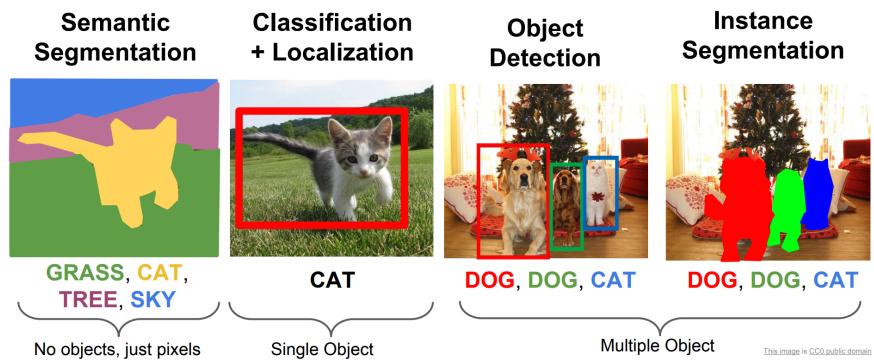


# Lecture 11) Detection and Segmentation

## Computer Vision Tasks

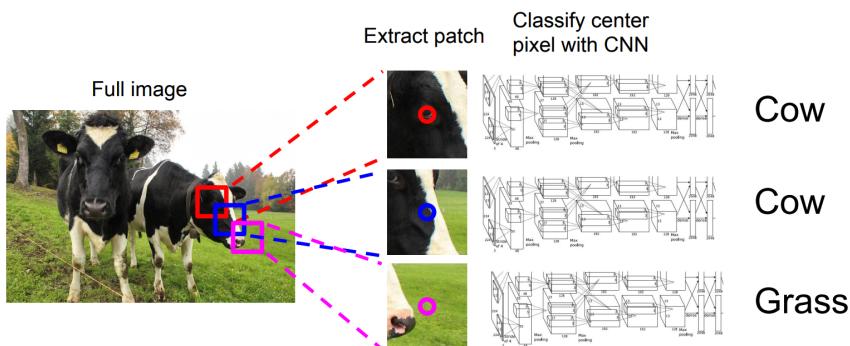
- Image Classification, Segmentation, Localization, Detection, ...
- Image Classification Review
  - Image를 CNN input으로 넣어서 feature map을 뽑고, 마지막에 Fully-Connected Layer를 통과시켜, 어떤 class(label)로 분류할지에 대한 score를 계산하게 된다.
  - Classification = CNN(특징뽑기) + FC Layer(분류하기)
- Other CV Tasks



### Semantic Segmentation

- 각 픽셀마다 카테고리를 분류한다.
- 픽셀 독립적으로 분류하므로 개체가 몇 개인지, 이 픽셀이 어떤 개체에 속하는지는 고려하지 않는다.

