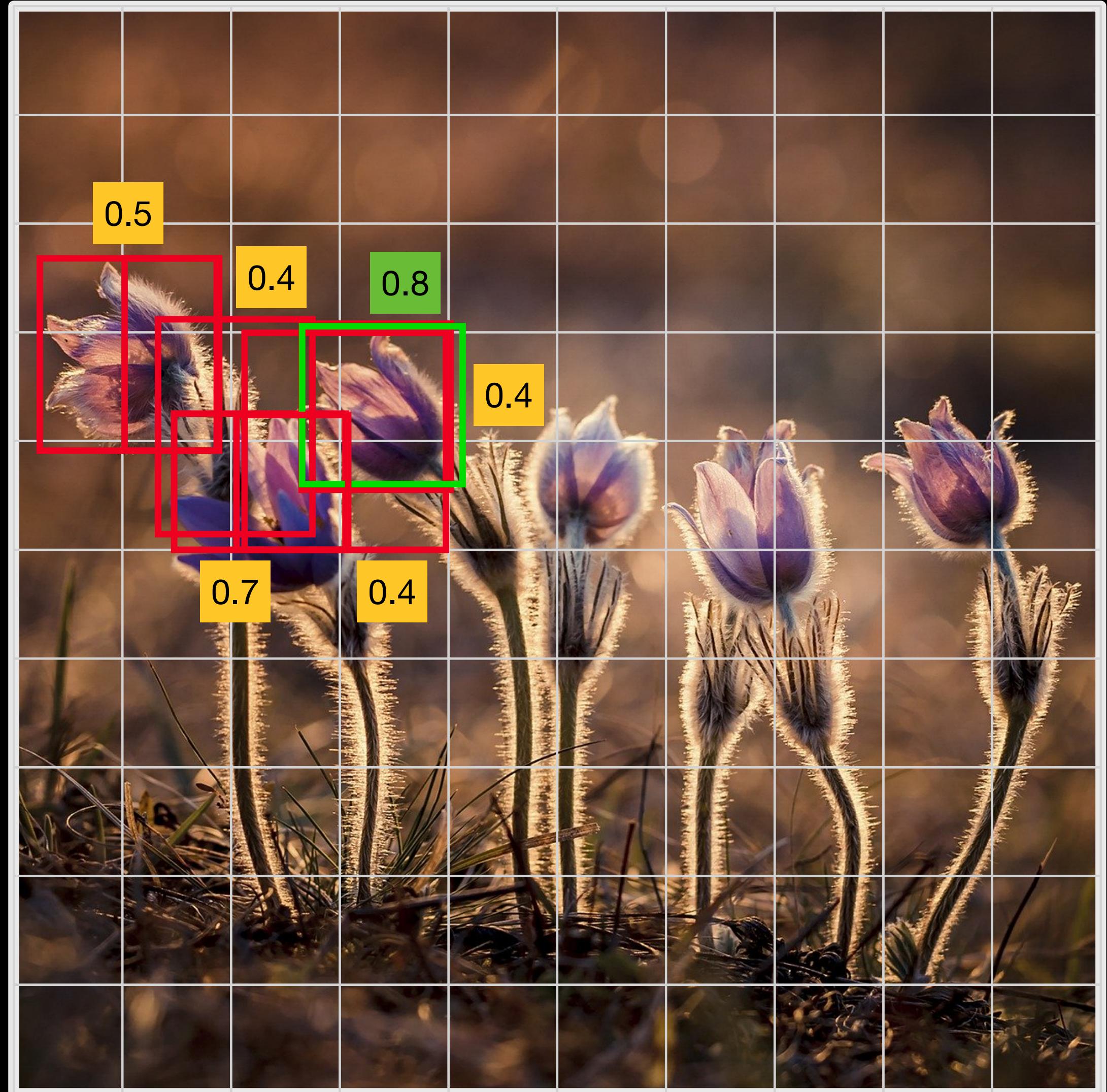
- 존재 여부가 **어떤 임곗값 이하**인 바운딩 박스 삭제
- 여기서는 **0.4**라고 하자.



사후처리

NMS(non-max suppression)

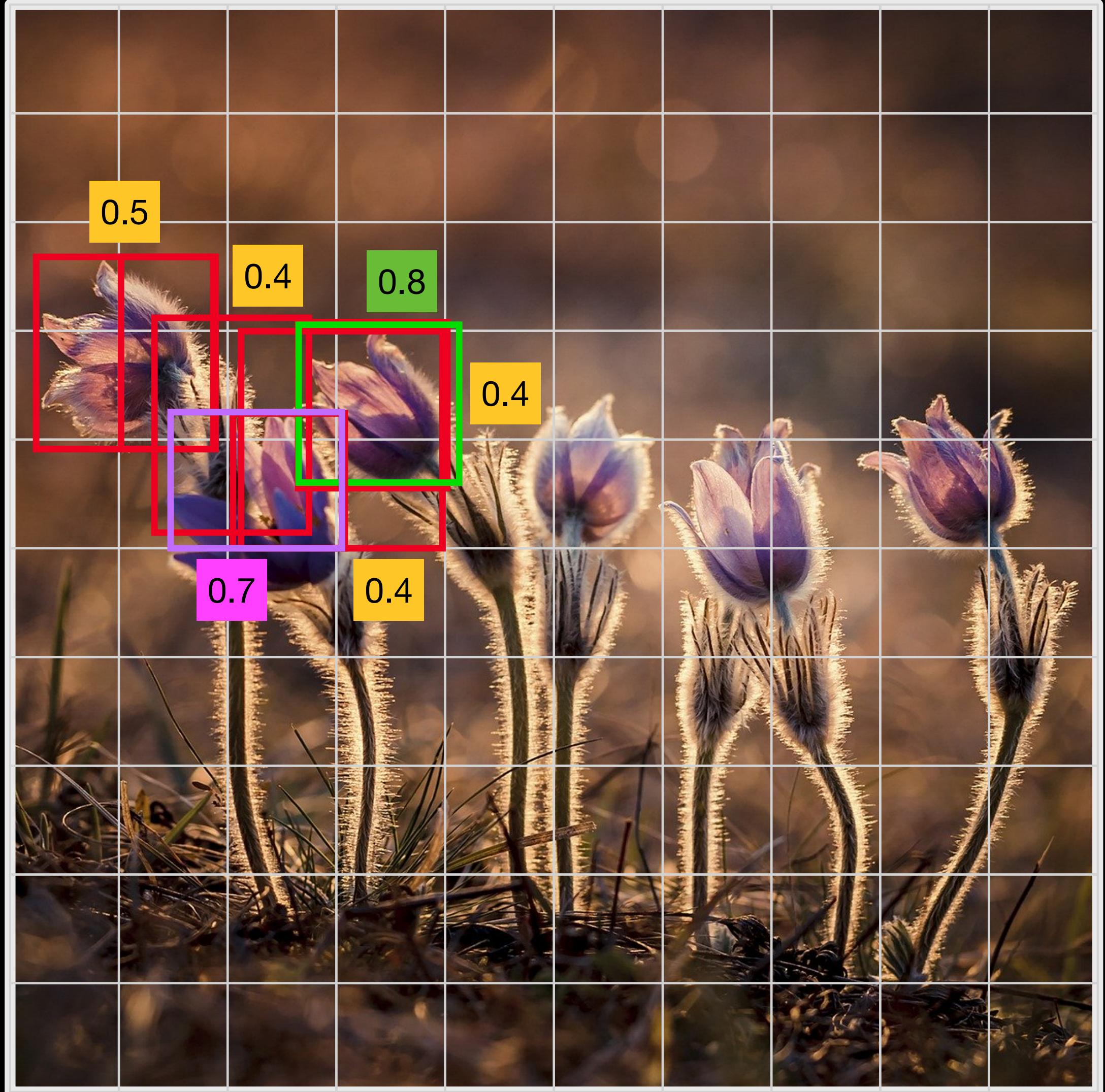
- 존재여부 점수가 가장 높은 바운딩 박스를 찾고,
- 존재여부 점수에 대해 가장 높은 점수부터 역순으로 배치



사후처리

NMS(non-max suppression)

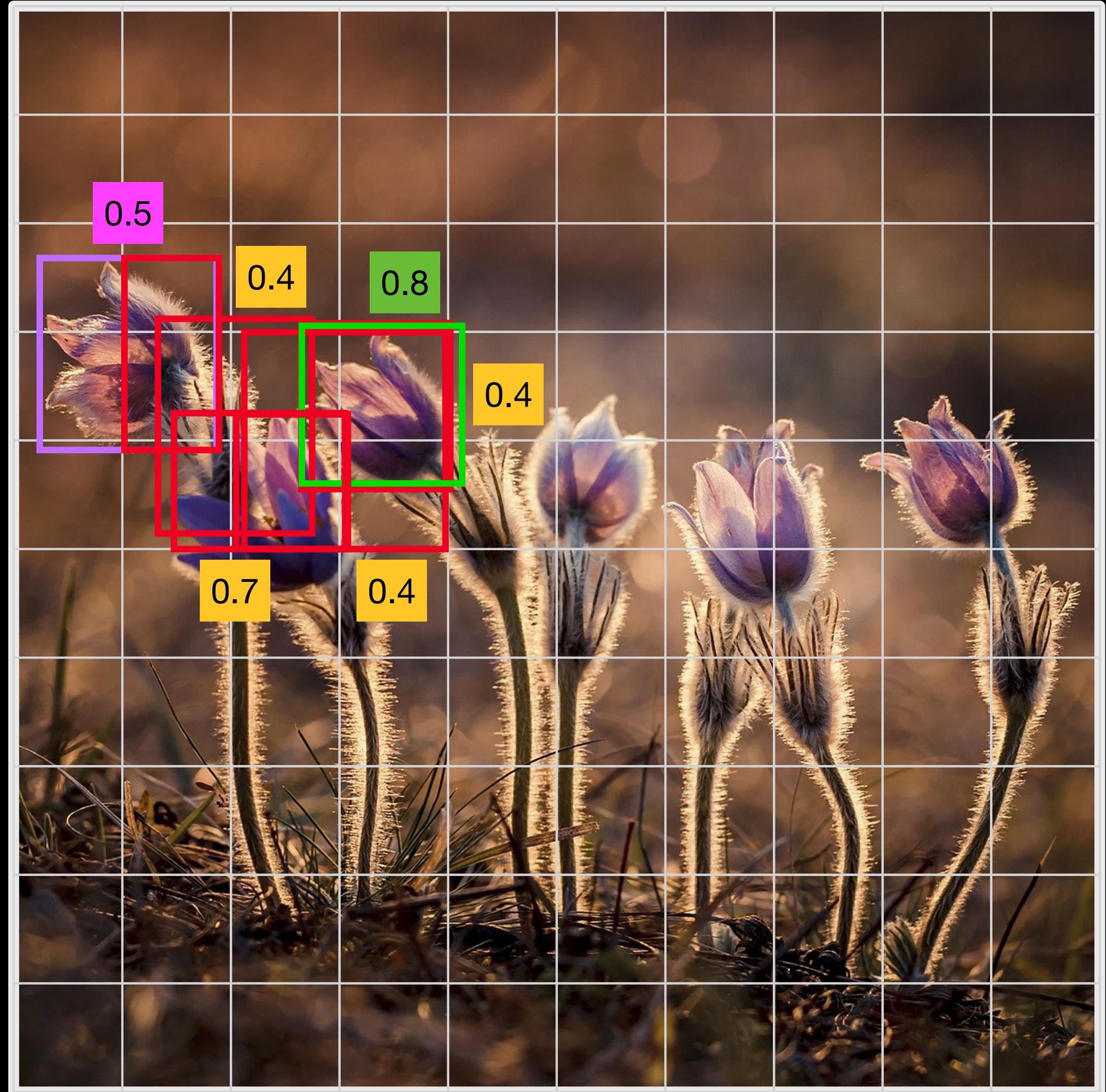
- 0.8과 0.7의 IoU를 비교, IoU가 **지정한 점수 (50%)** 이하라면,
- 그냥 지나간다.



사후처리

NMS(non-max suppression)

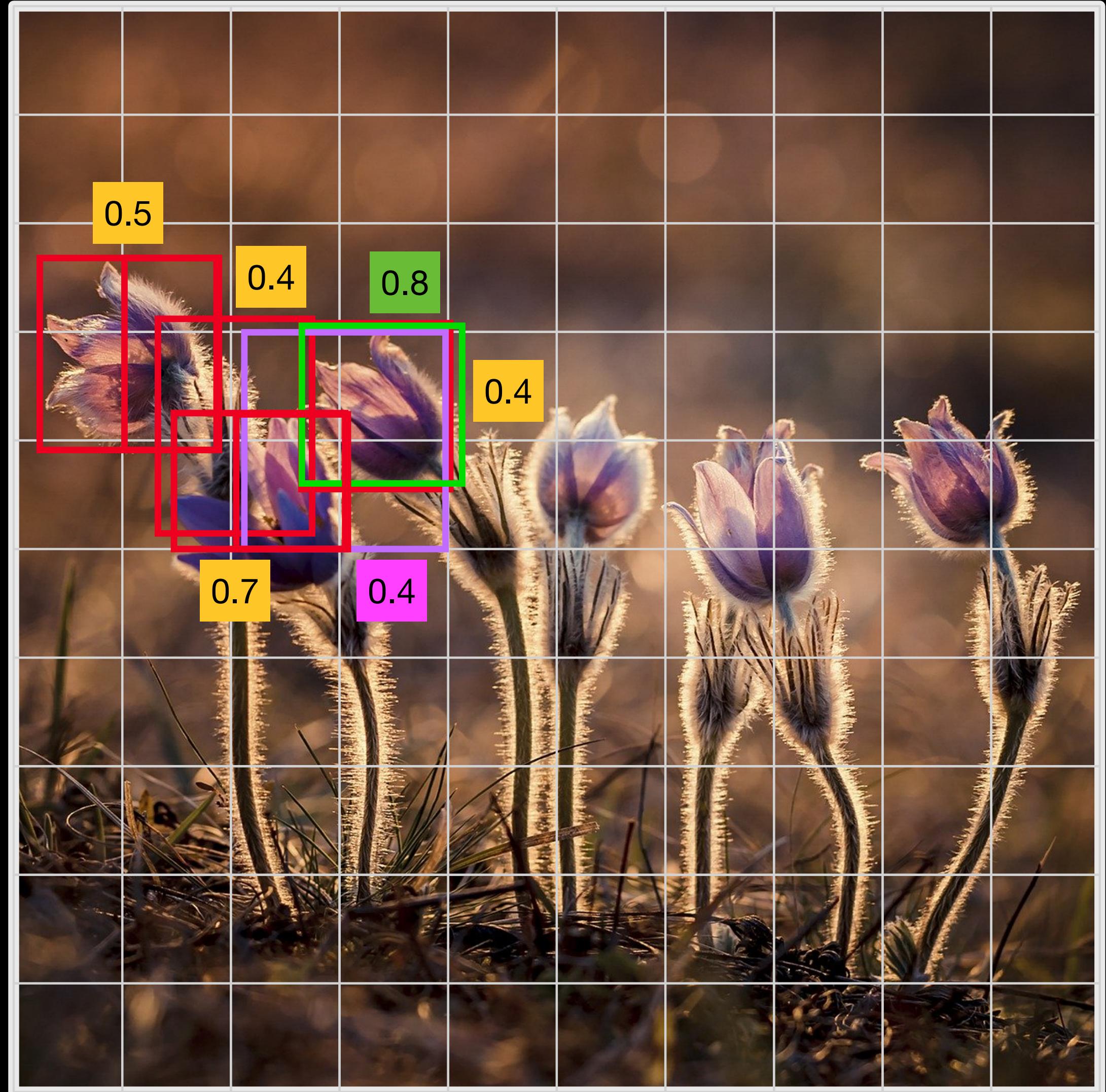
- 0.8과 0.5의 IoU를 비교, IoU가 **지정한 점수 (50%)** 이하라면,
그냥 지나간다.



