2 Stage Object Detection

팀 레모네이드 | 강희준 박은수 장우진 김선준 장재우

목 차

01 개요

02 R-CNN

03 Fast R-CNN

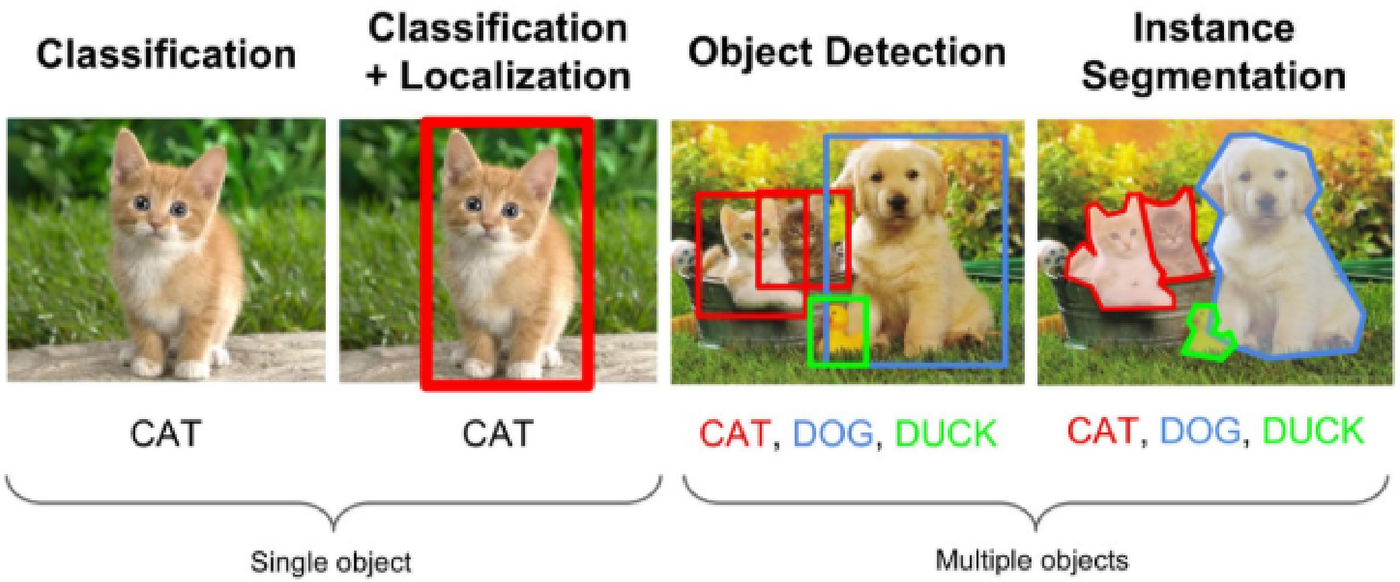
04 Faster R-CNN

05 Mask R-CNN

06 코드 구현

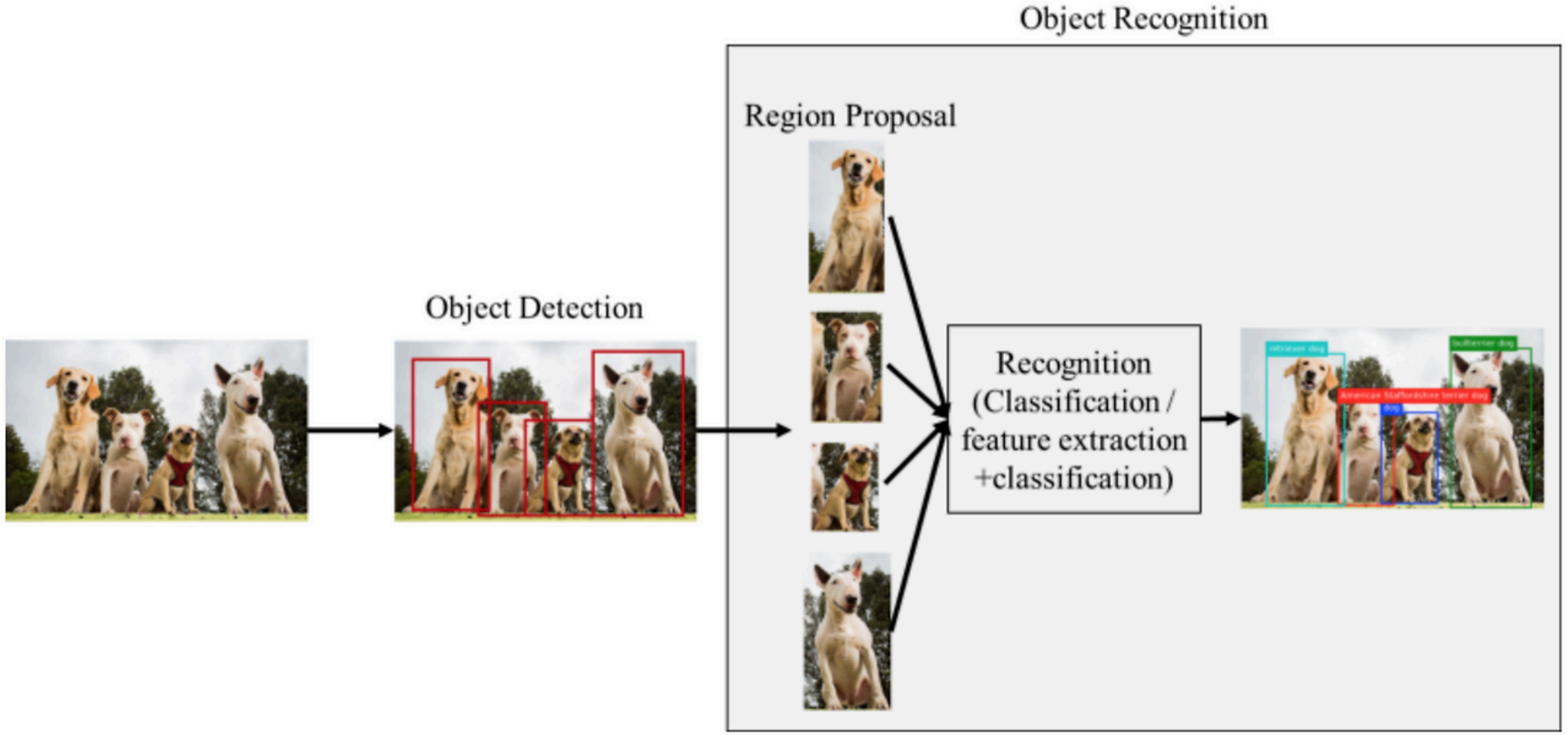
07 참조 자료

Object Detection



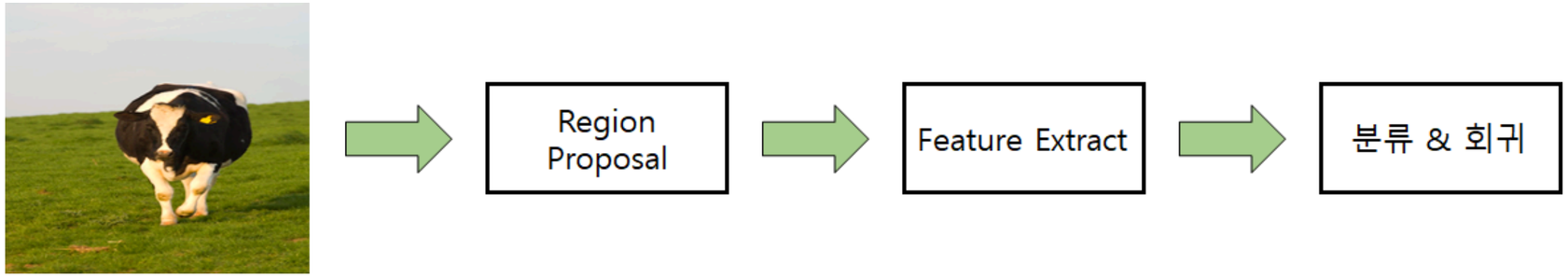
- Classification : Single object의 클래스를 분류하는 문제
- Classification + Localization : Single object에 대해서 object의 위치를 bounding box로 찾고 클래스를 분류하는 문제
- Object Detection : Multiple objects에서 각 object의 위치를 탐지하고 클래스를 분류하는 문제
- Instance Segmentation : Object Detection과 유사하지만, bounding box가 아닌 실제 edge로 찾는 것

Two-stage Object Detection



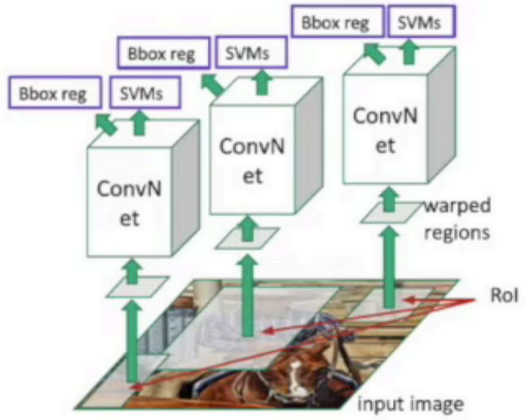
Region Proposal : object가 있을만한 영역을 우선 뽑아낸다 → RoI

convolution network를 통해 classification, box regression(localization)을 수행

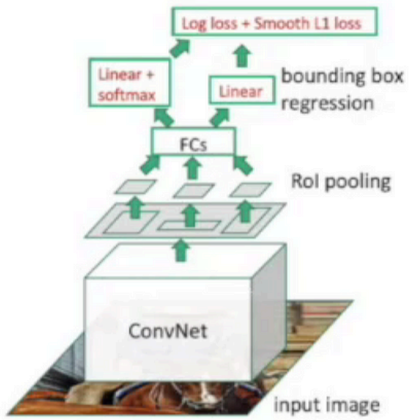


R-CNN 계열 모델

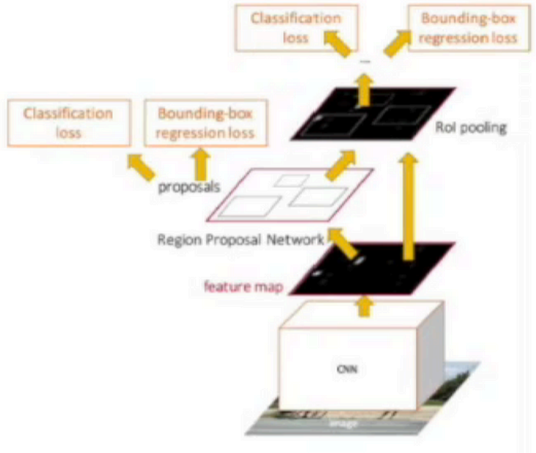
- **R-CNN** (2013)