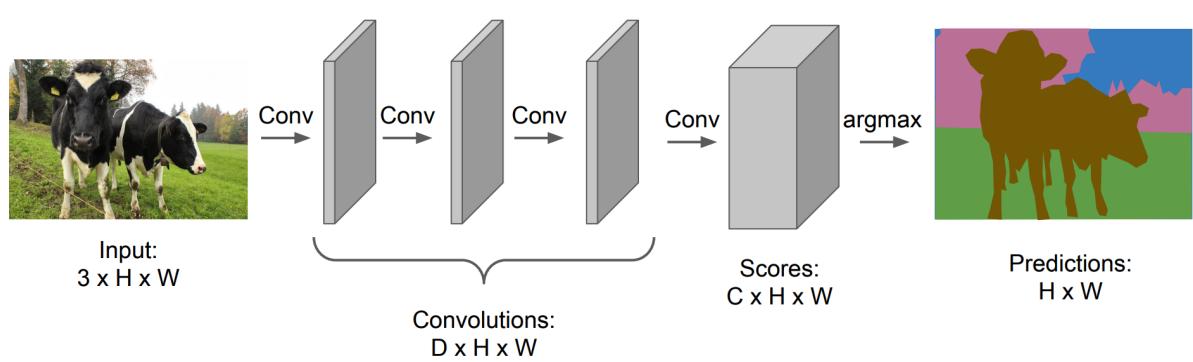
### 구현방법1) Sliding Window



#### • 방법

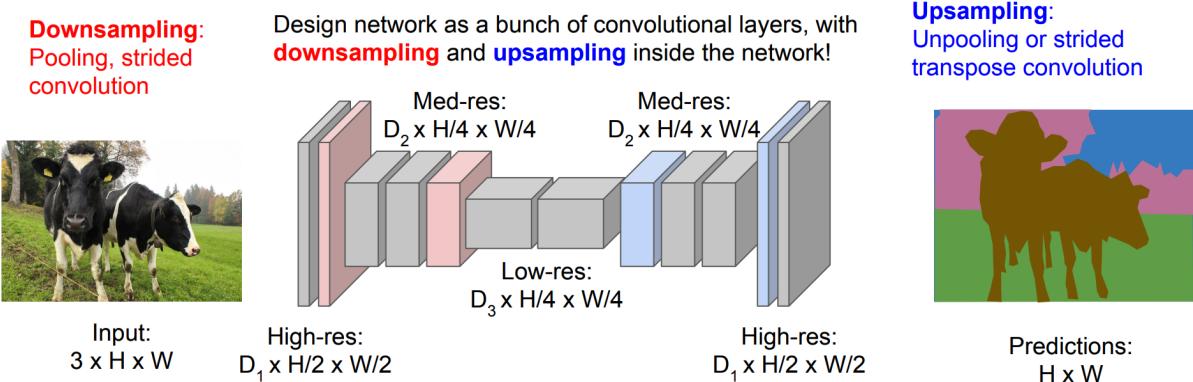
1. 이미지를 여러개의 local crop으로 자른다. (patch라고 부른다)
  2. 각 이미지 patch에 대한 classification problem으로 해결할 수 있다.
- 문제
  - 매우 비효율적(연산량이 많고, patch의 겹쳐진 부분을 계속하여 다시 계산하는 문제가 있다.)

## 구현방법2) Fully Convolutional Layer



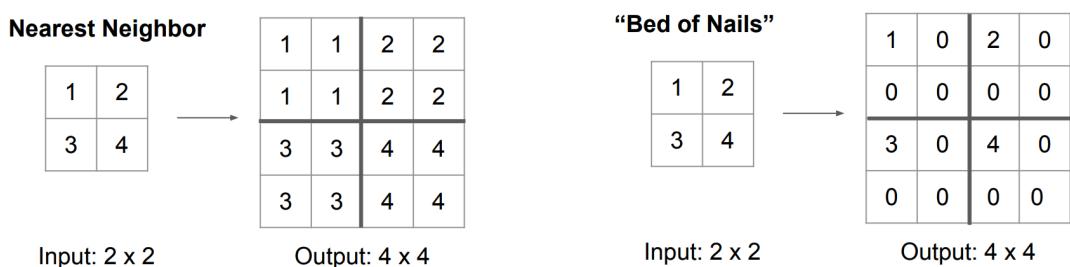
- 방법
  - 입력 이미지를 CNN에 통과시킨 후, 매 픽셀을 classification
  - 모든 픽셀에 대해 cross entropy 적용하여 분류
- 문제
  - 원본 이미지 resolution에 대한 convolution 연산량이 크다.

## 구현방법3) Fully Convolutional Layer (with Down/Up-sampling)

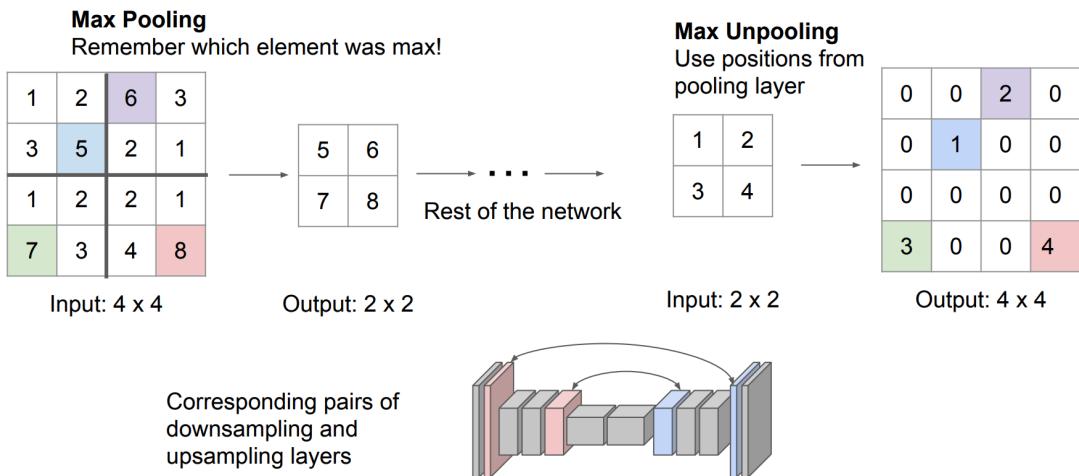


- 장점: Spatial resolution이 줄어드므로, 연산이 효율적이다.