사후처리

NMS(non-max suppression)

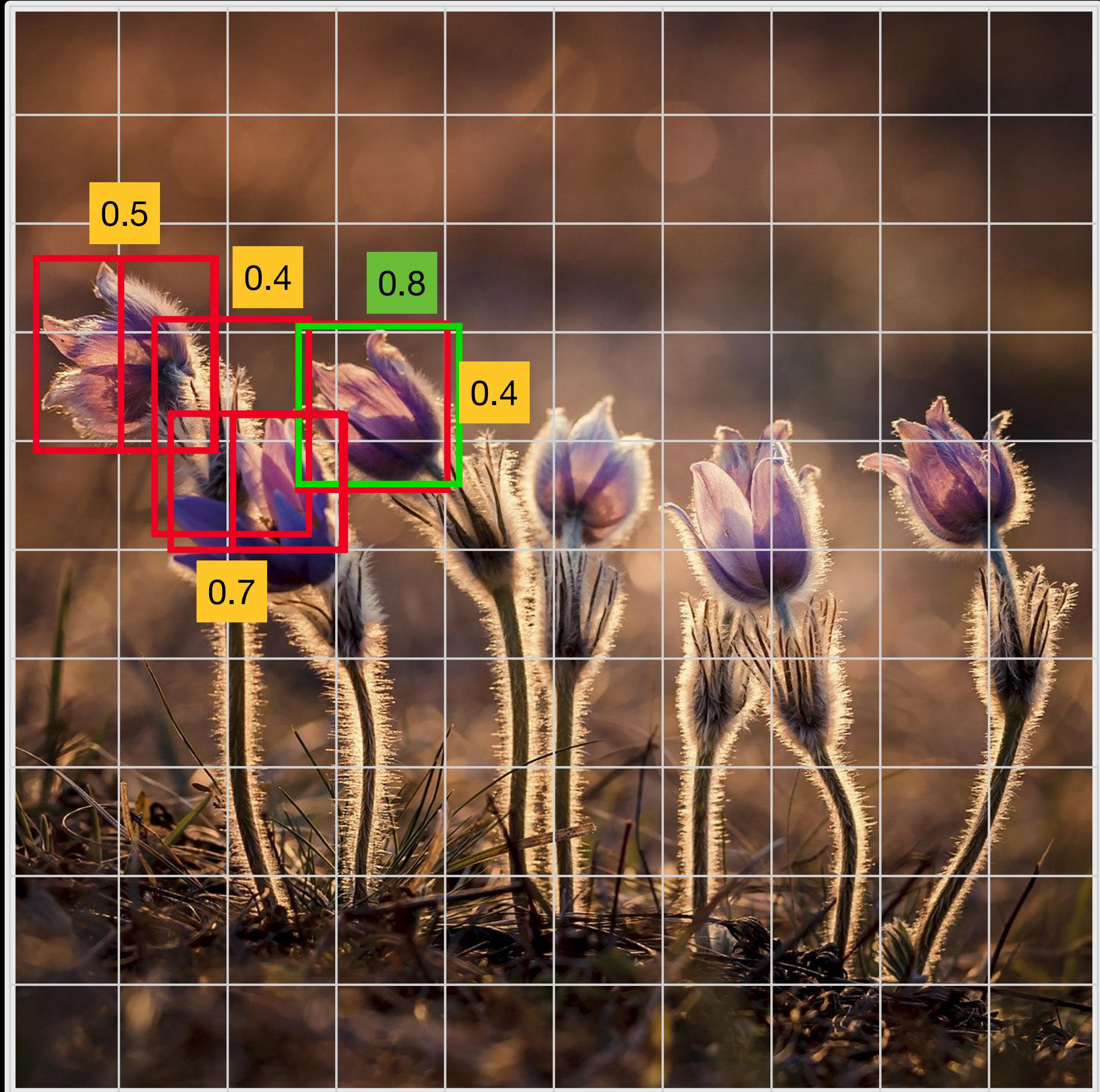
- 0.8과 0.4의 IoU를 비교, IoU가 **지정한 점수 (50%)** 이상이라면,
- 해당 영역 제안을 제거



사후처리

NMS(non-max suppression)

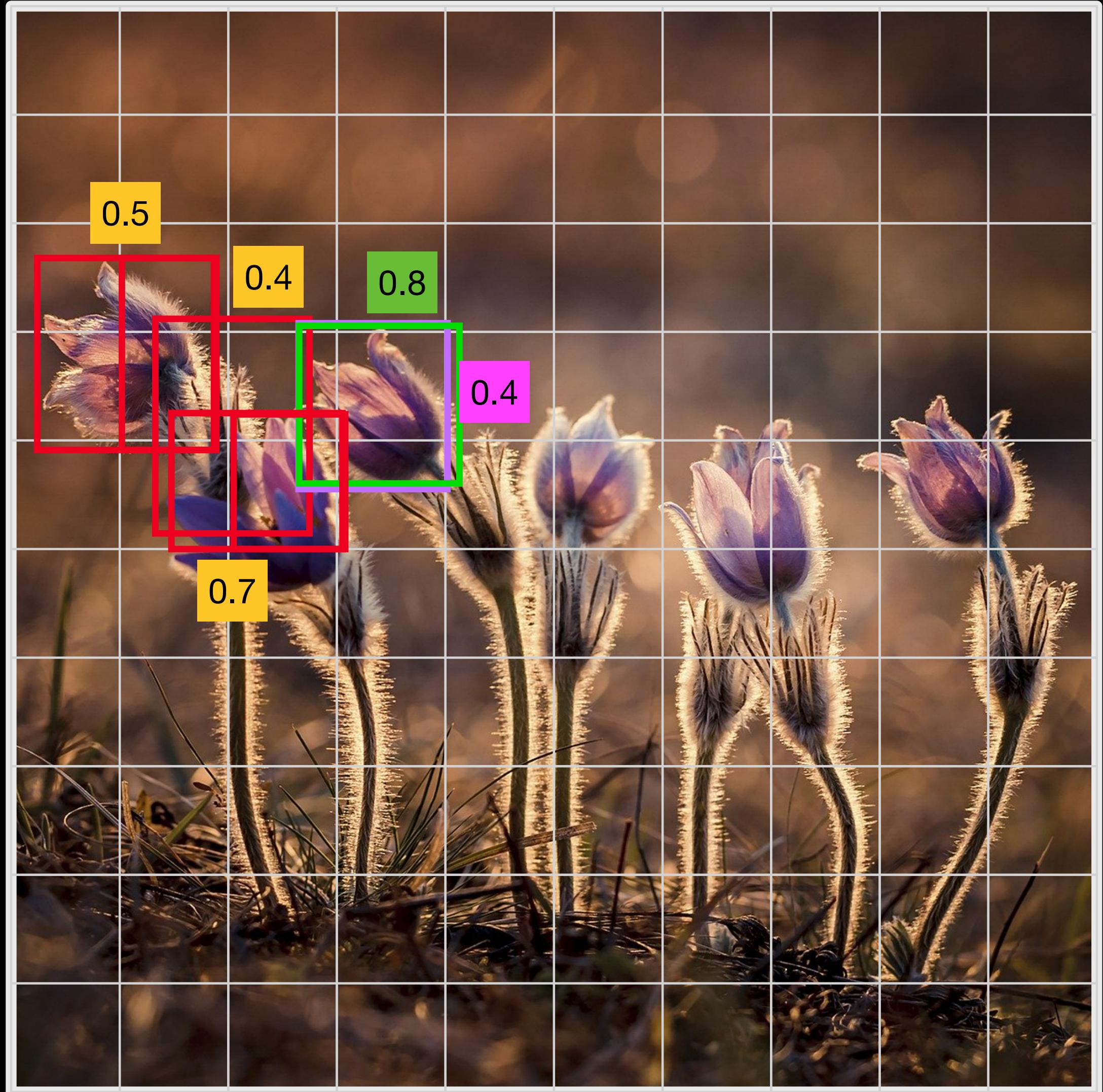
- 0.8과 0.4의 IoU를 비교, IoU가 **지정한 점수 (50%)** 이상이라면,
- 해당 영역 제안을 제거



사후처리

NMS(non-max suppression)

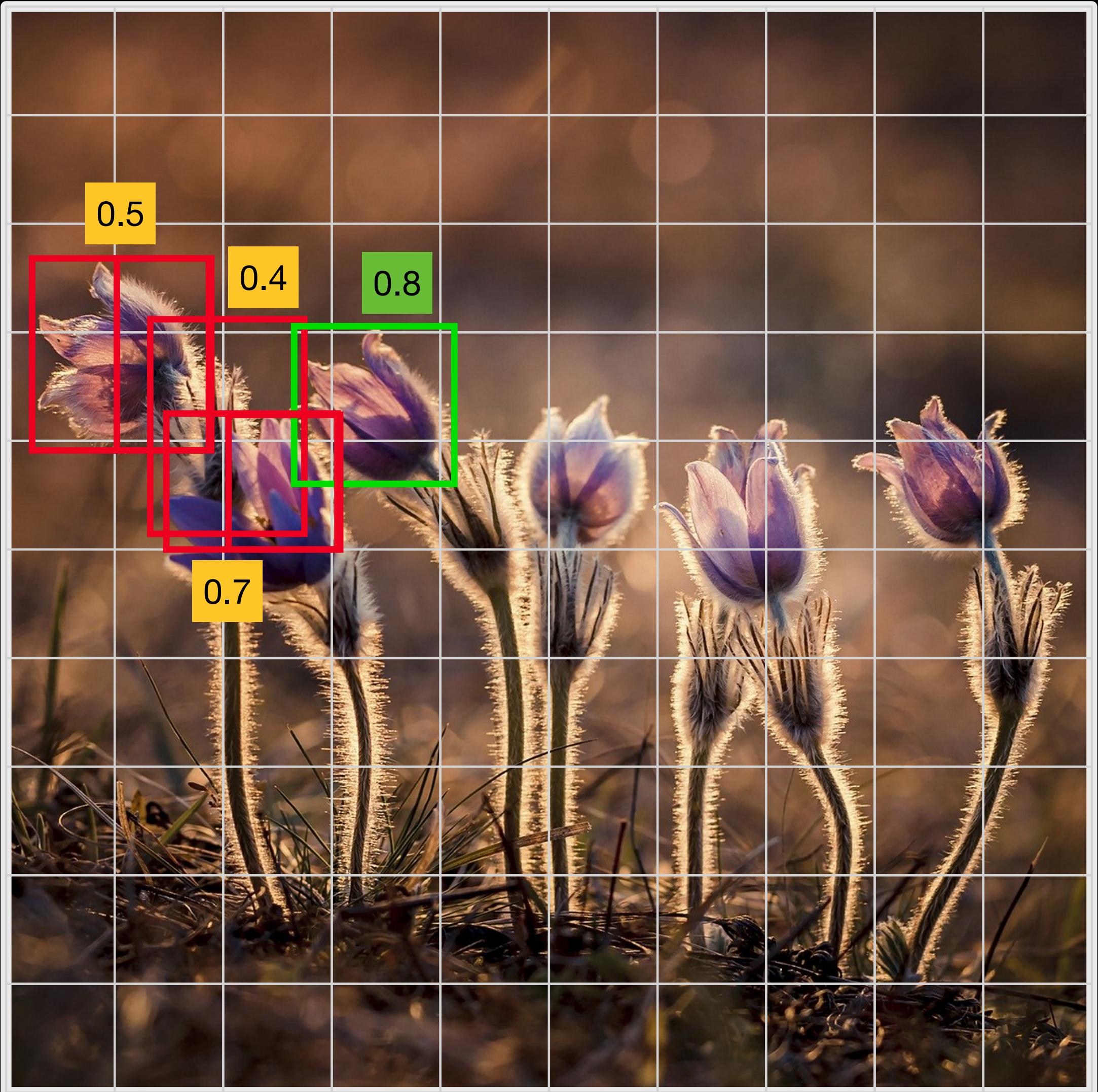
- 0.8과 0.4의 IoU를 비교, IoU가 **지정한 점수 (50%)** 이상이라면,
- 해당 영역 제안을 제거



사후처리

NMS(non-max suppression)

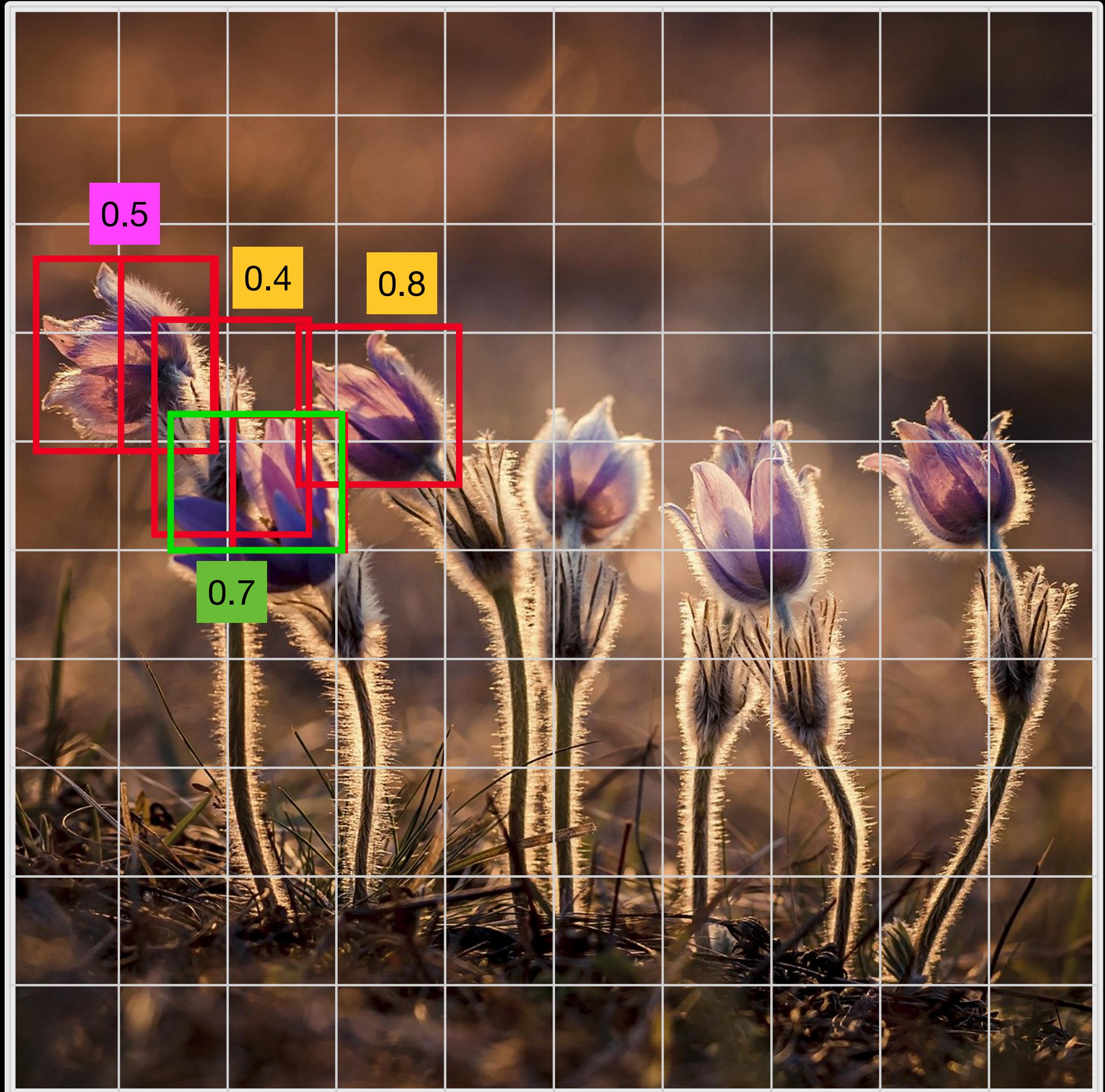
- 0.8과 0.4의 IoU를 비교, IoU가 **지정한 점수 (50%)** 이상이라면,
- 해당 영역 제안을 제거