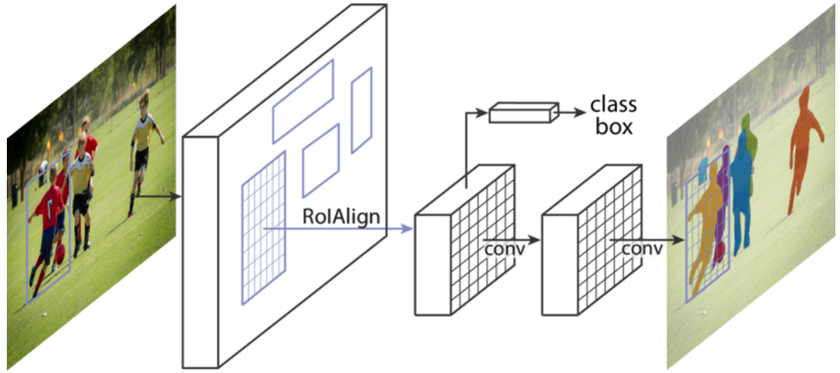
- **Fast R-CNN** (2015)



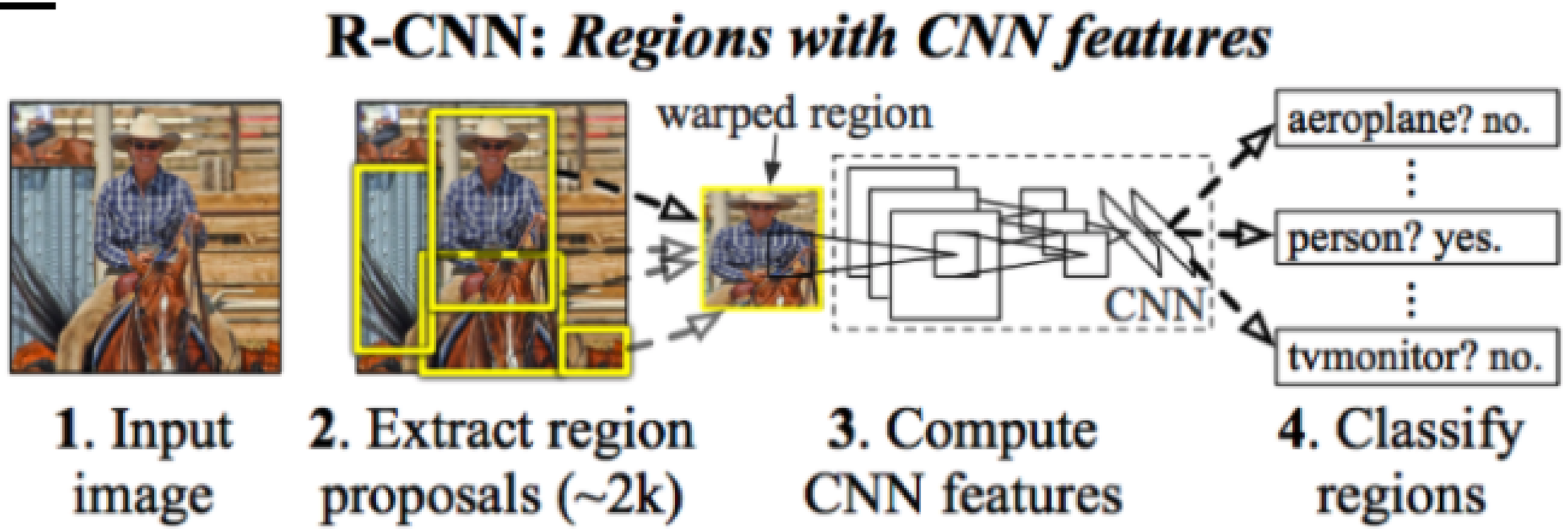
- **Faster R-CNN** (2015)



- **Mask R-CNN (2017)**



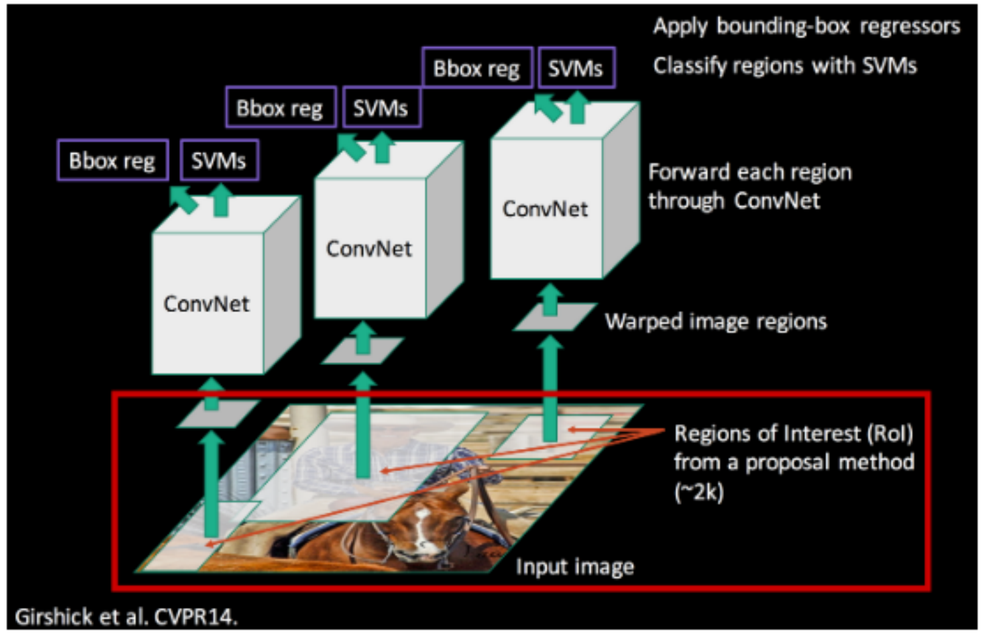
R-CNN

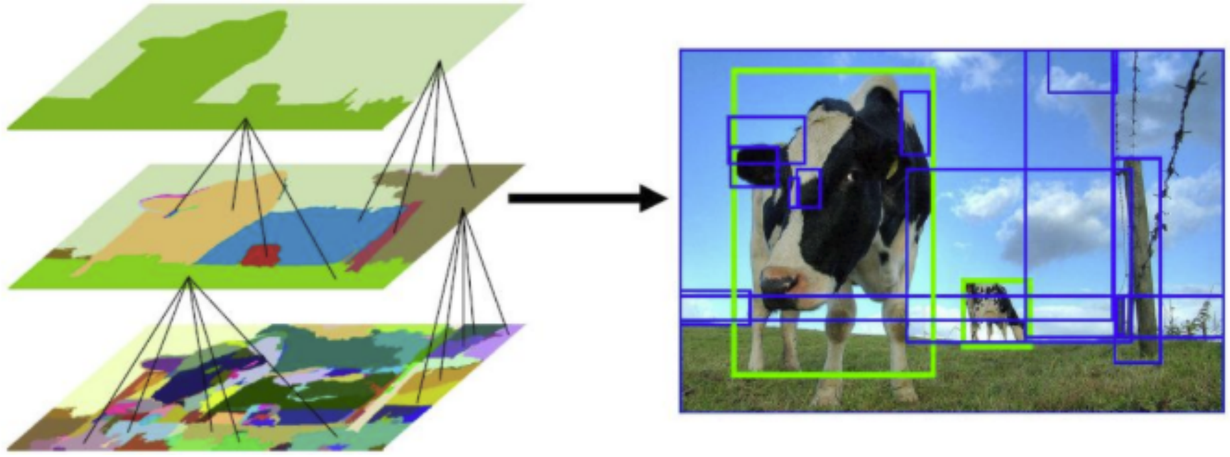


R-CNN 수행과정

1. Image 입력
2. Selective search 알고리즘에 의해 regional proposal output 약 2000개를 추출
3. 마지막 FC layer에서의 input size는 고정이므로 추출한 regional proposal output을 모두 동일 input size로 crop/resize
4. 2000개의 warped image를 각각 CNN 모델에 넣고 각각의 Convolution 결과에 대해 classification/bbox regression을 진행

Region Proposal





selective search

Selective Search 알고리즘 과정

1. 초기 분할

원본 이미지를 색상, 질감, 밝기, 크기 등의 시각적 특징을 기준으로 시각적으로 비슷한 영역끼리 묶어서 segmentation 수행

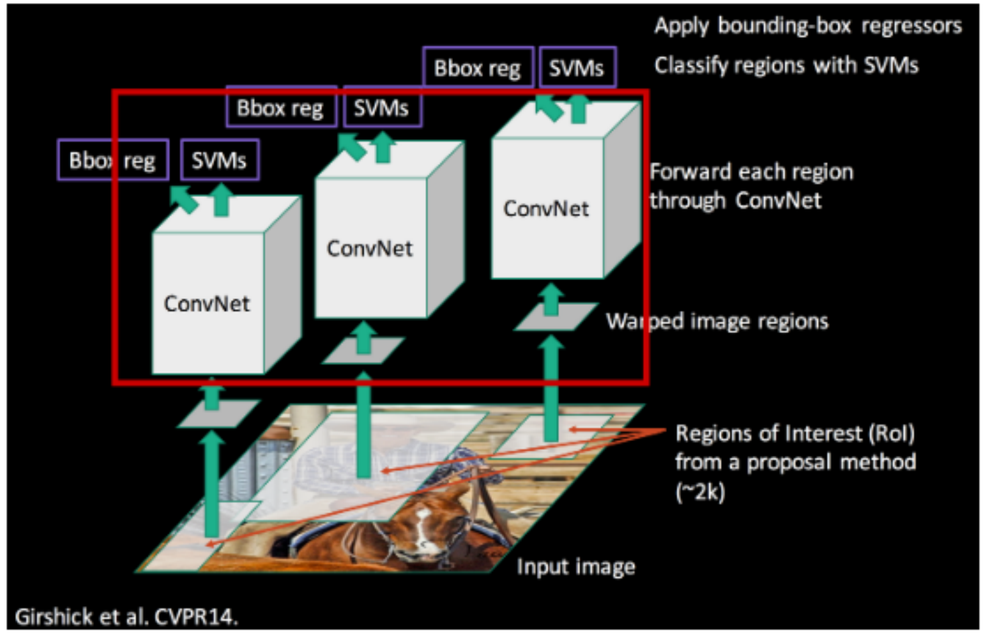
2. 영역 병합(Bottom-up)

작게 분할된 영역들은 영역 간의 유사도(색상, 질감, 크기, 공간적 인접성)를 기준으로 점차 병합합니다.

