

○ 2024.11.06 ○

# “ R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN ”

Team: 카피바라 | Name: 박현아, 배누리, 김호정, 전사영

• INDEX

# 목차

01

Object Detection

02

R-CNN

03

Fast R-CNN

04


Faster R-CNN

05

Mask R-CNN

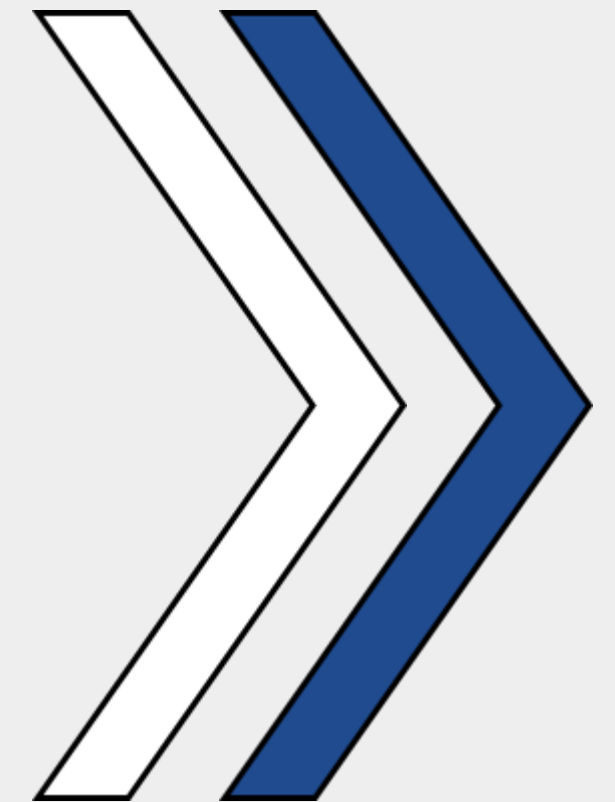
06

Mask R-CNN 코드 구현

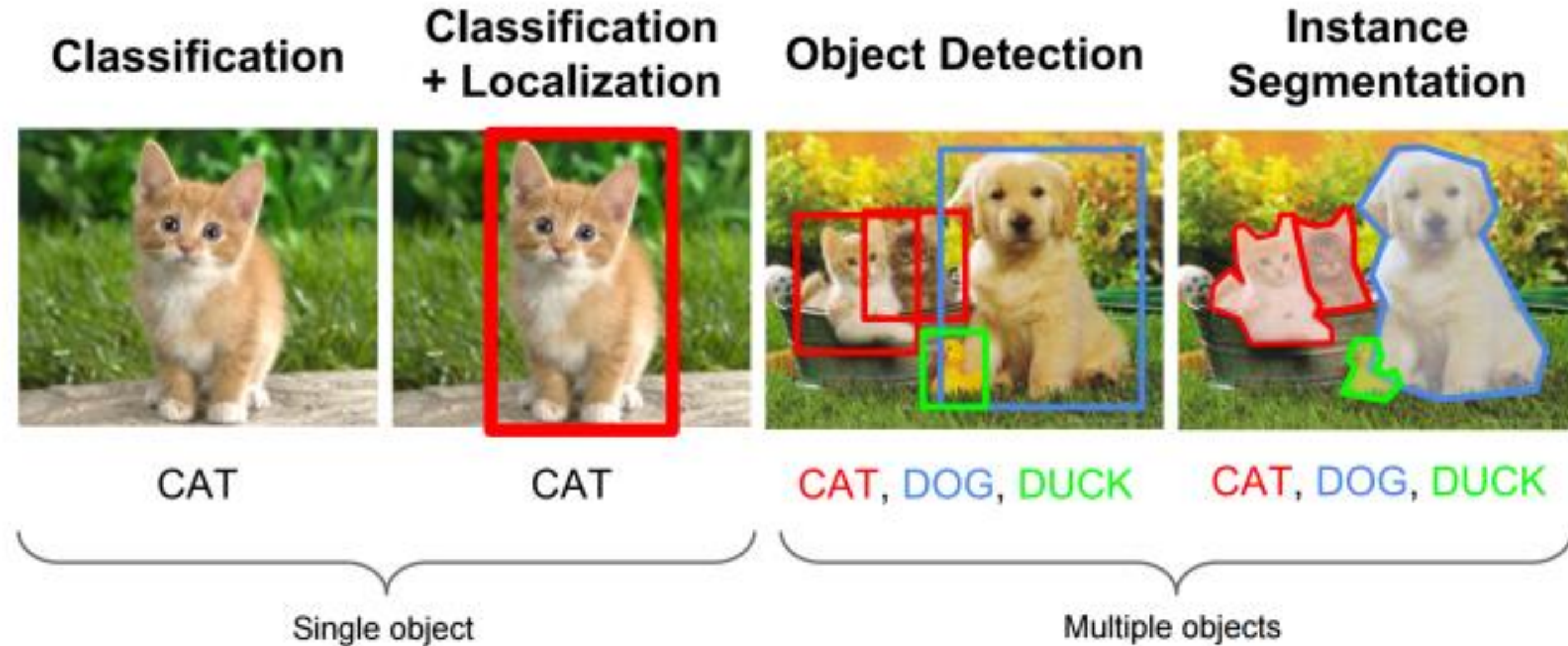


01

# Object Detection



# Object Detection



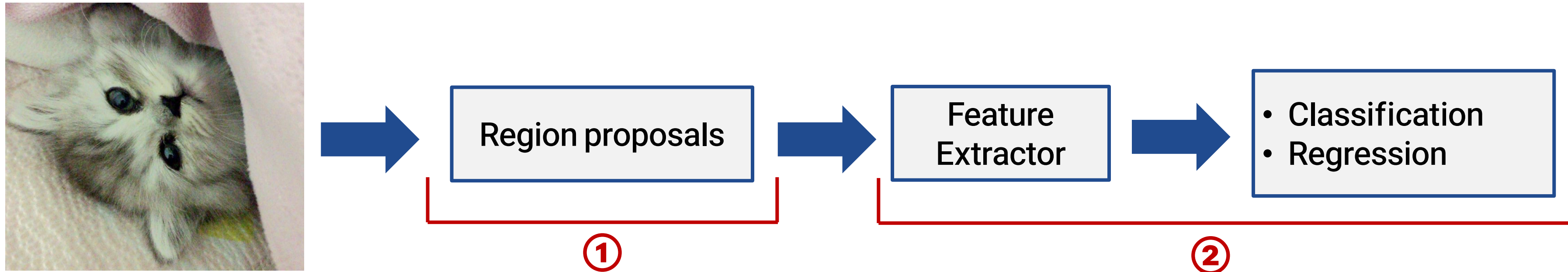
- Object Detection 은 다수의 사물이 존재하는 상황에서 각 사물의 위치와 클래스를 찾는 작업
- 비디오나 사진에서 어떤 물체가 있는지 분류하고 분류된 Object가 어디에 있는지 Bounding box를 통해 위치를 파악하여 Object를 감지함.
- Object의 수에 따라서 하나의 물체를 찾는 Single-object Detection과 여러 물체를 찾는 Multi-object Detection이 있음.



# 2-stage 방식 & 1-stage 방식

## • 2-Stage Detector

- 물체의 ①위치를 찾는 문제와 ②분류 문제를 **순차적**으로 해결함.

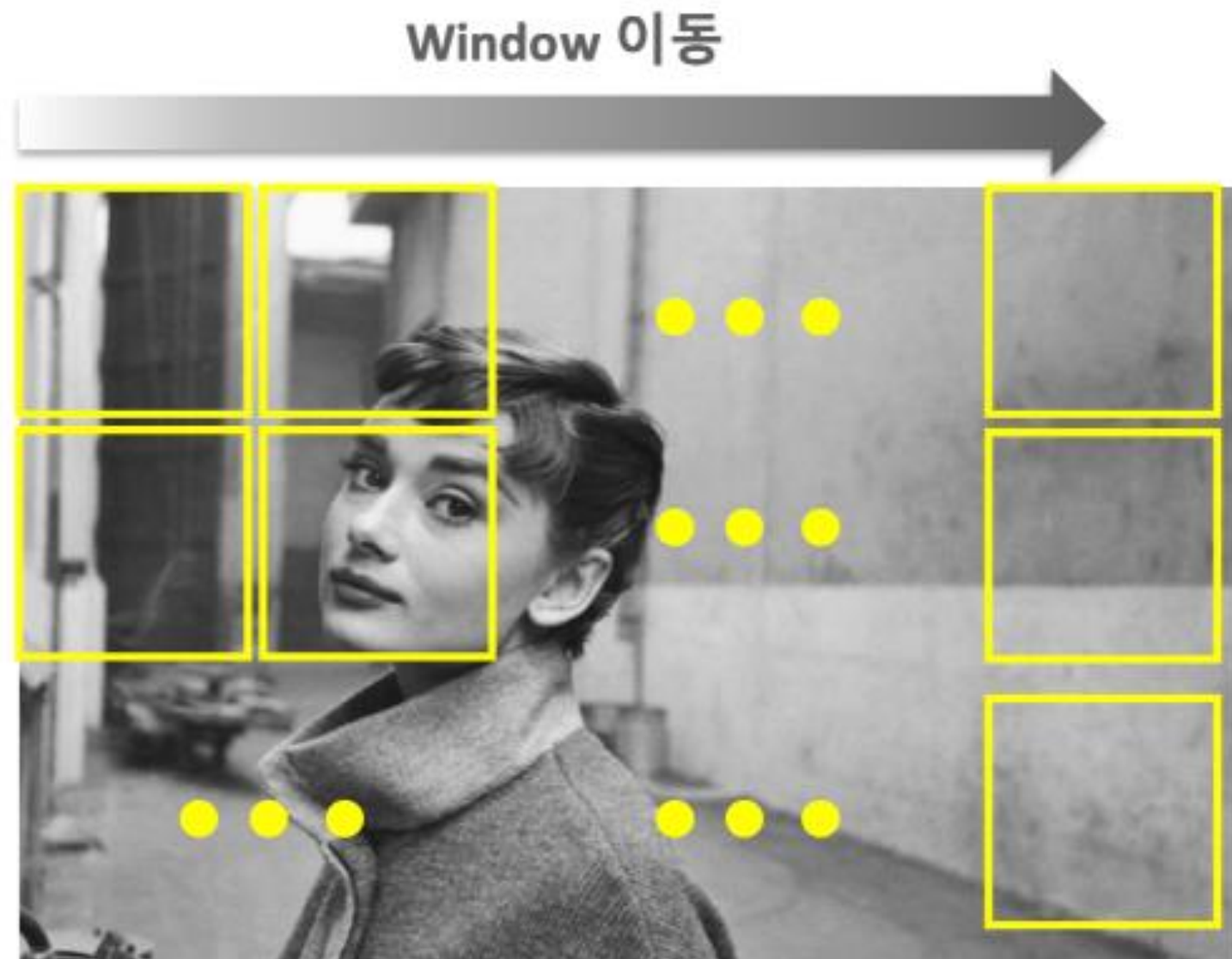


## • 1-Stage Detector

- 물체의 위치를 찾는 문제와 분류 문제를 **한 번에** 해결함.



# Sliding Window

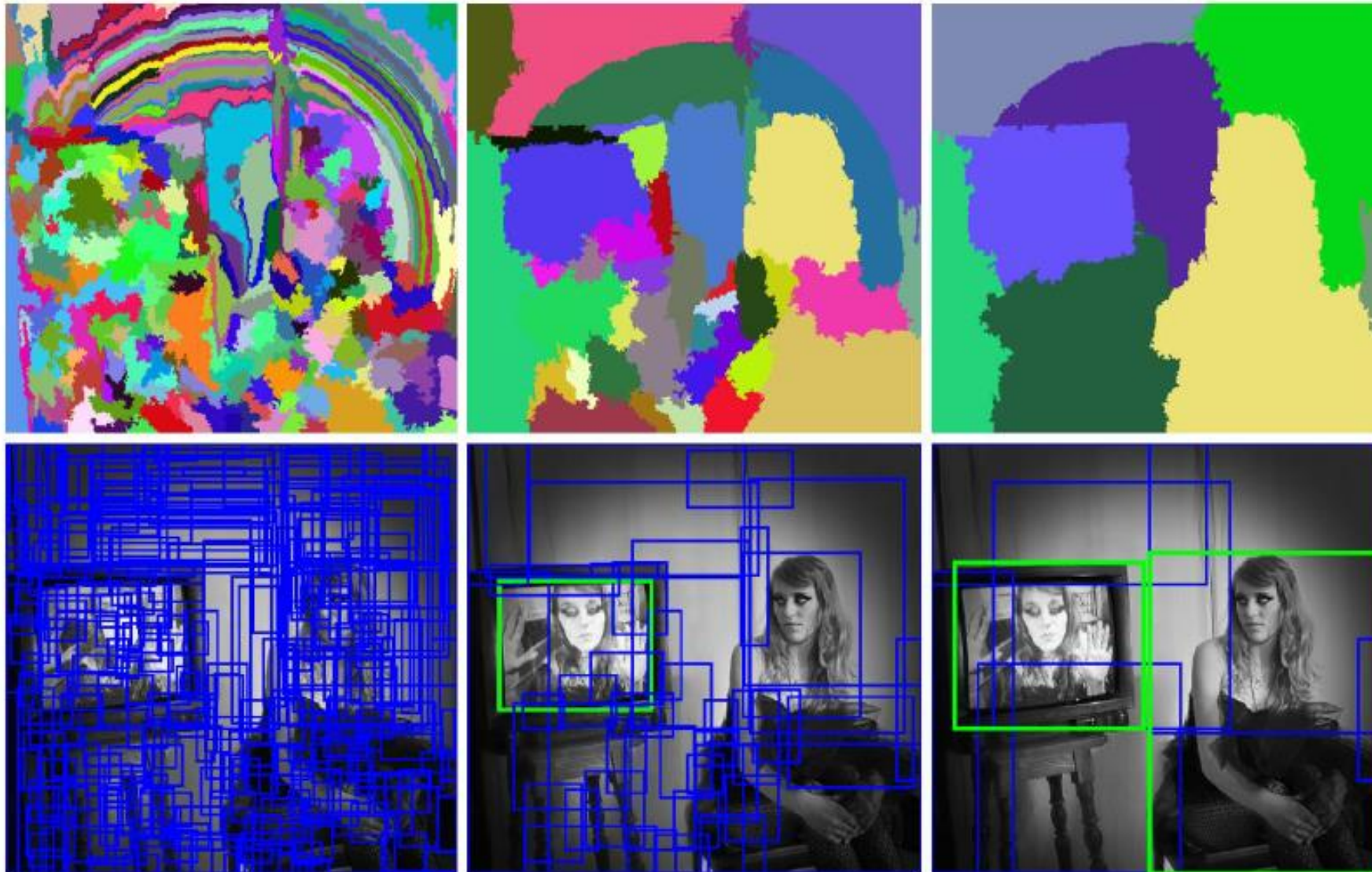


- 일정 크기의 window를 이동시키며, window 내에서 물체가 존재하는지 확인하는 방식
- 너무 많은 영역에 대하여 확인해야 한다는 단점이 있음



