Fast R-CNN

1. 기존 모델의 문제점

R-CNN

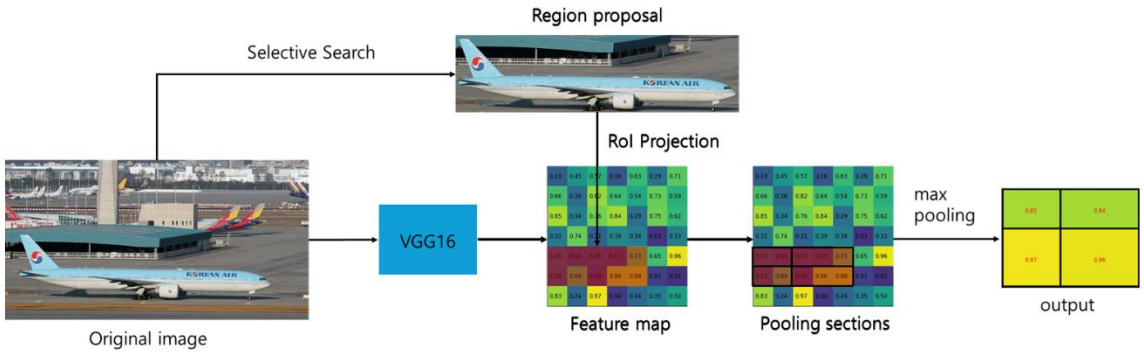
1. 속도 문제 : 입력 이미지에서 2,000개의 Region proposal을 추출한 뒤, 각 영역에 대해 모두 Conv연산을 하기에 많은 연산 비용 발생
2. 복잡한 학습 과정 : fine-tuning, SVM classification, bbox regression 으로 3번의 과정을 거쳐야 하기에 복잡하고 비효율적이다

SPPnet

1. 개선점 : 각 이미지에 대해 Conv 연산을 한 번만 계산하여 Feature map을 추출함으로써 속도 문제 개선
2. 한계 : 여전히 학습 과정이 R-CNN과 같이 여러 단계로 나누어져 있고, RoI 이전의 convolution layer를 업데이트할 수 없음

2. Fast R-CNN의 핵심 아이디어

RoI Pooling Layer



1) 원본 이미지에서 Feature map을 얻는다
2) 원본 이미지에 대하여 Selective search 알고리즘을 적용하여 region proposals을 얻는다

3) Feature map에서 각 region proposals에 RoI Projection을 통해서 해당하는 영역을 추출한다. Feature map은 VGG16 모델에 의해 높이와 너비가 축소되었지만 region propsals은 그렇지 않다. 따라서 Region proposals의 중심점 좌표, 높이, 너비와 original image에서 feature map으로 축소된 비율을 활용하여 feature map에 투영시킨다. 원본 RoI의 중심점 좌표, 높이, 너비를 (x, y, w, h) 라고 하고, 다운샘플링된 비율을 S라 하면 투영되는 것은 (x/S, y/S, w/S, h/S) 이다. (정수가 아닌 경우 반올림)

4) 추출한 RoI feature map(h * w)을 지정한 sub-window(h/H * w/W)의 크기에 맞게 grid로 나눠준다. 이때 H, W는 지정하는 파라미터로 output의 크기이다. VGG16의 경우 H=7, W=7을 사용한다.

5) 각 grid마다 max pooling을 수행하여 고정된 크기의 feature map을 얻는다.

이렇게 이루어진 RoI pooling layer는 어떤 문제를 해결했을까?

1) 기존 R-CNN은 각 이미지당 2천개의 RoI를 warping -> conv -> FC 과정을 거치므로 많은 연산 필요하였다. RoI pooling을 통해 이미지당 conv 1번만 필요하여 연산 속도 대폭 감소

2) 기존 R-CNN은 FC layer를 위해 warping을 하여 정보 왜곡이 발생하였다. 하지만 RoI pooling은 미리 지정한 크기의 sub-window에서 max pooling을 수행하다보니 region proposal의 크기가 서로 달라도 고정된 크기의 feature map을 얻을 수 있다.