유사도가 높은 두 영역을 반복적으로 병합하며 점점 더 큰 영역을 형성합니다.

3. Region Proposal 생성 (약 2,000개)

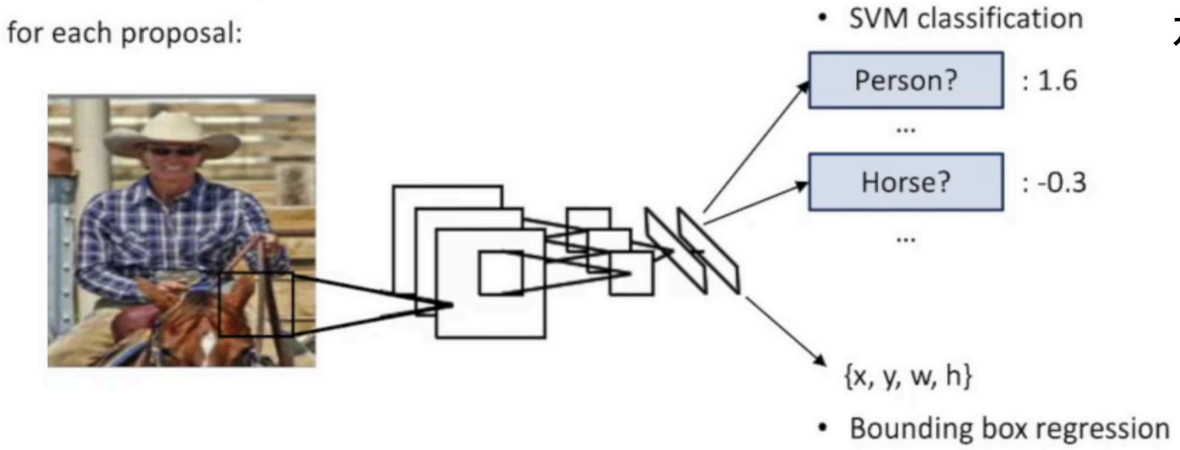
CNN -> Classification & BB regression



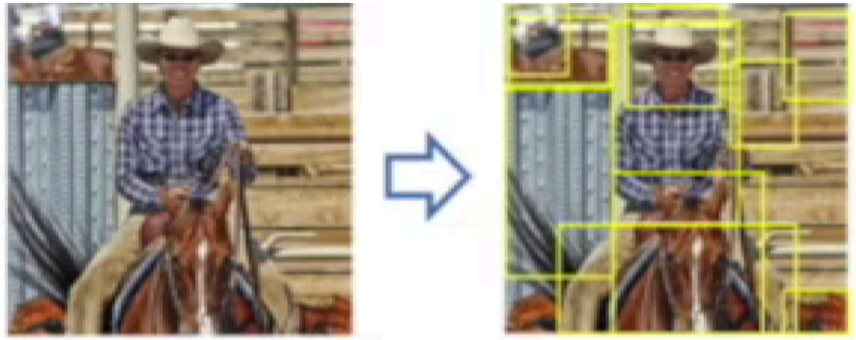
CNN모델로부터 feature가 추출되면 Linear SVM을 통해 각 RoI가 어떤 객체인지를 분류

Selective Search로 생성된 region proposal은 ground truth box와 비교해 위치 오차가 있음.

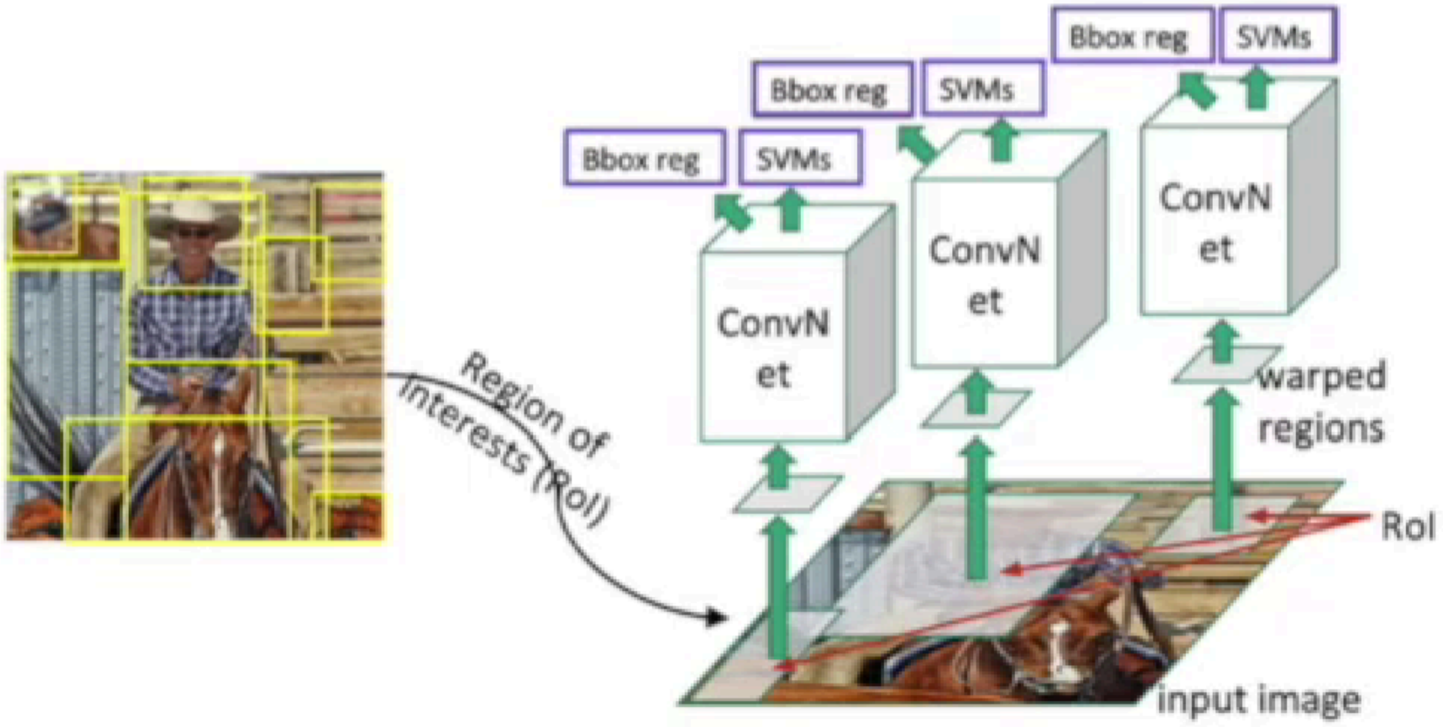
BB regression → CNN feature를 이용해 proposal box와 실제 GT box의 좌표 차이(offset)를 학습하는 선형 회귀 모델로, 객체의 위치를 미세 조정하는 역할을 수행



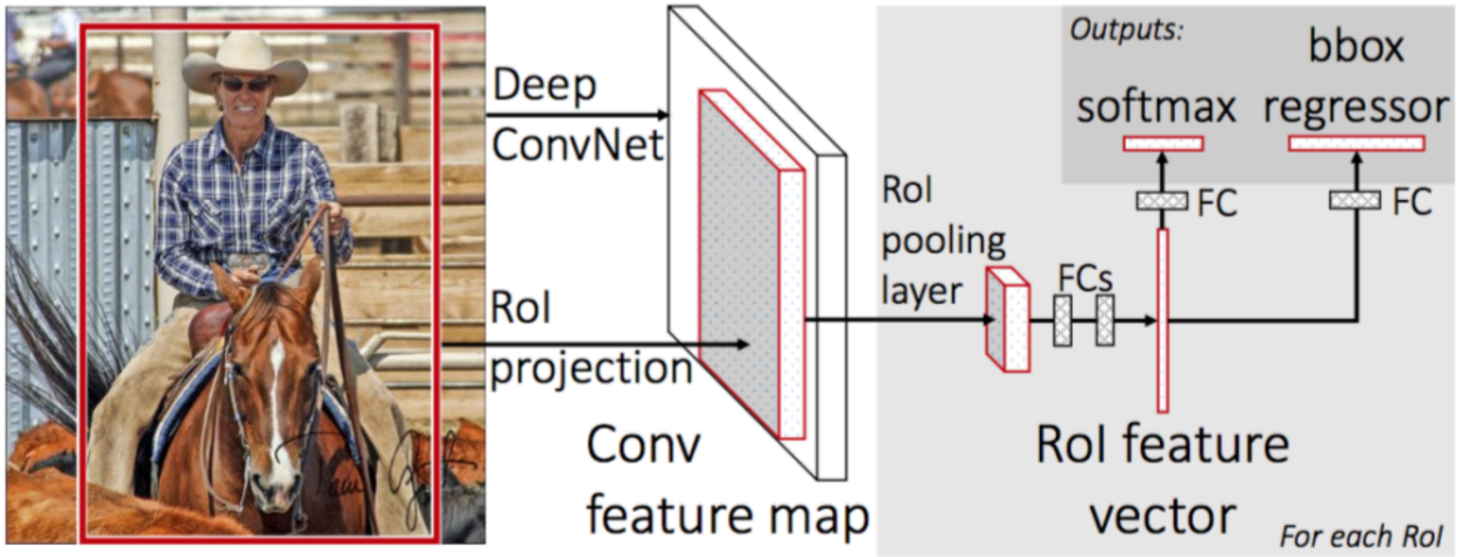
Selective Search → Region Proposals: 2 sec/image