- Downsampling 방법
  - strided conv, pooling, ... → spatial size를 줄인다.
- Upsampling 방법
  1. Unpooling



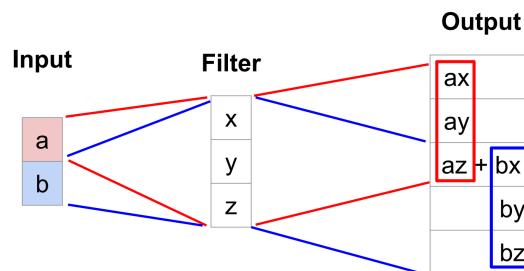
## 2. Max Unpooling



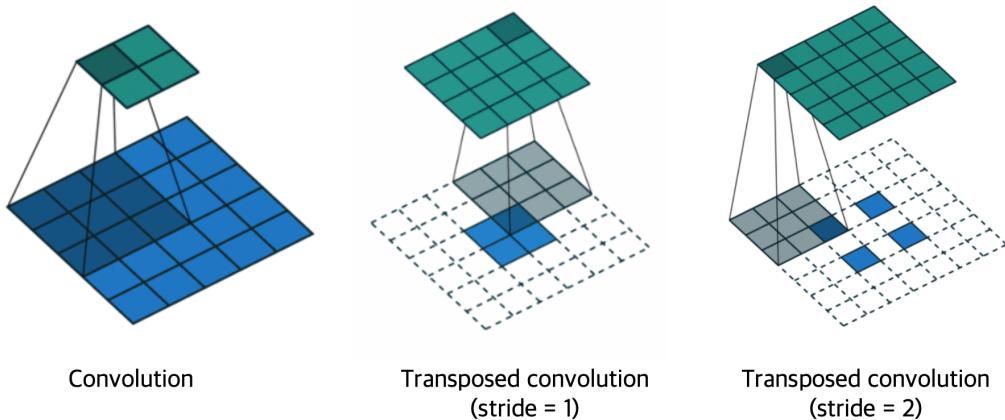
- Receptive field의 어떤 부분이 사용됐는지를 기억해두었다가 동일 위치에 재적용하고, 나머지 픽셀을 0으로 채워준다.
- Max pooling 시, receptive field에서 어떤 픽셀을 채택했는지에 대한 정보를 잊게 되는데, 여기에 벡터를 추가해서 spatial 정보를 기억하게 하는 역할이다.

## 3. Transpose Convolution

- 용어
  - ⇒ learnable layer, learnable upsampling  
(이전까지는 upsampling 하는 과정에서 학습이 이루어지지 않았음 = fixed function)
  - ⇒ 종종 deconvolution layer라고 쓰는 paper도 있으나, 명확한 명칭은 아님
  - ⇒ upconvolution, fractionally strided convolution, backwards strided convolution 등으로도 불림
- 과정
  - upsampling에서 사용되는 커널도 학습의 대상이 되며, input에 weight를 곱하여 output feature map을 도출한다.
  - 도중에 overlap되는 부분은 더해준다.
- 이미지를 upsampling하며, spatial size를 늘려주는 역할을 한다.
- 1D 예시



- 2D 예시

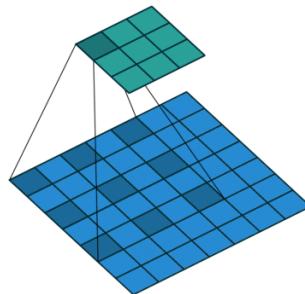


- stride가 1인 경우, 완전히 일반적인 convolution 계산과 동일하지만, stride가 2 이상인 경우에는 다르다. 입력 이미지에 변형을 준 후(입력에 padding을 줌)에 convolution 연산을 수행한다.