3) 기존 R-CNN은 SVM과 bbox regressor가 CNN과 완전히 분리된 별도의 모델이므로 classification/regression Loss를 CNN으로 다시 흘려 보낼 수 없어 end-to-end 학습이 불가능하였다. 또한 SPPnet에서는 SPP 연산이 conv feature map 이후에 들어가고, 그 앞단(conv 계층들)의 gradient 계산이 복잡해서 실제로는 conv층 일부를 고정한 채, SPP 이후의 FC 계층만 fine-tuning 하였다.

RoI Pooling은 max pooling 기반이기에 역전파를 적용하는데 유리하였다.

Max Pooling에서 forward과정에서는 pooling window 내의 최댓값을 선택하고

Backward과정에서는 순전파 때 저장해둔 최댓값의 위치로만 gradient를 전달하고 나머지는 0으로 전달한다

Multi-task Loss

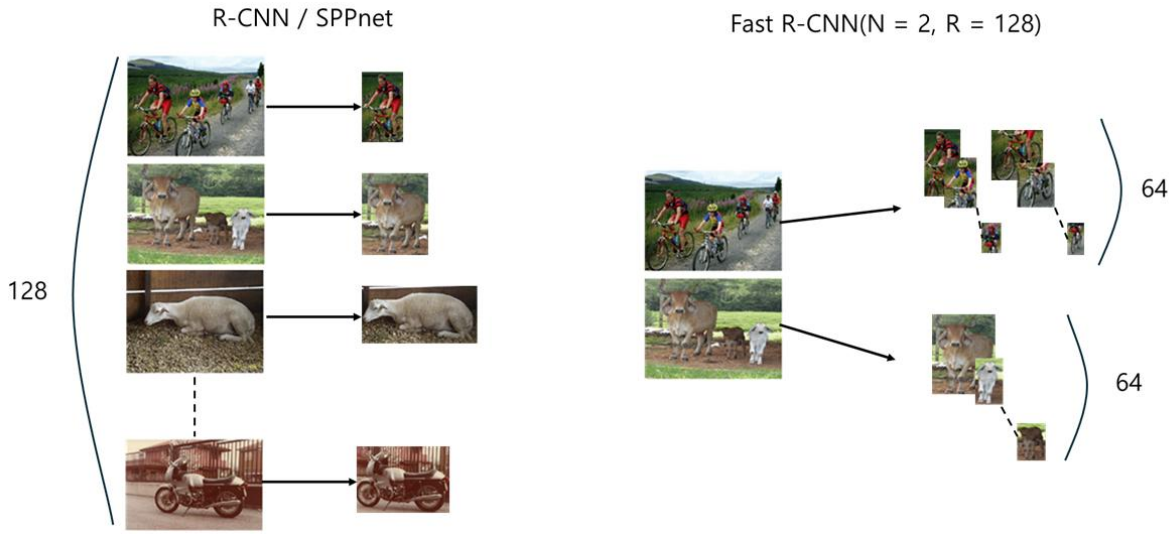
$$L(p,u,t_u,v)=L_{cls}(p,u)+\lambda[u\geq 1]L_{loc}(t_u,v)$$

1. $P = K+1$ 개의 class 예측값
2. u = ground truth class score 실제값(background=0)
3. t_u = bbox 예측 좌표값
4. v = ground truth bbox 좌표값
5. $L_{cls}(p,u) = -\log P_u$ = classification loss(Log loss)
6. $\lambda =$ balancing hyperparamter(논문에선 1)
7. $[u\geq 1] = u$ 가 1 이상일 때만(object) loss 를 계산
8. $$L_{\text{loc}}(t^u, v) = \sum_{i\in\{x,y,w,h\}} \text{smooth}_{L_1}(t_i^u - v_i)$$ (regression loss)
9. $$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

이렇게 이루어진 Multi-task Loss는 어떤 문제를 해결했을까?

Classification과 regression을 동시에 같은 네트워크의 파라미터를 업데이트하므로 feature가 두 작업에 모두 최적화된다. 또한 이전처럼 SVM과 bbox regressor를 쓰지 않고 하나의 네트워크로 모든 걸 처리하기에 end-to-end 학습이 가능하다. 또한 0.8~1.1% mAP를 상승시키는 효과도 있었다.