CNN Computation (for each region proposal) → SVM & BB reg.: 47 sec/image



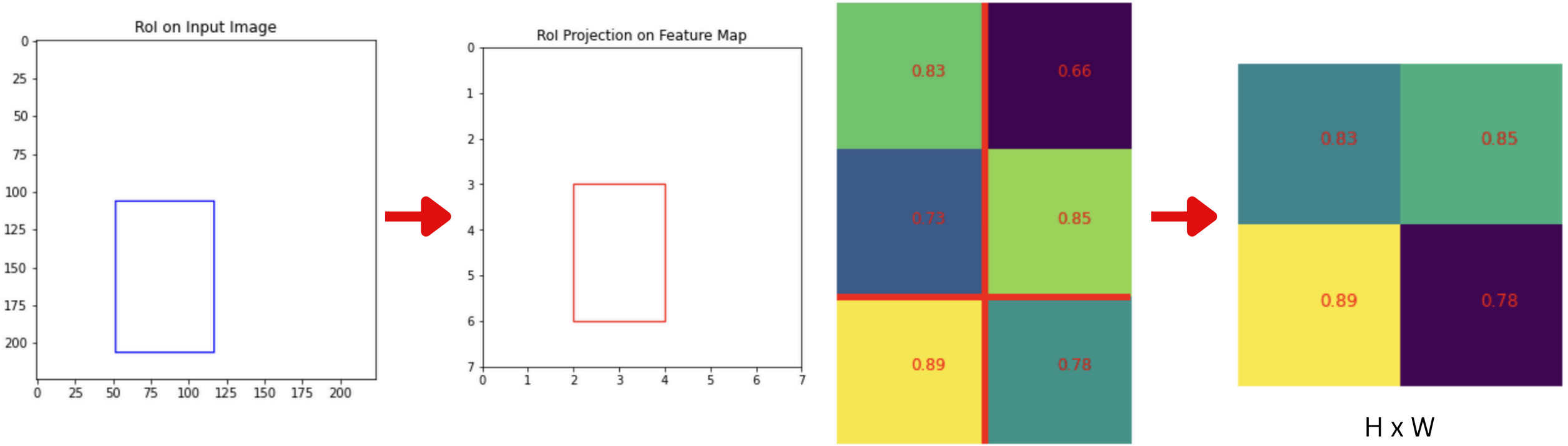
Fast R-CNN



Fast R-CNN 수행과정

1-1. Selective Search를 통해 RoI를 생성
1-2. 원본 이미지를 CNN에 통과시켜 feature map을 추출
2. Selective Search로 찾았었던 RoI를 feature map크기에 맞춰서 projection 하기
3. projection시킨 RoI에 대해 RoI Pooling을 진행하여 고정된 크기의 feature vector 얻기
4. feature vector는 FC layer를 통과한 뒤, 2개의 브랜치로 나뉨
5-1. 하나는 softmax를 통과하여 RoI에 대해 object classification 진행
5-2. bounding box regression을 통해 selective search로 찾은 box의 위치를 조정

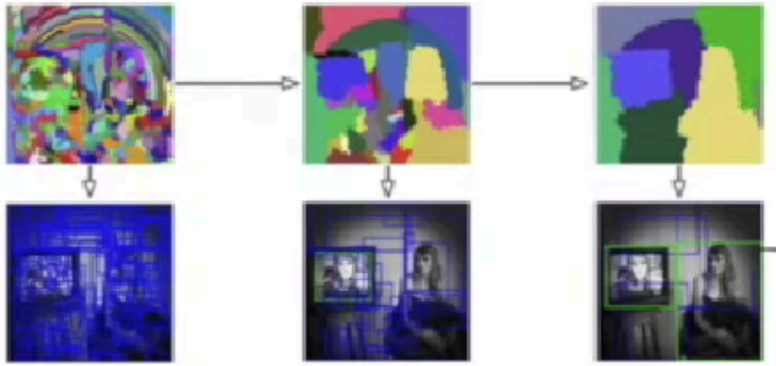
RoI Pooling

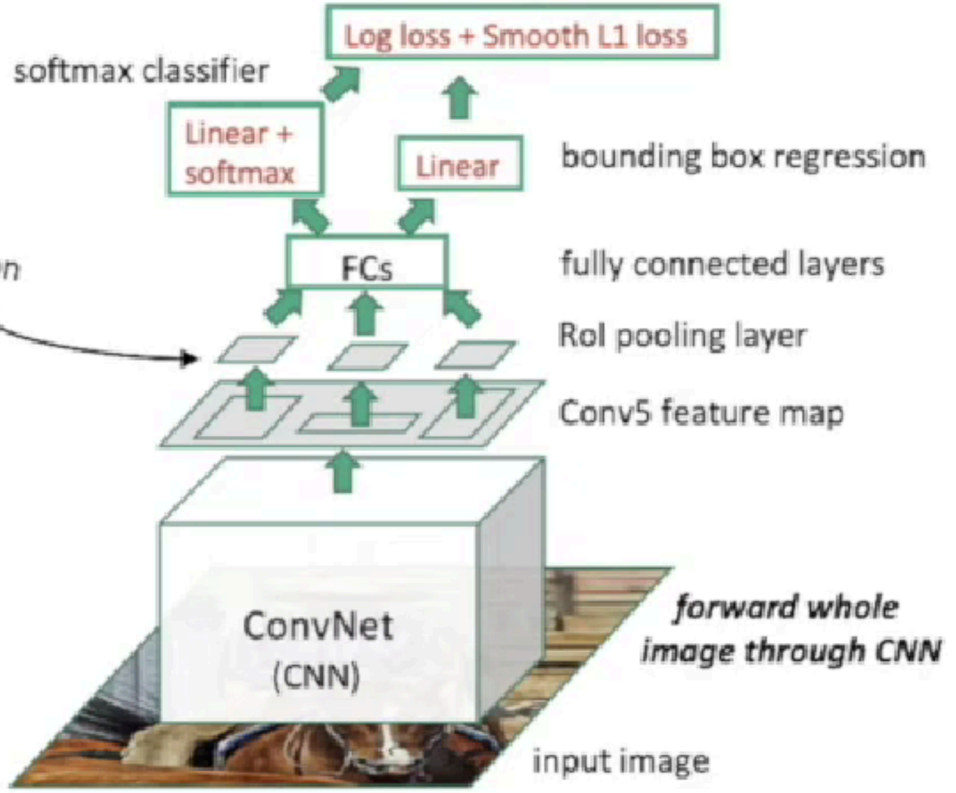


(1) 미리 설정한 피라미드 HxW크기로 만들어주기 위해서 (h/H) * (w/H) 크기만큼 grid를 RoI위에 만든다.
(2) RoI를 grid크기로 split시킨 뒤 max pooling을 적용
H×W, feature map위의 RoI가 h×w일 때 pool size는 h/H, w/W으로 계산
(이 예시에서는 projected roi를 2×2크기로 pooling한다고 할 때 반올림해서 2×1만큼의 영역을 우선적으로 잘라서 max pooling을 수행함)

● Fast R-CNN

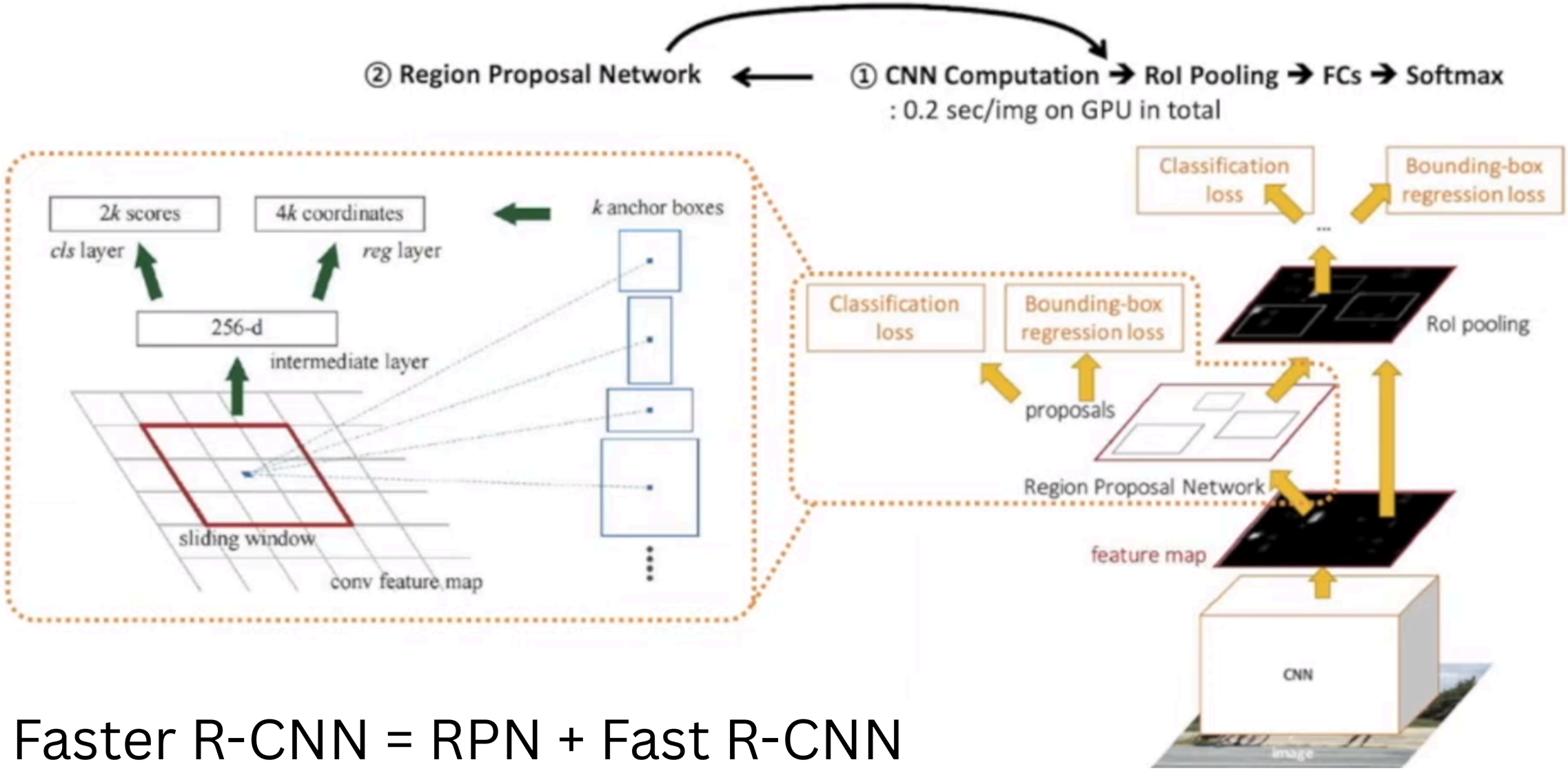
- Image
- Selective Search ➔ Region Proposal
  : 2 sec/img on CPU
- CNN Computation ➔ RoI Pooling ➔ FCs ➔ Softmax & Reg.
  : 0.3 sec/img on GPU