사후처리

NMS(non-max suppression)

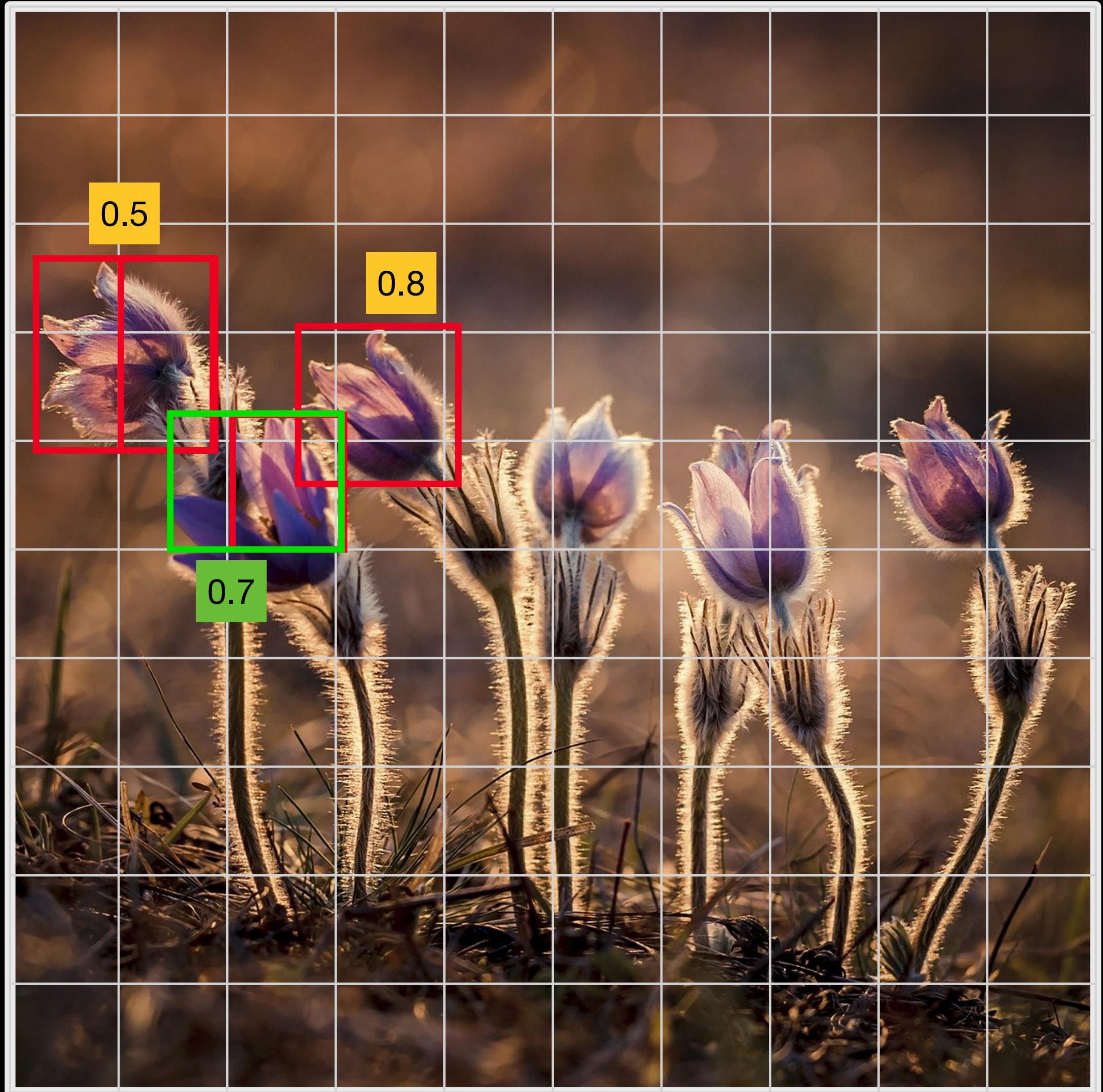
- 0.8에 대해서 다 끝났다면,
- 다음 단계인 0.7에 대해서 영역 비교 시작
- IoU가 지정된 값 이하면, 남겨두고, 이상이라면, 영역 제안 제거



사후처리

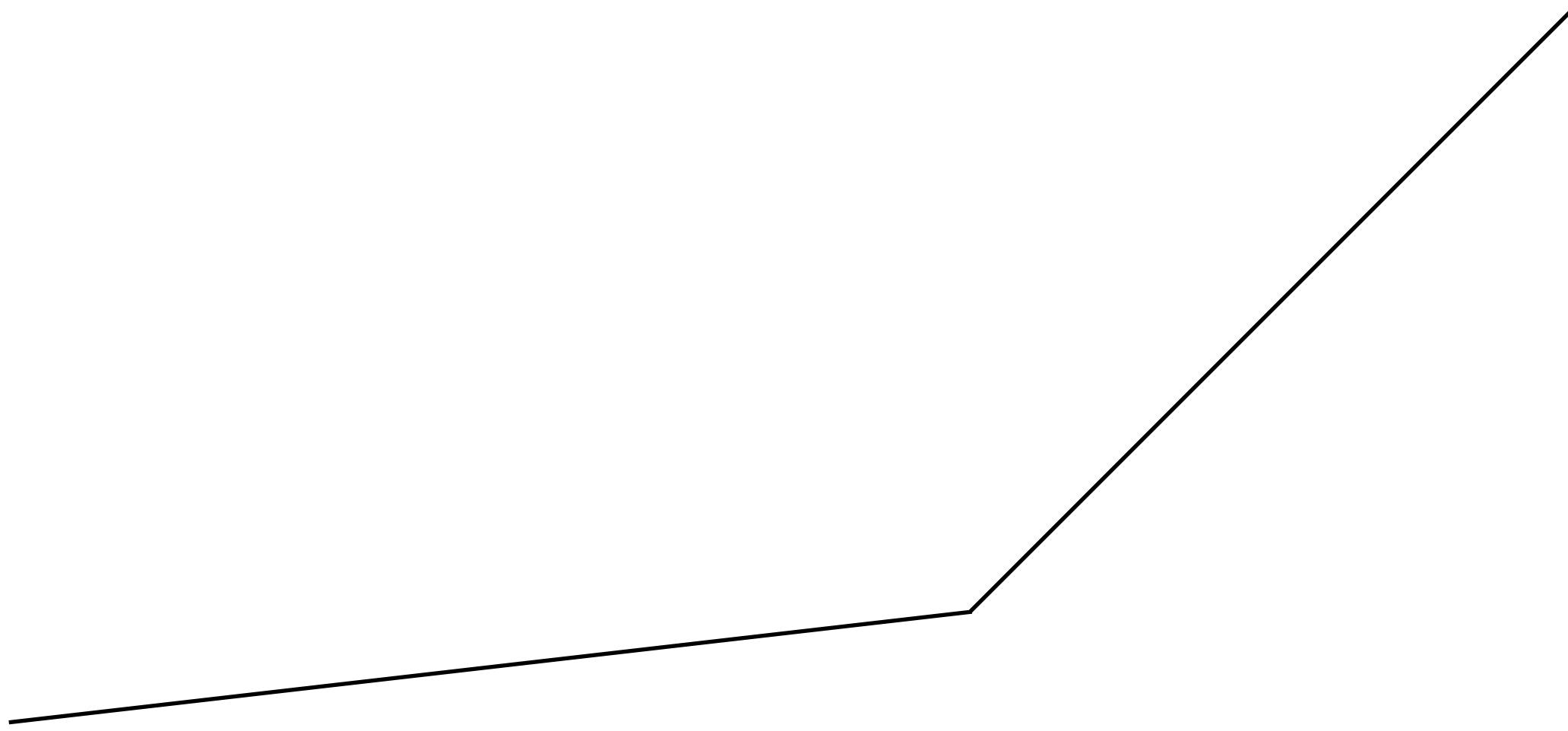
NMS(non-max suppression)

- 0.8에 대해서 다 끝났다면,
- 다음 단계인 0.7에 대해서 영역 비교 시작
- IoU가 지정된 값 이하면, 남겨두고, 이상이라면, 영역 제안 제거



Leaky ReLU

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$



LOSS

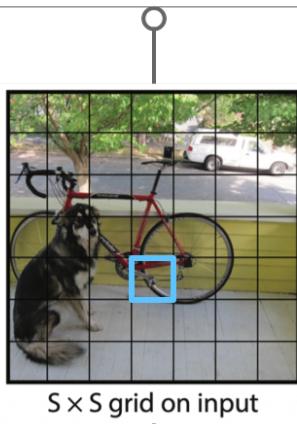
$$\begin{aligned} \mathbb{1}_{ij}^{\text{obj}} & \quad (1) \\ \mathbb{1}_{ij}^{\text{noobj}} & \quad (2) \\ \mathbb{1}_i^{\text{obj}} & \quad (3) \end{aligned}$$

- (1) Object가 존재하는 grid cell i의 predictor bounding box j
- (2) Object가 존재하지 않는 grid cell i의 bounding box j
- (3) Object가 존재하는 grid cell i

- Object가 존재하는 grid cell i의 predictor bounding box j에 대해, x와 y의 loss를 계산
- Object가 존재하는 grid cell i의 predictor bounding box j에 대해, w와 h의 loss를 계산. 큰 box에 대해서는 small deviation을 반영하기 위해 제곱근을 취한 후, sum-squared error를 한다.(같은 error라도 larger box의 경우 상대적으로 IOU에 영향을 적게 준다.)
- Object가 존재하는 grid cell i의 predictor bounding box j에 대해, confidence score의 loss를 계산. ($C_i = 1$)
- Object가 존재하지 않는 grid cell i의 bounding box j에 대해, confidence score의 loss를 계산. ($C_i = 0$)
- Object가 존재하는 grid cell i에 대해, conditional class probability의 loss 계산. (Correct class c: $p_i(c) = 1$, otherwise: $p_i(c) = 0$)

$$\begin{aligned} \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} & \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} & \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} & (C_i - \hat{C}_i)^2 \\ + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} & (C_i - \hat{C}_i)^2 \\ + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} & \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3) \end{aligned}$$

$$\begin{aligned}
& \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3)
\end{aligned}$$

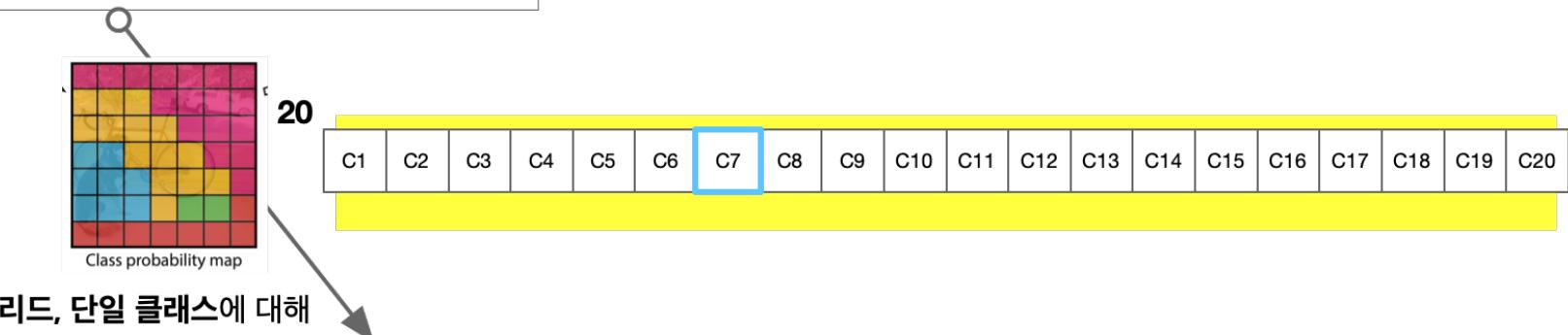
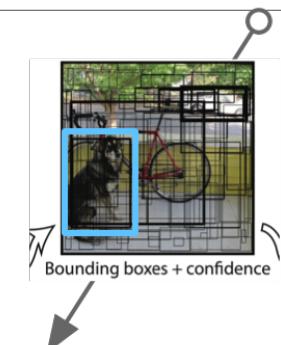


단일 그리드에서

$$\begin{aligned}
& \lambda_{\text{coord}} \sum_{j=0}^B \mathbb{1}_j^{\text{obj}} \left[(x_j - \hat{x}_j)^2 + (y_j - \hat{y}_j)^2 \right] \\
& + \lambda_{\text{coord}} \sum_{j=0}^B \mathbb{1}_j^{\text{obj}} \left[(\sqrt{w_j} - \sqrt{\hat{w}_j})^2 + (\sqrt{h_j} - \sqrt{\hat{h}_j})^2 \right] \\
& + \sum_{j=0}^B \mathbb{1}_j^{\text{obj}} (C_j - \hat{C}_j)^2 \\
& + \lambda_{\text{noobj}} \sum_{j=0}^B \mathbb{1}_j^{\text{noobj}} (C_j - \hat{C}_j)^2 \\
& + \sum_{j=0}^B \mathbb{1}_j^{\text{obj}} \sum_{c \in \text{classes}} (p_j(c) - \hat{p}_j(c))^2 \quad (3)
\end{aligned}$$