▼ Transposed Convolution과 Deconvolution의 개념적인 차이

- **Deconvolution**은 Convolution 연산할 때 사용한 kernel과 output을 알고 있어야 하며, 역 연산을 통해서 input 을 재현하는 목적이라고 볼 수 있다.
- **Transposed Convolution**에서 사용하는 kernel은 어떤 convolution layer와 공유하는 것이 아니다. Transposed convolution layer가 학습을 통해서 kernel을 찾아간다는 점에서 차이가 있다. 공간(크기)을 재현하는 목적이라고 볼 수 있다.
- 출처: <https://realblack0.github.io/2020/05/11/transpose-convolution.html#Transposed-Convolution>

▼ 추가) Dilated Convolution(Downsampling)

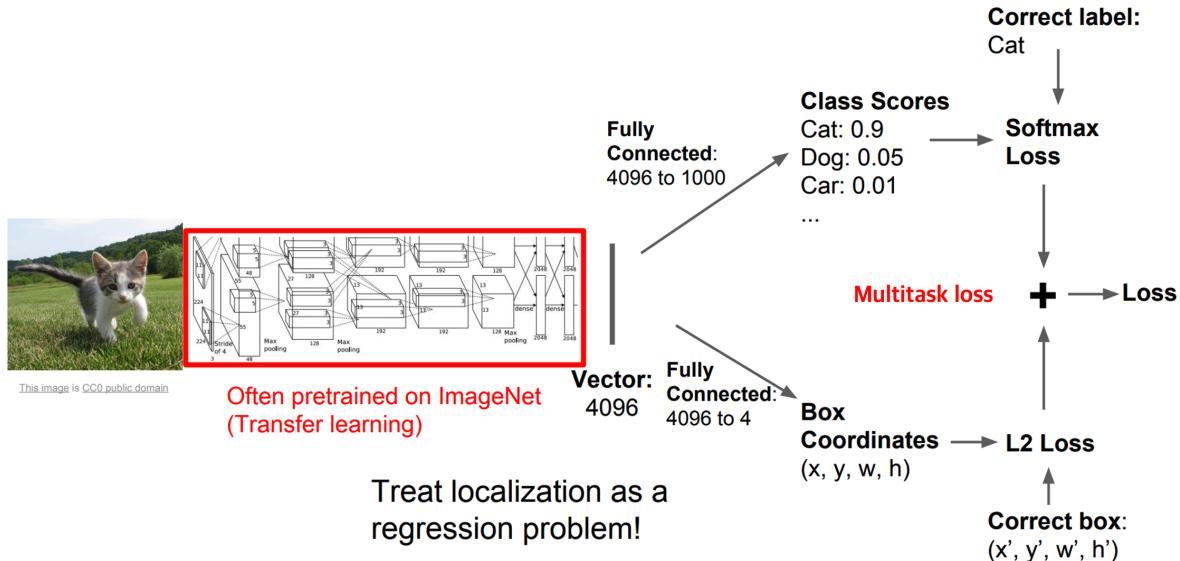


- Dilated Convolution은 필터 내부에 zero padding을 추가해 강제로 receptive field를 늘리는 방법이다. 위 그림은 파란색이 인풋, 초록색이 아웃풋인데, 진한 파랑 부분에만 weight가 있고 나머지 부분은 0으로 채워진다. receptive field란 필터가 한 번의 보는 영영으로 볼 수 있는데, 결국 필터를 통해 어떤 사진의 전체적인 특징을 잡아내기 위해서는 receptive field는 높으면 높을 수록 좋다. 그렇다고 필터의 크기를 크게하면 연산의 양이 크게 늘어나고, 오버 피팅의 우려가 있다. 그래서 일반적인 CNN에서는 이를 conv-pooling의 결합으로 해결한다. pooling을 통해 dimension을 줄이고 다시 작은 크기의 filter로 conv를 하면, 전체적인 특징을 잡아낼 수 있다. 하지만 pooling을 수행하면 기존 정보의 손실이 일어난다. 이를 해결하기 위한 것이 Dilated Convolution으로 Pooling을 수행하지 않고도 receptive field의 크기를 크게 가져갈 수 있기 때문에 spatial dimension의 손실이 적고, 대부분의 weight가 0이기 때문에 연산의 효율도 좋다. 공간적 특징을 유지하는 특성 때문에 Dilated Convolution은 특히 Segmentation에 많이 사용된다.