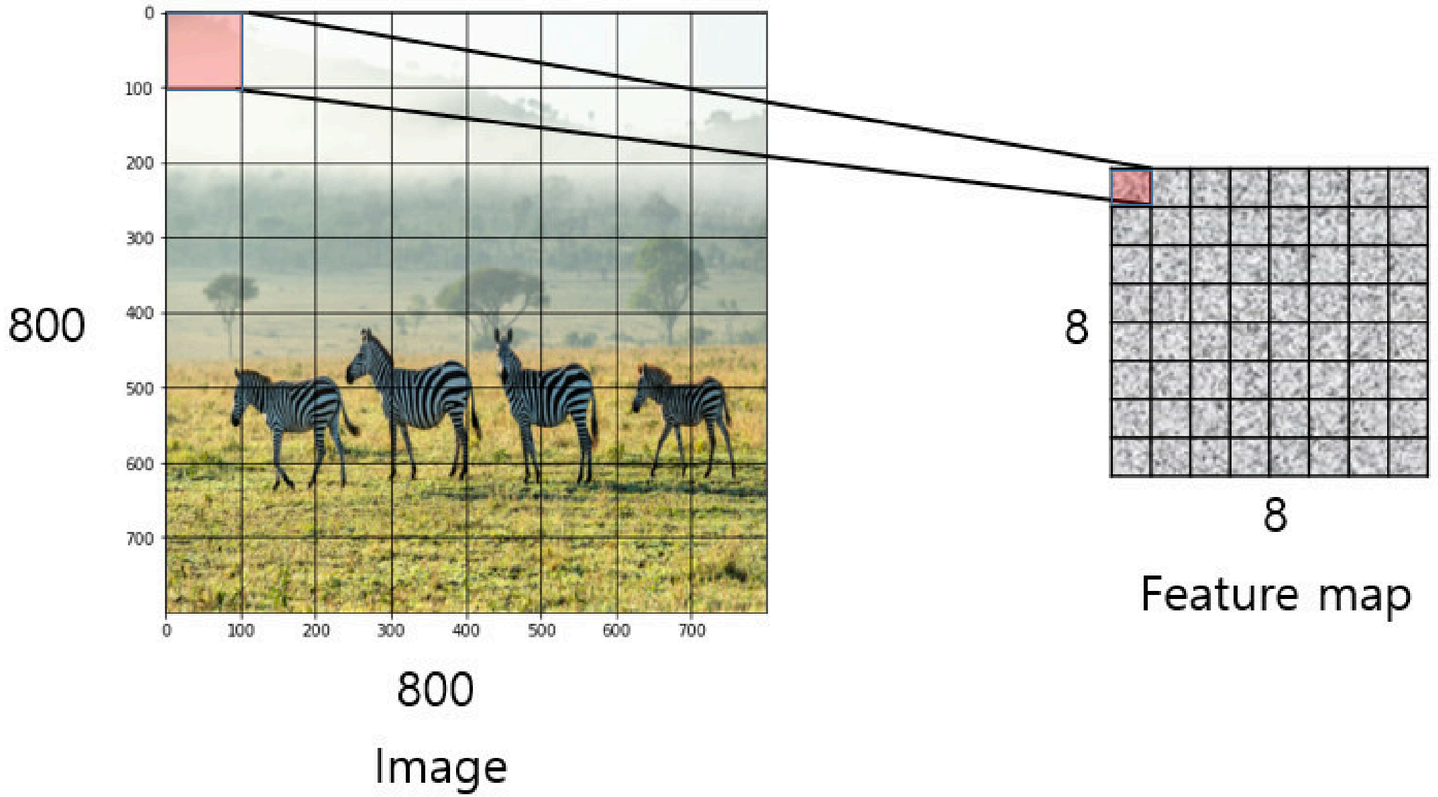
New Bottleneck !!

Faster R-CNN



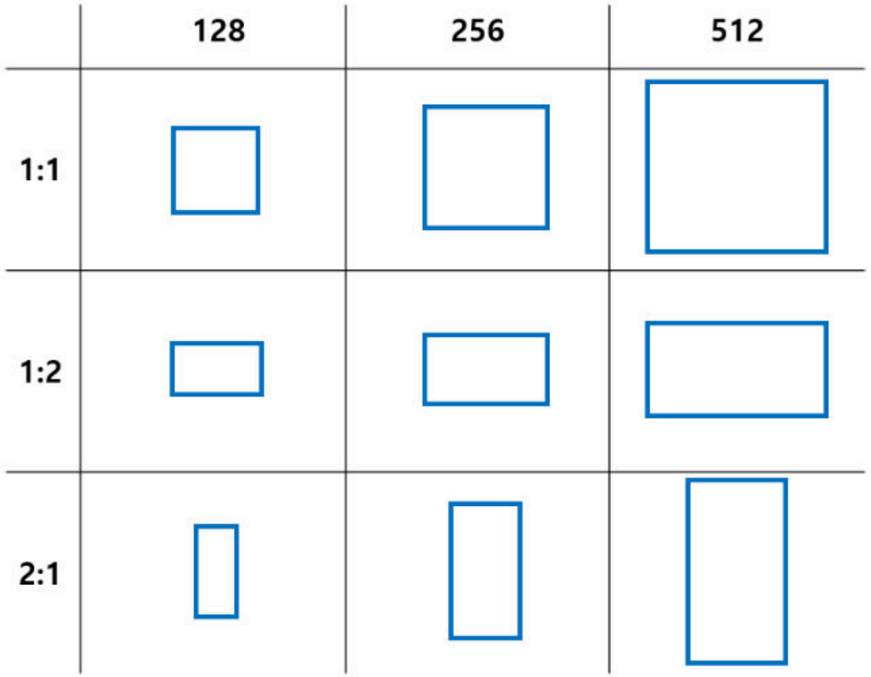
Faster R-CNN = RPN + Fast R-CNN

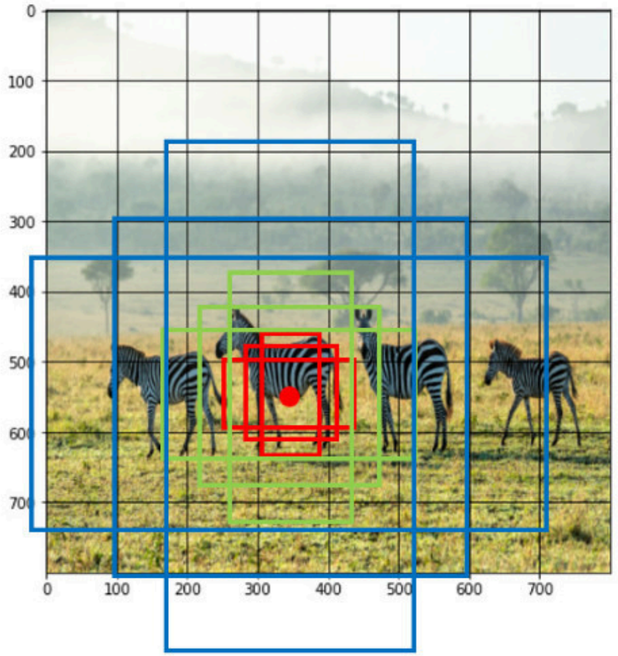
Region Proposal Network



- 원본 이미지는 CNN backbone을 통과하여 feature map 생성
- feature map 상의 한 좌표는 원본 이미지 상의 일정한 위치에 대한 시각적 정보를 요약
  - 원본 이미지 상의 100×100 영역의 시각적 정보를 함축

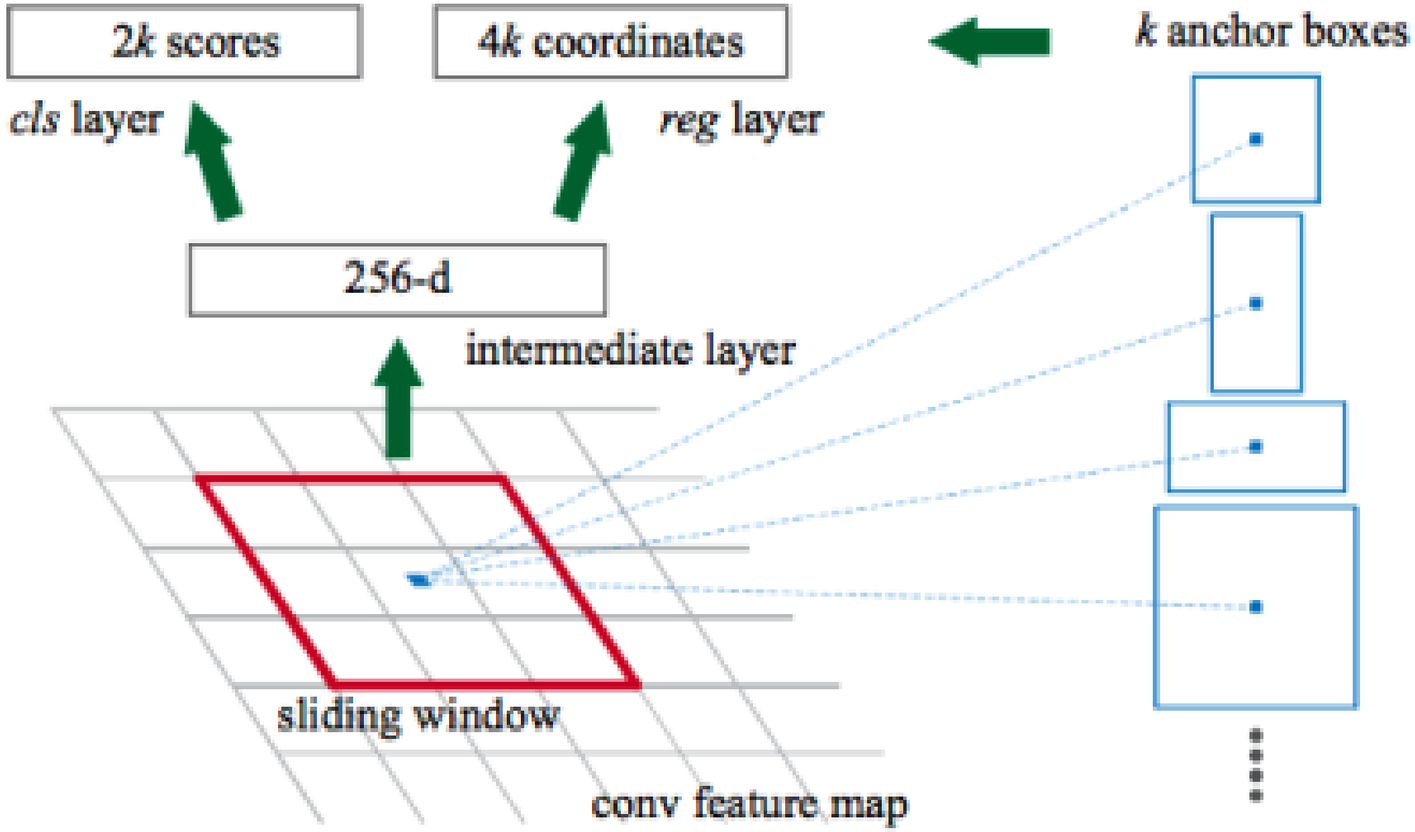
Region Proposal Network





- feature map의 각 픽셀은 anchor box의 중심점
- 각 영역에 대해 3개의 scale과 3개의 aspect ratio를 조합하여 총 9개의 anchor box를 생성
  - 총 8x8x9 = 576 개의 anchor box 생성

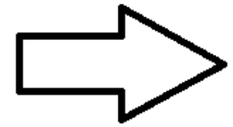
Region Proposal Network



단계	출력 형태	의미
CNN backbone	(50×38×512)	기본 시각적 feature
3×3 conv (256 filters)	(50×38×256)	주변 context를 포함한 RPN 전용 feature
1×1 conv (18 filters)	(50×38×18)	9개 anchor의 objectness score (2×9)
1×1 conv (36 filters)	(50×38×36)	9개 anchor의 bbox regression (4×9)

"모든 가능한 위치에 미리 anchor box를 뿌려놓고,
그중 진짜 물체일 것 같은 것만 골라내고,
약간의 이동(offset)을 줘서 진짜 물체를 감싸도록 보정한다."

