

# Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

<https://arxiv.org/abs/1506.01497>

## 목차

1. [Introduction](#)
2. [배경지식\(Related Work\)](#).
3. [Faster R-CNN](#)
4. [Experiments](#)
5. [Conclusion](#)

---

## 1. Introduction

본 논문이 나오기 이전까지의 **object detection** 분야에서는 가장 높은 성능을 달성한 SOTA 모델로 SPP-Net과 Fast R-CNN 등이 있었다. 두 모델은 그 이전에 제안되었던 R-CNN보다 상대적으로 속도가 더욱 빨랐지만, 여전히 네트워크 바깥에서 CPU 방식으로 돌아가는 **region proposal** 단계에서 많은 시간이 소요된다는 단점이 존재했다.

- 여기서 **Object detection**이란 영상 안에 존재하는 여러 object의 위치 찾기 + classification까지 해야 하는 문제이다.



출처: [1]

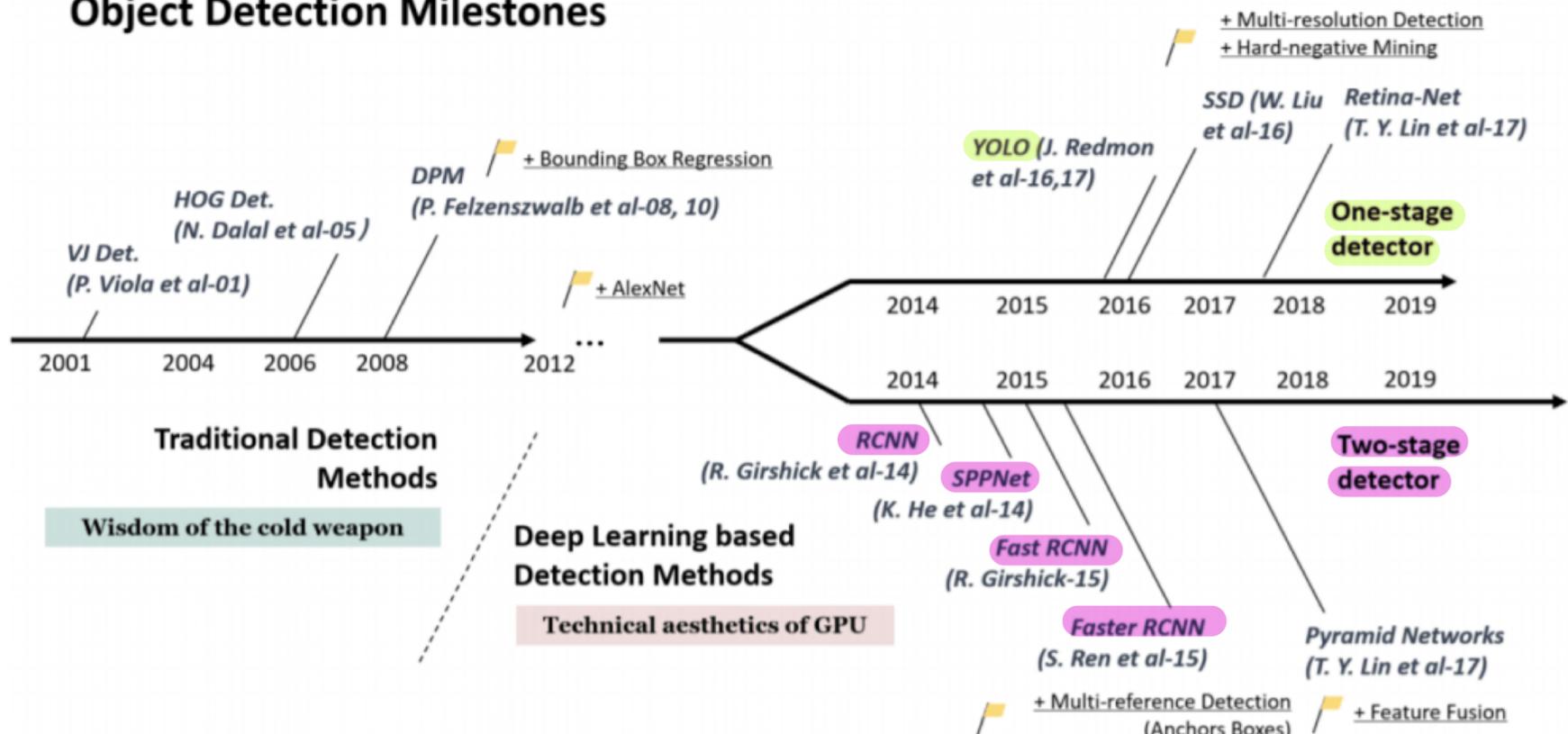
- Region proposal: Input image에서 object가 있을법한 영역을 제안해주는 것이다.

---

## 2. 배경지식(Related Work)

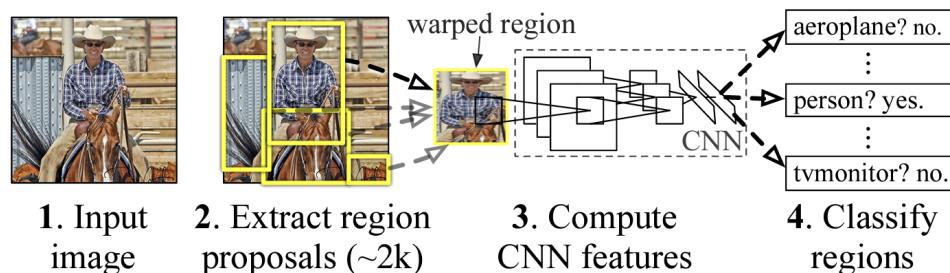
Faster R-CNN은 R-CNN, SPP-Net, 그리고 Fast R-CNN에 이어서 나온 논문이다. 따라서 Faster R-CNN 이전의 논문들에 대해 먼저 알아보도록 하자.