# Region Proposal : Selective search

- 객체가 있을만한 후보 영역을 미리 찾고 그 영역 내에서만 객체를 찾는 방식
- Segmentation 분야에 많이 쓰이는 알고리즘이며, 객체와 주변간의 색감(Color), 질감(Texture) 차이, 다른 물체에 애워 쌓여 있는지(Enclosed) 여부 등을 파악해서 인접한 유사한 픽셀끼리 묶어 물체의 위치를 파악할 수 있도록 하는 알고리즘



처리 과정

## Algorithm 1: Hierarchical Grouping Algorithm

**Input:** (colour) image

**Output:** Set of object location hypotheses  $L$

Obtain initial regions  $R = \{r_1, \dots, r_n\}$  using [13]

Initialise similarity set  $S = \emptyset$

**foreach** Neighbouring region pair  $(r_i, r_j)$  **do**

    Calculate similarity  $s(r_i, r_j)$

$S = S \cup s(r_i, r_j)$

**while**  $S \neq \emptyset$  **do**

    Get highest similarity  $s(r_i, r_j) = \max(S)$

    Merge corresponding regions  $r_t = r_i \cup r_j$

    Remove similarities regarding  $r_i : S = S \setminus s(r_i, r_*)$

    Remove similarities regarding  $r_j : S = S \setminus s(r_*, r_j)$

    Calculate similarity set  $S_t$  between  $r_t$  and its neighbours

$S = S \cup S_t$

$R = R \cup r_t$

Extract object location boxes  $L$  from all regions in  $R$

<Selective search 알고리즘>

# Intersection over Union (IoU)

- IoU란 두 바운딩 박스가 겹치는 비율을 의미함
  - 성능 평가 예시 : mAP@ 0.5 는 정답과 예측의 IoU 가 50% 이상일 때 정답으로 판정하겠다는 의미
  - NMS 계산 예시 : 같은 클래스끼리 IoU 가 50% 이상일 때 낮은 confidence의 box를 제거함

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


(교집합)

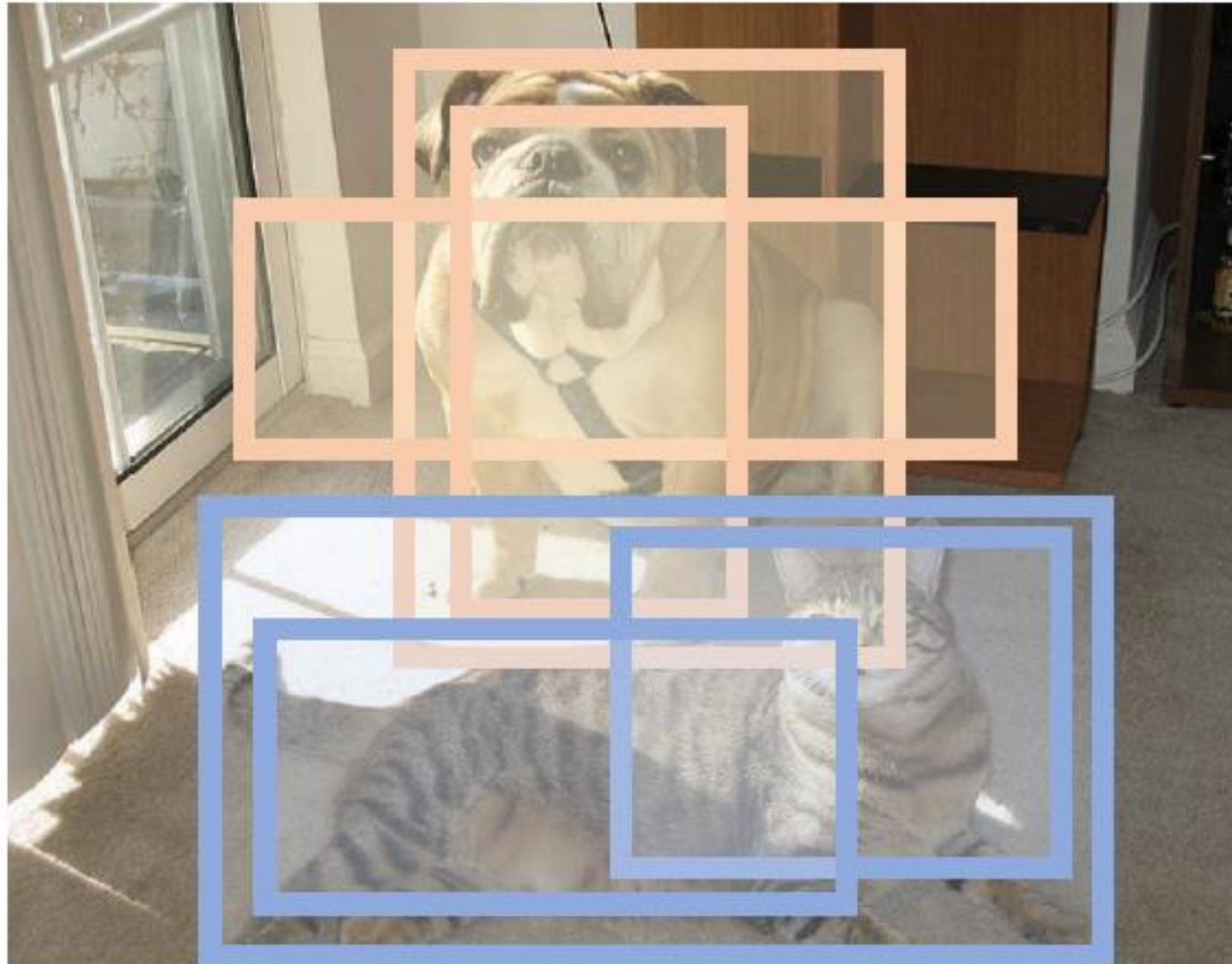
(합집합)



# NMS (Non Maximum Suppression)

- 객체 검출 (object detection)에서는 하나의 객체에 하나의 bounding box가 적용되어야 함

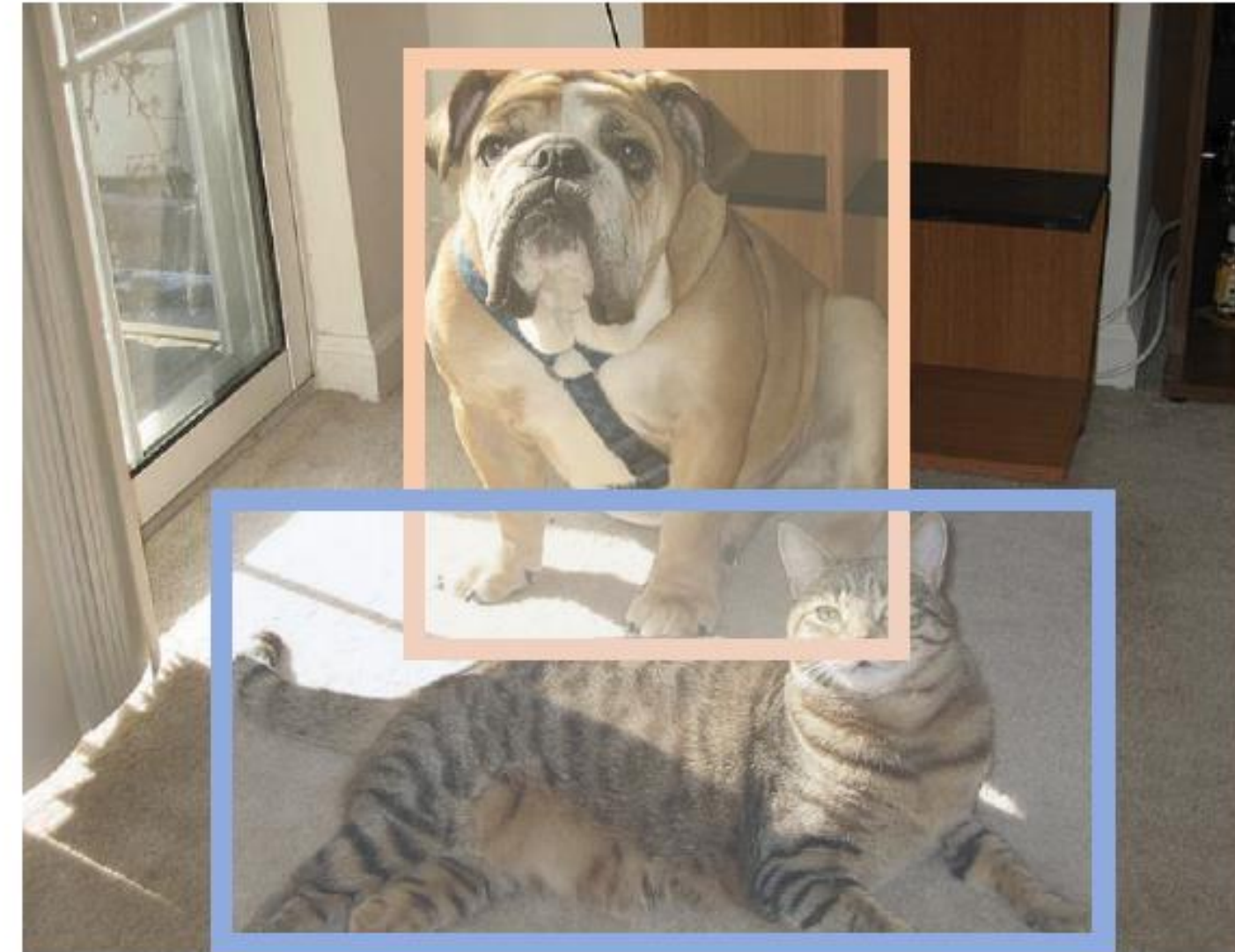
■ : dog ■ : cat



NMS

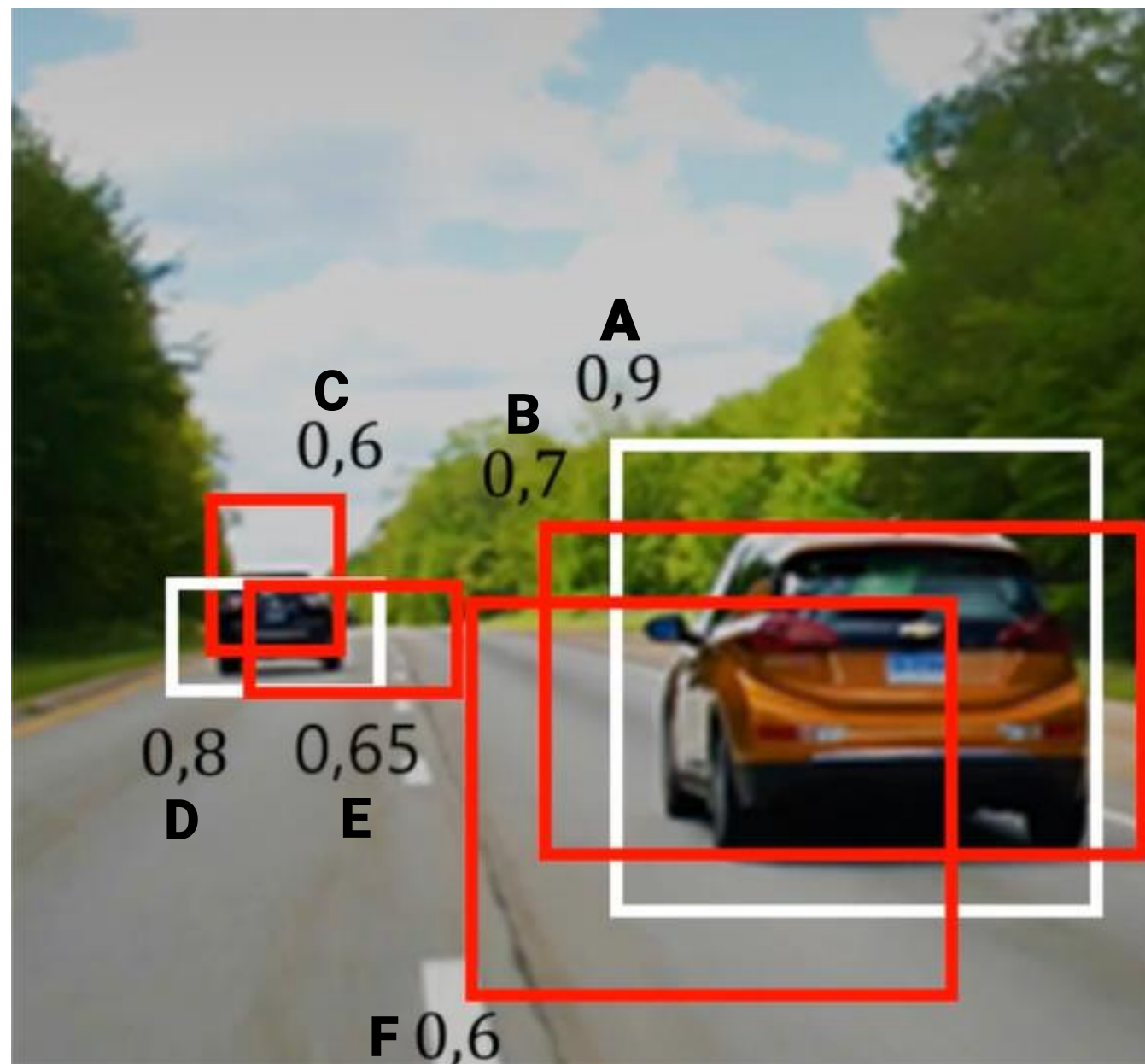


IoU가 특정 임계점(threshold) 이상인 중복 box 제거



# NMS (Non Maximum Suppression)

- (예시) IoU의 threshold를 0.5 라고 지정



1. Bounding box를 Confidence score 기준 내림차순 정렬

→ [A(0.9), D(0.8), B(0.7), E(0.65), C(0.6), F(0.6)]

2. 가장 높은 Confidence score 를 가진 A를 기준으로 잡고 뒤의 모든 박스를 비교.