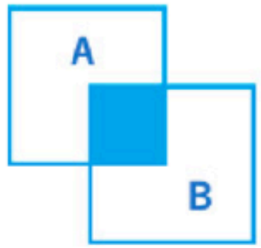
Region Proposal Network

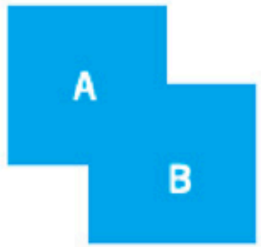


- Binary classification을 기준으로 score가 높은 상위 anchor box(예: 200~300개)**만 남겨 RoI로 사용

Intersection over Union

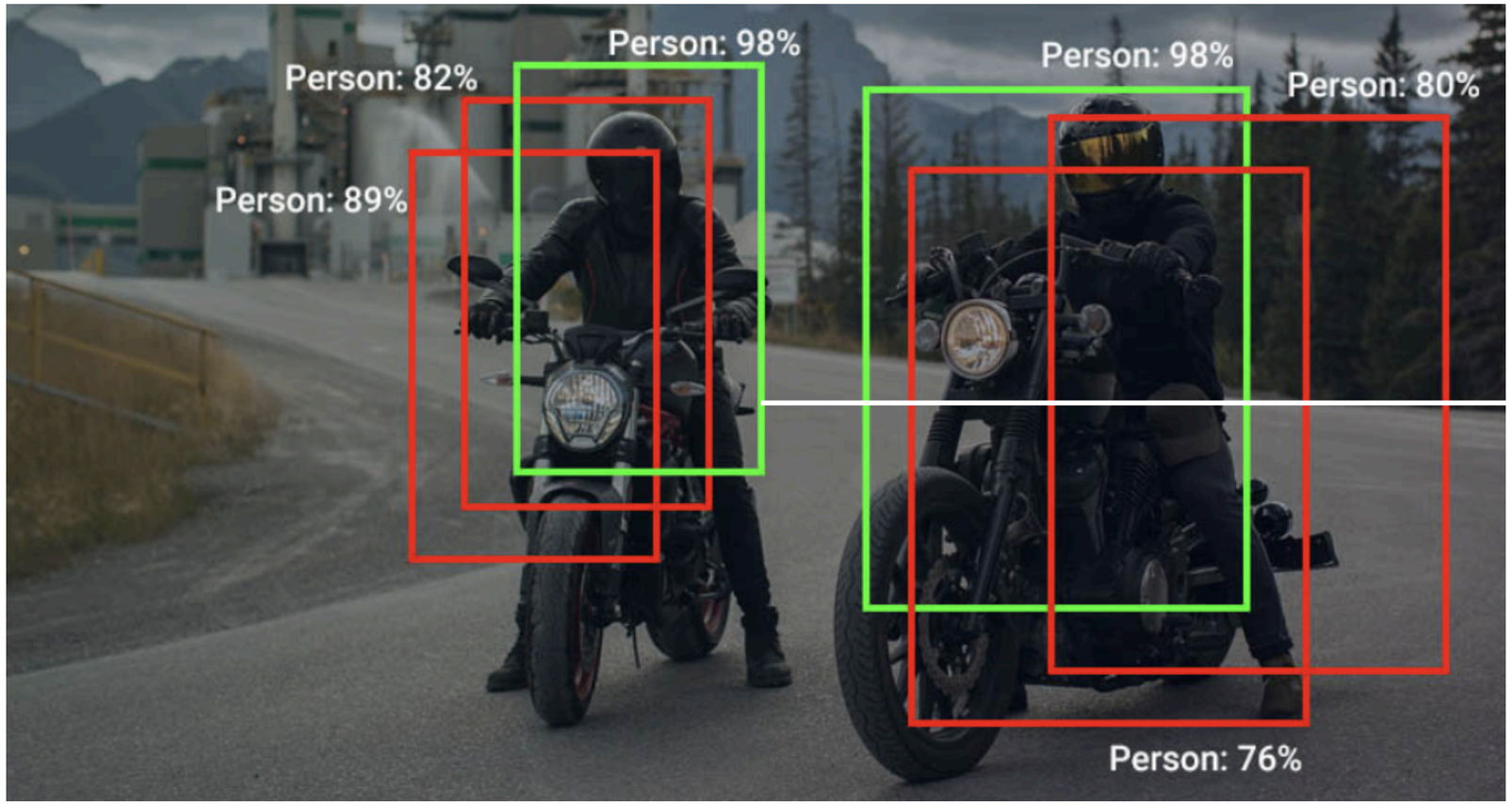
$$\text{Intersection over Unit (IoU)} = \frac{\text{Area of overlap}}{\text{Area of union}}$$





$$= \frac{|A \cap B|}{|A \cup B|}$$

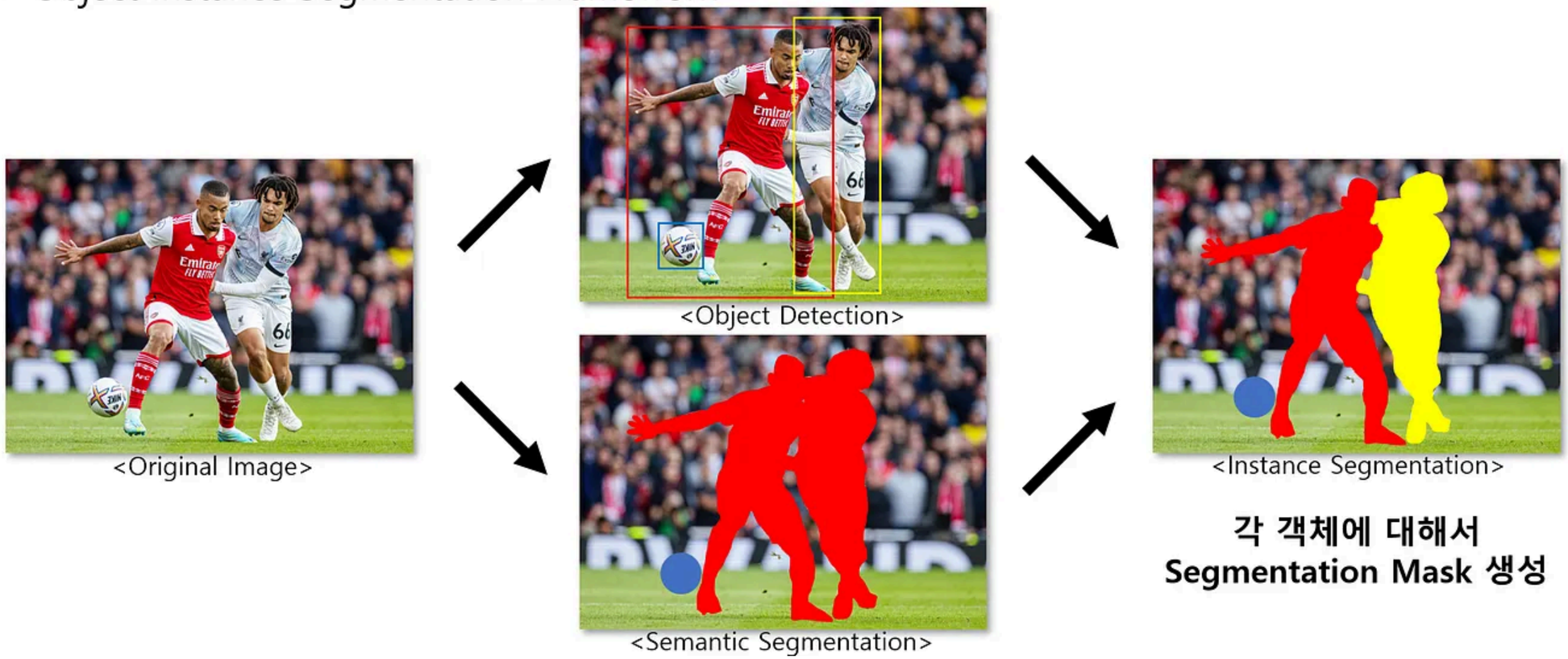
Non-maximum Suppression



- 목적: anchor box 중 중복된 후보를 제거
- 동작 절차
  a. score 기준으로 내림차순 정렬
  b. 가장 높은 박스를 선택
  c. 선택된 박스와 다른 박스 간 IoU 계산
  d. IoU threshold 이상일 경우 제거

Mask R-CNN

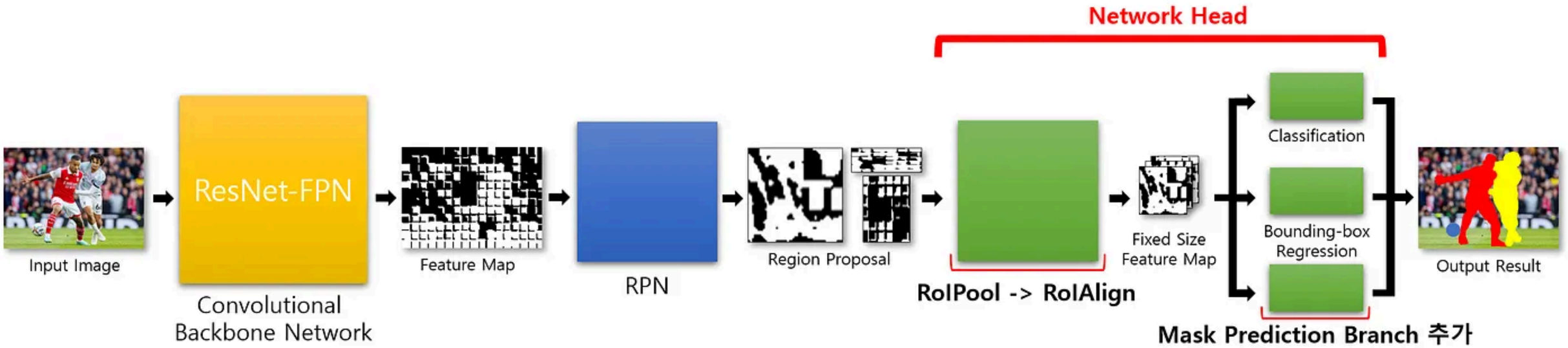
▪ Object Instance Segmentation Framework