Hierarchical Sampling



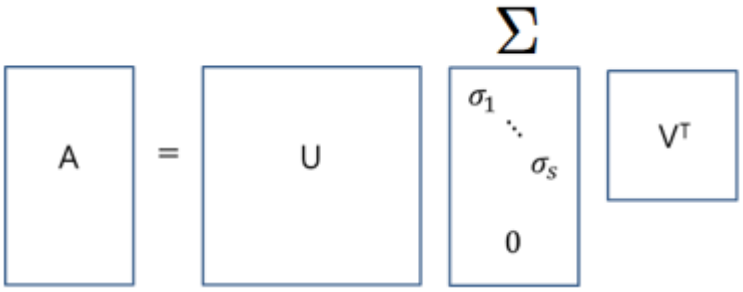
기존 R-CNN이나 SPPnet 방식에서는 미니배치에 128개의 RoI를 모두 다른 이미지에서 하나씩 샘플링하였다. 이는 메모리적으로 매우 비효율적이었다.

Fast R-CNN에서는 먼저 N개의 이미지를 무작위로 선택 후 각각의 이미지에서 R/N 개의 RoI를 샘플링한다. 예를 들어 N = 2, R = 128일 경우, 2장의 이미지에서 각 64개의 RoI를 추출하여 총 128개의 RoI를 구성한다. 이렇게 구성하면 하나의 미니배치(128개 RoI)는 단 2장의 이미지에서만 추출된다. 따라서 128개의 서로 다른 이미지를 CNN에 입력할 필요 없이, 단 2개의 이미지만 CNN에 입력하여 feature map을 계산하면 된다.

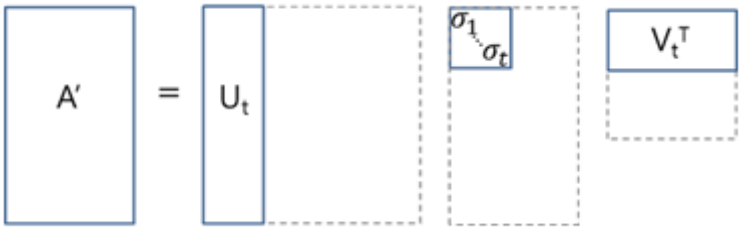
각 이미지의 region proposals 중 25%(=16장)은 ground truth와의 IoU 값이 0.5 이상인 sample을 추출하고, 나머지 75%에 대해서는 IoU 값이 0.1~0.5 사이의 sample을 추출한다. 전자의 경우 positive sample로, 위에서 정의한 multi-task loss의 u=1이며, 후자는 u=0으로 background로 처리한다.

미니배치 내의 64개 RoI는 동일한 이미지의 feature map을 공유하므로 순전파와 역전파 과정에서 발생하는 막대한 양의 Conv 연산을 공유할 수 있게 되면서 학습 속도가 매우 빨라졌다. (R-CNN의 64배, SPPnet의 3배)

Truncated SVD



Full SVD는 행렬 A를 m*m 크기인 U, m*n 크기인 Σ, n*n 크기인 V^T로 분해하는 것을 말한다.

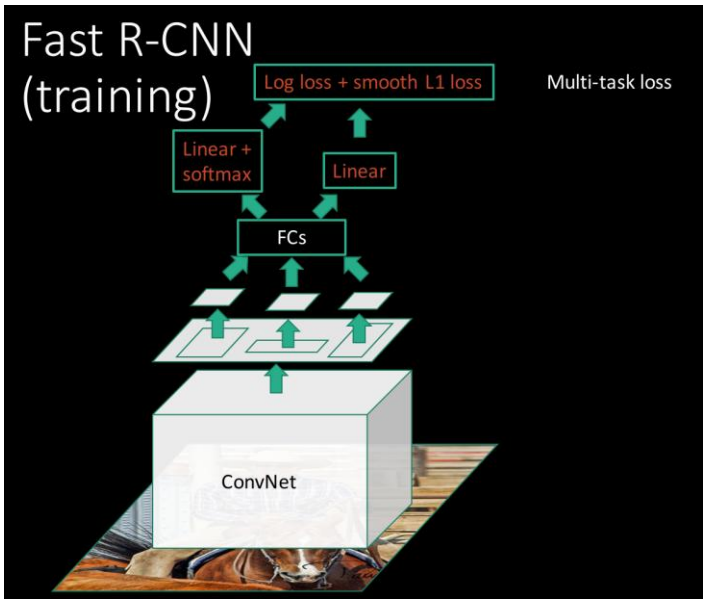


Truncated SVD는 Σ의 비대각 부분과 대각 원소 중 특이값이 0인 부분을 모두 제거하고, 제거된 Σ에 대응되는 U, V 원소도 함께 제거하여 차원을 줄인 형태이다. U_t 의 크기는 m*t이며, Σ_t 의 크기는 t*t, 그리고 V_t^T 의 크기는 t*n이 된다.