- D(0.8) 와는 겹치지 않으므로 남겨둠.
- B(0.7) 와 IoU가 threshold 이상이므로 이 박스는 0.9 박스와 같은 것을 가리킨다고 간주하고 제거함.
- E(0.65), C(0.6) 와는 겹치지 않으므로 남겨둠.
- F(0.6)와 IoU가 threshold 이상이므로 제거함.
- 이제 A 다음으로 높은 Confidence score 를 가진 D(0.8)를 기준으로 뒤의 모든 박스와 비교

3. 최종적으로 A와 D가 남음

# NMS (Non Maximum Suppression)

- (예시) IoU의 threshold를 0.5 라고 지정

구분	Confidence score	A 기준	B 기준	C 기준	D 기준
A	0.9		0.3	0.4	0.2
B	0.7	0.7		0.8	0.3
C	0.6	0.4	0.7		0.6
D	0.8	0.1	0.6	0.6	0.2

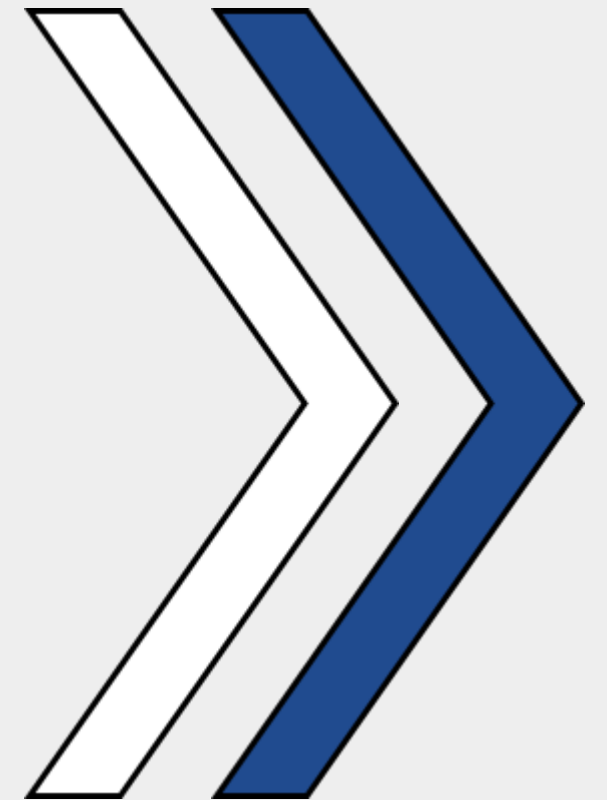
- Confidence score가 가장 높은 A를 기준으로 IoU 0.5 이상인 B를 제거
- A 다음으로 Confidence score가 높은 D 기준으로 IoU가 0.5 이상인 C 제거
- 최종적으로 A와 D가 남음





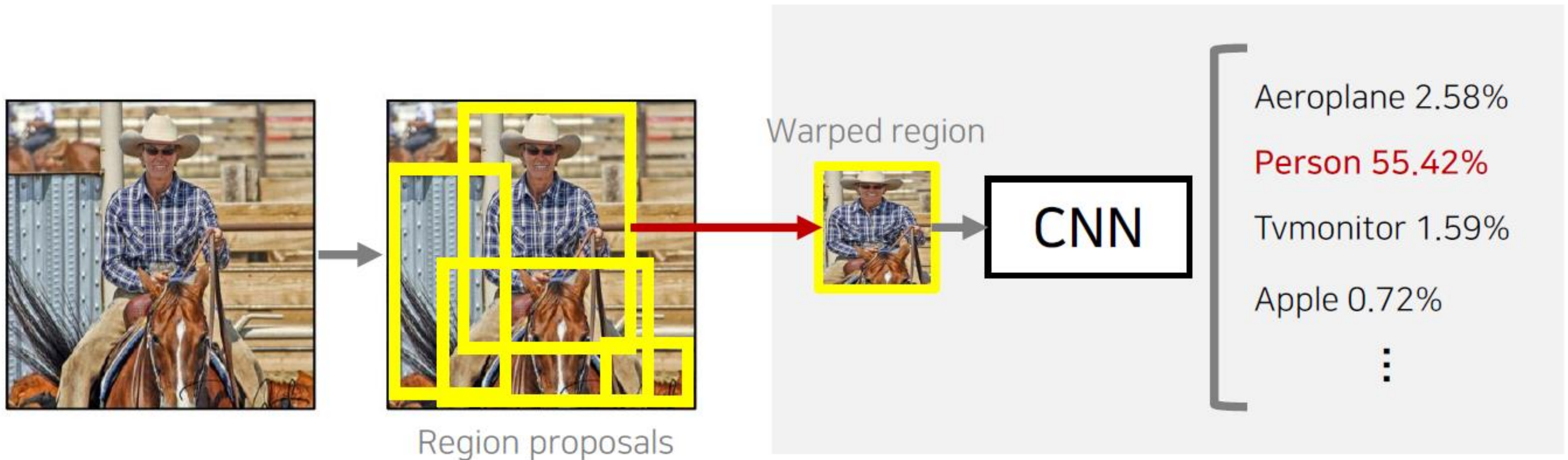
02

**R-CNN**



# R-CNN : Regions with CNN features

- 논문에서는 Selective Search 를 이용해 2,000 개의 Region Proposal 을 생성함
- 각 Region Proposal 을 일일이 CNN 에 넣어서 결과를 계산함

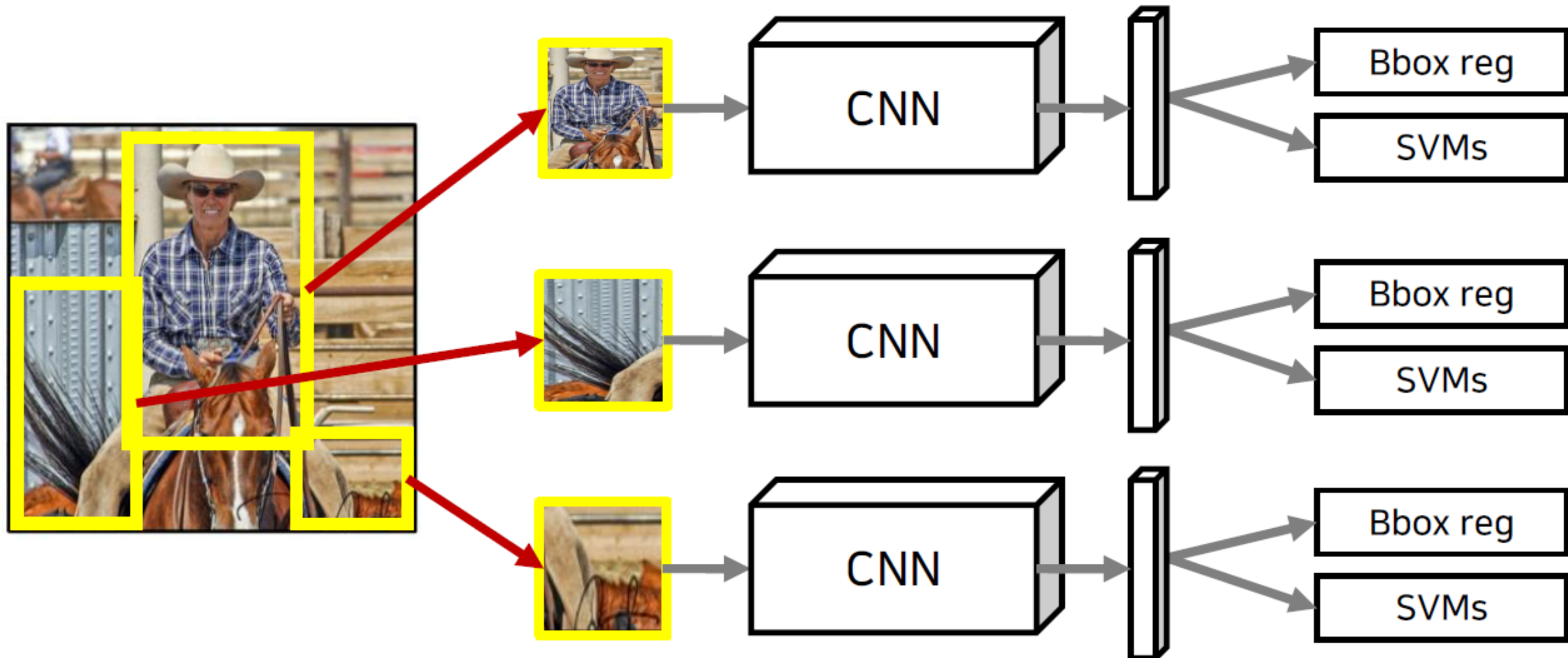


# R-CNN : Regions with CNN features

① Selective Search

② CNN을 통해 Feature vector 추출

③ SVM + Regression





# R-CNN 동작과정

1. Selective search를 이용해 2,000개의 RoI(Region of Interest)를 추출함.
2. 각 RoI에 대하여 warping을 수행하여 동일한 크기의 입력 이미지로 변경.
3. Warped image를 CNN에 넣어서(forward)이미지 feature를 추출함.
4. 해당 feature를 SVM에 넣어 클래스(class)분류 결과를 얻음.
  - 이때 각 클래스에 대해 독립적으로 훈련된 이진(binary)SVM을 사용함.
5. 해당 feature를 regressor에 넣어 위치(bounding box)를 예측함.

## warping

이미지를 특정 변환에 따라 왜곡하거나 변형하여 새로운 형태로 매핑하는 작업.

이미지 속 픽셀을 다른 위치로 이동시키면서 원근, 회전, 확대, 축소 등의 변형을 적용할 수 있음.

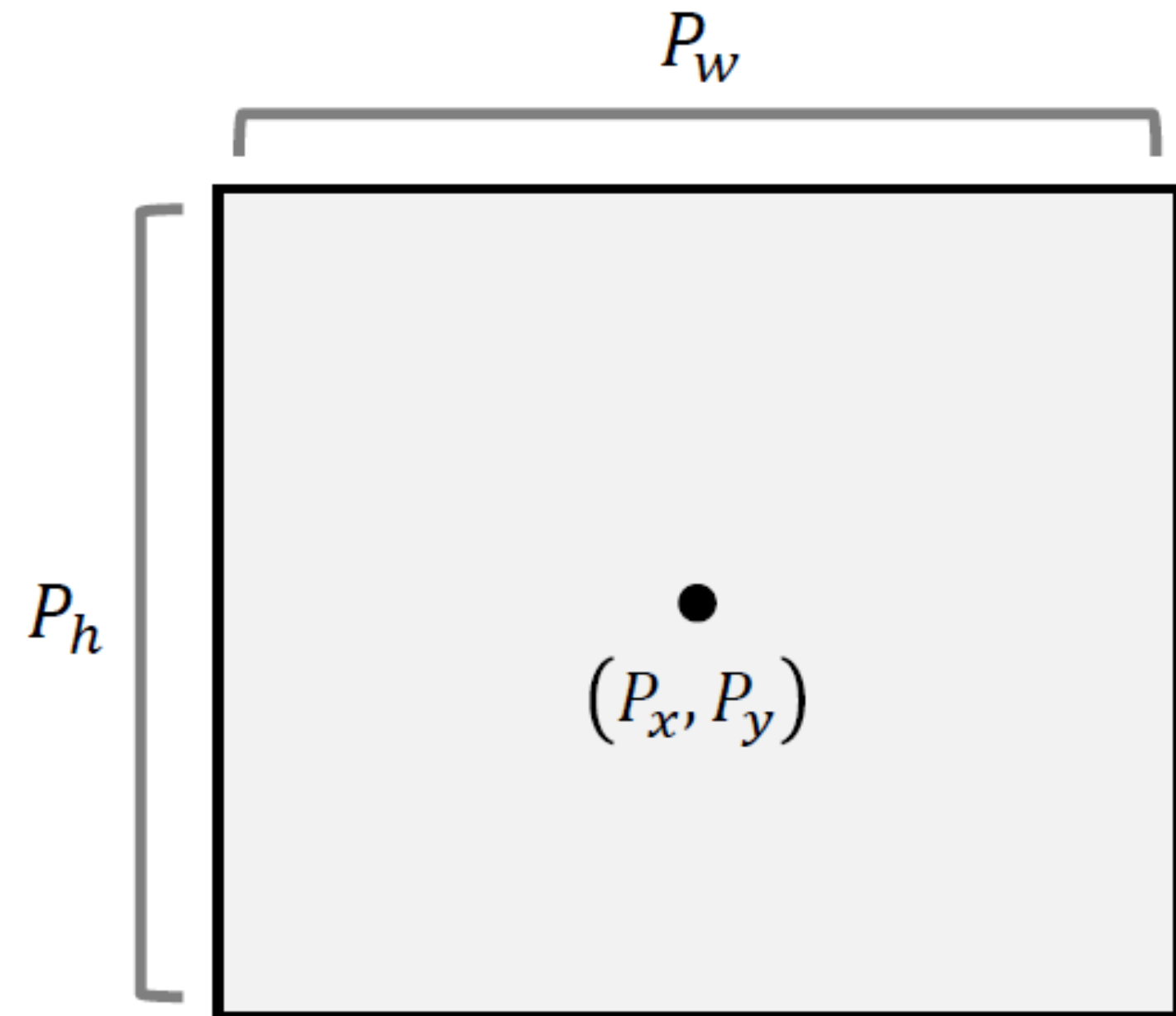
# R-CNN: Bounding Box Regression

지역화(localization)성능을 높이기 위해 bounding-box regression을 사용함.