- Semantic Segementation은 한 클래스에 대해 하나의 Segmentation Mask를 생성
- Instance Segmentation은 각 객체에 대해서 Segmentation Mask를 생성

Mask R-CNN

- Mask R-CNN은 세 개의 Stage로 구성



Faster R-CNN과의 차이점 2가지

- RoI pooling → RoI Align
- Mask Prediction Branch에서 Segmentation Mask를 생성

RoI Align

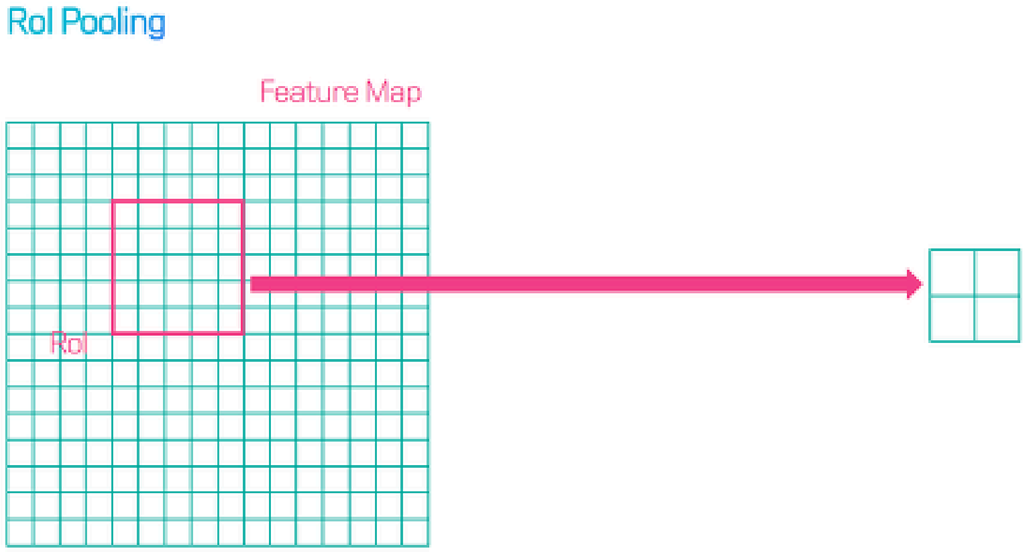
0.8	1.1	2.1	3.9	0.4
2.2	4.3	4.2	1.8	6.3
0.2	5.5	4.8	7.7	9.5
1.1	0.8	2.2	0.8	5.2
5.0	1.8	1.9	2.0	4.1

5 x 5 Feature Map
Red Dotted Line = RoI

Quantization

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

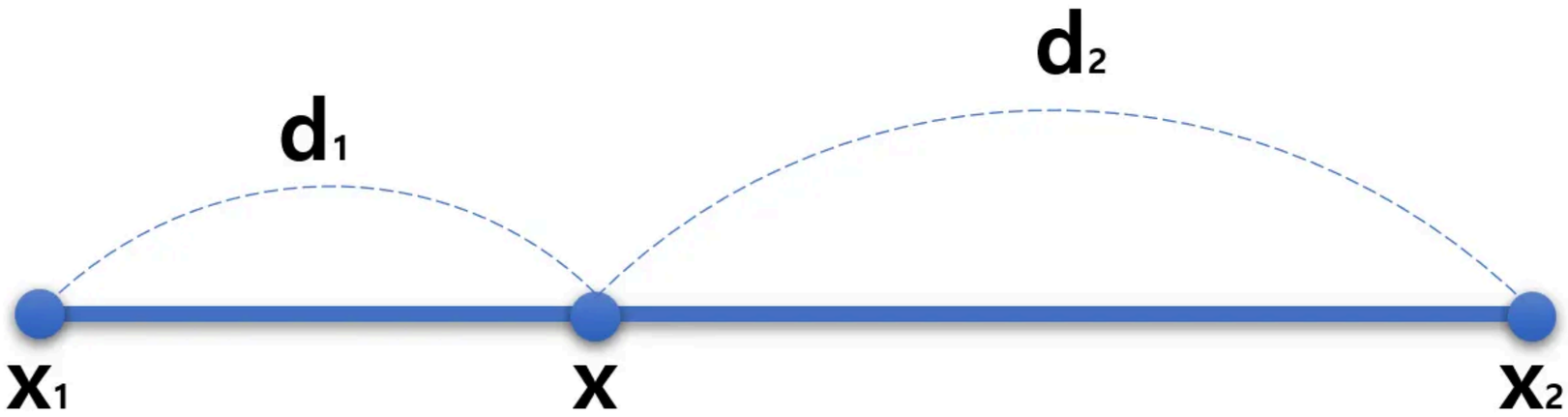
Misalignment 발생
Segmentation 성능 저하



RoI pooling의 문제

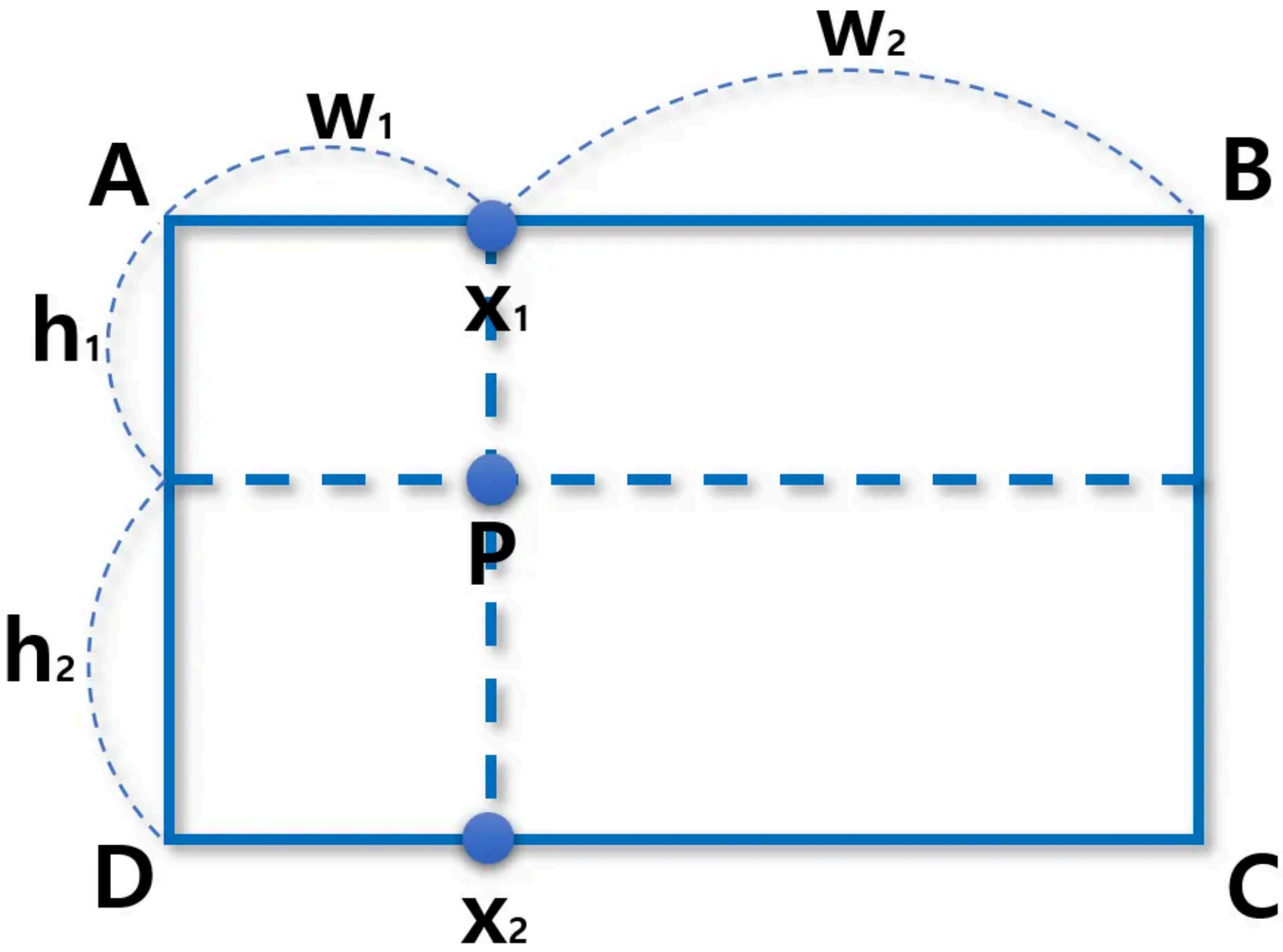
- Misalignment로 인해 데이터 유실이 발생하고, 원하지 않는 데이터가 RoI에 포함
- 따라서 segmentation 성능 저하

Linear Interpolation



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

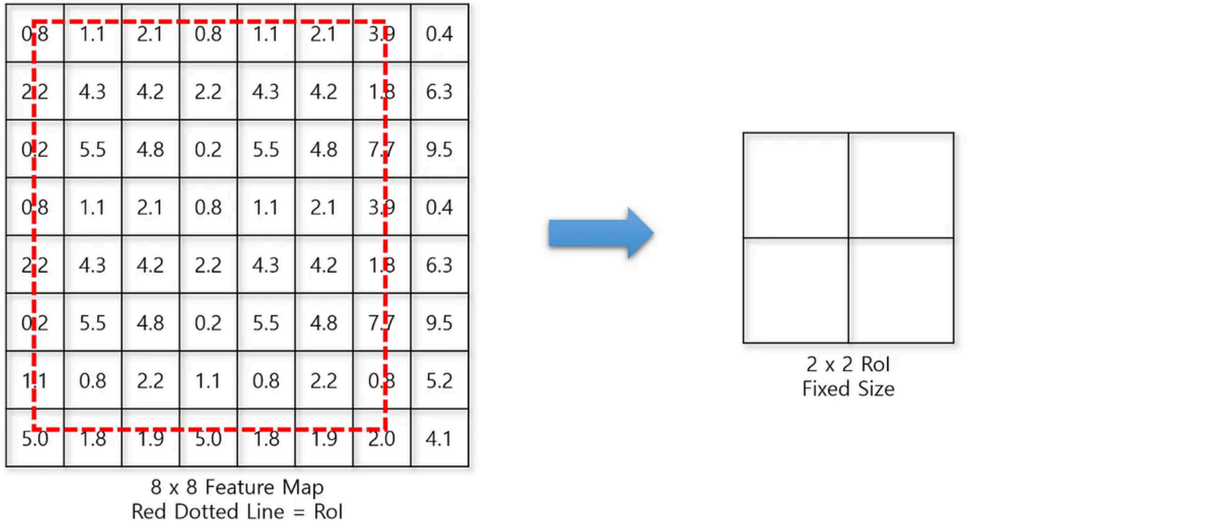
Bilinear Interpolation



1. 점 A와 B 사이에 Linear Interpolation을 수행하여 x1 값 구하기.
2. 점 D와 C 사이에 Linear Interpolation을 수행하여 x2 값 구하기.
3. 점 x1와 x2 사이에 Linear Interpolation을 수행하여 P값 구하기.