$$W \approx U\Sigma_t V^T$$

Fc layer의 가중치 행렬이 W(=u*v)라고 할 때, Truncated SVD를 통해 위와 같이 근사하는 것이 가능하다. 이를 통해 파라미터 수를 u*v 에서 t(u+v)로 감소시키는 것이 가능하다. Truncated SVD를 fc layer의 가중치 행렬 W에 적용하면, fc layer는 두 개의 fc layer로 나눠진다. 첫 번째 fc layer는 $\Sigma_t V^T$ 가중치 행렬, 두 번째 fc layer는 U 가중치 행렬이다. 이를 통해 네트워크를 효율적으로 압축하는 것이 가능하며, 논문의 저자는 Truncated SVD를 통해 mAP는 66.9%에서 66.6%로 감소한 반면 detection 시간은 30% 정도 감소되었다고 말한다.

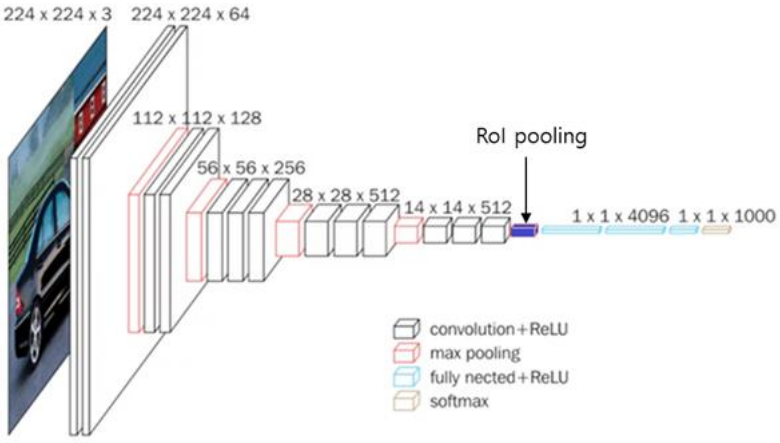
3. Fast R-CNN Architecture



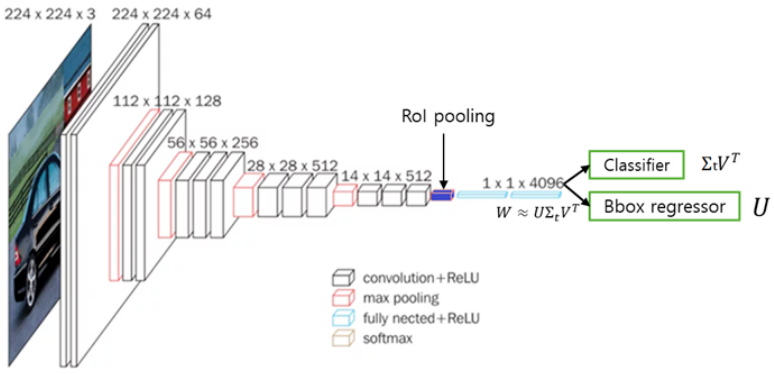
1. VGG16 fine tuning

Feature map의 추출을 위해 ImageNet을 학습한 VGG16 모델을 사용한다. 먼저 네트워크를 detection을 수행하기 위해 조절하는 과정이 필요하다.