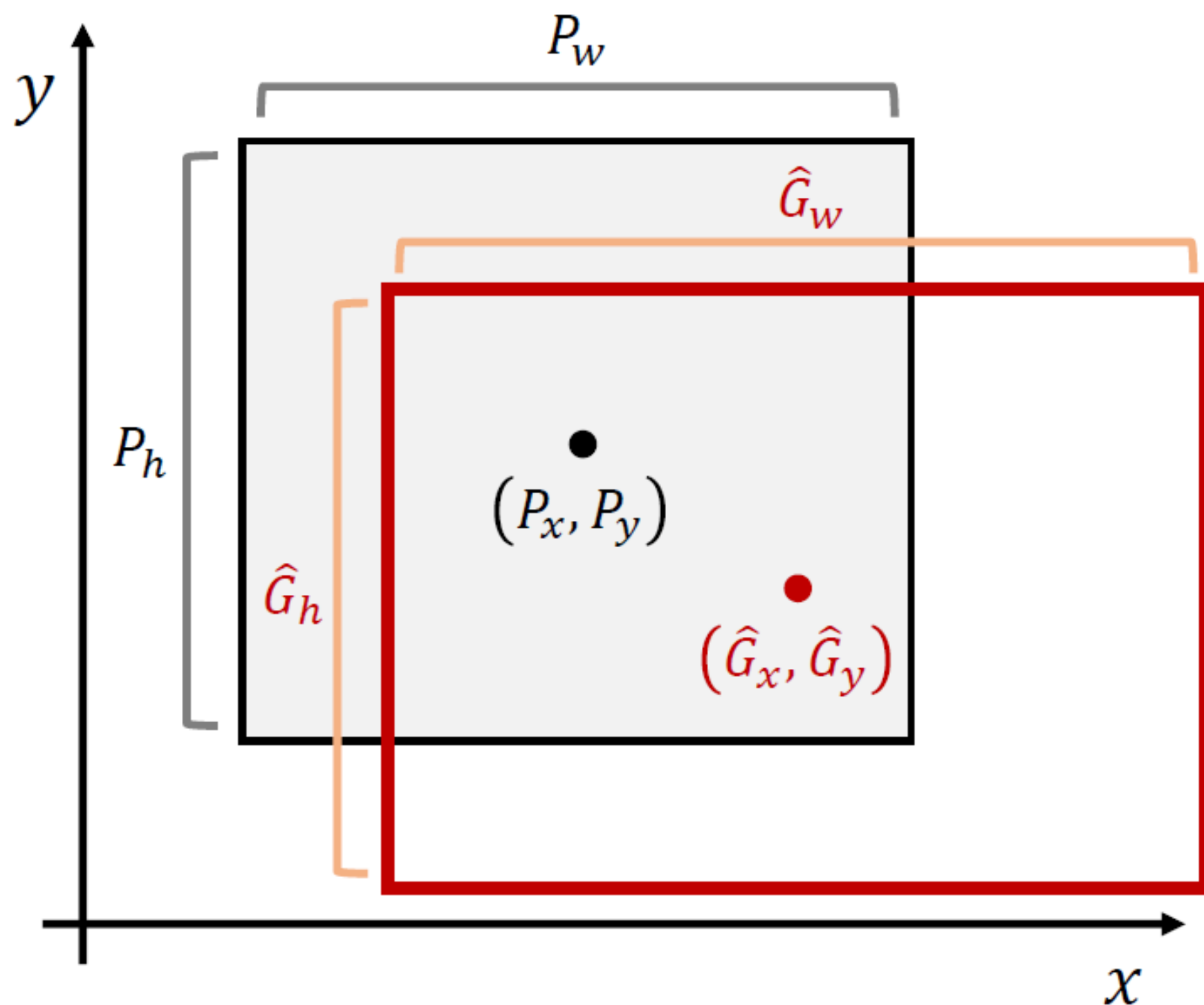
· 학습데이터셋:  $\{(P^i, G^i)\}_{i=1, \dots, N}$

· 예측위치:  $P^i = (P_x^i, P_y^i, P_w^i, P_h^i)$

· 실제위치:  $G^i = (G_x^i, G_y^i, G_w^i, G_h^i)$



# R-CNN: Bounding Box Regression



- 학습할 4개의 파라미터

- $d_x(P), d_y(P), d_w(P), d_h(P)$

- Linear regression 진행

$$\hat{G}_x = P_w d_x(P) + P_x$$

$$\hat{G}_y = P_h d_y(P) + P_y$$


$$\hat{G}_w = P_w \exp(d_w(P))$$

$$\hat{G}_h = P_h \exp(d_h(P))$$



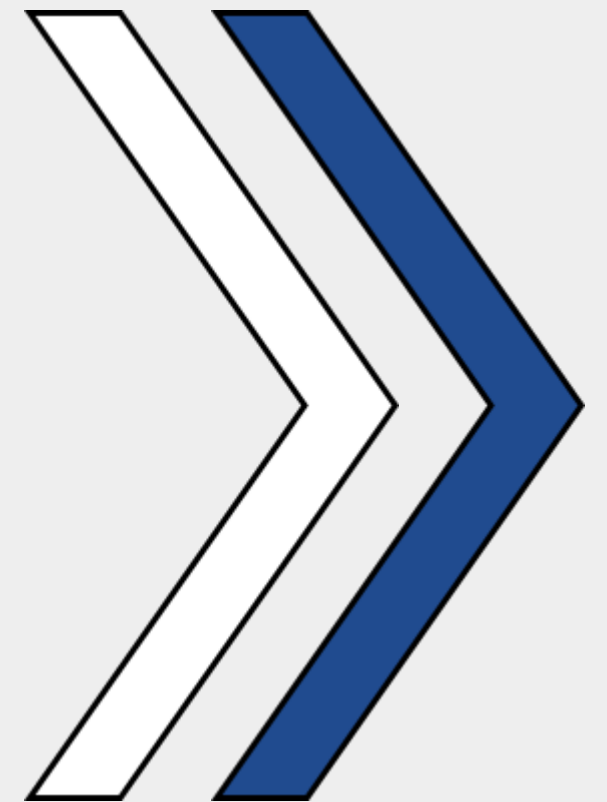
## R-CNN 한계점

- 입력 이미지에 대하여 **CPU기반의 Selective Search**를 진행해야 하므로 많은 시간이 소요됨.
- 전체 아키텍처에서 SVM, Regressor 모듈이 CNN과 분리되어 있음.
  - CNN은 고정되므로 SVM과 Bounding Box Regression 결과로 CNN을 업데이트 할 수 없음.
  - 다시말해 end-to-end 방식으로 학습할 수 없음.
- 모든 RoI(RegionofInterest)를 CNN에 넣어야하기 때문에 각 RoI마다 CNN 연산이 필요함.
- 학습(training)과 평가(testing)과정에서 많은 시간이 필요함.



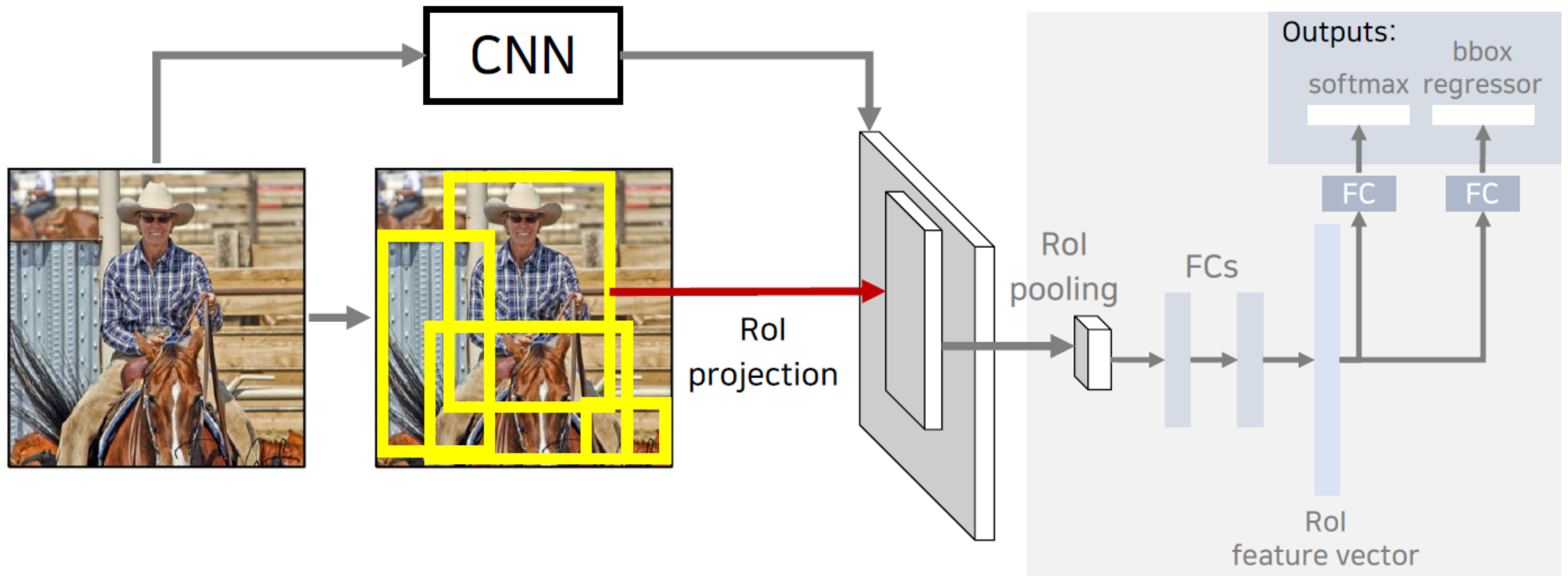
03

**Fast R-CNN**



# Fast R-CNN

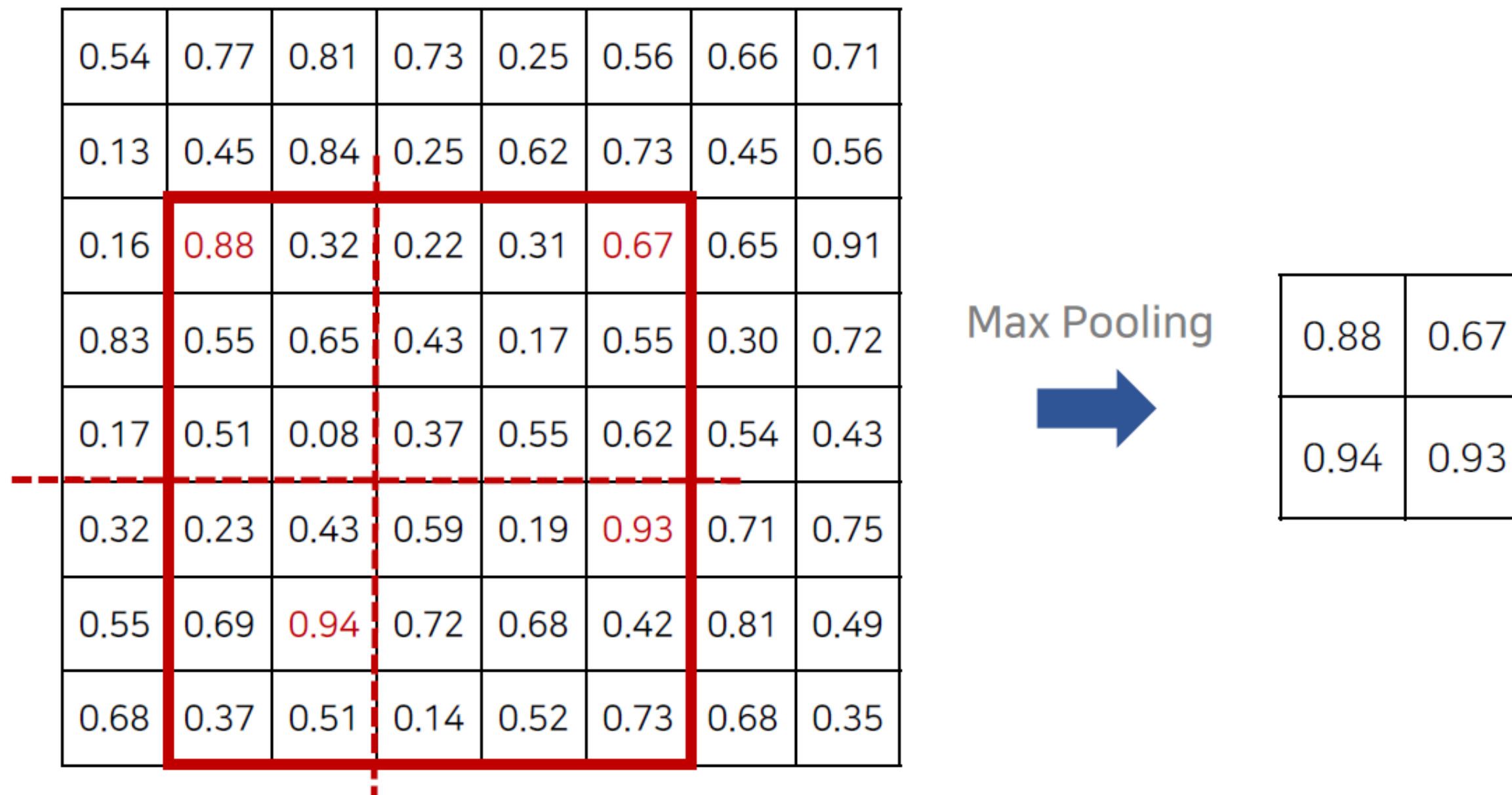
- 동일한 Region proposal 을 이용하되 이미지를 **한 번만 CNN 에 넣어** Feature Map 을 생성함