## Object Detection Milestones



### 2-1. R-CNN

R-CNN: Regions with CNN features



#### R-CNN 모델 작동 방식

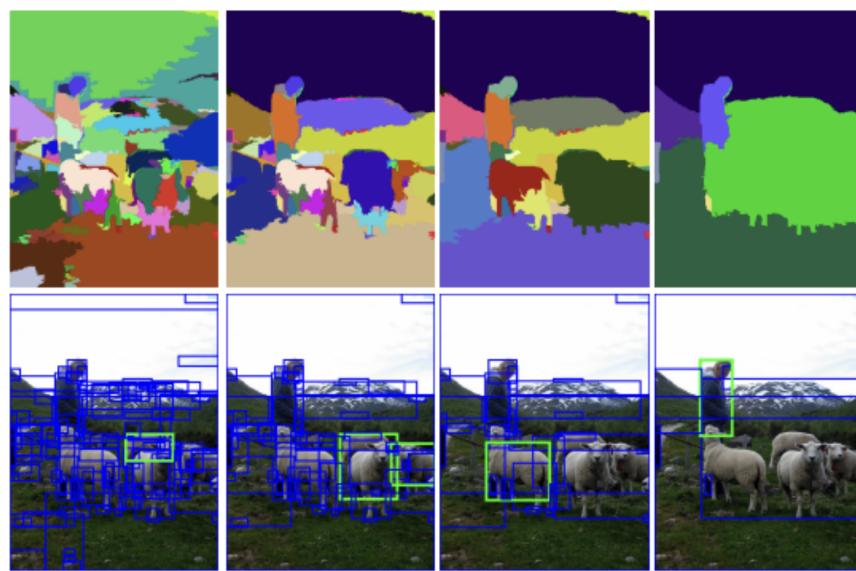
- (1) Input Image를 입력으로 받는다
- (2) 이미지에서 무작위로 2,000개의 region proposals(Roi)를 추출한다.(Selective search 사용)
- (3) 추출된 Roi의 크기가 모두 다르므로, 특정 크기(227x227)로 리사이즈(warp)한다.
- (3) CNN(pre-trained된 AlexNet 사용)을 통과시켜 각 proposals의 feature를 뽑는다.
- (4) SVM으로 class를 분류하고, bounding box regression으로 bounding box 좌표를 조정한다.

#### 단점

- Region proposal에서 **Selective search** 알고리즘을 사용하여 **오래 걸린다**.
- 2000개의 각각의 패치에서 CNN으로 feature를 추출해야하기 때문에 **오래 걸린다**. (13s per image in GPU)
- 마지막단에서 **SVM** 분류기를 학습시키기 때문에, 분류 결과로 **CNN을 업데이트시키지 못한다**.

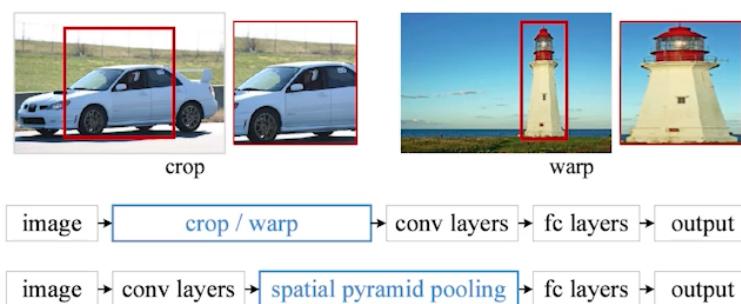
#### Selective Search

특정 이미지가 주어졌을 때, 인접한 영역끼리 유사성(색상, 위치 등의 특성 사용)을 측정해 큰 영역으로 차례대로 통합해 나가는 알고리즘이다[8]. 이렇게 만들어진 segmentation 결과를 기준으로 region proposal을 하는 것이 바로 selective search이다. 룰 베이스의 알고리즘이기 때문에, end-to-end 모델 학습이 어려워 처리시간이 길다는 단점이 있다. (Region Proposal이 RPN과 같이 역전파를 통해 학습의 일부가 되면서, 이 알고리즘은 사용되지 않는다.)



출처: [2]

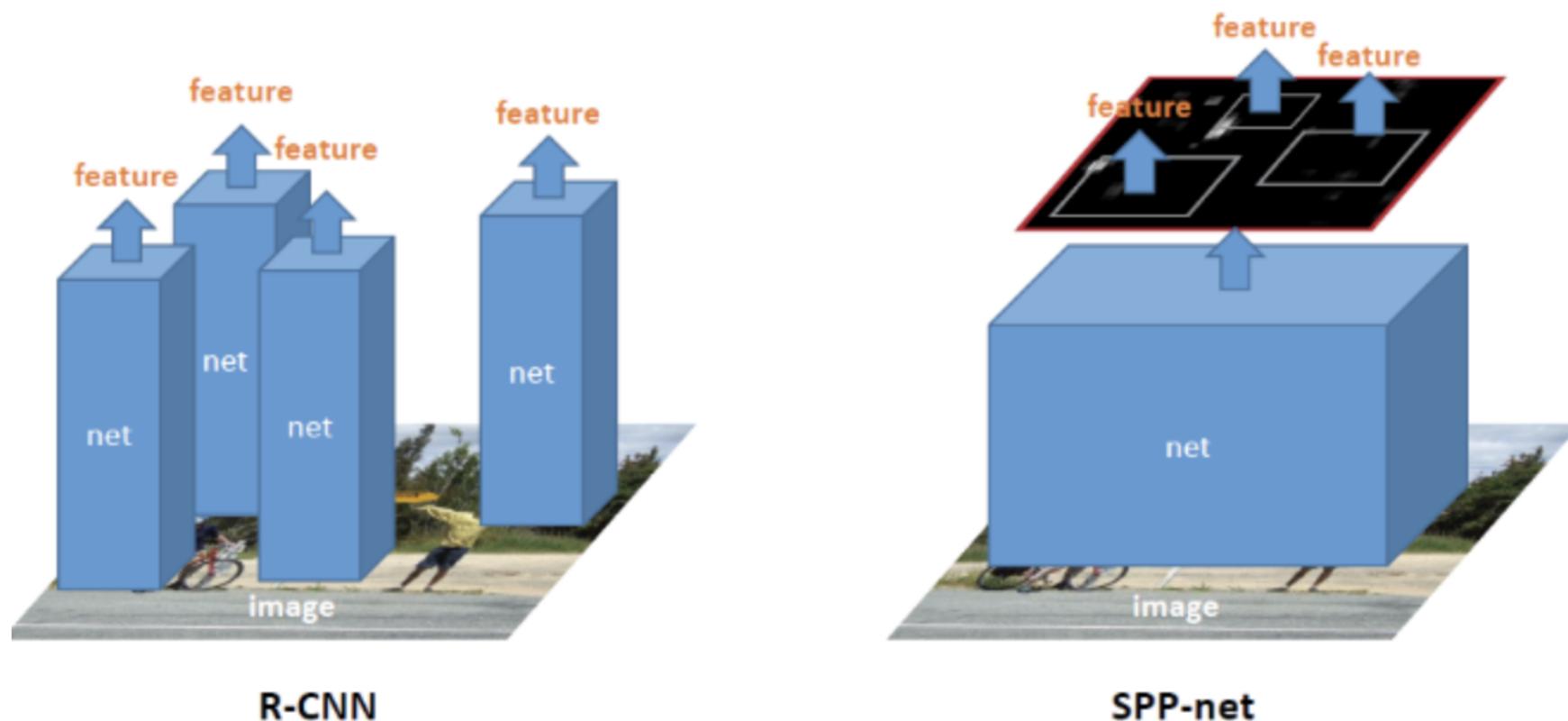
## 2-2. SPP-Net(Spatial Pyramid Pooling Network)