단일 그리드에서

$$\begin{aligned}
 & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{\text{obj}}^{\text{obj}} \left[(x - \hat{x})^2 + (y - \hat{y})^2 \right] \\
 & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{\text{obj}}^{\text{obj}} \left[(\sqrt{w} - \sqrt{\hat{w}})^2 + (\sqrt{h} - \sqrt{\hat{h}})^2 \right] \\
 & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{\text{obj}}^{\text{obj}} \left(C - \hat{C} \right)^2 \\
 & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{\text{obj}}^{\text{noobj}} \left(C - \hat{C} \right)^2 \\
 & + \sum_{i=0}^{S^2} \mathbb{1}_{\text{obj}}^{\text{obj}} \sum_{c \in \text{classes}} (p(c) - \hat{p}(c))^2 \quad (3)
 \end{aligned}$$



단일 그리드, 단일 경계 상자에서

$$\begin{aligned}
 & \lambda_{\text{coord}} \sum_{j=0}^B \mathbb{1}_{\text{obj}}^{\text{obj}} \left[(x - \hat{x})^2 + (y - \hat{y})^2 \right] \\
 & + \lambda_{\text{coord}} \sum_{j=0}^B \mathbb{1}_{\text{obj}}^{\text{obj}} \left[(\sqrt{w} - \sqrt{\hat{w}})^2 + (\sqrt{h} - \sqrt{\hat{h}})^2 \right] \\
 & + \sum_{j=0}^B \mathbb{1}_{\text{obj}}^{\text{obj}} \left(C - \hat{C} \right)^2 \\
 & + \lambda_{\text{noobj}} \sum_{j=0}^B \mathbb{1}_{\text{obj}}^{\text{noobj}} \left(C - \hat{C} \right)^2
 \end{aligned}$$

$$+ \mathbb{1}_{\text{obj}}^{\text{obj}} \sum_{c \in \text{classes}} (p(c) - \hat{p}(c))^2 \quad (3)$$

단일 그리드, 단일 클래스에 대해

단일 그리드, 단일 경계 상자에서

$$\lambda_{\text{coord}} \sum_{j=0}^B \mathbb{1}^{\text{obj}} [(x - \hat{x})^2 + (y - \hat{y})^2]$$

$$+ \lambda_{\text{coord}} \sum_{j=0}^B \mathbb{1}^{\text{obj}} [(\sqrt{w} - \sqrt{\hat{w}})^2 + (\sqrt{h} - \sqrt{\hat{h}})^2]$$

$$+ \sum_{j=0}^B \mathbb{1}^{\text{obj}} (C - \hat{C})^2$$

$$+ \lambda_{\text{noobj}} \sum_{j=0}^B \mathbb{1}^{\text{noobj}} (C - \hat{C})^2$$

단일 그리드, 단일 클래스에 대해

$$+ \mathbb{1}^{\text{obj}} \sum_{c \in \text{classes}} (p(c) - \hat{p}(c))^2 \quad (3)$$

단일 그리드, 단일 경계 상자에서 객체 존재

$$\lambda_{\text{coord}} \mathbb{1}^{\text{obj}} [(x - \hat{x})^2 + (y - \hat{y})^2]$$

$$+ \lambda_{\text{coord}} \mathbb{1}^{\text{obj}} [(\sqrt{w} - \sqrt{\hat{w}})^2 + (\sqrt{h} - \sqrt{\hat{h}})^2]$$

$$+ \mathbb{1}^{\text{obj}} (C - \hat{C})^2$$

단일 그리드, 단일 경계 상자에서 객체 미존재 confidence

$$+ \lambda_{\text{noobj}} \mathbb{1}^{\text{noobj}} (C - \hat{C})^2$$

단일 그리드, 단일 경계 상자에서 객체 존재

$$\lambda_{\text{coord}} \quad \mathbb{I}^{\text{obj}} \left[(x - \hat{x})^2 + (y - \hat{y})^2 \right]$$

$$+ \lambda_{\text{coord}} \quad \mathbb{I}^{\text{obj}} \left[(\sqrt{w} - \sqrt{\hat{w}})^2 + (\sqrt{h} - \sqrt{\hat{h}})^2 \right]$$

$$+ \quad \mathbb{I}^{\text{obj}} \left(C - \hat{C} \right)^2$$

단일 그리드, 단일 경계 상자에서 객체 미존재 confidence

$$+ \lambda_{\text{noobj}} \quad \mathbb{I}^{\text{noobj}} \left(C - \hat{C} \right)^2$$

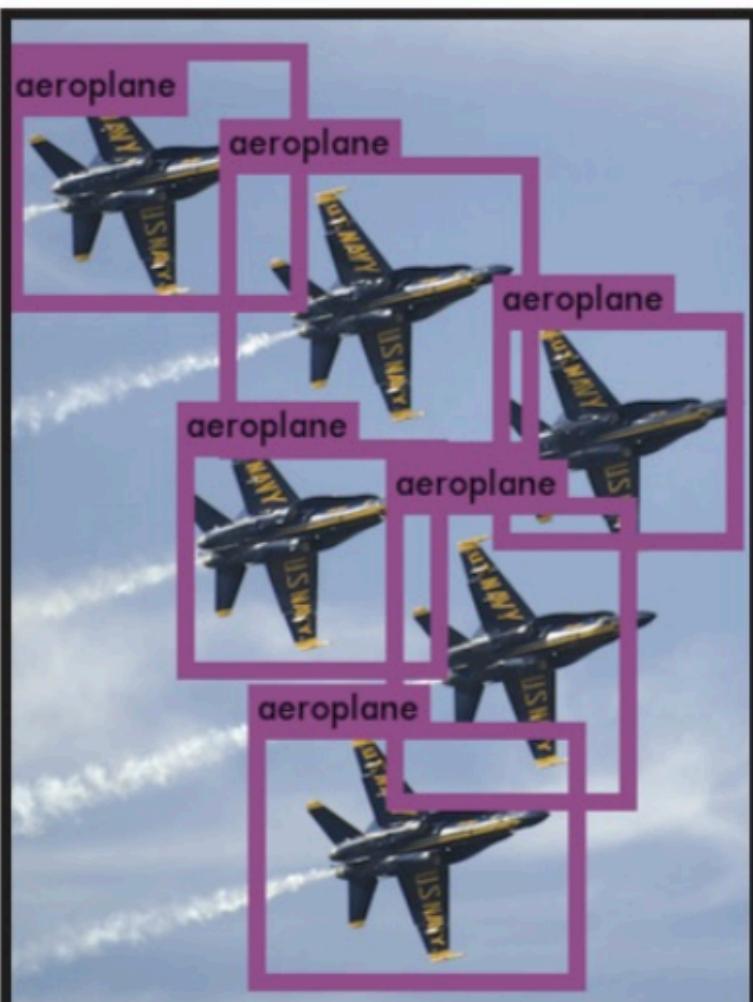
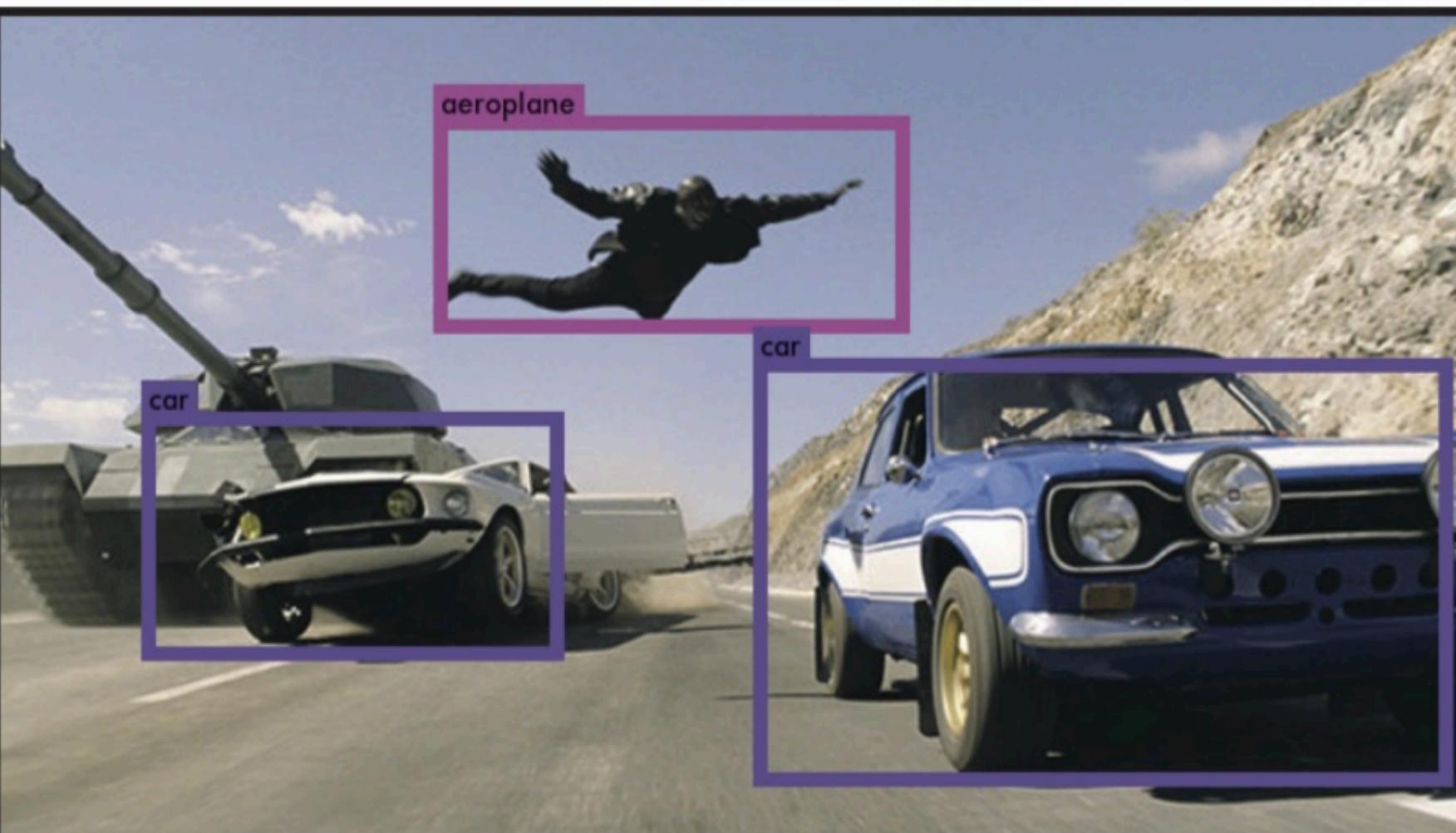
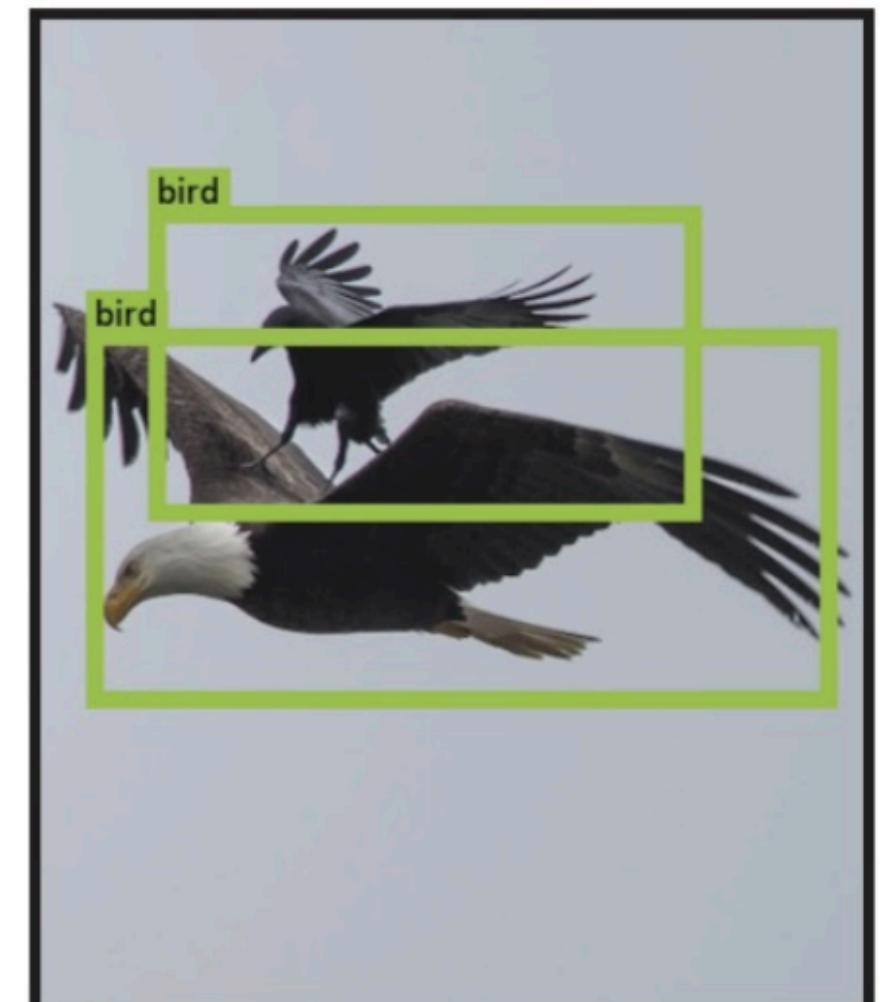
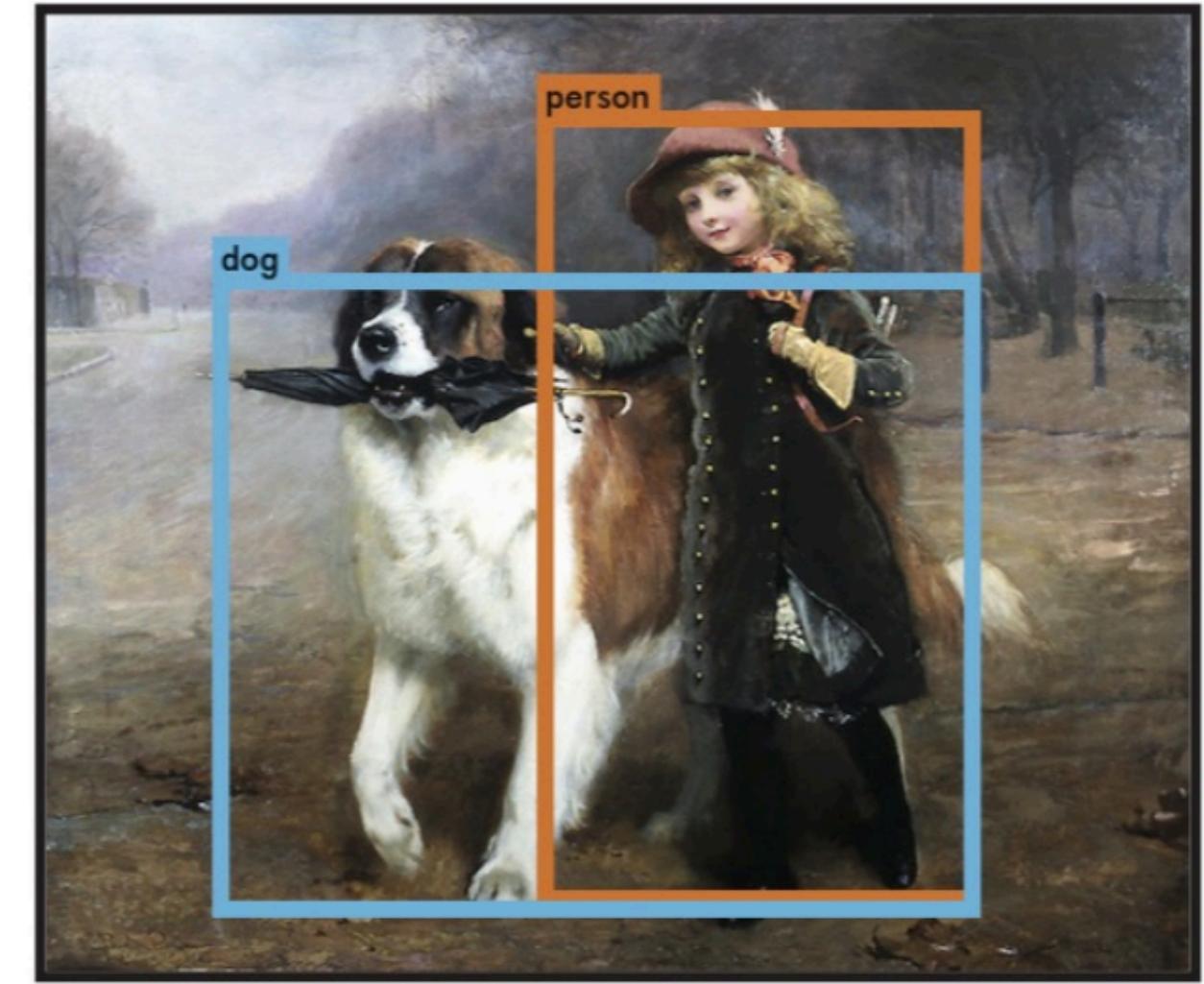
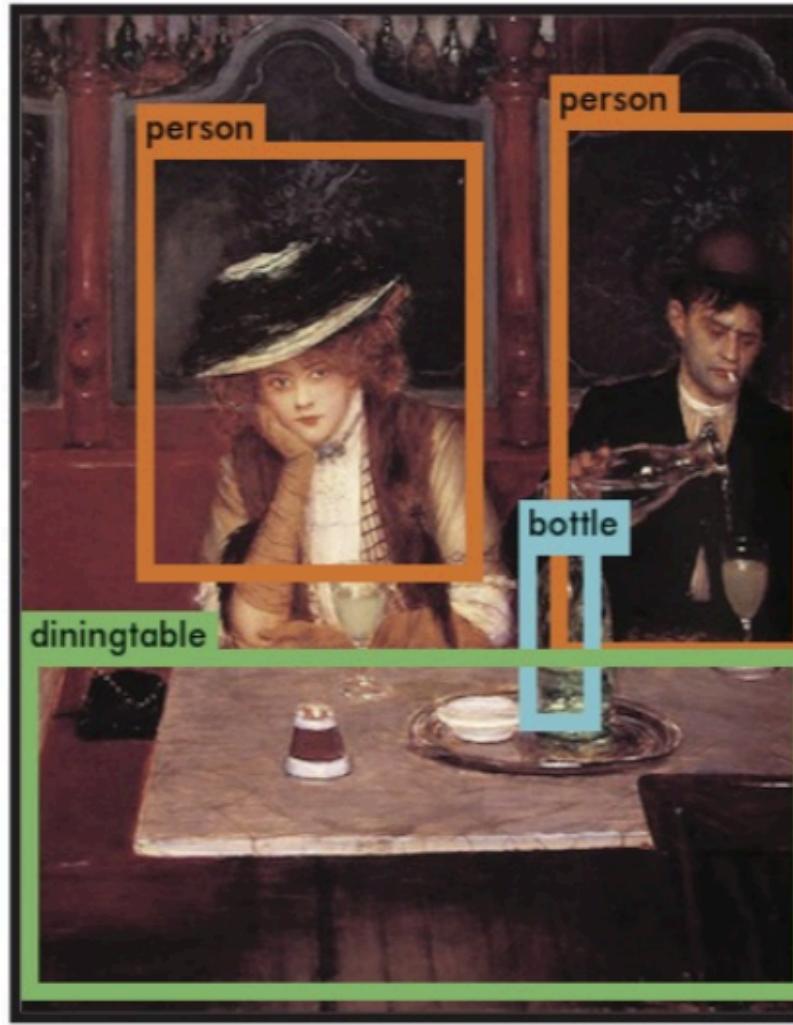
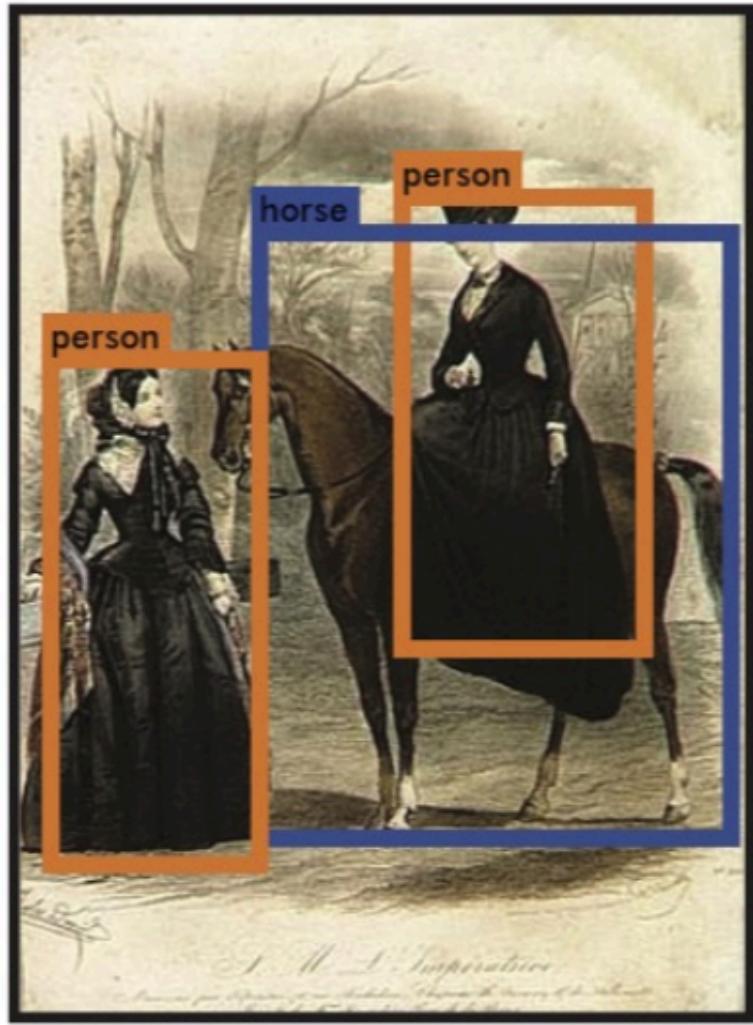
단일 그리드, 단일 경계 상자에서 객체 존재, 좌표 및 크기