# Fast R-CNN: RoI Pooling

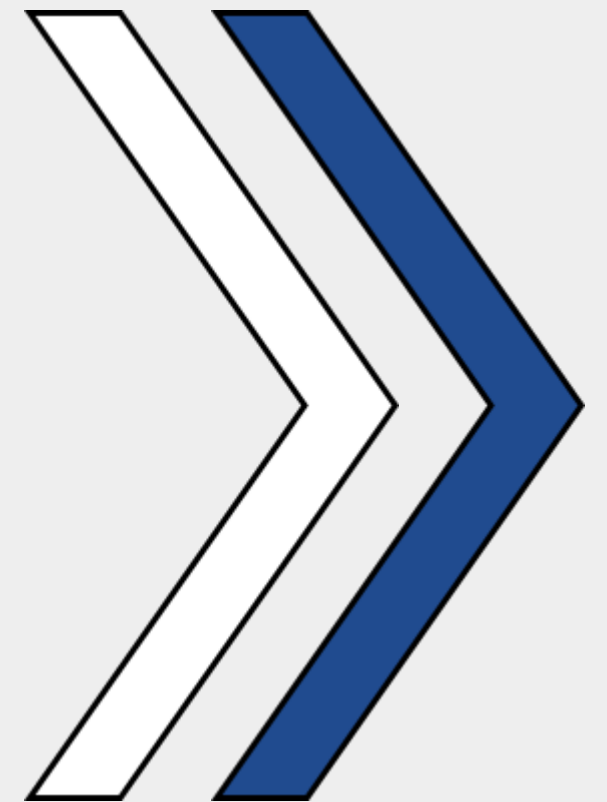
- 특정 영역(Region of Interest; RoI)을 고정된 크기의 피쳐 맵으로 변환하기 위해 사용하는 방법
- 각 RoI 영역에 대하여 맥스 풀링 (max pooling) 을 이용해 고정된 크기의 벡터를 생성함
- 이 과정에서 양자화(quantization)가 발생하는데, 좌표값이 정수로 반올림되면서 공간 정보가 약간 왜곡될 수 있음





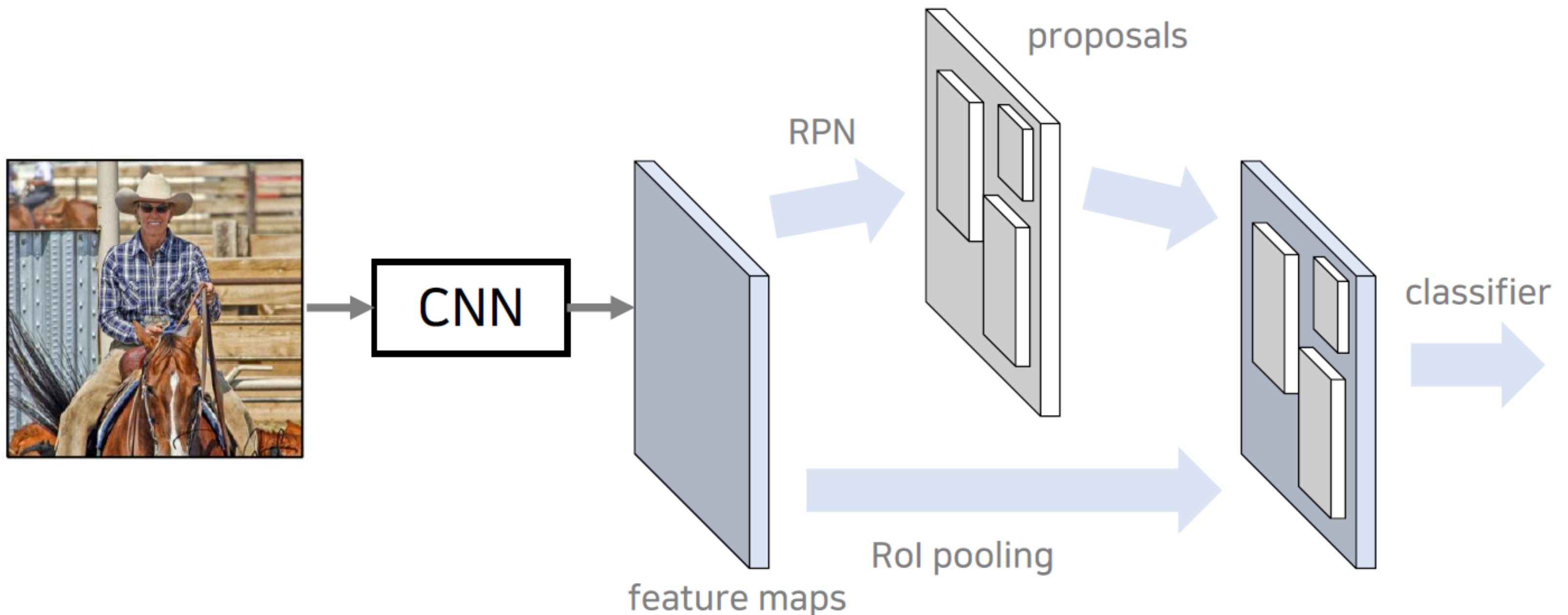
04

# Faster R-CNN



# Faster R-CNN

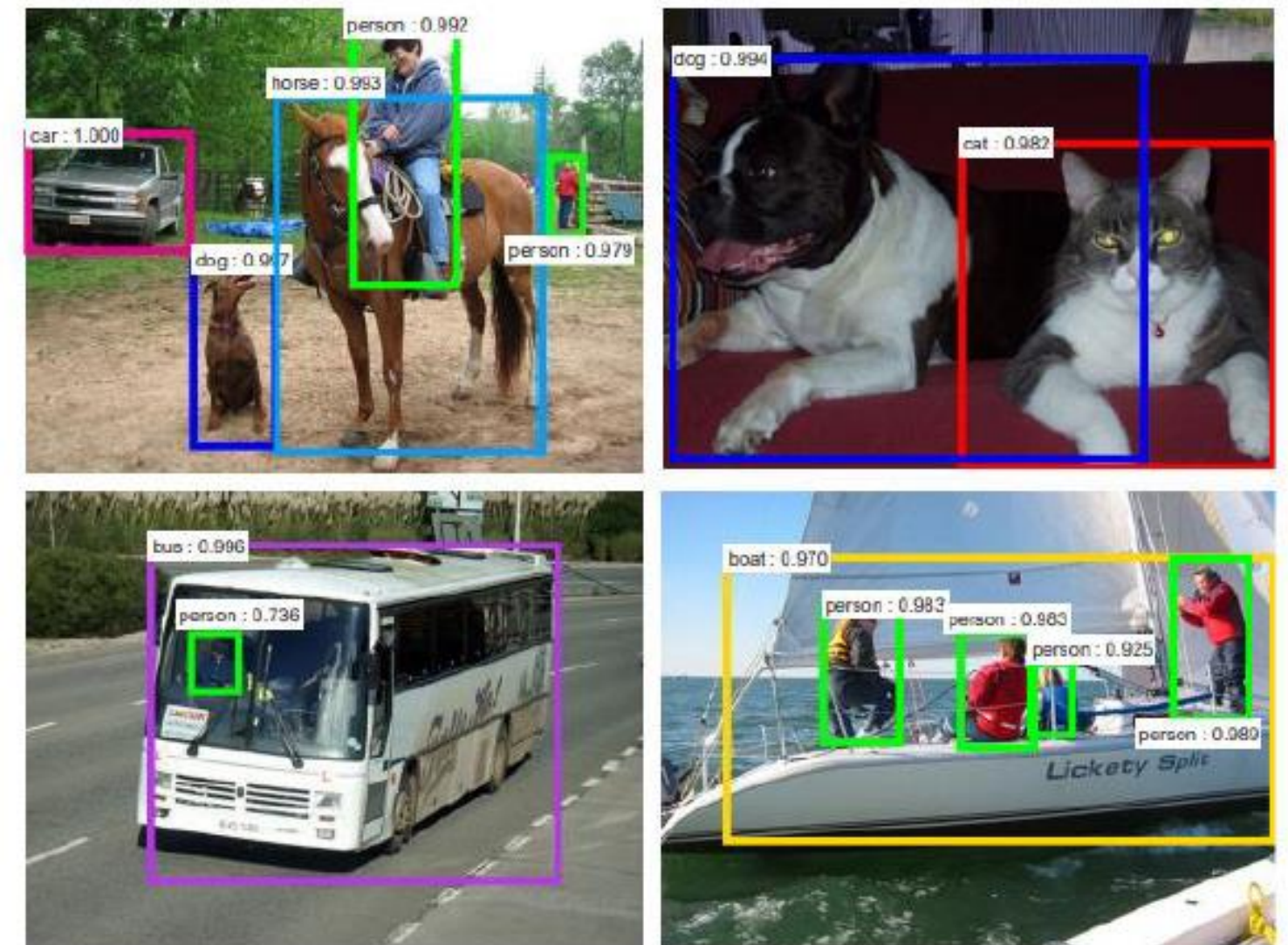
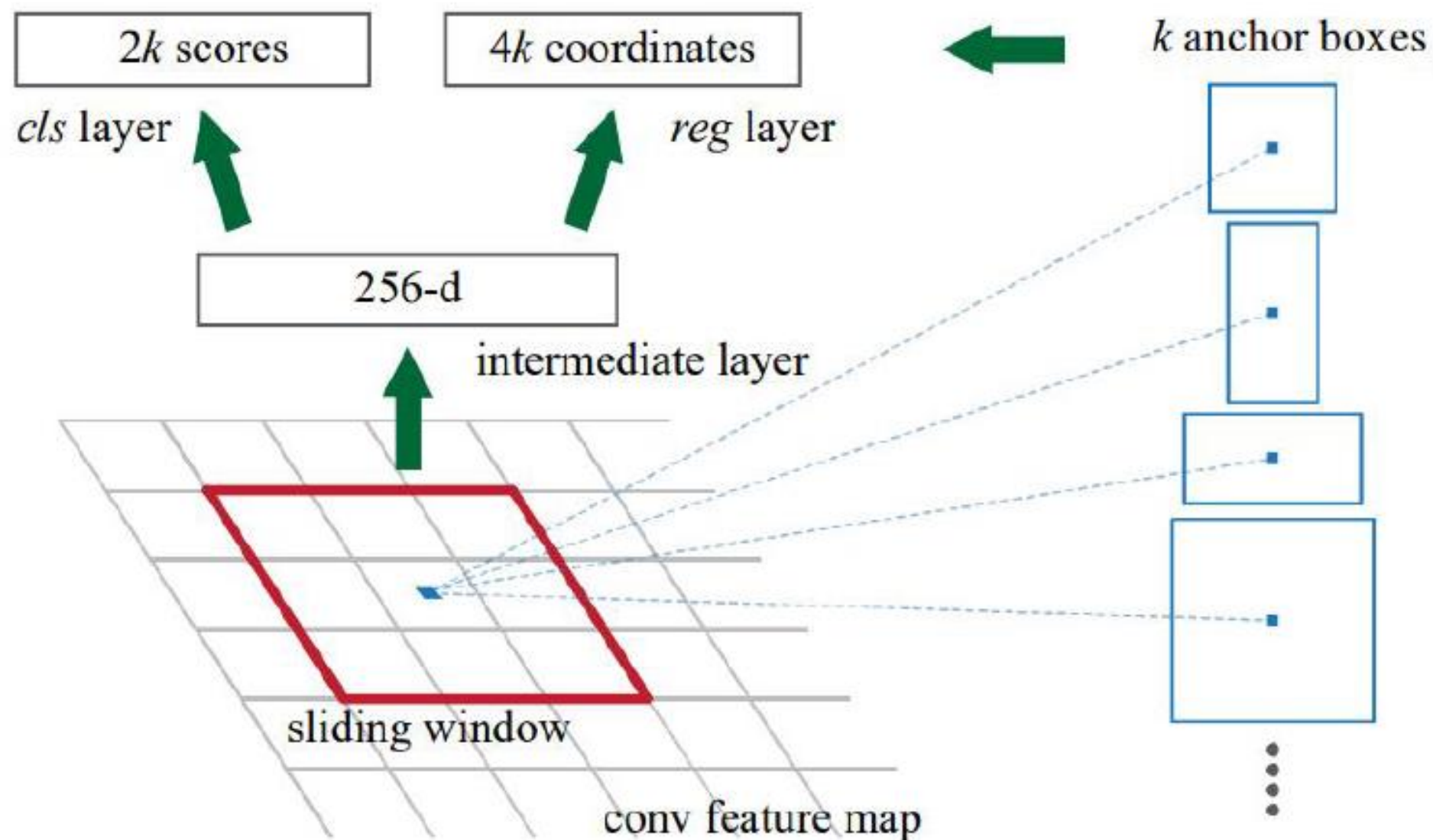
- 병목 (bottleneck) 에 해당하던 Region Proposal 작업을 **GPU 장치에서 수행하도록 함**(RPN 적용)
- 전체 아키텍처를 **end to end** 로 학습 가능함






# Faster R-CNN : Region Proposal Networks (RPN)

- RPN 네트워크는 feature map 이 주어졌을 때 물체가 있을 법한 위치를 예측 함
- $k$  개의 앵커 박스 (anchor box) 를 이용함
- 슬라이딩 윈도우 (sliding window) 를 거쳐 각 위치에 대해 Regression 과 Classification 을 수행함