### SPP-Net 모델 작동 방식

- (1) 인풋 이미지에서 ROI를 여러개 추출한다.(Selective search 방식)
- (2) 인풋 이미지 전체(ROI가 아님)를 CNN을 통과시켜 feature map을 생성한다.
- (3) (1)에서 추출한 bounding box 각각의 영역에 대응하는 ROI에 대해(크기가 모두 다름), **SPP(Spatial pyramid pooling)**을 통해 고정된 크기의 feature를 추출하고 fully connected layer를 통과시킨다.
- (4) SVMClassifier로 앞서 추출된 벡터의 class를 분류하고, bounding box regression으로 bounding box 좌표를 조정한다.

### R-CNN vs SPP-Net



R-CNN에서는 각각의 ROI이 모두 CNN을 거치지만, SPP-Net은 인풋 이미지가 한번의 CNN을 거친 뒤 SPP로 feature의 크기를 모두 같게 만들어 학습한다.

R-CNN에서 모든 ROI를 CNN에 통과시키는 부분을 **개선**하였지만, 여전히 아래와 같은 단점이 존재한다.

## 단점

- 여전히 **Selective Search**를 사용한다.
- 여전히 뒷단에서 **SVM**을 사용한다. (end-to-end 방식이 아님)

## SPP(Spatial pyramid pooling)

각기 크기가 다른 convolution feature map을 입력으로 받아, 고정된 크기의 feature를 뽑아낸다.(크기가 다른 여러 feature map을 동일한 크기로 resize해준다고 이해하면 된다.)

4x4, 2x2, 1x1로 미리 정해진 영역에 따라 max-pool을 진행하고, 이렇게 pooling으로 추출된 값들을 하나의 fixed-length representation으로 이어붙이면 21크기의 벡터가 추출되는 것이다. SPP-Net의 CNN에서 마지막 레이어인 conv<sub>5</sub>의 채널 수가 256이므로, 여기선 input feature map의 크기에 상관없이 21x256의 fixed-length feature가 추출된다.

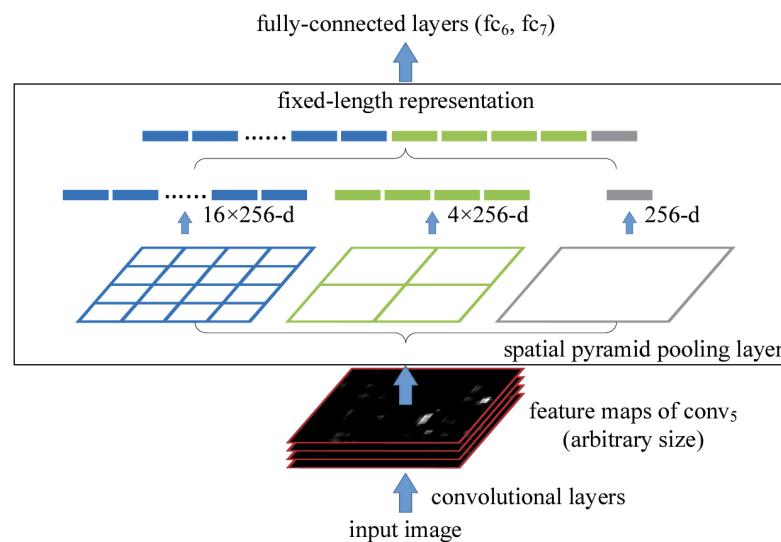
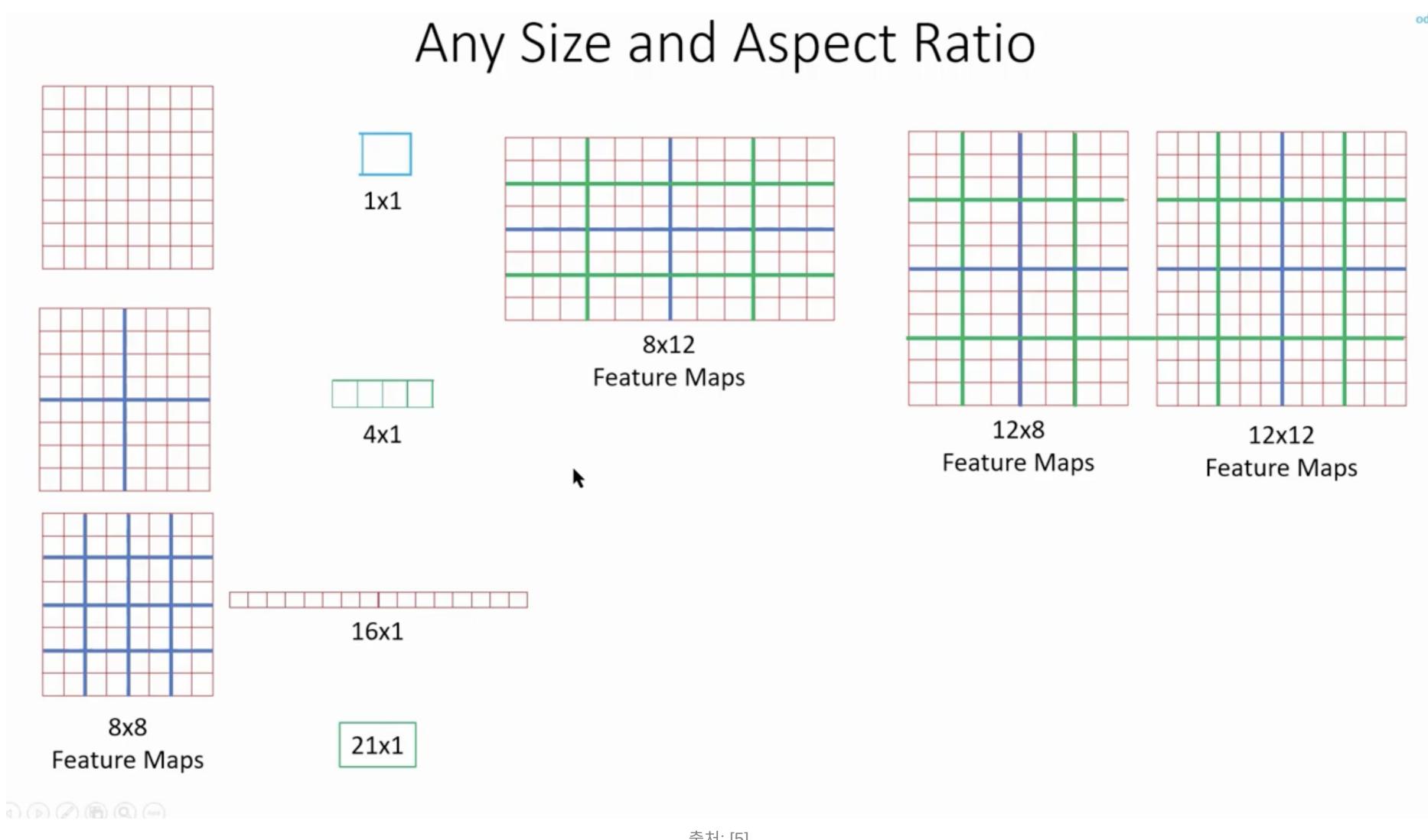


Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the conv<sub>5</sub> layer, and conv<sub>5</sub> is the last convolutional layer.

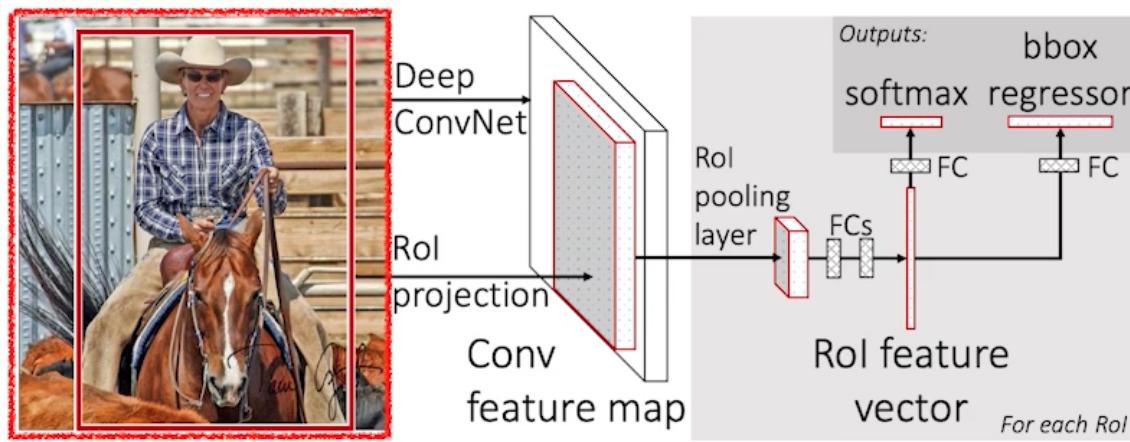
아래의 그림을 참고하면 조금 더 쉽게 이해할 수 있을 것이다. 어떤 크기의 feature map이 들어와도, 지정한 영역대로 분할하여 max-pooling을 진행한다.



구현 방식은 다음 공식 깃허브 링크를 참고해보자. [링크](#)

출처: [5]

## 2-3. Fast R-CNN



- 기본 컨셉은 SPPNet과 거의 동일하다. 다만 뒷단에서 neural network 구조를 통해 bbox regression과 classification을 했다는 점에서 차이가 있다.(SVM → neural network)

### Fast R-CNN 작동 방식

- (1) 인풋 이미지에서 bounding box를 여러개 추출한다.(Selective search 방식)
- (2) 인풋 이미지를 CNN을 통과시켜 feature map을 생성한다.
- (3) (1)에서 추출한 bounding box 각각의 영역에 대응하는 ROI에 대해, **ROI pooling**을 통해 fixed length feature를 추출하고 fully connected layer를 통과시킨다.
- (4) bounding box regressor과 class에 대한 output을 추출한다.

### 단점

- 여전히 **Selective search** 방식을 사용한다.

### ROI pooling

SPP-Net의 SPP의 동작 방식과 같다. 어떤 ROI가 들어와도, max-pooling의 연산을 통해 동일한 크기의 output을 만들어내는 것이다.

ROI max pooling works by dividing the  $h \times w$  ROI window into an  $H \times W$  grid of sub-windows of approximate size  $h/H \times w/W$  and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. The ROI layer is simply the special-case of the spatial pyramid pooling layer used in SPPnets [11] in which there is only one pyramid level. We use the pooling sub-window calculation given in [11].

Fast-RCNN 논문에서 언급된 ROI pooling 설명 부분

## R-CNN vs SPP-Net vs Fast R-CNN

### 모델별 한 이미지에 대한 test time 비교(단위: 초)

| Model     | Region proposal 과정 포함 | Region proposal 과정 미포함 |
|-----------|-----------------------|------------------------|
| R-CNN     | 49                    | 47                     |
| SPP-Net   | 4.3                   | 2.3                    |
| Fast-RCNN | 2.3                   | 0.32                   |

- 2초가 region proposal에 사용된다. 이를 개선하기 위해 제안된 것이 Faster-RCNN의 RPN인 것이다.  
→ Region proposal을 위해 네트워크 바깥에서 동작하던 selective search 알고리즘을 버리고, 네트워크 안에서 학습이 가능한 알고리즘으로 바꿔줌으로써 GPU를 사용하여 end-to-end 학습이 가능하도록 하였다.