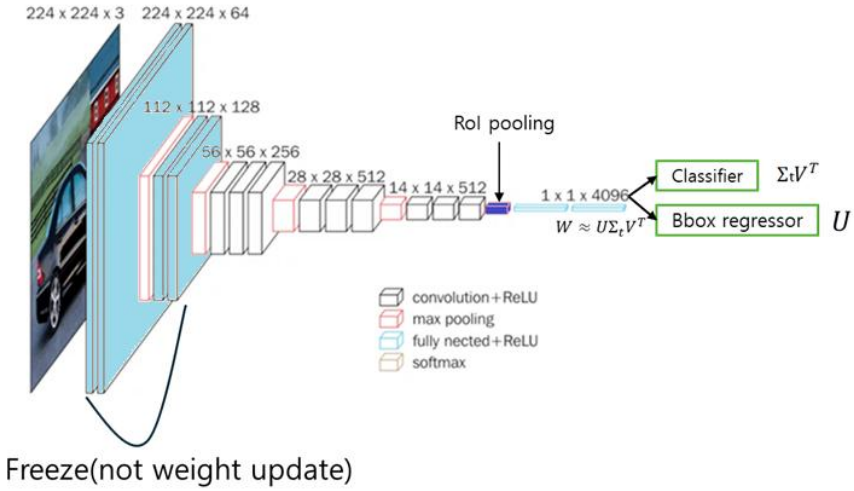
첫번째 과정은 VGG16 모델의 마지막 max pooling layer를 RoI pooling layer로 대체한다. 이때 RoI pooling을 통해 출력되는 feature map의 크기인 H,W는 fc layer를 위하여 7*7(VGG16이 학습한 크기)으로 설정한다.



두번째 과정은 마지막 fc layer를 2개의 fc layer로 대체한다. 첫번째 fc layer는 K개의 class와 배경을 포함한 K+1개의 output을 가지는 Classifier이며, 두 번째 layer는 각 class별로 bounding box의 좌표를 조정하므로 K개의 class에 대해 bounding box의 좌표와 크기를 가지는 bounding box regressor이다.



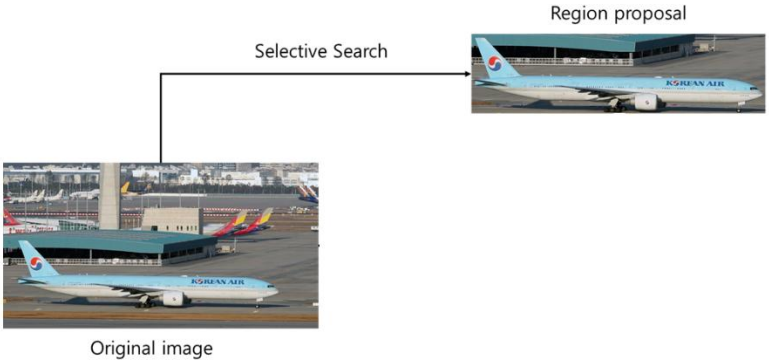
세번째 과정은 conv layer2_2까지의 가중치값은 고정시키고, 이후의 가중치가 업데이터되도록 fine tuning을 한다. 논문에서는 fc layer만 fine tuning했을 때보다 conv layer까지 포함시켰을 때 더 좋은 성능이었다고 말한다. 또한 성능은 conv2_1부터 fine tuning 했을 때가 더 좋았지만 conv3_1부터 fine tuning 했을 때보다 시간이 9.5시간에서 12.5시간으로 1.3배 증가하여 conv2_2까지 고정시킨 VGG를 사용하였다.



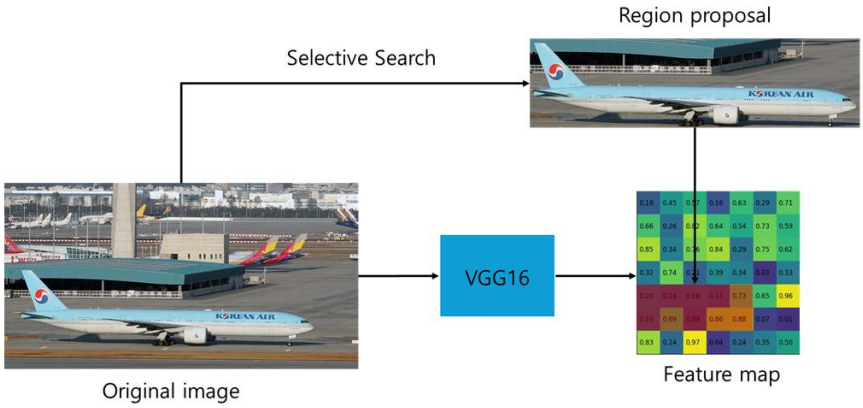
	layers that are fine-tuned in model **L**			SPPnet **L**
	≥ fc6	≥ conv3_1	≥ conv2_1	≥ fc6
VOC07 mAP	61.4	66.9	**67.2**	63.1
test rate (s/im)	**0.32**	**0.32**	**0.32**	2.3