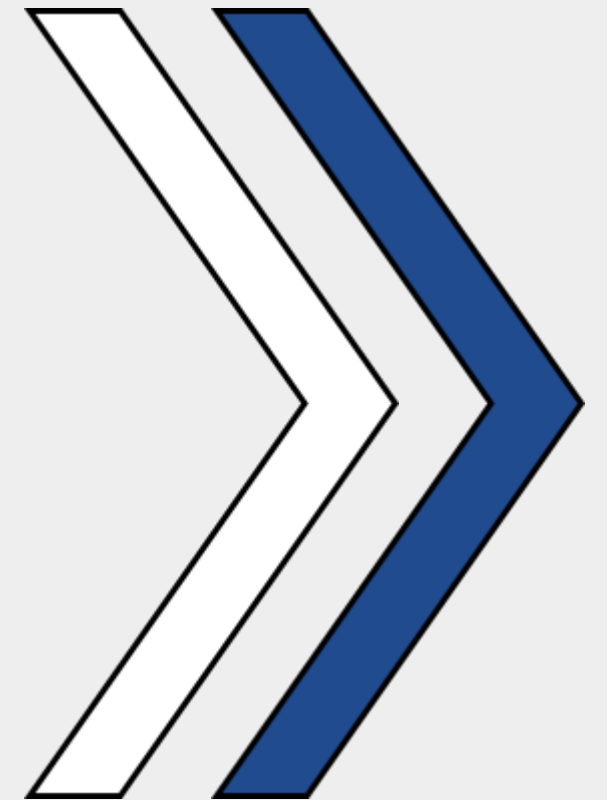






05

**Mask R-CNN**

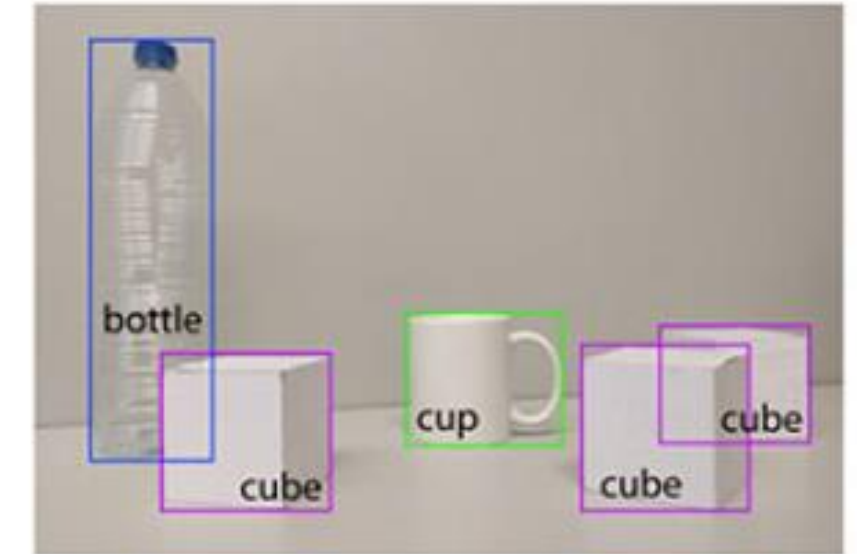


# Mask R-CNN

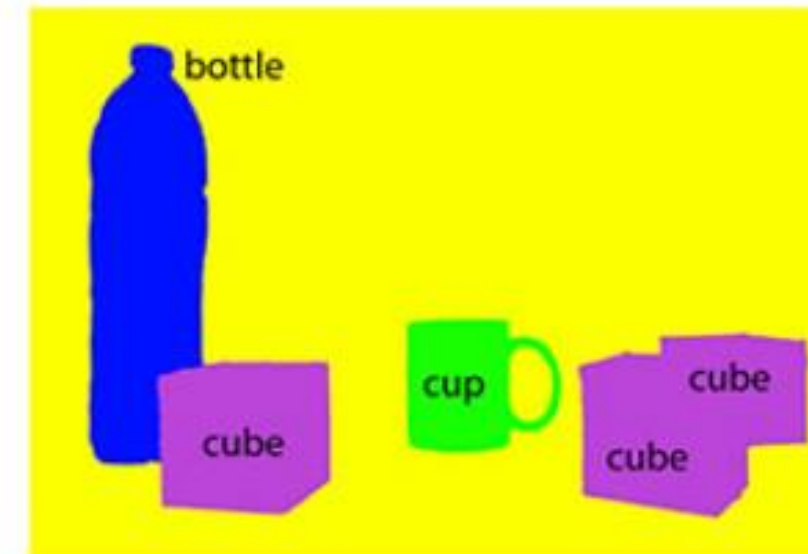
- **Instance segmentation** 목표
  - 이미지의 모든 객체를 올바르게 감지하는 동시에 각 인스턴스를 정확하게 분할
  - 기존 분류 및 bounding box regression을 위한 분기와 각 관심 영역(RoI)에서 segmentation mask를 예측하기 위한 분기가 병렬로 수행됨
  - 분류 및 회귀 분기는 객체의 클래스와 bounding box를 예측하는 데 사용되고, segmentation mask 분기는 각 객체의 픽셀 단위 마스크를 예측하는 데 사용됩니다.



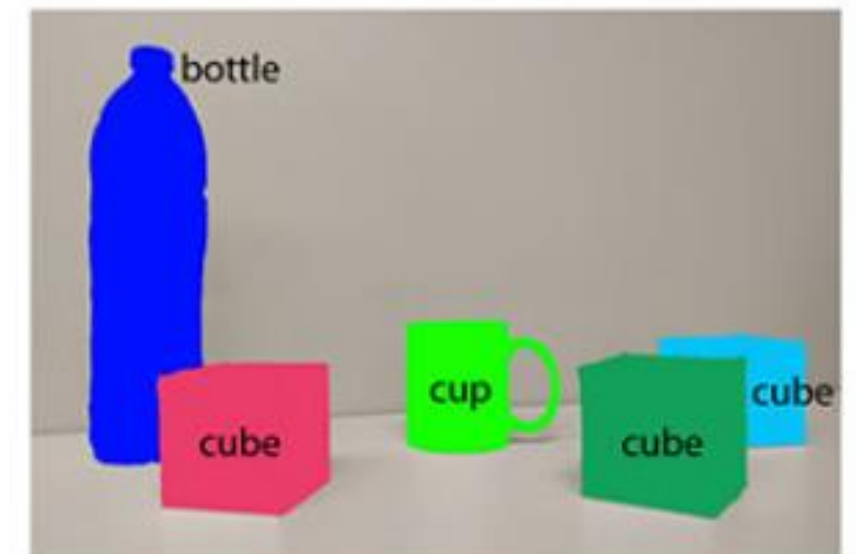
(a) Image classification



(b) Object localization



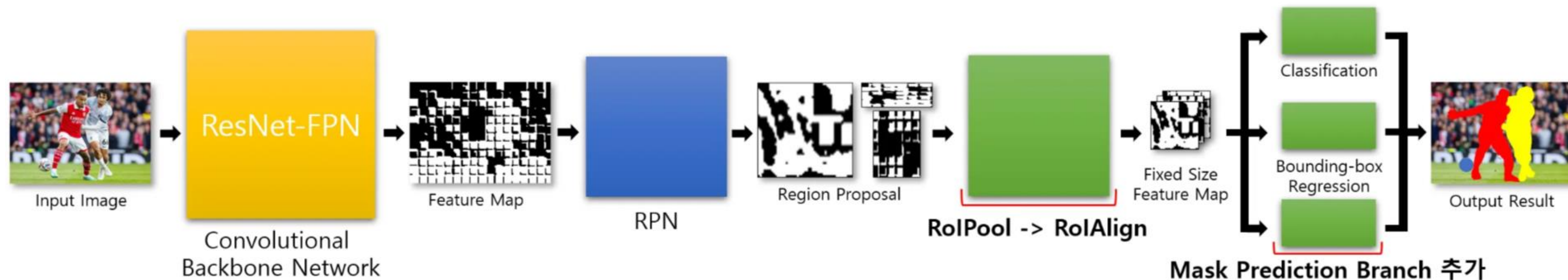
(c) Semantic segmentation



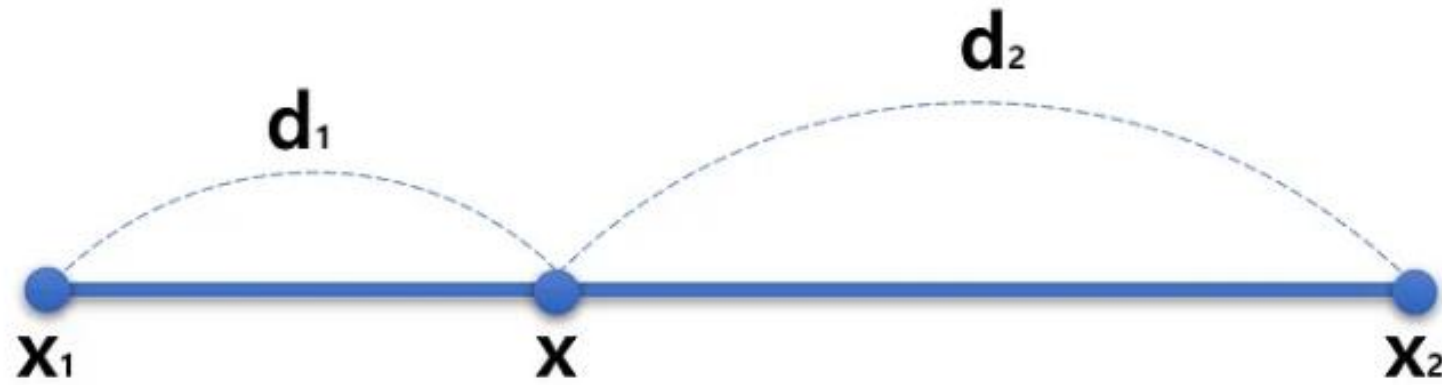
(d) Instance segmentation

# Mask R-CNN

- 객체 인스턴스 분할을 위한 프레임워크로, faster R-CNN에 객체 마스크 예측 기능을 추가한 모델
- 정확한 위치 정렬을 위해 RoI Align 레이어를 도입하였으며, 기존의 RoI Pool의 양자화 문제점을 해결함.
- 양자화를 하지 않고 부동 소수점 좌표를 사용하여 정확한 위치 정보를 유지
- 대상 RoI를 고정된 크기로 변환할 때, 각 셀에서의 값을 가져올 때 양자화 대신 bilinear interpolation을 사용함
- RoI Pooling에 비해 훨씬 정밀하게 위치 정보를 유지하며, 이를 통해 마스크 예측 정확도가 크게 향상됨

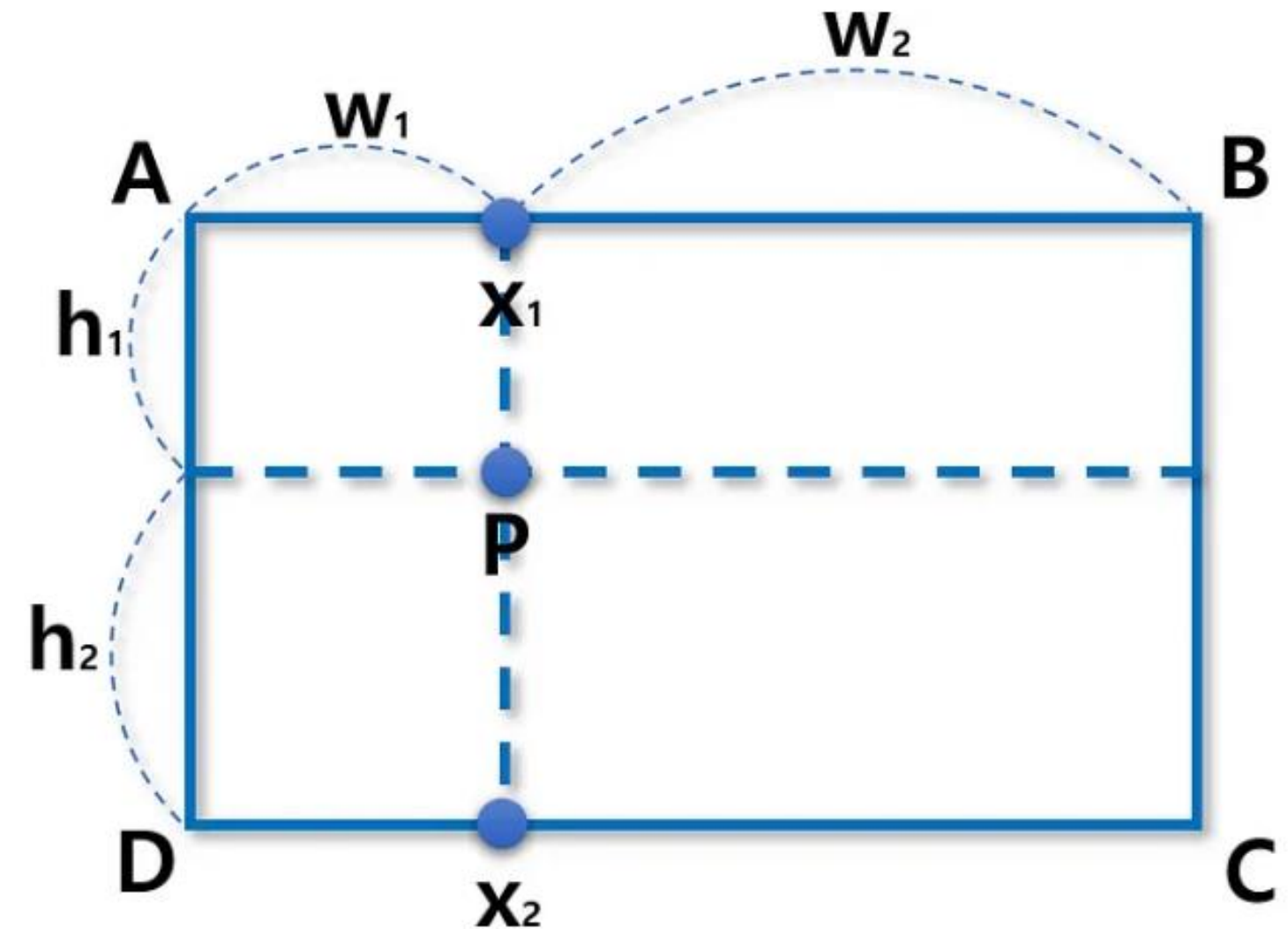


# bilinear interpolation



$$x = \frac{d_1}{d_1 + d_2} x_2 + \frac{d_2}{d_1 + d_2} x_1$$

**Linear Interpolation**



**Bilinear Interpolation**



# Rol Align

0.8	1.1	2.1	0.8	1.1	2.1	3.9	0.4
2.2	4.3 2.8	4.2 3.9	2.2	4.3	4.2	1.8	6.3
0.2	5.5 1.7	4.8 2.2	0.2	5.5	4.8	7.7	9.5
0.8	1.1	2.1	0.8	1.1	2.1	3.9	0.4
2.2	4.3	4.2	2.2	4.3	4.2	1.8	6.3
0.2	5.5	4.8	0.2	5.5	4.8	7.7	9.5
1.1	0.8	2.2	1.1	0.8	2.2	0.8	5.2
5.0	1.8	1.9	5.0	1.8	1.9	2.0	4.1