$$\lambda_{\text{coord}} \quad \mathbb{I}^{\text{obj}} \left[(x - \hat{x})^2 + (y - \hat{y})^2 \right]$$

$$+ \lambda_{\text{coord}} \quad \mathbb{I}^{\text{obj}} \left[(\sqrt{w} - \sqrt{\hat{w}})^2 + (\sqrt{h} - \sqrt{\hat{h}})^2 \right]$$

$$+ \quad \mathbb{I}^{\text{obj}} \left(C - \hat{C} \right)^2$$

단일 그리드, 단일 경계 상자에서 객체 존재 confidence



Semantic Segmentation을 활용한 차량 파손 탐지 딥러닝 모델 개발기



차량 파손 탐지 모델 배경

- 1) 사용자가 차량을 이용할 쏘카존을 선택합니다.
- 2) 원하는 시간을 선택하고,
- 3) 원하는 차량을 선택하게 됩니다.
- 4) 약속한 이용 시간이 가까워지면 선택한 쏘카존에 방문해, 차량의 상태를 확인합니다.
 - 이때 사용자는 차량의 당시 외관 사진을 앱 내에 업로드해야 합니다.
- 5) 문제가 없는 경우, 운행을 시작합니다.

The image displays three screenshots of a mobile application interface for car damage detection:

- Screenshot 1: Home Screen**

Shows the title "차량 파손 탐지 모델 배경".
- Screenshot 2: Vehicle Inspection Start Screen**

Shows the title "차량 확인하기" and a sub-section "외관 촬영 시작하기". It includes a note: "차량 외관에서 흠집이나 사고 흔적을 발견했다면 반드시 촬영해주세요. 운행이 불가능한 손상이 있다면 고객센터로 문의해주세요." A blue button labeled "외관 촬영 시작하기" is at the bottom, and a link "꼭 해야 하나요 ⓘ" is above it.
- Screenshot 3: Vehicle Inspection Result Screen**

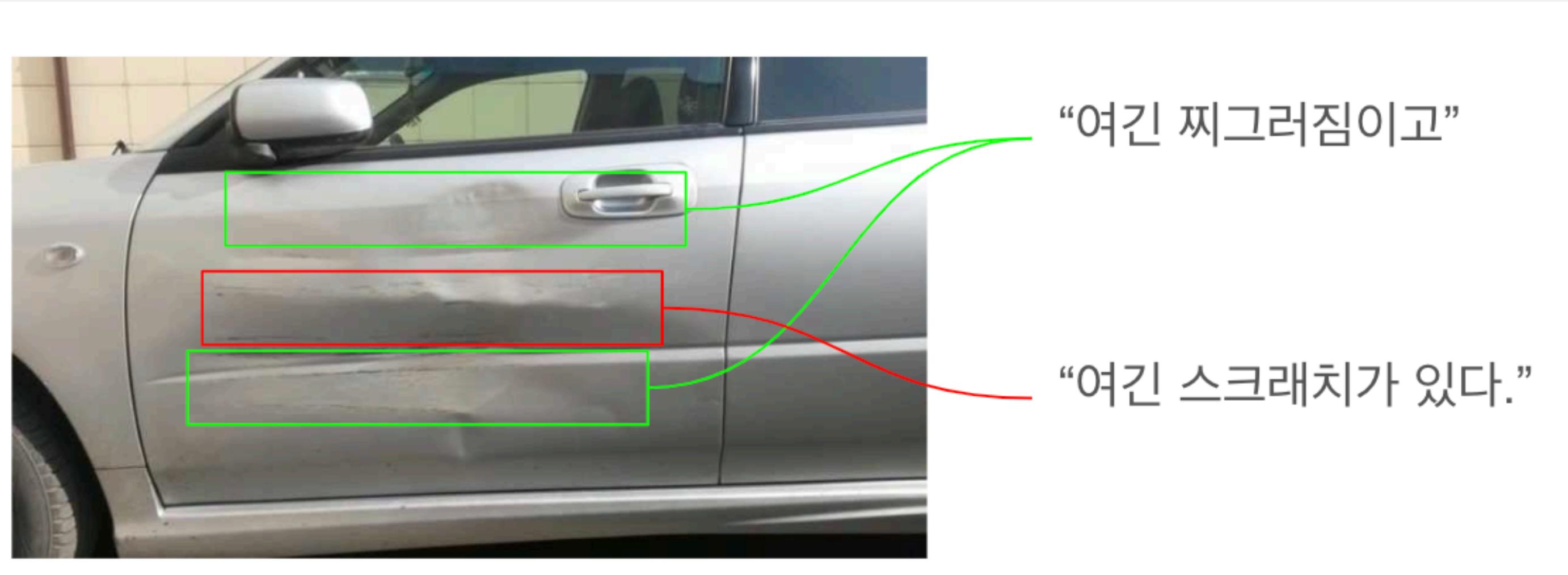
Shows the title "외관 촬영" and a note: "차량의 여섯 면을 가이드에 맞춰 촬영해주세요. 사진 전송 후에는 수정할 수 없습니다." Below this are six camera icons representing different vehicle angles. A section titled "운행 전 추가 확인" shows a note: "차량 내부 상태와 관련된 태그를 선택하거나 메모를 남겨주세요. 차량 계기판에 경고등이 들어와있다면 고객센터로 문의해주세요." Below this are two rows of buttons:
 - Top row: 동물털, 음식물, 흙, 먼지, 모래, 담배냄새, 악취
 - Bottom row: 쓰레기, 끈적임, 오염, 타인물품, 창문얼룩A note below these buttons says: "불편사항이 태그에 없나요? 메모로 남겨주세요." A large blue button at the bottom right says "외관에 이상이 없습니다".

문제 정의

- 차량의 파손 상태를 꾸준히 모니터링
- 기존 업무 프로세스는 업무 담당자분들이 차량 외관 이미지를 직접 검수
- 일평균 7-8만장, 최대 11만장의 차량 외관 이미지가 업로드.
- **파손 시점 추적**은 담당자가 직접 차량의 외관 이미지를 현재부터 과거로 추적하며 파손 시점을 찾아내는 식으로 진행되었습니다.

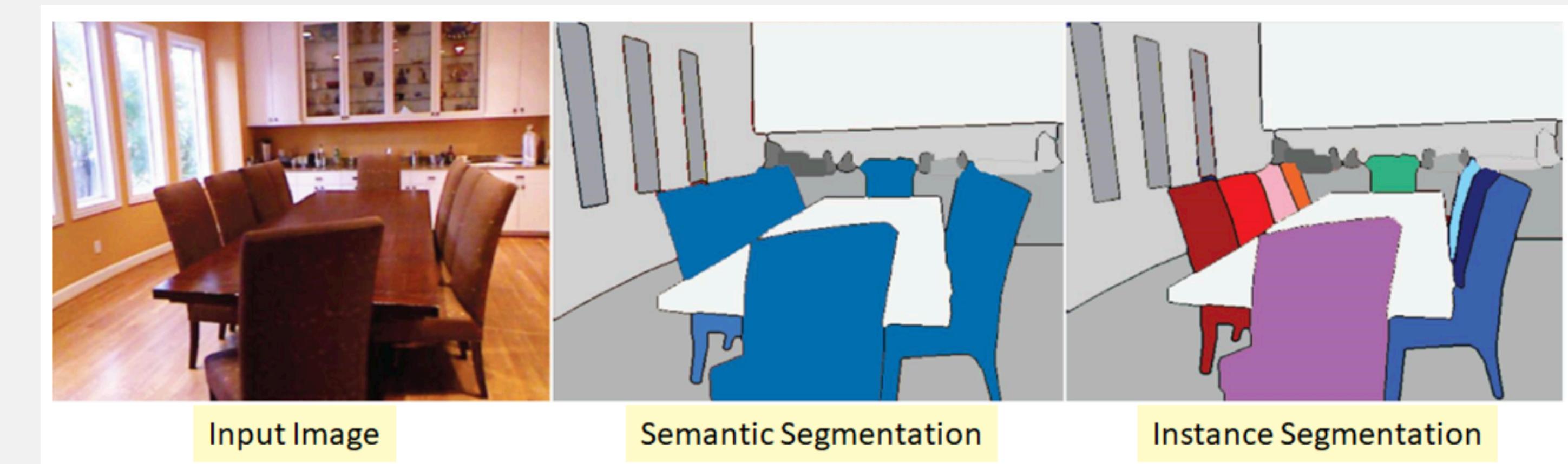
문제 해결 방식

- 딥러닝 모델을 이용한 차량 파손 탐지 자동화 프로젝트
- 구체화가 필요
 - 딥러닝 분야의 다양한 문제 접근 방식 중, 어떤 Task인지?
 - 어떤 데이터를 사용할 것인지?
 - 어떤 구조의 모델을 사용하여 문제 해결 목표를 달성할 것인지?