출처: <https://3months.tistory.com/213>

## Classification + Localization

- 정해진 수의 개체에 대해, 카테고리 값과 그 위치를 반환하는 task
- Localization을 regression 문제로 해결한다.



- CNN을 통과시켜 feature를 얻은 후, class score를 얻기 위해 FC layer를 통과하고, bounding box(bbox) 좌표를 얻기 위해 FC layer를 통과한다.
- Class score를 얻기 위해 categorical loss(categorical output), bbox 좌표를 얻기 위해 regression loss(continuous output)를 사용한다. 둘을 합쳐서 multitask loss라고 한다.
  - Hard sharing: 동일한 뿌리모델을 사용하되, 서로 다른 representation을 학습하는 것  
(참고: Multitask learning(MTL) - <https://mapadubak.tistory.com/40>)
  - + 각 클래스에 대한 bbox를 구하고, 정답 bbox에 대한 L2 loss만 backpropagation하기도 한다.
- Human Pose Estimation 등에 활용 가능  
(정해진 joint 수에 대해, feature extraction 부분은 공유하고, 마지막 분류 레이어만 다르게 학습하여 사용)

## Object Detection

- 이미지에서 찾아야하는 개체의 수를 모른 채로 시작한다.
- 이미지마다 찾아야하는 개체의 수(bbox)가 다르다.
- 모든 bbox의 위치와 그 박스들이 각각 어떤 카테고리인지를 예측하는 문제이다.
- Object detection은 regression으로 풀지 않는다.  
→ 각 이미지가 서로 다른 수의 output(bbox)를 내기 때문이다.

### 방법1) Sliding Window

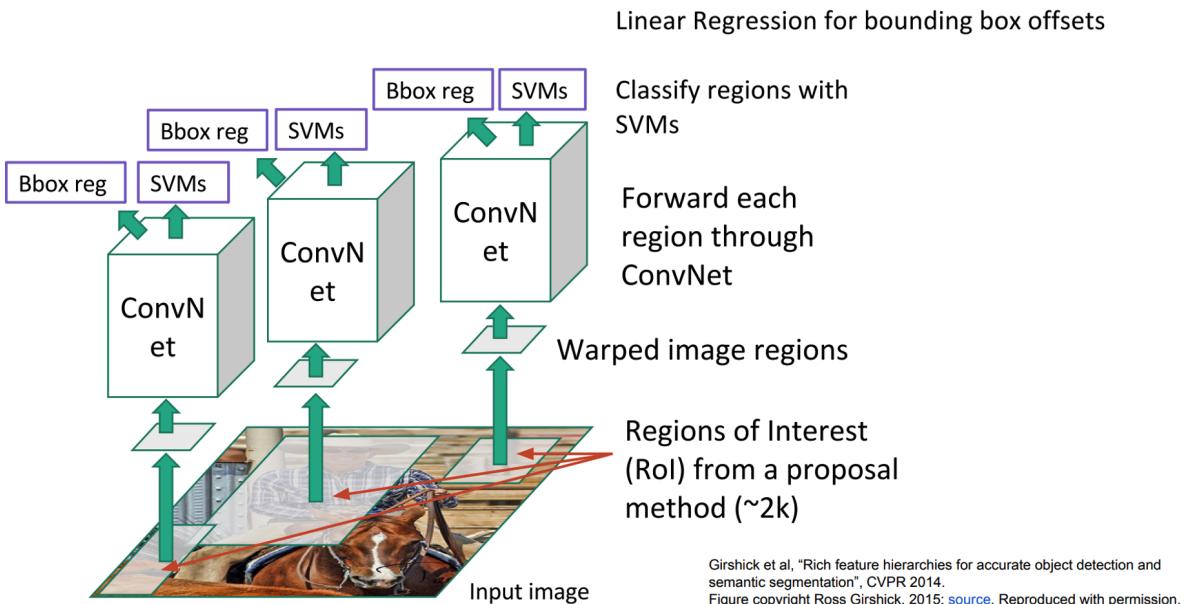
- 입력 이미지를 여러 다른 crop으로 잘라서, CNN classify task를 수행한다.  
각 crop이 배경인지, 물체인지 분류한다.
- 문제: 엄청난 양의 계산량  
→ crop 위치와 크기에 따라 다양한 patch가 나오게 되는데, 이를 브루트포스로 찾게 되면 연산양이 매우 커지게 된다.  
→ 해결책: Region Proposal