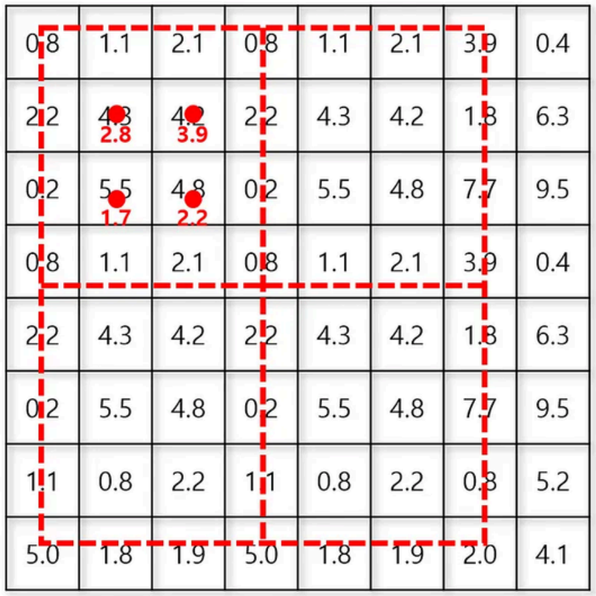
1~3 과정을 통해 P 값을 구한 계산이 바로 Bilinear Interpolation입니다.

Bilinear Interpolation



8 x 8 Feature Map
Red Dotted Line = RoI

2 x 2 RoI
Fixed Size



8 x 8 Feature Map
Red Dotted Line = RoI

2 x 2 RoI
Fixed Size

1. 왼쪽의 RoI를 2 x 2로 나눠서 4개의 Bin을 생성하기.
2. 하나의 Bin을 3 x 3로 나눴을 때 교차점이 되는 지점 4개를 찍기.
3. 각 점에 대해 Bilinear Interpolation을 수행하여 추정값을 구하기.
4. 추정 값 4개에 대해 Max Pooling을 사용해서 최대값을 뽑기.
5. 2~4의 과정을 나머지 3개의 Bin에 대해서도 수행하기.
6. 그렇게 되면 오른쪽에서 보이는 2 x 2 크기의 고정된 RoI를 생성할 수 있게 됩니다.

코드구현 - Mask RCNN

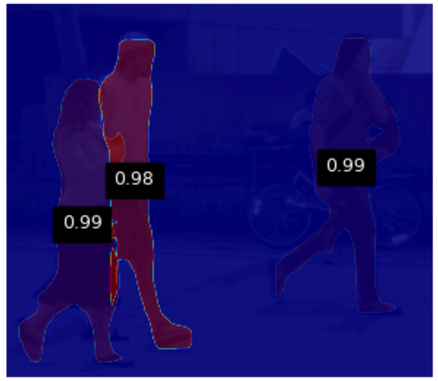
```python
# ----------------------------
# 모델 불러오기/교체(2 classes: bg/person)
# ----------------------------
def get_mask_model(num_classes=2):
    model = torchvision.models.detection.maskrcnn_resnet50_fpn(weights="DEFAULT")
    # box predictor 교체
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = torchvision.models.detection.faster_rcnn.FastRCNNPredictor(in_features, num_classes)
    # mask predictor 교체
    in_mask = model.roi_heads.mask_predictor.conv5_mask.in_channels
    model.roi_heads.mask_predictor = torchvision.models.detection.mask_rcnn.MaskRCNNPredictor(in_mask, 256, num_classes)
    return model

# ----------------------------
# 학습 루프
# ----------------------------
def train(model, loader, epochs=EPOCHS, lr=LR, wd=WD, device=DEVICE):
    model.to(device).train()
    opt = torch.optim.AdamW([p for p in model.parameters() if p.requires_grad], lr=lr, weight_decay=wd)
    for ep in range(1, epochs+1):
        total = 0.0
        t0 = time.time()
        for imgs, tgts in loader:
            imgs = [i.to(device) for i in imgs]
            tgts = [{k:v.to(device) if torch.is_tensor(v) else v for k,v in t.items()} for t in tgts]
            loss_dict = model(imgs, tgts)
            loss = sum(loss_dict.values())
            opt.zero_grad(); loss.backward(); opt.step()
            total += float(loss)
        print(f"[Train] epoch {ep}/{epochs} loss={total/len(loader):.4f} time={(time.time()-t0):.1f}s")
    model.eval()
    return model
```

```
[INFO] Train size=127, Test size=43, Device=cuda
[Train] epoch 1/5 loss=0.4545 time=40.3s
[Train] epoch 2/5 loss=0.2693 time=56.1s
[Train] epoch 3/5 loss=0.2350 time=60.2s
[Train] epoch 4/5 loss=0.2122 time=55.1s
[Train] epoch 5/5 loss=0.1961 time=43.7s
```

코드구현 - Mask RCNN

PennPed00090.png • Mask R-CNN (person, score≥0.5)

PennPed00066.png • Mask R-CNN (person, score≥0.5)



FudanPed00008.png • Mask R-CNN (person, score≥0.5)


```python
# ----------------------------
# 평가 루틴 (bbox/mask AP@0.5 + latency/FPS)
# ----------------------------
@torch.no_grad()
def evaluate(model, loader, device=DEVICE, score_thresh=SCORE_THRESH):
    model.eval().to(device)
    preds_box, gts_box = [], []
    preds_mask, gts_mask = [], []
    latencies = []

    for imgs, tgts in loader:
        imgs = [i.to(device) for i in imgs]
        t0 = time.time()
        outputs = model(imgs)  # list of dict
        lat = (time.time() - t0) * 1000.0 / len(imgs)  # ms/img
        latencies.append(lat)

        for out, gt in zip(outputs, tgts):
            # 사람만 + score_thresh
            keep = (out["labels"] == PERSON_ID) & (out["scores"] >= score_thresh)
            boxes = out["boxes"][keep].detach().cpu().numpy()
            scores = out["scores"][keep].detach().cpu().numpy()
            masks = (out["masks"][keep, 0] > 0.5).detach().cpu().numpy()

            preds_box.append({"boxes": boxes, "scores": scores})
            preds_mask.append({"masks": masks, "scores": scores})

            g_boxes = gt["boxes"].detach().cpu().numpy()
            g_masks = gt["masks"].detach().cpu().numpy().astype(bool)
            gts_box.append({"boxes": g_boxes})
            gts_mask.append({"masks": g_masks})

    ap_box = compute_ap_50(preds_box, gts_box, is_mask=False)
    ap_mask = compute_ap_50(preds_mask, gts_mask, is_mask=True)
    avg_lat = float(np.mean(latencies))
    fps = 1000.0 / avg_lat if avg_lat > 0 else 0.0
```

```
=== Evaluation (test) ===
AP@0.5 (BBox): 0.958
AP@0.5 (Mask): 0.950
Latency (ms/img): 92.91   |   FPS: 10.76
```

References

- ㅇMohammed, S. Y. (2025). Architecture review: Two-stage and one-stage object detection. Engineering Applications of Artificial Intelligence. Elsevier. https://doi.org/10.1016/j.engappai.2025.108312
- Lu, X., Li, Q., Li, B., & Yan, J. (2020). MimicDet: Bridging the Gap Between One-Stage and Two-Stage Object Detection. European Conference on Computer Vision (ECCV). https://arxiv.org/abs/2009.11528
- Zhao, Z., Zheng, P., Xu, S., & Wu, X. (2019). Object Detection With Deep Learning: A Review. IEEE Transactions on Neural Networks and Learning Systems, 30(11), 3212–3232. https://doi.org/10.1109/TNNLS.2018.2876865
- iissaacc. (2023, June 20). RoI Pooling — 개념 정리. Velog. https://velog.io/@iissaacc/RoI-Pooling
- Rubber-Tree. (2023, August 5). [Object Detection] Faster R-CNN 구조 완벽 정리. Rubber-Tree Blog. https://rubber-tree.tistory.com/116
- Wsshin. (2023, July 12). [Computer Vision] Fast R-CNN & Faster R-CNN 정리. Wsshin Tech Blog. https://wsshin.tistory.com/9
- Ganghee Lee. (2023, May 30). Mask R-CNN 구조 및 동작 원리 (1). Ganghee’s AI Blog. https://ganghee-lee.tistory.com/35
- Ganghee Lee. (2023, June 2). Mask R-CNN 구조 및 동작 원리 (2). Ganghee’s AI Blog. https://ganghee-lee.tistory.com/36
- Lionhong, H. (2024, April 15). [AI CV] Faster R-CNN & RPN 이해하기. Developer Lionhong Blog. https://developer-lionhong.tistory.com/64
- Yun, S. (2024, March 4). Object Detection: One-Stage vs Two-Stage Detectors. Medium. https://sharkyun.medium.com/computer-vision-object-detection-one-stage-vs-two-stage-b05dbff88195
- Computer Vision Zone. (2024, June 1). Faster R-CNN Explained – Full Architecture Review [YouTube Video]. YouTube. https://www.youtube.com/watch?v=q-ewMdPUnBk
- Coding AI Lab. (2024, June 15). Mask R-CNN 논문 리뷰 및 코드 구현 [YouTube Video]. YouTube. https://www.youtube.com/watch?v=Jo32zrxr6l8
- Deep Learning Camp. (2024, May 10). R-CNN → Faster R-CNN → Mask R-CNN 발전 과정 [YouTube Video]. YouTube. https://www.youtube.com/watch?v=xdQwCeGCHgA

Q&A