2. Region proposal by Seletive search

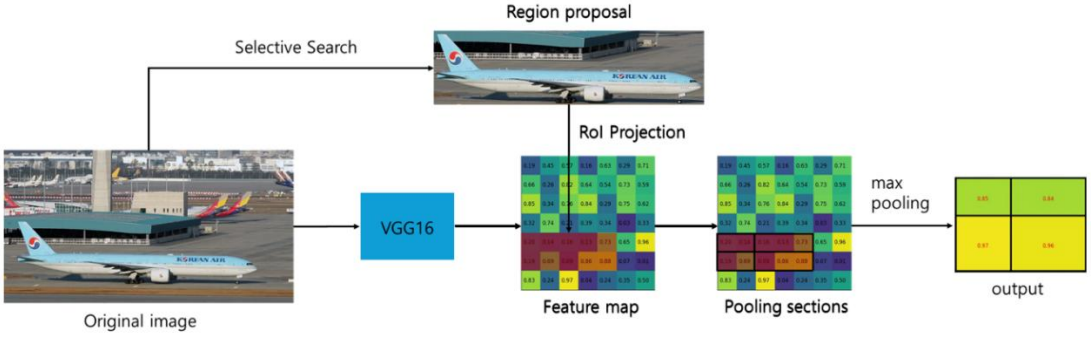
원본 이미지에 대하여 Seletive search 알고리즘을 적용하여 region proposal을 추출한다.



3) Feature extraction by VGG16

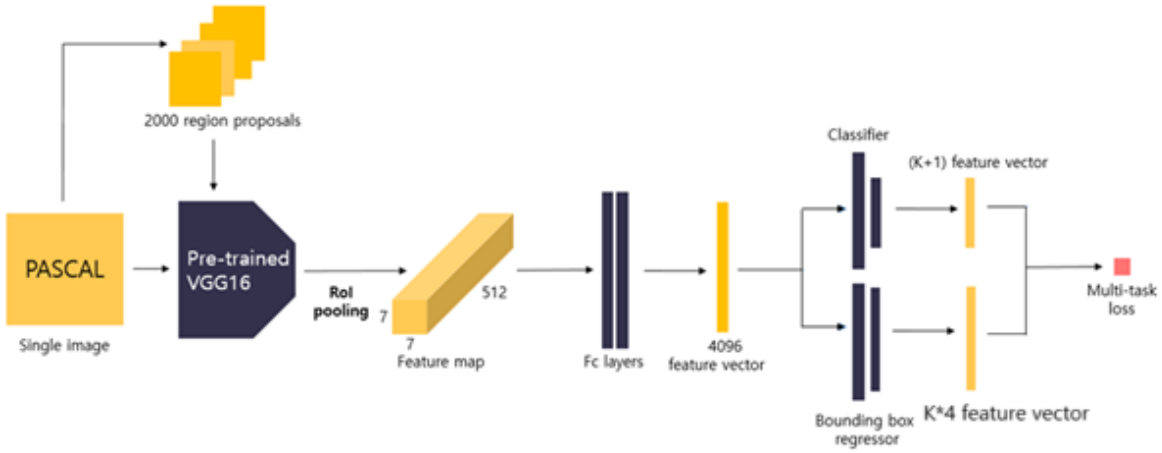


4) RoI pooling



5) Feature vector extraction by FC layers

Region proposal별로 7*7*512의 feature map을 faltten한 후 FC layer에 입력하여 4096 크기의 feature vector를 얻는다.



이후 K개의 class와 background를 포함한 K+1개에 대해 classification을 진행한다. Bounding box regressor는 K개의 class에 대해 중심좌표 x,y와 크기 h,w로 regression을 진행한다.

6) Train Classifier and Bounding box regressor by Multi-task loss

Multi-task loss를 사용하여 하나의 region proposal에 대한 Classifier와 Bounding box regressor의 loss를 반환한다. 이후 역전파를 통해 두 모델을 한번에 학습시킨다.

7) Non maximum suppression

R-CNN과 같이 추론 시에는 최적의 bounding box만 출력하기 위해 Non maximum suppression을 사용한다. 이 알고리즘에 대한 설명은 이전 pdf를 참고하면 된다.