## 3. Faster R-CNN

Faster R-CNN의 핵심은 기존 모델 학습/추론에서 대부분의 시간을 잡아먹는 **Selective Search**을 버리고, **region proposal** 역시 네트워크 안에서 학습을 시키자는 것이다. 따라서 region proposal을 포함한 전체 네트워크를 end-to-end로 학습이 가능하다.

$$\text{Faster R-CNN} = \text{Selective Search} + (\text{Fast R-CNN} + \text{RPN})$$

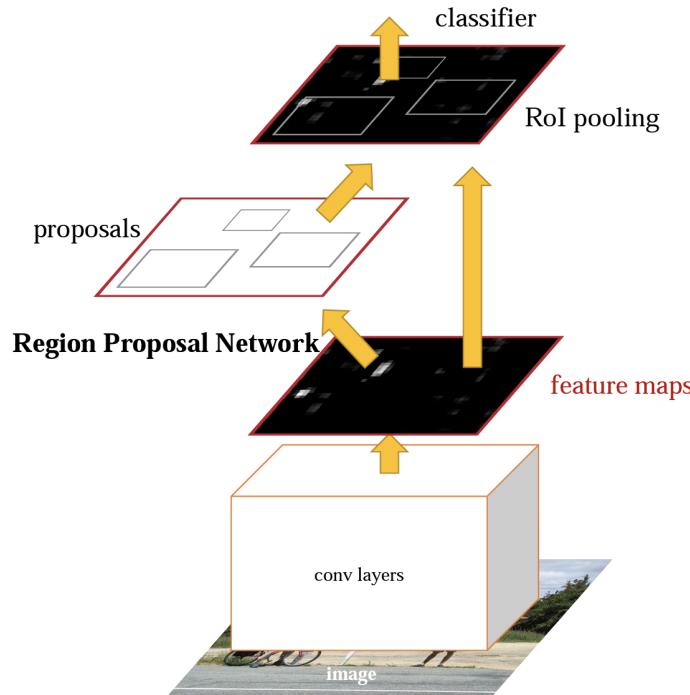
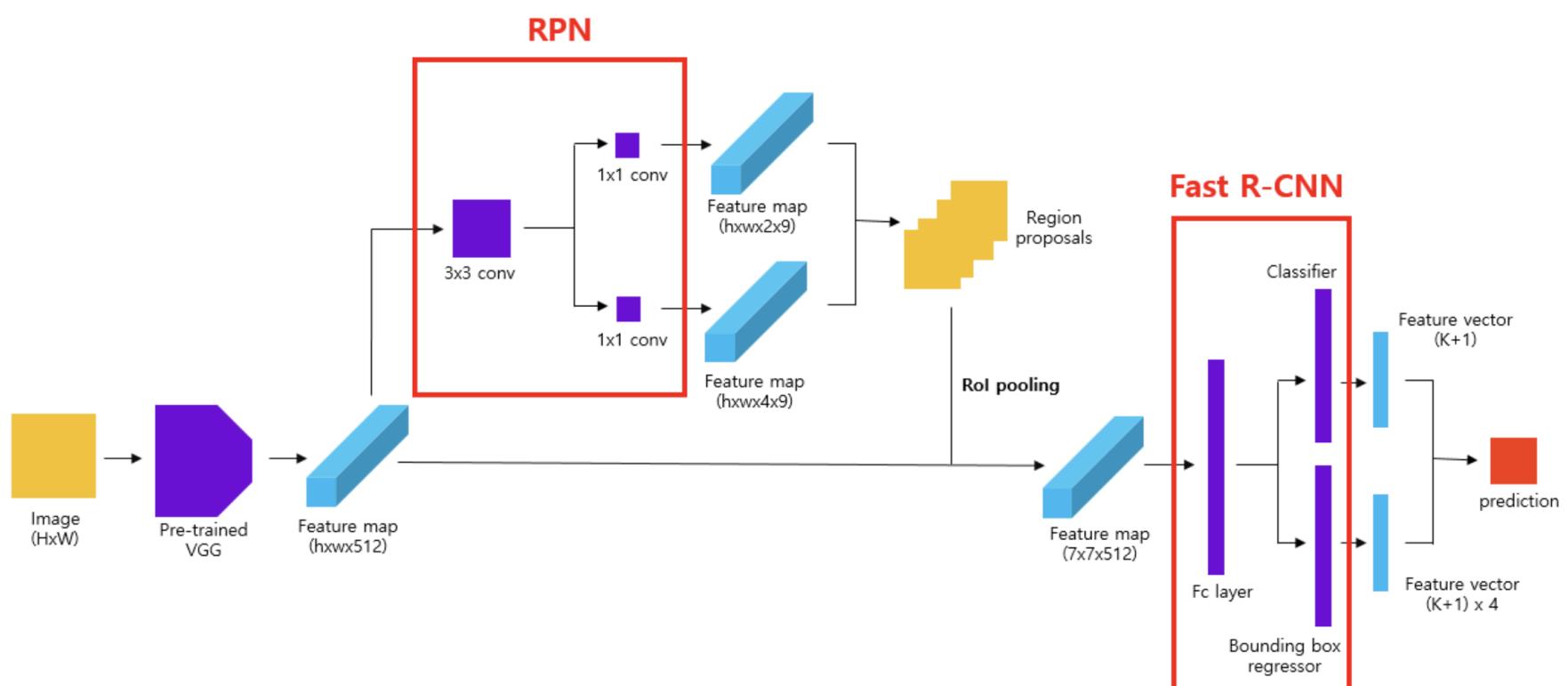


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the ‘attention’ of this unified network.

네트워크 내부에 region proposal이 학습된다.

### Faster R-CNN의 작동 방식



참고: [9]

(1) 인풋 이미지가 conv layers를 통하여 feature maps가 추출된다.

(2) RPN을 통해 feature maps에서 물체가 있을 법한 위치를 찾는다. 즉, RPN을 통해 region proposals를 얻는 것이다.

- RPN의 output: **Bounding box 좌표**와 해당 영역에 물체가 있을 확률(**Objectness Score**)
- Bounding box 좌표:  $[x, y, w, h]$
- Objectness Score: [object일 확률, no object일 확률]

(3) RPN에서 추출된 region proposals와 (1)의 feature maps을 통해 RoI Pooling을 수행하여 고정된 크기의 feature map을 얻는다.

(4) Classifier(기존의 Fast R-CNN 구조와 똑같음)은 (3)에서 얻은 feature map을 통해 classification과 regression을 진행한다.