문제 접근 방식 정의

- Classification
 - 무엇이 있는가?
- Localization
 - 어디에 있는가?
- Object Detection
 - 무엇이, 어디에 있는가?
- Semantic Segmentation
 - 내가 찾고자 하는 물체의 종류 및 위치 (클래스)
- Instance Segmentation
 - 내가 찾고자 하는 객체들의 위치 (객체)
 - 어떤 객체인지는 모름



데이터 정의 및 준비

- 데이터 정의
 - 이미지 단위의 파손 존재 여부 (이미지 전체에서 파손이 존재하는지)
 - 픽셀 단위의 파손 클래스 분류
 - 스크래치 (Scratch) : 차량에 스크래치가 난 영역
 - 찌그러짐 (Dent): 차량이 찌그러진 영역
 - 이격 (Spacing): 차체 패널이 벌어져 들뜸, 틈이 생긴 영역
 - 해당 없음 (Background): 파손이 존재하지 않는 영역
 - 입력 데이터(Feature)는 차량 이미지가 될 것이며, 출력 데이터(Label)는 이미지 단위 파손 존재 여부와 픽셀 단위 파손 클래스 분류

```
{"image_id": "20190218_12985_20129968_2e0da3c3e2ace9c780abb1eeb060d43f.jpg",
"damage_level": 1,
"regions": [
{
"points": [
[245.0, 260.0], [285.0, 228.0], [333.0, 211.0], [390.0, 209.0],
[448.0, 216.0], [538.0, 184.0], [594.0, 174.0], [595.0, 204.0],
[524.0, 220.0], [475.0, 240.0], [464.0, 291.0], [453.0, 366.0],
[424.0, 415.0], [379.0, 448.0], [312.0, 456.0], [271.0, 442.0],
[221.0, 377.0]
],
"class_name": 1
}
]
```

- image_id: 이미지 파일명
- damage_level: 이미지 단위 파손 존재 여부. 파손이 존재할 경우 1, 파손이 존재하지 않을 경우 0
- regions: 파손 영역 단위 파손 클래스 분류
데이터 관리의 편리함을 위해 픽셀 단위로 정의하는 mask 방식이 아닌, 영역 단위로 정의하는 polygon 방식을 채택했습니다. 학습 시, 해당 polygon 정보를 픽셀 단위 mask로 변환해 이용했습니다.
- points: 해당 파손 영역의 외곽선 좌표 배열
- class_name: 해당 파손 영역의 종류
- 0: 해당 없음 (Background)
- 1: 스크래치 (Scratch)
- 2: 찌그러짐 (Dent)
- 3: 이격 (Spacing)

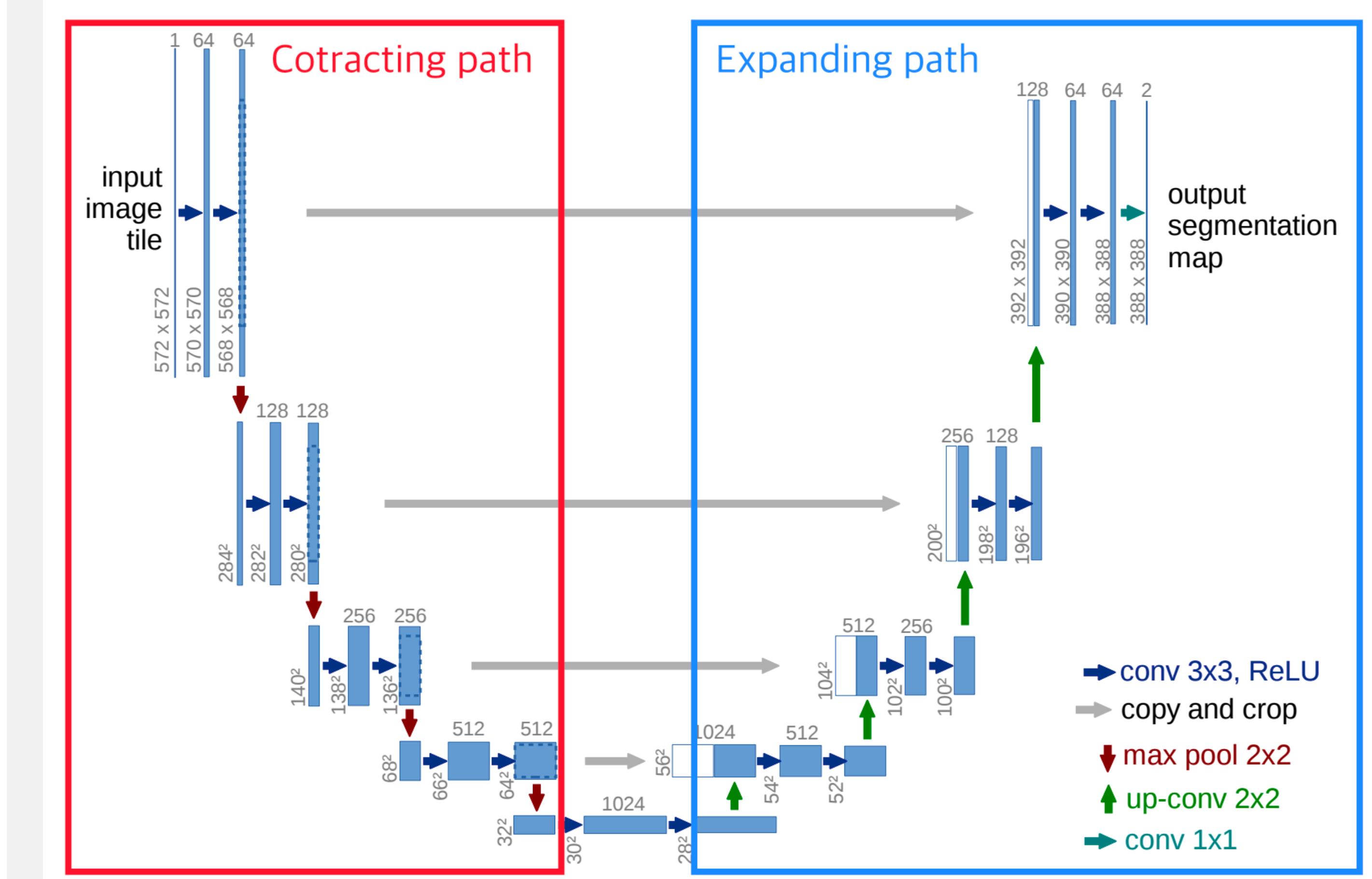
데이터 정의 및 준비

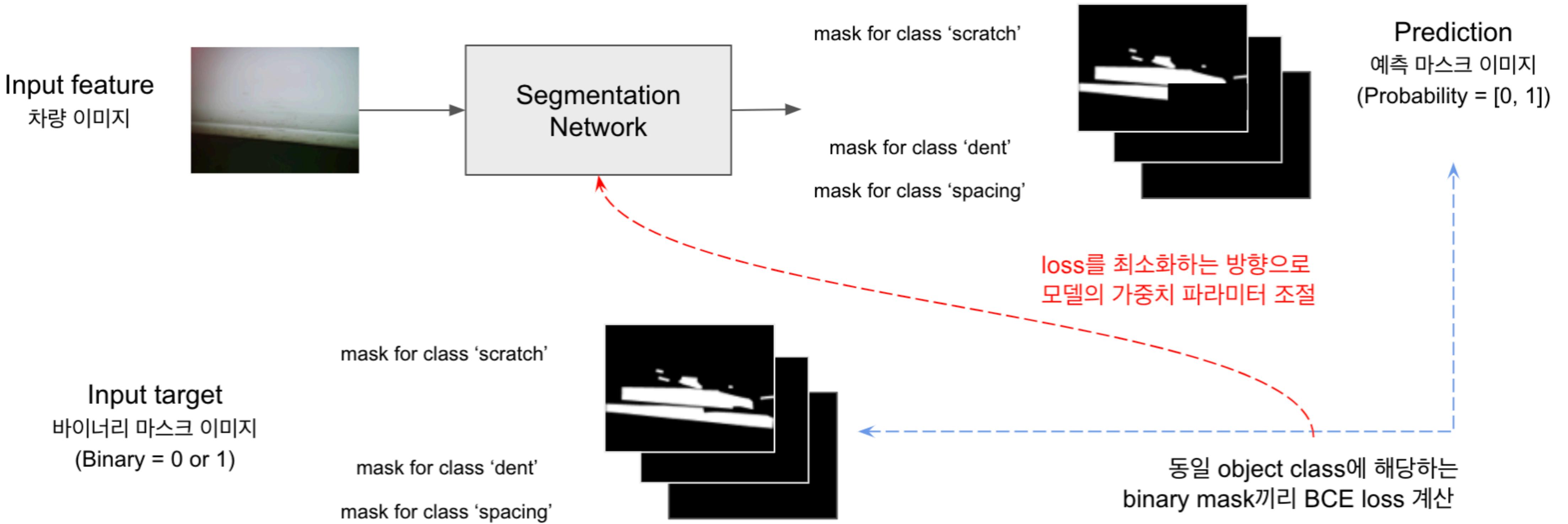
Dataset 분리

- 정의된 데이터를 이용 목적에 따라 분리
- 전체 2,000개 데이터. Training, Validation, Test Set 8:1:1.
- 데이터 세트 분리 시, 특정 데이터 세트에 데이터가 편향되는 것을 막기 위해 Stratified Split을 사용
 - 데이터가 편향되는 것은 짜그러진 파손에 해당하는 이미지 대부분이 Validation set에만 포함되어 있으면 학습시 해당 특징이 모델에 충분히 반영되지 못하는 경우를 의미
- 해당 프로젝트에서는 파손 클래스와 전체 이미지 면적 대비 파손 영역이 차지하는 면적 비율이 편향되지 않도록 설정해 Training Set, Validation Set, Test Set으로 분리

사용한 모델의 구조

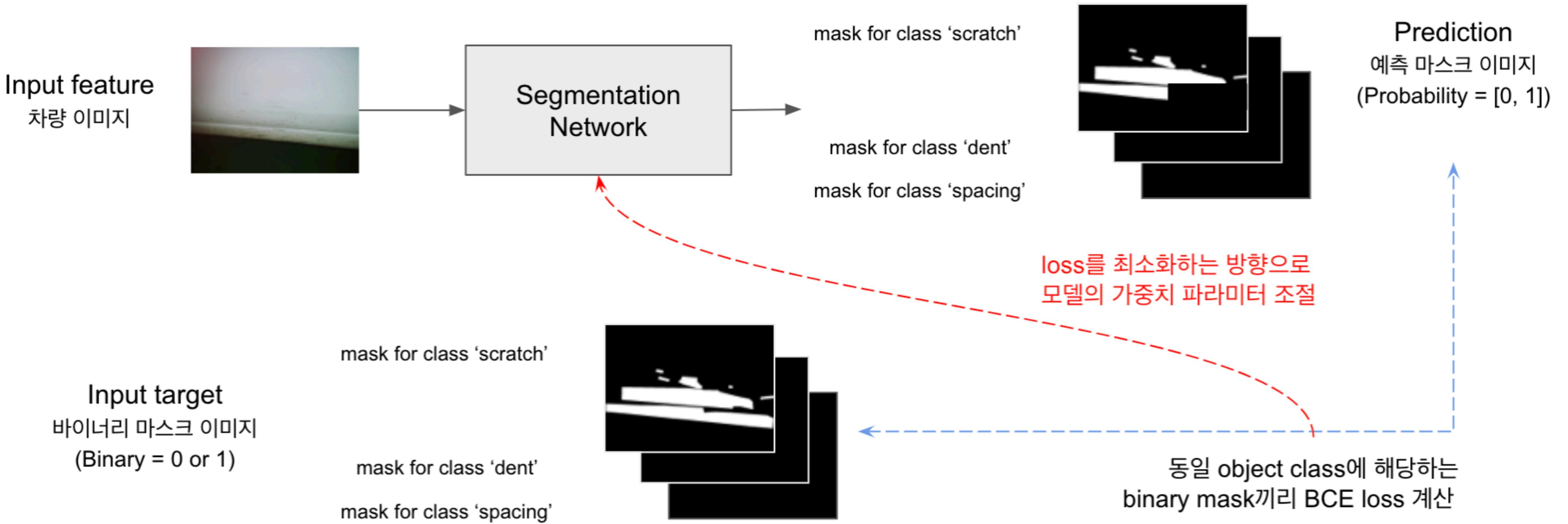
- ImageNet 데이터 세트로 학습된 EfficientNet을 Contracting Path의 Backbone으로 사용
- 베이스라인 성능 비교 결과 DeepLab v3은 mIOU 80.7, U-Net은 mIOU 92.2를 기록





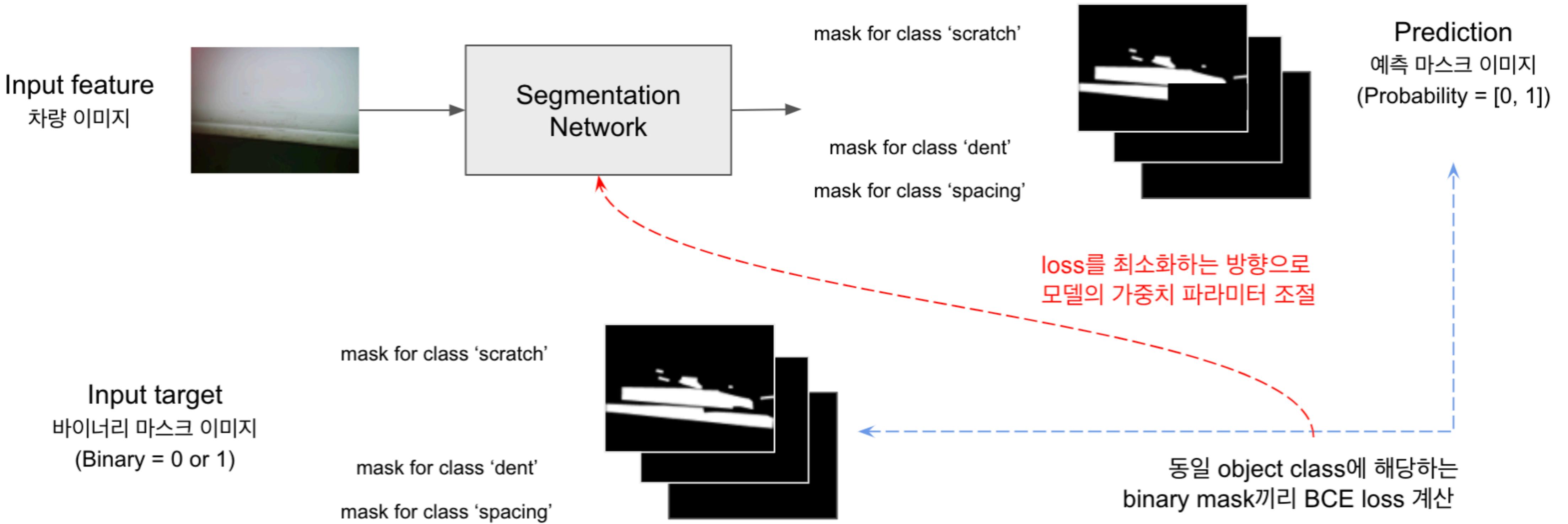
- Feature를 입력으로, 모델의 각 계층(Layer)의 파라미터와의 연산을 통해 예측값을 도출 (Prediction)
- 모델이 예측한 값과 실제 값(Target)이 얼마나 차이가 있는지 (Loss) 계산
- 이 차이(Loss)를 최소화하는 방향으로 모델 각 계층의 파라미터를 조절하여 최적화