8 x 8 Feature Map  
Red Dotted Line = Rol



3.9	4.2
3.6	3.7

2 x 2 Rol  
Fixed Size

# Rol Pooling, Rol Align 비교

비교 항목	Rol Pooling	Rol Align
좌표 처리 방식	좌표를 정수로 반올림 (손실 발생 가능)	좌표를 소수점까지 고려 (양선형 보간)
정밀도	좌표 반올림으로 세부 정보 손실 가능	높은 좌표 정밀도로 더 나은 특징 추출 가능
적용 모델	Fast R-CNN, Faster R-CNN 등	Mask R-CNN

0.3	0.4	0.2	0.1
0.5	0.1	0.9	0.7
0.3	0.6	0.2	0.2
0.1	0.7	0.9	0.1

2 x 2

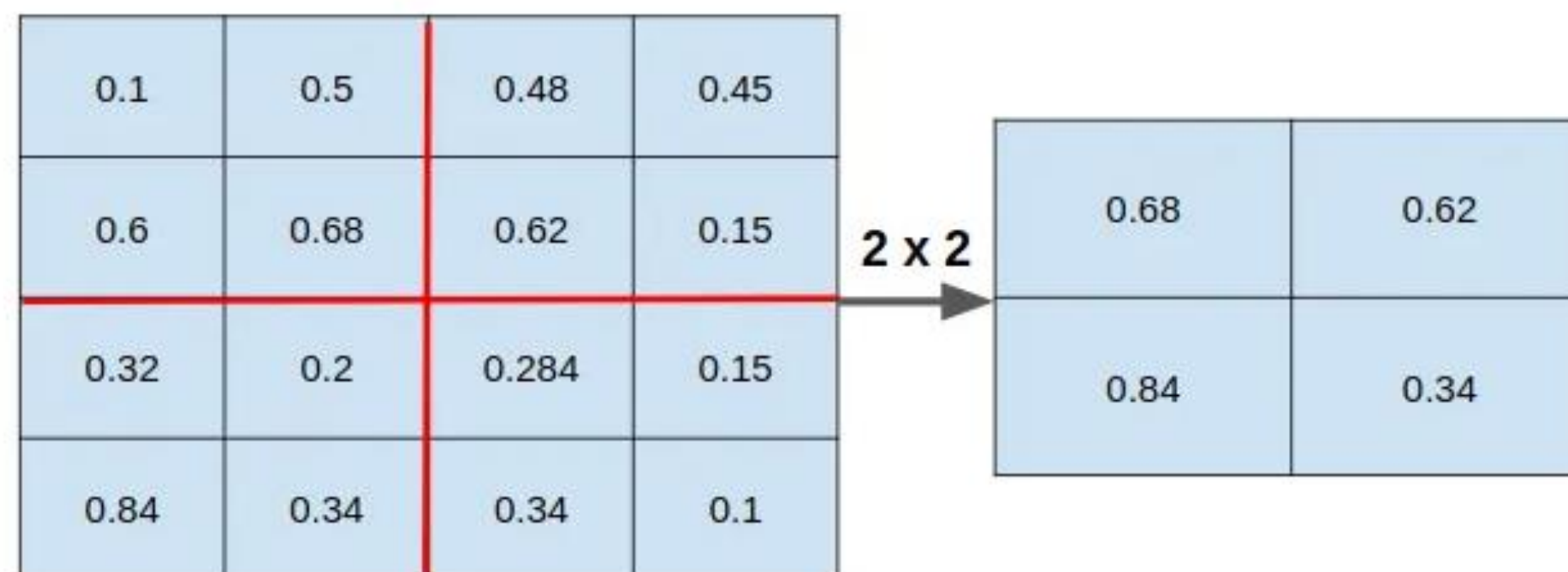
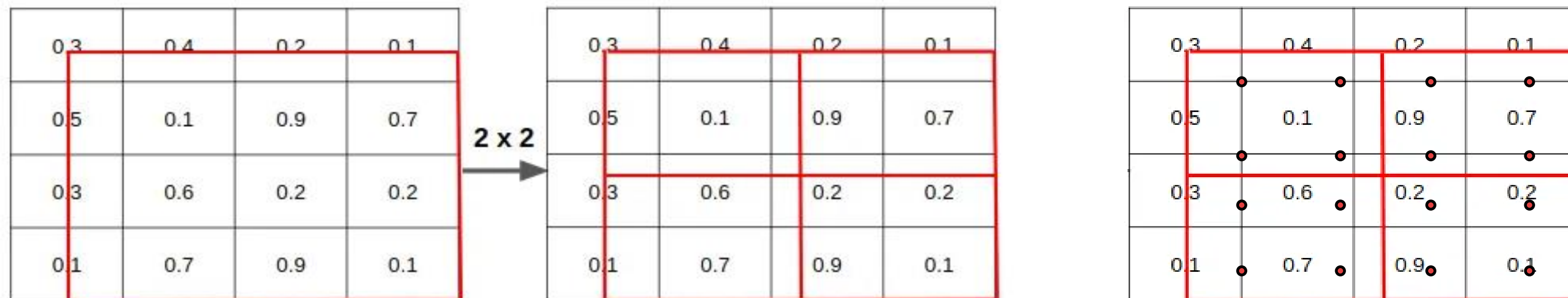
0.3	0.4	0.2	0.1
0.5	0.1	0.9	0.7
0.3	0.6	0.2	0.2
0.1	0.7	0.9	0.1

Rol Pooling


max pool ROI

0.9	0.7
0.9	0.1

# Roi Align

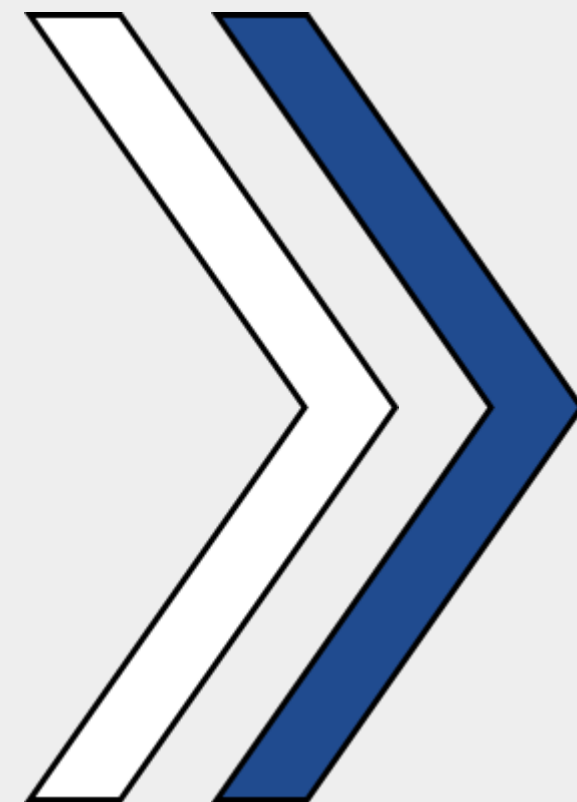






06

**Mask R-CNN 코드구현**



## 코드 구현

```
# Mask R-CNN 모델을 coco 데이터셋 가중치로 초기화
model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)
model.eval() # 모델을 평가 모드로 설정

# 사용할 장치 설정 (GPU가 사용 가능하다면 GPU로 설정)
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
model.to(DEVICE)

# 테스트할 이미지 경로 설정
test_image_path = 'c:/Users/hyeonah/Desktop/cat/11.png' # 이미지 경로

# 이미지 불러오기 및 전처리
image = cv2.imread(test_image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # OpenCV는 BGR 형식이므로 RGB로 변환
image_rgb = image_rgb.astype(np.float32) / 255.0 # 정규화
image_tensor = torch.tensor(image_rgb).permute(2, 0, 1).unsqueeze(0).to(DEVICE) # 텐서로 변환

# 예측 수행
with torch.no_grad():
    predictions = model(image_tensor)

# 예측 결과 시각화
pred_boxes = predictions[0]["boxes"].cpu().numpy()
pred_scores = predictions[0]["scores"].cpu().numpy()
pred_masks = predictions[0]["masks"].cpu().numpy()

# 신뢰도 임계값 설정 (0.5 이상만 표시)
threshold = 0.5
sample = (image_rgb * 255).astype(np.uint8) # 원본 이미지로 변환
```

# 코드 구현

```
# 원하는 색상 리스트 정의 (RGB 값으로 설정)
# 각 객체마다 다른 색상 설정
colors = [
    [0, 255, 0],    # 초록색
    [255, 0, 0],    # 빨간색
    [0, 0, 255],    # 파란색
    [0, 255, 255],  # 시안
    [255, 0, 255],  # 마젠타
    [255, 255, 0],  # 노란색
] # 필요한 만큼 색상 추가

for i, box in enumerate(pred_boxes):
    if pred_scores[i] >= threshold:
        # 바운딩 박스 그리기
        box = box.astype(int)
        cv2.rectangle(sample, (box[0], box[1]), (box[2], box[3]), (255, 0, 0), 2)

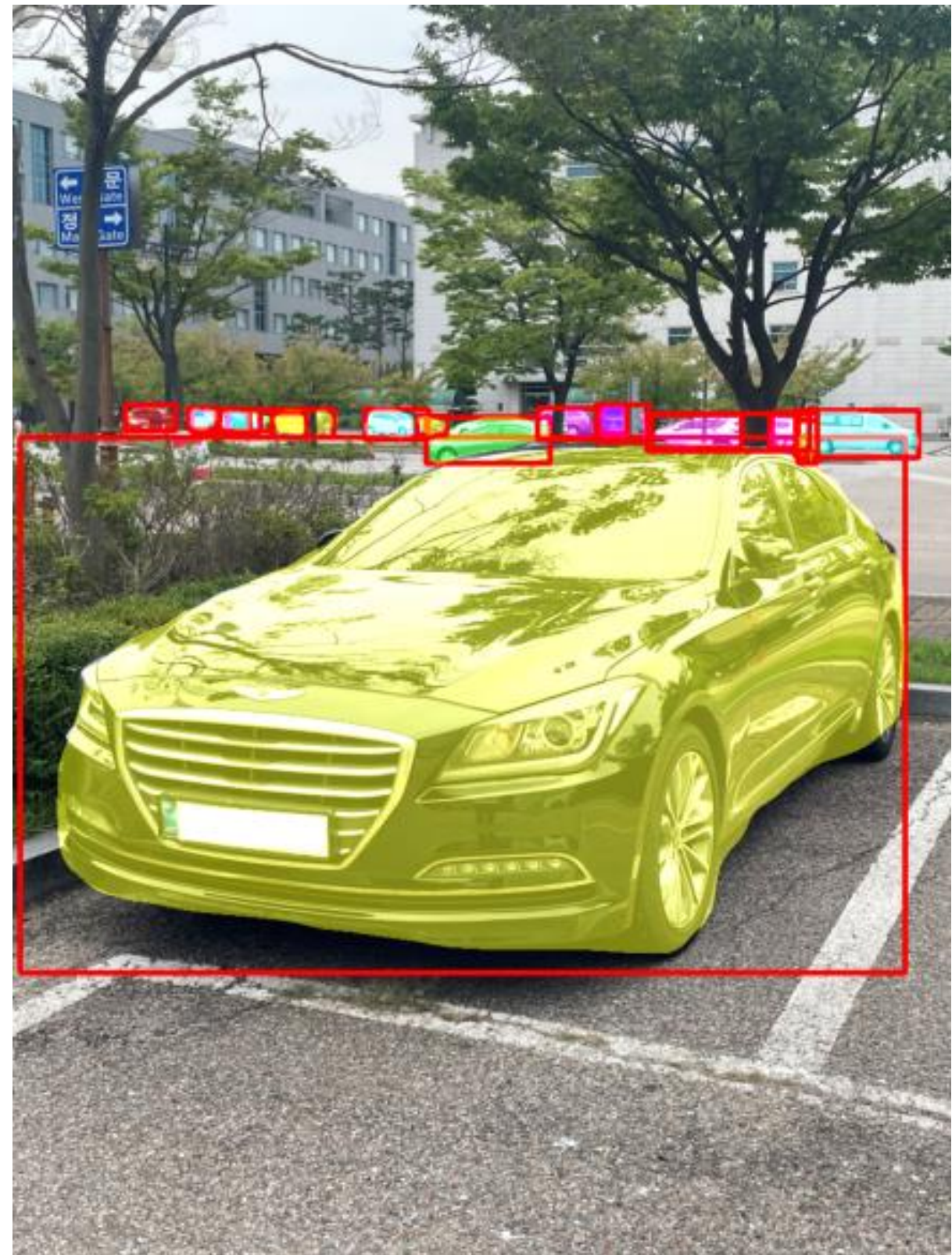
        # 마스크 그리기
        mask = pred_masks[i, 0] > threshold
        colored_mask = np.zeros_like(sample, dtype=np.uint8)

        # 색상을 순서대로 할당 (i 값을 색상 리스트의 길이로 나눈 나머지를 사용)
        color = colors[i % len(colors)]
        colored_mask[mask] = color # 객체별로 지정한 색상을 마스크에 적용

        sample = cv2.addWeighted(sample, 1, colored_mask, 0.5, 0)

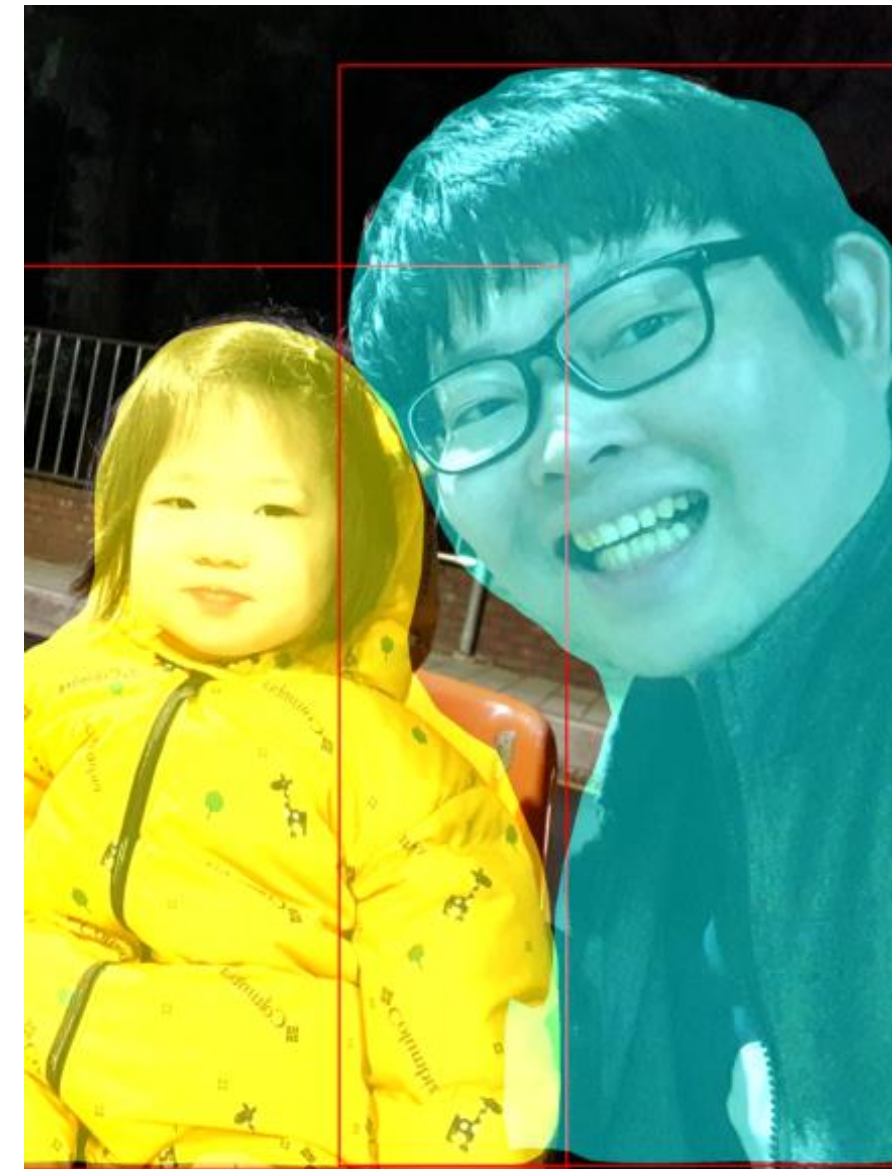
# 결과 이미지 출력
plt.imshow(sample)
plt.axis("off")
plt.show()
```