## 방법2) Region Proposal

- Crop의 위치와 크기를 정하는 방법으로, 딥러닝이 아닌 전통적인 CV 방식을 사용한다.
- 상대적으로 빠르게 동작하며, 여기에 selective search와 같은 방법이 존재한다.
- Object가 있을만한 bbox(후보가 되는 region)를 제공하는 네트워크이다.
- Region proposal 과정에서 반환하는 후보군을 Region of Interest(Roi)라고 부른다.

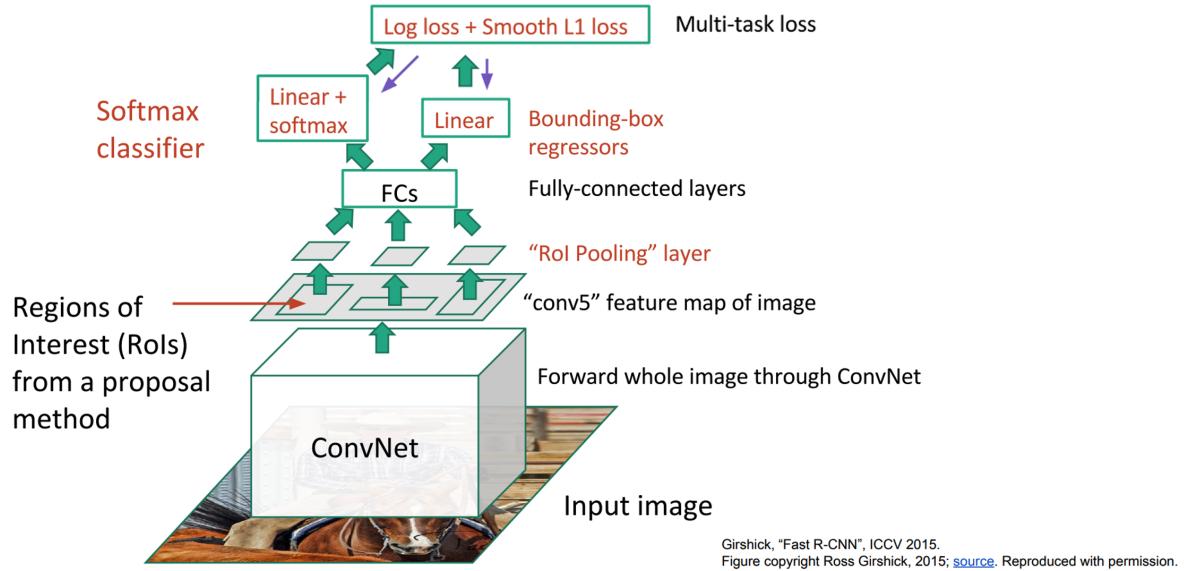
### Region Based Method

#### Region proposal 활용1) R-CNN



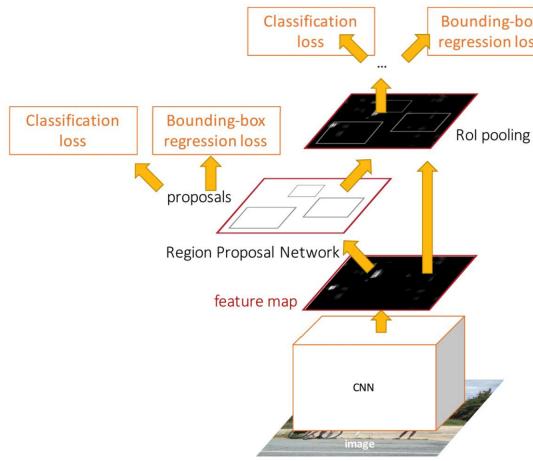
1. Region proposal 사용하여, Roi를 뽑아낸다.
2. Roi들을 동일한 이미지 사이즈(fixed img size)로 만들어준다.(CNN 통과를 위해)
3. 각각을 CNN에 통과시켜서, class label(classify)과 bbox offset(regression)을 구한다.
  - Regression 과정을 통해 region proposal에서 도출한 output box의 범위를 벗어난 최종 bbox를 낼 수 있다.
  - 총 5개의 output: `class score`, `(x, y, h, w)`
- 문제점
  - 연산량이 많고, (학습과 실행 속도가) 느리다.
  - 고정된 region proposal(not learning)