모델 학습(Training) 과정



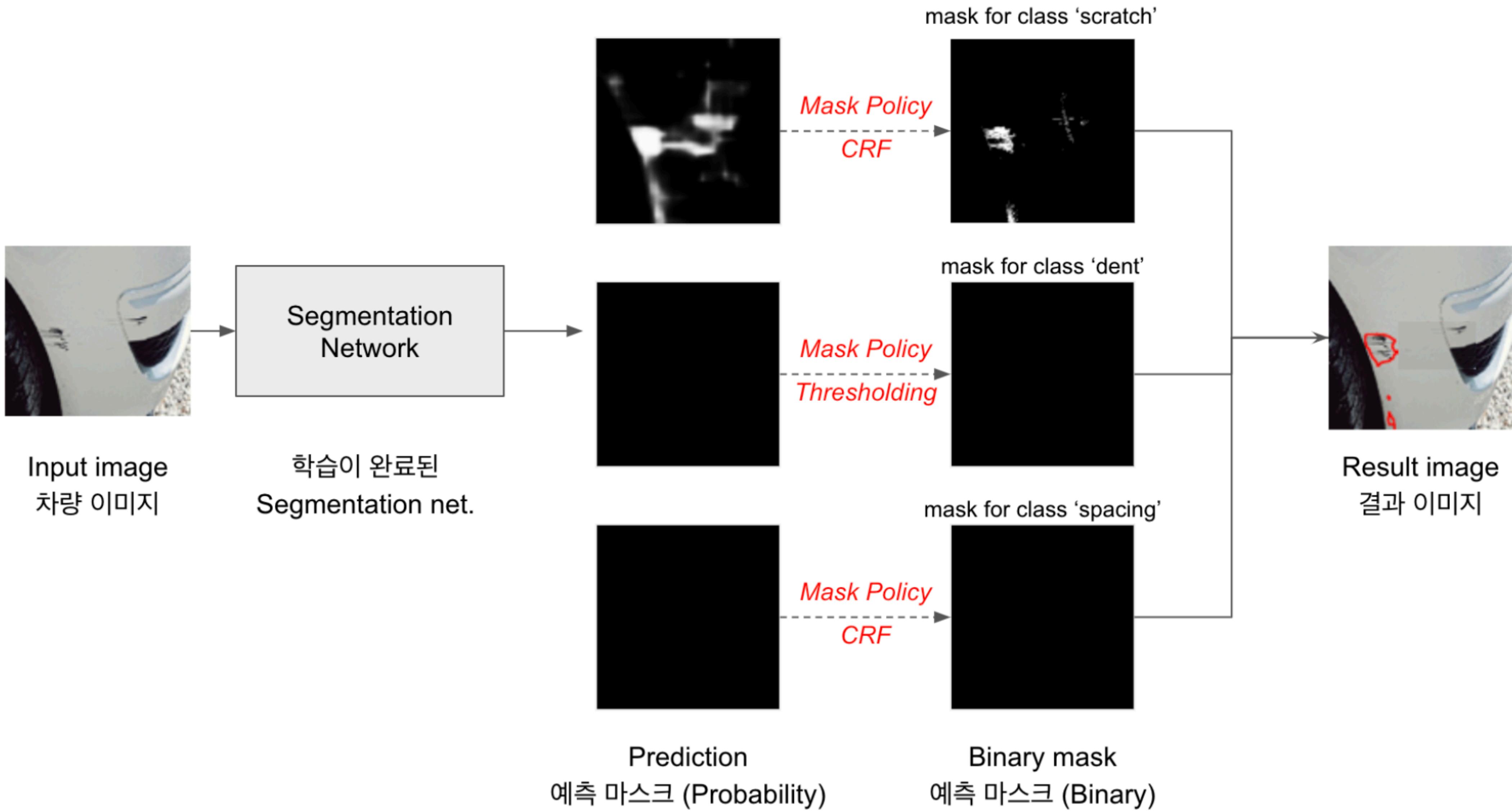
- Feature는 차량 이미지, Target으로는 마스크(Binary Mask Image)가 사용
- 파손 영역별 좌표 형태(Polygon)로 주어진 입력 파일을 마스크 형식으로 가공
- 하나의 파손 클래스당 한 장, 총 3장의 2차원 이진 마스크

모델 학습(Training) 과정



- 출력되는 값은 Target과 동일한 형태로 3장의 2차원 예측 마스크의 형태
- 출력되는 예측 마스크는 이진(Binary) 마스크가 아닌, 대응되는 입력 이미지의 픽셀별로 해당 파손 클래스에 속할 확률(Probability Score)을 나타내는 마스크
- 각 파손 클래스마다 해당 클래스의 Target 마스크와 예측 마스크 사이의 오차를 계산
- 오차 계산을 위해 Binary Cross Entropy(BCE) 함수를 사용

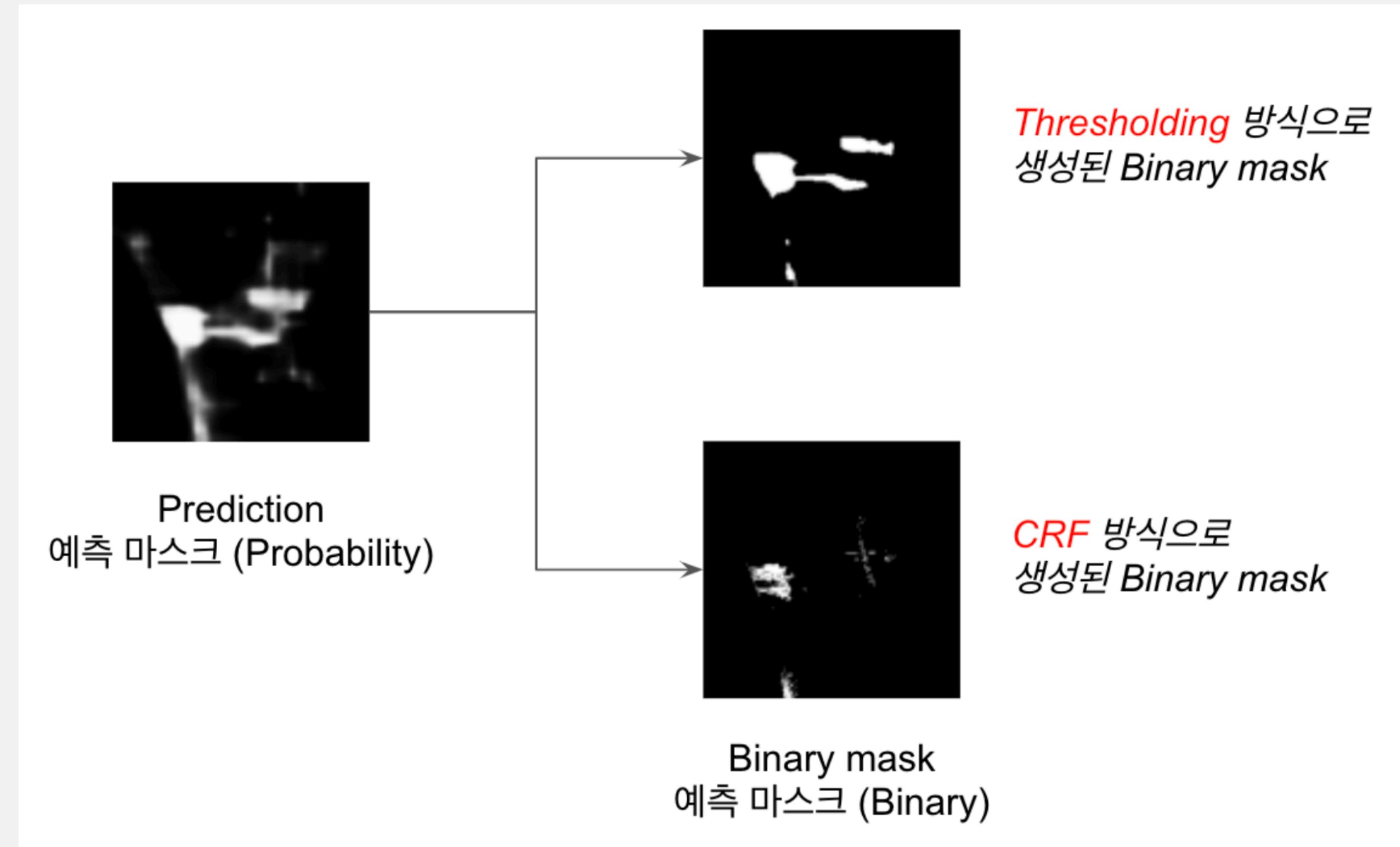
모델 학습(Training) 과정



모델 Inference(Prediction) 후처리

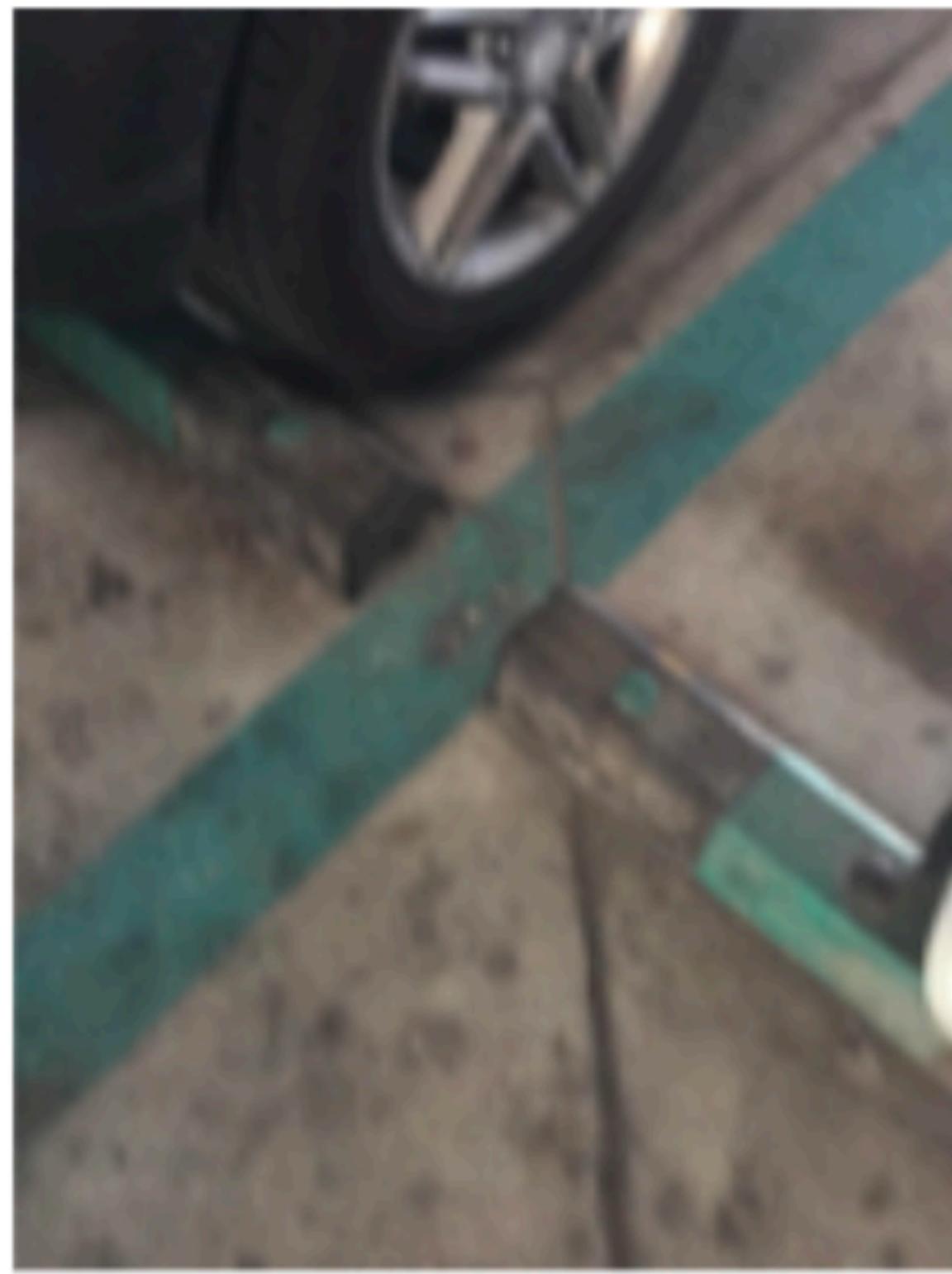
모델 학습 (Training) 과정

- 해당 프로젝트에서 고안해 사용한 Mask Policy는 각 파손 클래스 별 특성을 반영한 규칙
 - 스크래치 (Scratch), 찌그러짐 (Dent), 이격 (Spacing)
 - 스크래치와 이격의 경우, 세밀한 영역 경계 설정이 필요(CRF)하고,
 - 찌그러짐의 경우 비교적 넓은 면적으로 발생할 가능성이 높은 파손(Thresholding)



실제 데이터 검증 시 생긴 문제

- 차량의 파손 상태를 꾸준히 모니터링
- 기존 업무 프로세스는 업무 담당자분들이 차량 외관 이미지를 직접 검수
- 일평균 7-8만장, 최대 11만장의 차량 외관 이미지가 업로드.
- **파손 시점 추적**은 담당자가 직접 차량의 외관 이미지를 현재부터 과거로 추적하며 파손 시점을 찾아내는 식으로 진행되었습니다.