- Classifier가 중심적으로 봐야할 영역을 알려준다는 점에서 논문에서는 이러한 RPN이 **attention** 역할을 한다고 말한다.

### 3.1 Region Proposal Networks

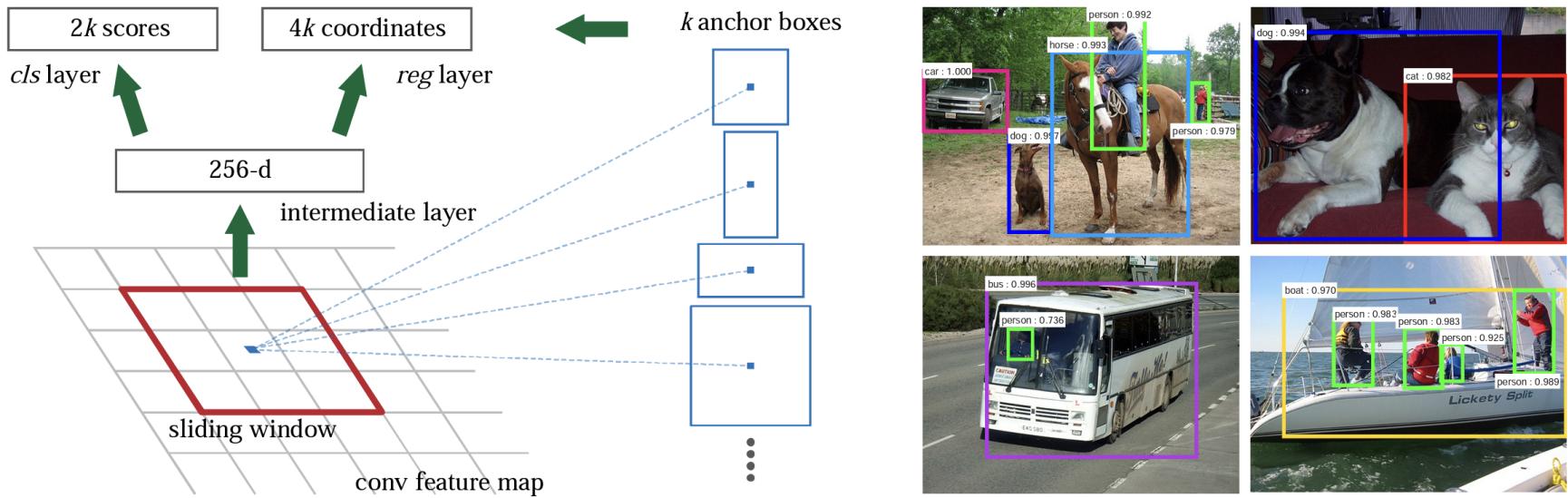


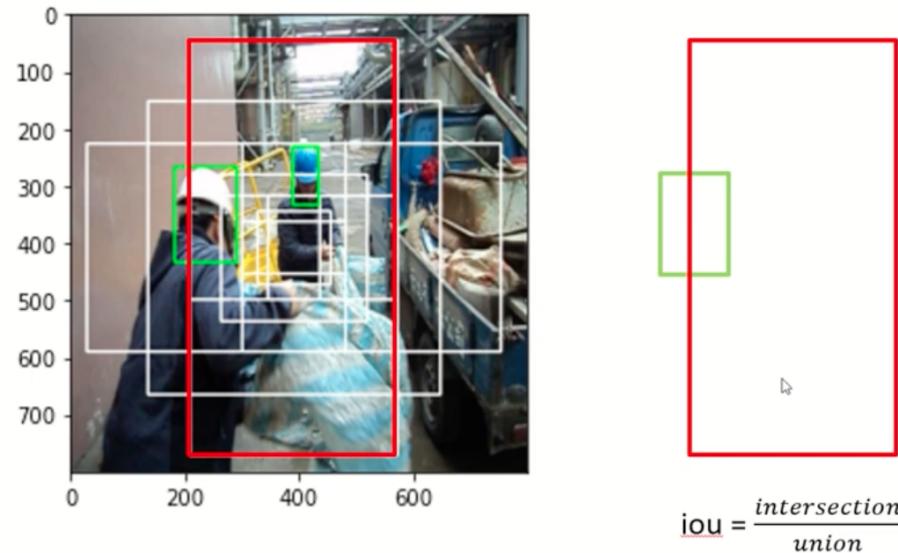
Figure 3: **Left:** Region Proposal Network (RPN). **Right:** Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.

RPN은 feature map을 입력으로 받아 물체가 있을 법한 위치를 예측한다.

- 이때 feature map의 크기는  $(W \times H \times C)$ 이므로,  $W \times H$  개의 grid가 존재한다.(이때,  $C = 256$  or  $512$ )
- $k$  개의 anchor boxes를 사용한다.(논문에서는  $k = 9$ )
  - 각각의 anchor box는 서로 다른 비율(3)과 크기(3)를 갖고 있다. $(3 \times 3 = 9)$
- 따라서 전체 앵커 수는  $WHk$  개인 것이다.
- Feature map의 왼쪽 위부터 sliding window 방식으로 이동하면서, 각각의 영역에서 256차원으로 mapping을 수행하는 동시에 9개의 anchor를 생성한다.
- 각각의 영역에서 mapping된 feature를 통해 output으로 2k개의 scores와 4k개의 bounding box 좌표가 나오게 된다.