## 결과 – 자동차



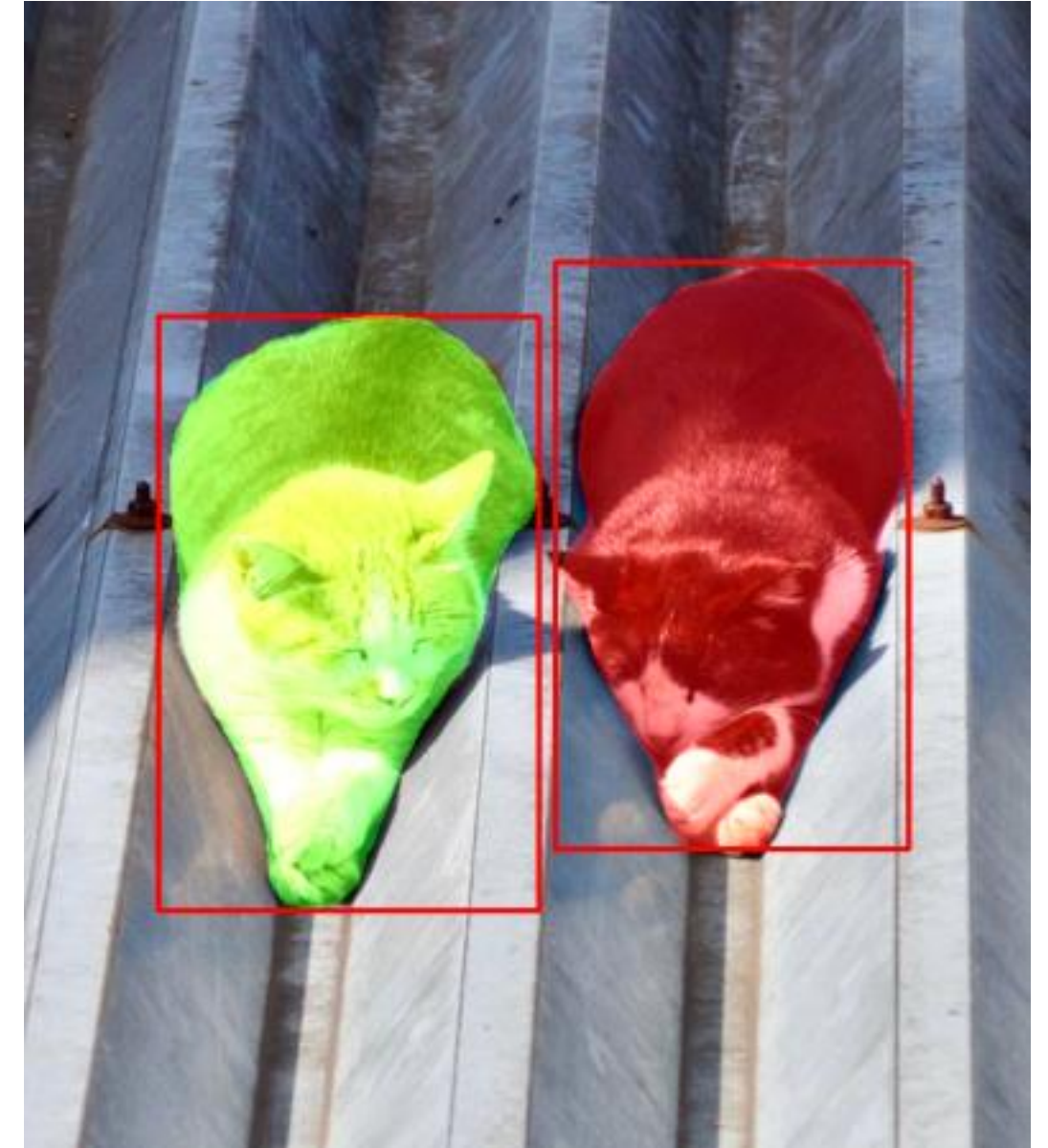
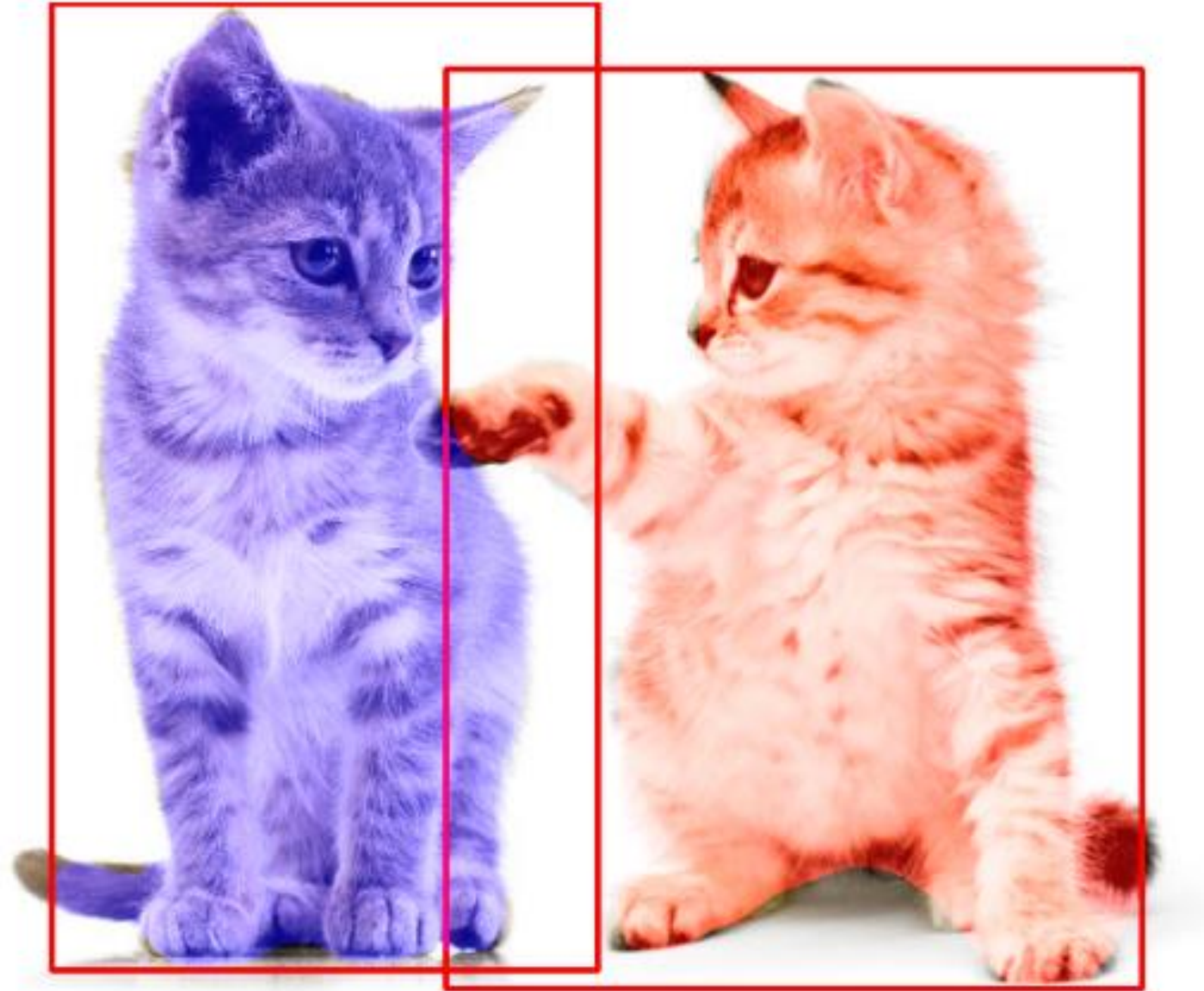
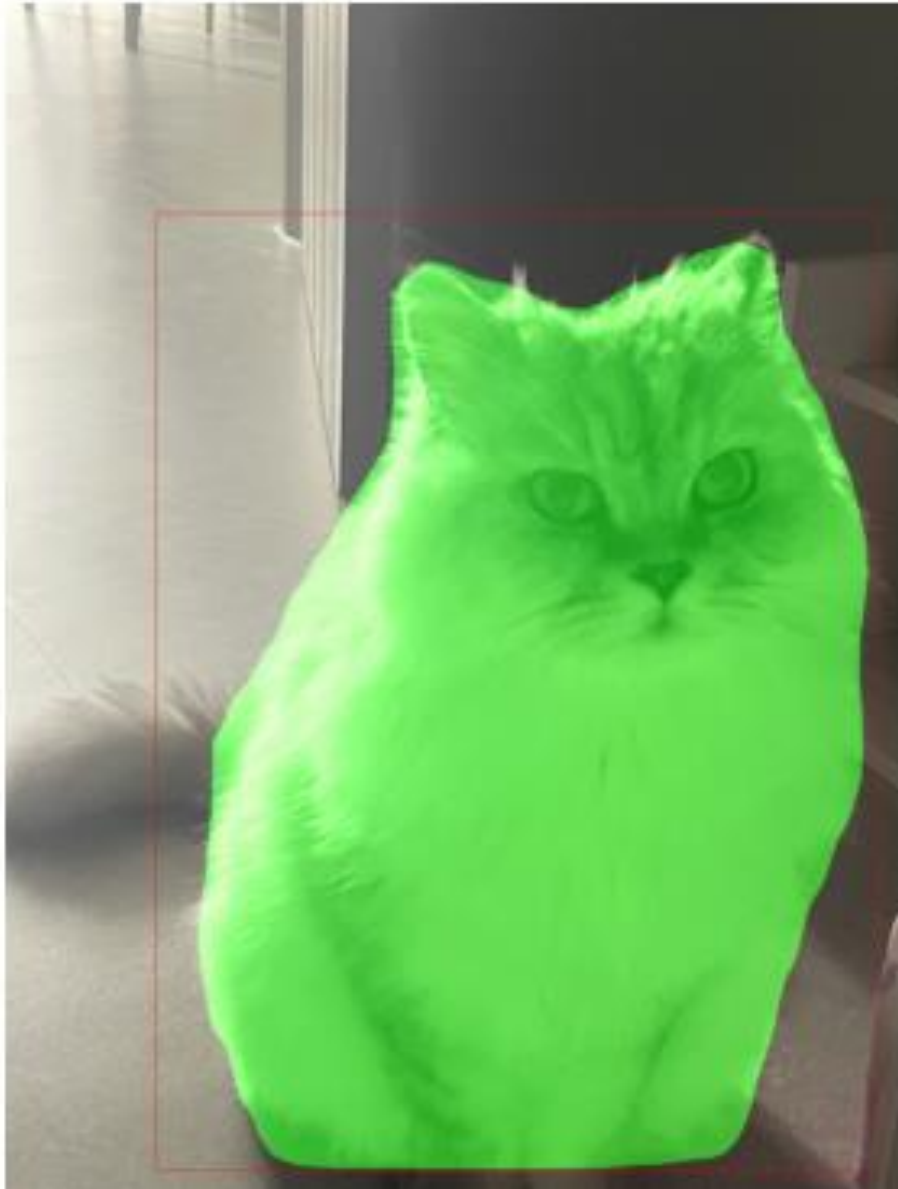


## 결과 - 사람





## 결과 - 동물





**THANK YOU**  
**Q&A**