실제 데이터 검증 시 생긴 문제

- 어두운 곳에서 촬영된 차량 이미지에 취약



실제 데이터 검증 시 생긴 문제

시간적인 제한에 대한 문제

- Semantic Segmentation을 수행하기 위하여, 모델은 픽셀 단위로 해당 픽셀이 각 클래스에 속할 확률을 예측해야.
- 이를 위해 모델의 구조가 **굉장히 깊고 복잡**해지는 것은 불가피한 일이며, **방대한 연산량**으로 연산 속도 저하의 위험성
 - CPU 머신 위에서 실제 테스트 시, 차량 이미지 한 장 당 약 15초의 처리 시간
- 이는 하루 평균 7-8만장의 이미지를 처리해야 하는 업무 상황에 적절하지 못하다고 판단
- 이러한 문제점들을 해결하기 위해, 1) 데이터 측면에서 보완할 수 있는 점과 2) 모델 측면에서 보완할 수 있는 점으로 구분해 전체적인 틀을 수정했습니다.

문제점 보완

데이터 측면에서의 보완

- Data Augmentation을 사용
- 어두운 곳에서 촬영된 이미지에 취약점을 보이는 문제점을 해결하기 위하여, Training Set에 무작위로 사진의 밝기와 대비를 변경하는 전처리 (Random Brightness & Contrast Transformation)

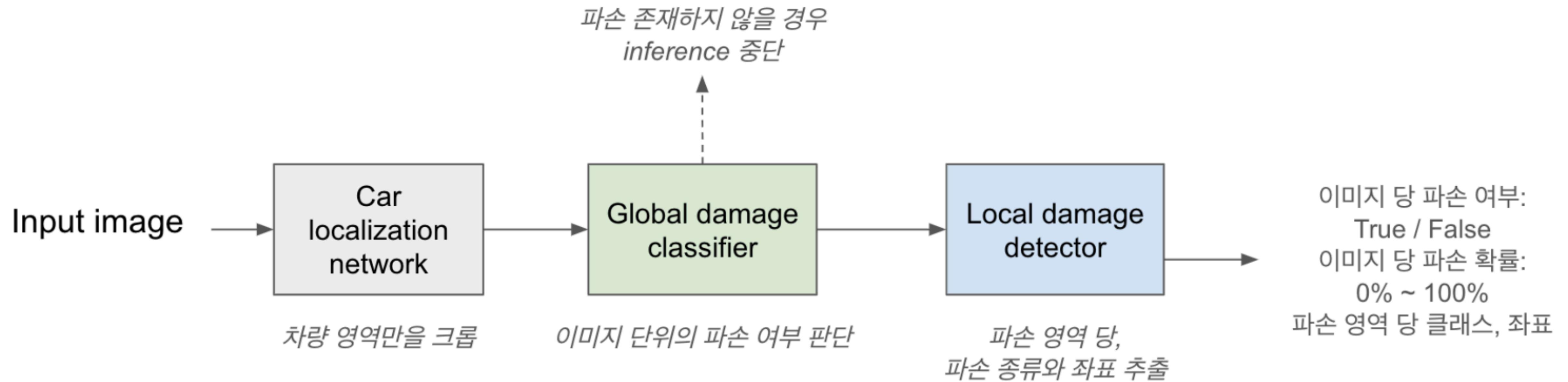
문제점 보완

모델 측면에서의 보완

- 사용자들이 **사진을 촬영하는 방식이 제각각이라는 점을 해결하기 위해 모델의 시작 부분에 Localization Network를 추가했습니다.** 즉, 이미지 전체에서 차량이 존재하는 영역만을 Crop해 이후 모델의 입력으로 사용하겠다는 전략이었습니다.
 - Localization Network는 COCO 데이터 세트로 미리 학습된 **YOLO v3 모델**을 사용했습니다.
- 모델의 연산량으로 인한 처리 속도 저하 해결을 위해 **Global Damage Classifier**를 추가하는 방법을 택했습니다. 실제로 쏘카 앱을 통해 업로드되는 이미지 중 **손상이 존재하는 이미지는 소수**이기 때문에, 굳이 **모든 이미지를 깊은 Segmentation Network에 통과시킬 필요가 없다**고 생각했습니다.
- Segmentation Network 바로 앞단에 이미지 단위의 **파손 존재 여부를 0과 1로 예측하는 비교적 얇은 Classification Network를 추가**했고, 이를 Global Damage Classifier라는 이름으로 칭하겠습니다.
- Global Damage Classifier의 예측 결과가 “**이 이미지 내에는 파손이 없다. (=0, False)**”로 정해진 이미지에 대해서는 더 이상의 Inference를 진행하지 않는 방법을 택했습니다. 이미지 내에 파손이 있다고 판단된 이미지에 대해서만 Segmentation Network 내의 연산을 통해 파손 영역 및 클래스 예측을 수행하게 됩니다. = 일종의 파이프라인.

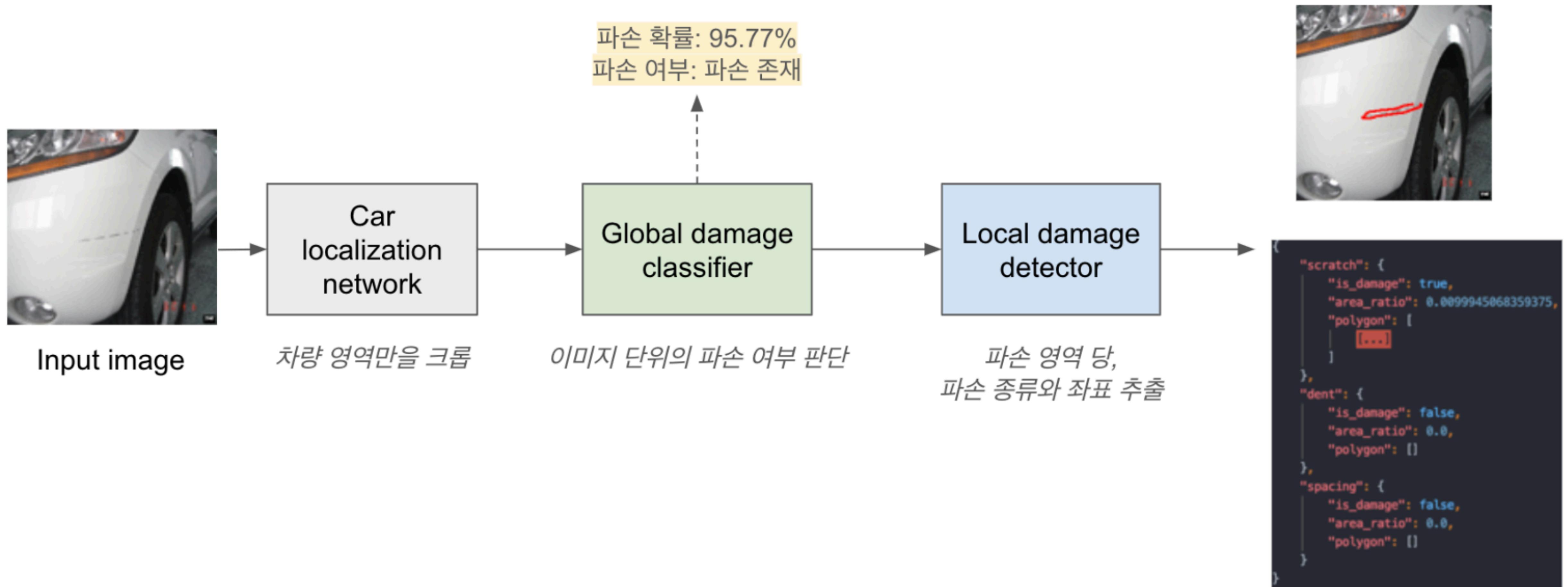
문제점 보완

모델의 전체 구조



문제점 보완

전체 Inference 흐름도



성능 평가

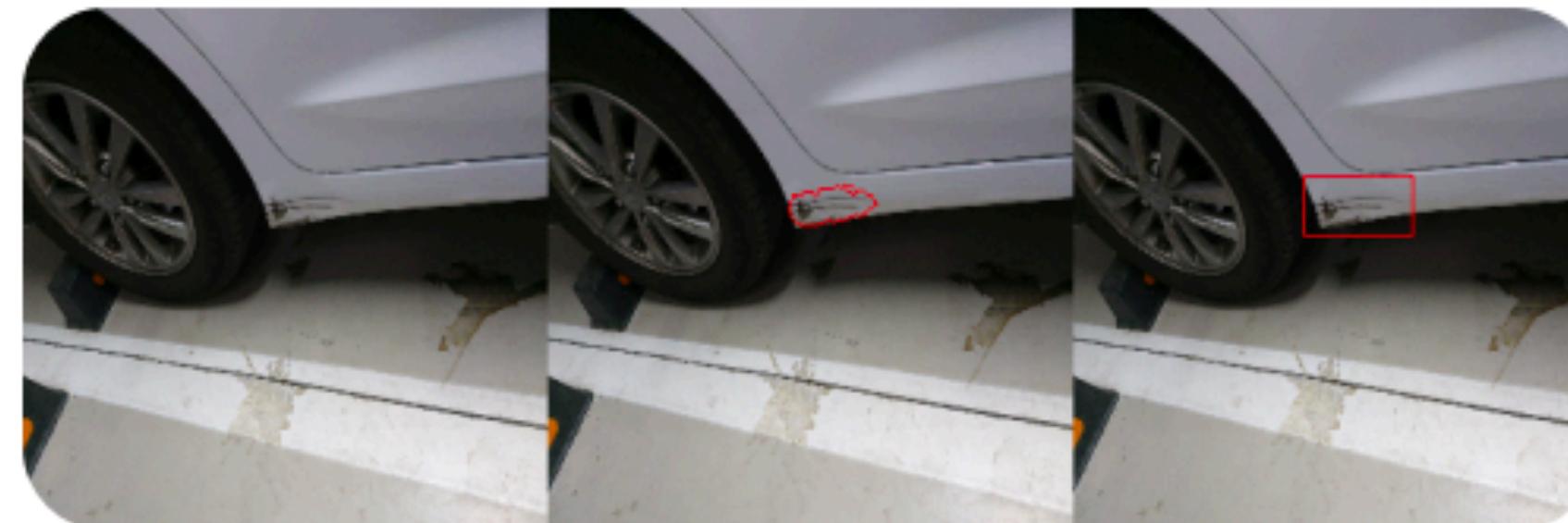
- 모델의 처리 속도
 - 이렇게 완성된 모델은 GPU 머신 위에서 초당 약 5장의 이미지를, CPU 머신 위에서 초당 약 0.7장의 이미지를 처리할 수 있습니다.
- 모델의 정확도
 - 성능 평가를 위해 분리해둔 200장의 Test Set 대상으로 계산된 결과입니다.
 - Global Damage Classifier의 분류 성능: 96%의 Accuracy, 96%의 F1-score를 기록했습니다.
 - Segmentation Network의 성능: Threshold 0.5 기준 96.7 의 IoU를 기록했습니다.
 - 이 때 Threshold 값은 한 픽셀이 특정 클래스에 속할 확률값이 몇 이상일 때 클래스에 속한다고 판단을 내릴 지에 대한 경곗값

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


실제 데이터에 적용 예시

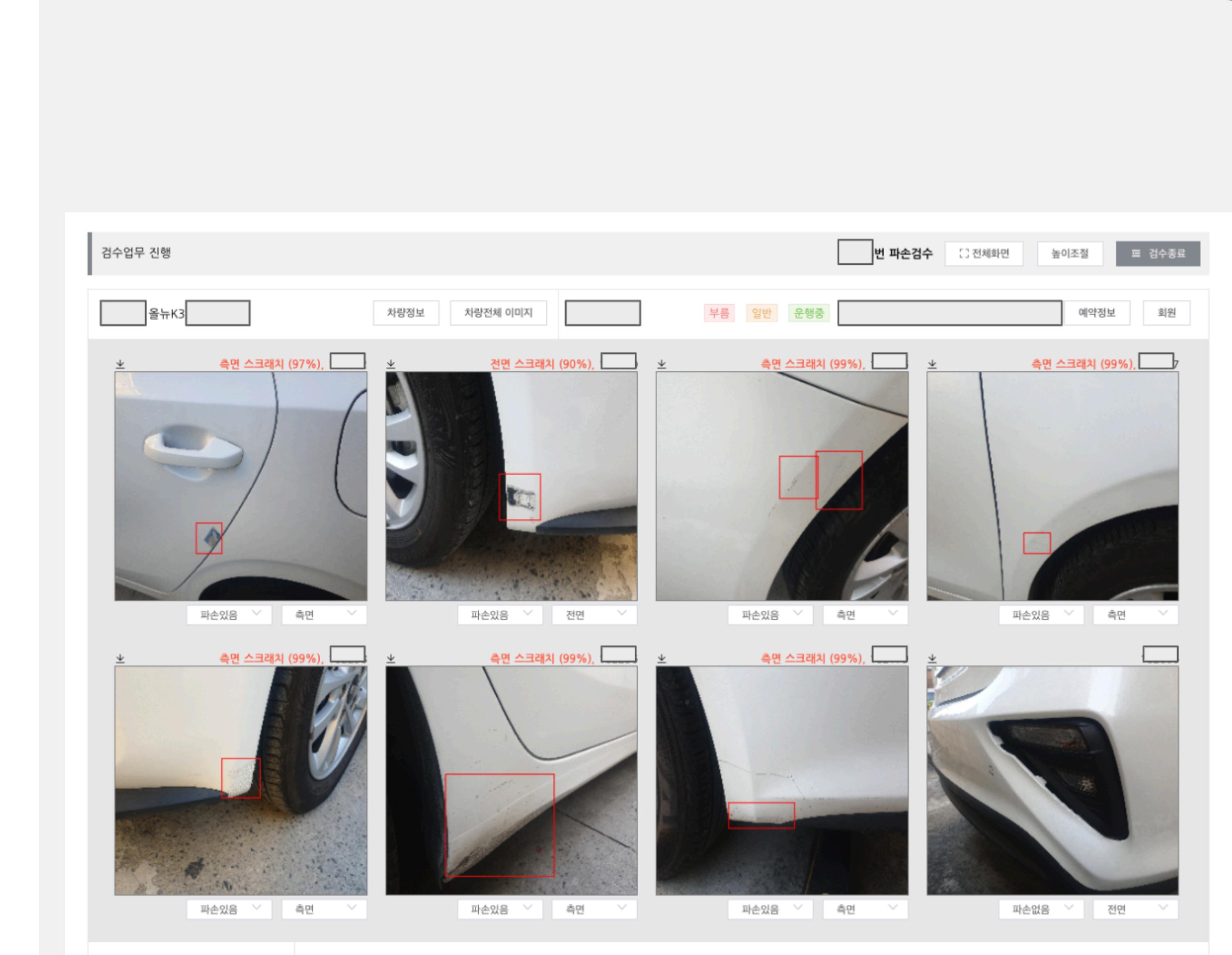
- 실제 앱을 통해 업로드된 차량 이미지들을 입력으로 한 모델의 Inference 결과

입력 이미지 파손 탐지 결과 보정 결과



실제 데이터에 적용 예시

- 차량 파손 검수 결과를 제공하고 있는 실제 운영 페이지의 일부



딥러닝 모델 Serving 간단 구축기

(feat. AWS SQS + Python Application + Kubernetes + Git & Rancher)

- <https://tech.socarcorp.kr/data/2020/03/10/ml-model-serving.html>