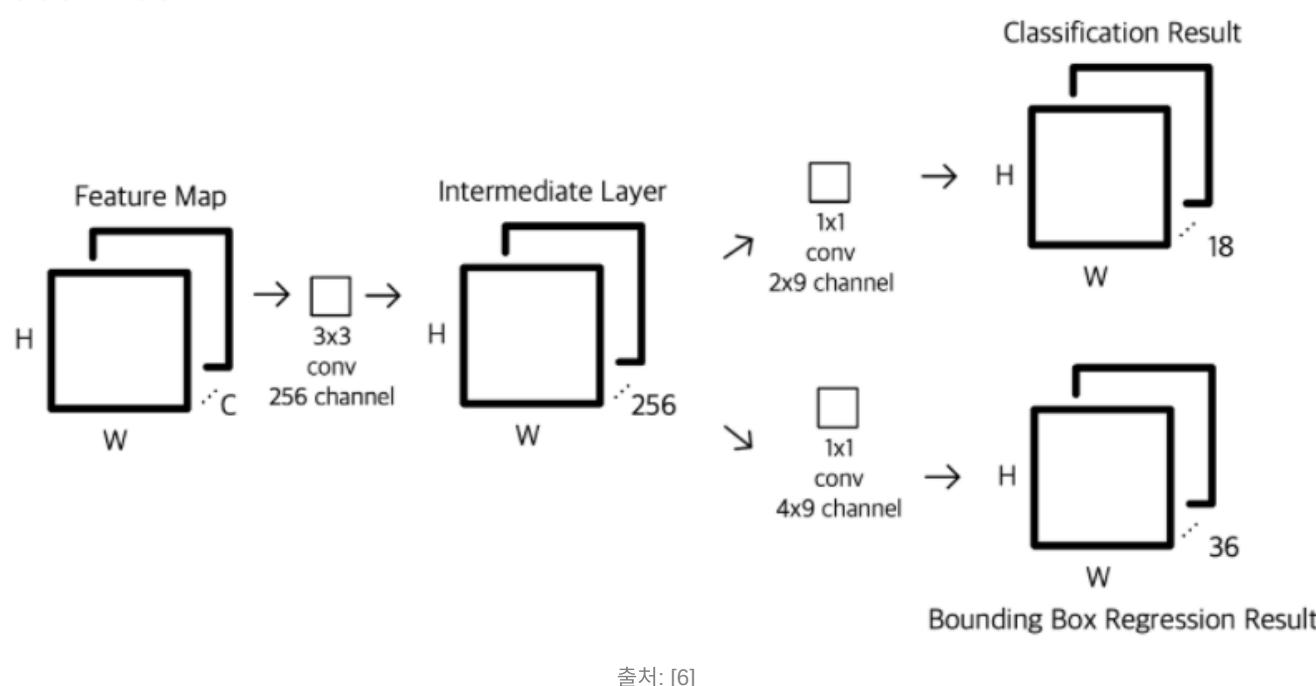
#### Anchor의 positive/negative 할당 조건

- Ground-truth box와 IoU가 0.7보다 큰 경우, positive anchor로( $p_i^* = 1$ ), IoU가 0.3 미만일 경우엔 negative anchor( $p_i^* = 0$ )로 분류된다.
- 만약  $0.3 \leq \text{IoU} \leq 0.7$  이라면, 학습에서 무시된다.(Anchors that are neither positive nor negative do not contribute to the training objective.)
- 만약 하나의 single ground-truth box에서 위 조건을 만족하는 positive anchor가 없다면, 그 중 가장 높은 IoU를 가진 anchor를 채택한다.

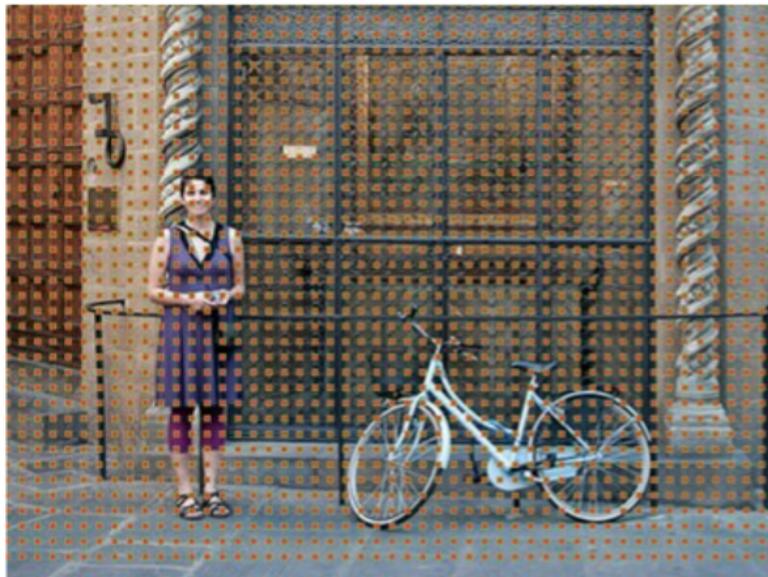


출처: [11]

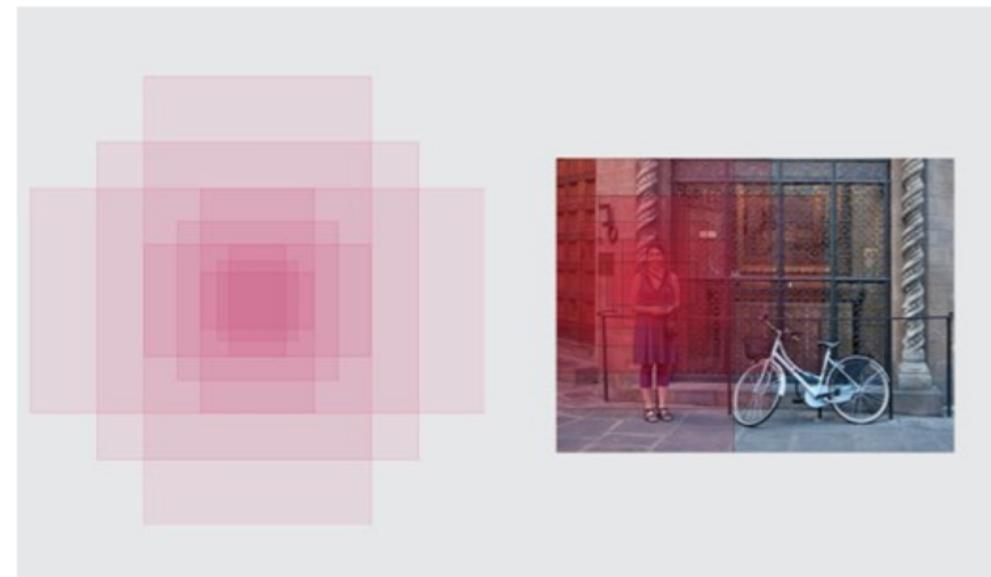
#### 조금 더 직관적으로 표현한 그림



### Anchor 예시



### Anchor box 적용



이렇게 만들어진 anchor 값들을 모두 사용하는 것은 아니고, 한 이미지당 랜덤하게 256개를 sampling한다. positive : negative 비율을 1:1 비율로 만들어 RPN에 넣어주면, 해당 anchor에 object가 있는지 없는지 이진 분류를 하는 classifier를 학습하고, 앵커 내 물체 위치를 찾는 bbox regression가 이루어진다. 만약 positive anchor의 개수가 128개보다 낮을 경우, 빈 자리는 negative anchor sample로 채운다[10].

### 3.2 Loss Function(RPN)

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

$i$ : mini-batch 내의 anchor의 index

$p_i$ : 예측된 확률(object vs not object)

$p_i^*$ : positive anchor:1, negative anchor:0

$N_{cls}$ ,  $N_{reg}$ ,  $\lambda$ : Normalize term(논문에서는 256, 2400, 10으로 설정하였다.)

$t_i$ : 예측된 bounding box

$t_i^*$ : ground-truth box

$L_{cls}$ : Classification loss(log loss over two classes - object vs. not object)

$p_i^* L_{reg}$ : Regression loss(smooth  $L_1$  을 사용하였다고 한다.),  $p_i^*$  가 붙은 이유는 positive anchor

( $p_i^* = 1$ )에 대해서만 활성화된다는 의미이다.

### 3.3 Training RPN

학습에는 추출된 전체 anchor를 사용하지 않고, positive anchor와 negative anchor의 비율이 최대 1:10이 되게끔 256개를 sampling하여 학습을 진행했다.

|           |                           |
|-----------|---------------------------|
| Loss      | cls + reg Multi-task loss |
| Optimizer | SGD(momentum=0.9)         |

|                 |   |
|-----------------|---|
| Learning rate   | 0.001로 첫 75%의 미니배치 학습, 이후 25% 미니배치 학습시 0.0001로 감소 |
| Weight Decay    | 0.0005  |
| CNN backbone 모델 | ZF net, VGG net를 각각 사용하여 실험 진행                    |

## 4. Experiments

논문의 저자들은 여러가지 실험들을 진행했다. Faster R-CNN의 backbone 모델, anchor의 개수를 바꿔가며 성능 비교를 하는 등 많은 실험들이 Experiments 파트에 실려 있다. 다음은 RPN을 사용했을 때의 성능과 시간 측면에 대한 결과이다. 다른 실험 결과들은 본 논문을 참고하자.

Table 4: Detection results on **PASCAL VOC 2012 test set**. The detector is Fast R-CNN and VGG-16. Training data: “07”: VOC 2007 trainval, “07++12”: union set of VOC 2007 trainval+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. <sup>†</sup>: <http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html>. <sup>‡</sup>: <http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html>. <sup>§</sup>: <http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html>.

| method                       | # proposals | data        | mAP (%)     |
|------------------------------|-------------|-------------|-------------|
| SS                           | 2000        | 12          | 65.7        |
| SS                           | 2000        | 07++12      | 68.4        |
| RPN+VGG, shared <sup>†</sup> | 300         | 12          | 67.0        |
| RPN+VGG, shared <sup>‡</sup> | 300         | 07++12      | <b>70.4</b> |
| RPN+VGG, shared <sup>§</sup> | 300         | COCO+07++12 | <b>75.9</b> |

논문에서 제안한 네트워크 구조가 성능이 제일 잘 나왔다.

Table 5: **Timing** (ms) on a K40 GPU, except SS proposal is evaluated in a CPU. “Region-wise” includes NMS, pooling, fully-connected, and softmax layers. See our released code for the profiling of running time.

| model | system           | conv | proposal  | region-wise | total      | rate          |
|-------|------------------|------|-----------|-------------|------------|---------------|
| VGG   | SS + Fast R-CNN  | 146  | 1510      | 174         | 1830       | 0.5 fps       |
| VGG   | RPN + Fast R-CNN | 141  | <b>10</b> | 47          | <b>198</b> | <b>5 fps</b>  |
| ZF    | RPN + Fast R-CNN | 31   | <b>3</b>  | 25          | <b>59</b>  | <b>17 fps</b> |

RPN을 사용했을 때 더 빨랐다.

## Conclusion

본 논문에서는, 네트워크와 별개로 특정 영역을 CPU 단에서 추출했던 방식에서 벗어나 GPU에서 end-to-end로 학습이 가능한 **Region Proposal Network(RPN)**을 제안하여 region proposal에 필요한 시간을 획기적으로 단축하는 것은 물론, 성능 역시 향상시켰다. RPN은 어떠한 위치(bounding box)에 물체가 존재하는가(0/1)를 학습하는 모듈이다. 여기서 물체가 존재한다면(=1), 이 box 내 Object가 어떤 class인지 분류하는 것은 기존의 Fast R-CNN의 classifier 구조를 활용한다. 따라서 Faster R-CNN은 기존의 Fast R-CNN 모델 구조에 Region Propasal Network 모듈을 합친 형태인 것이다.

$$\text{Faster R-CNN} = \text{Fast R-CNN} + \text{RPN}$$

## References

- [1] <https://medium.com/analytics-vidhya/beginners-guide-to-object-detection-algorithms-6620fb31c375> (Object detection)
- [2] <https://velog.io/@jaehyeong/R-CNNRegions-with-CNN-features-논문-리뷰> (R-CNN)
- [3] <https://go-hard.tistory.com/33> (Selective Search)
- [4] [https://blog.naver.com/PostView.nhn?blogId=jaeyoon\\_95&logNo=221785990158&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=1&from=postView](https://blog.naver.com/PostView.nhn?blogId=jaeyoon_95&logNo=221785990158&categoryNo=0&parentCategoryNo=0&viewDate=&currentPage=1&postListTopCurrentPage=1&from=postView) (SPP-Net)
- [5] [https://www.youtube.com/watch?v=2IoHC\\_fhrFU](https://www.youtube.com/watch?v=2IoHC_fhrFU) (SPP-Net)
- [6] <https://yeomko.tistory.com/17> (Faster R-CNN)
- [7] <https://www.youtube.com/watch?v=kcpAGIgBGRs> (Faster R-CNN)
- [8] <https://www.youtube.com/watch?v=jqNCdjOB15s> (Faster R-CNN)

- [9] <https://herbwood.tistory.com/10> (Faster R-CNN) 
- [10] <https://chacha95.github.io/2020-02-14-Object-Detection2/> (2-Stage detector)
- [11] [https://www.youtube.com/watch?v=4yOcsWg\\_7g8](https://www.youtube.com/watch?v=4yOcsWg_7g8) (Faster R-CNN)