## Region proposal 활용2) Fast R-CNN

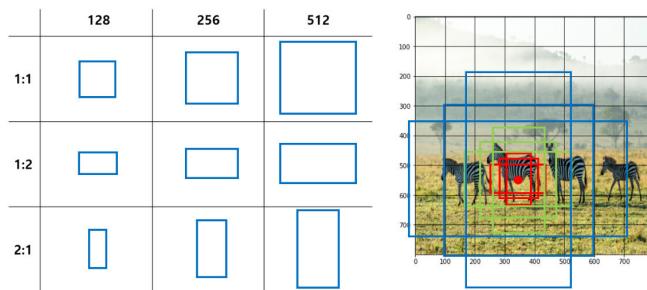


1. 이미지를 잘라내서 각각을 모두 convolution layer에 통과시키는 것이 아닌,  
입력 이미지를 convolution layer에 통과시키고, 이후 ROI 영역을 잘라내서 사용하는 방법이다.
  2. convolution feature map의 crop들을 reshape한다.(RoI Pooling)
  3. 각각을 FC layer에 통과시킨 후,  
softmax classifier를 통해 class를 예측하고, linear regression을 통해 bbox offset을 구한다.
- 문제
    - R-CNN에 비해 Fast R-CNN이 매우 빨라지긴 했으나, 여전히 region proposal 연산으로 인해 속도가 느림(bottleneck)

## Region proposal 활용3) Faster R-CNN



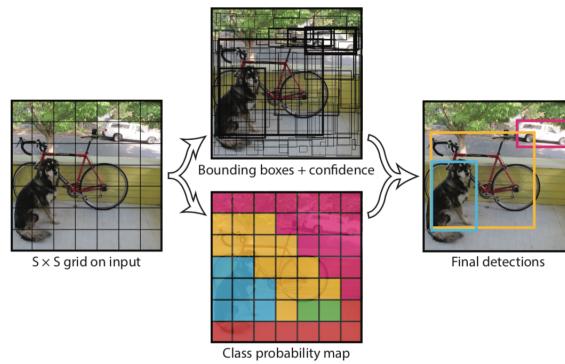
- CNN 내에서 region proposal을 예측한다.
- **Region Proposal Network(RPN)**을 추가한다. CNN의 결과로 나온 feature map을 사용하여 proposal을 예측하여 반환하는 역할을 한다.
- 모델의 전체 과정이 GPU 상에서 동작하여 병목 현상이 발생하지 않는다.
- 학습에 사용되는 loss 4개
  1. RPN classify(물체인지 배경인지, binary classification loss)
  2. RPN regress(box 좌표) → 정답과 overlap 된 정도에 따라 loss 계산
  3. Final classification score(object class)
  4. Final box 좌표
- 과정
  1. 원본 이미지를 pre-trained된 CNN 모델에 입력하여 feature map을 얻는다.
  2. feature map은 RPN에 전달되어 적절한 region proposals를 산출한다.
  3. region proposals와 1) 과정에서 얻은 feature map을 통해 RoI pooling을 수행하여 고정된 크기의 feature map을 얻는다. ( $\Rightarrow$  RoI pooling을 사용하면 입력 이미지 크기와 상관없이 고정된 크기의 feature map을 얻을 수 있다.)
  4. Fast R-CNN 모델에 고정된 크기의 feature map을 입력하여 Classification과 Bounding box regression을 수행한다.
- Anchor box 개념의 도입
  - Selective search 방법을 사용하지 않는 경우, 원본 이미지를 일정 간격의 grid로 나눠서, 각 grid cell을 bounding box로 간주하는 방식을 사용한다.
  - 이처럼(grid cell)고정된 크기의 bbox를 사용할 경우, 다양한 크기의 객체를 포착하기 어려울 수 있으므로, 서로 다른 크기와 가로세로비를 가지는 bbox(anchor box)를 도입하게 된다.
  - $8 \times 8$  grid cell마다 9개의 anchor box가 생성되어 총 576( $=8 \times 8 \times 9$ )개의 region proposals가 추출됨



## Non-proposal Method

### YOLO

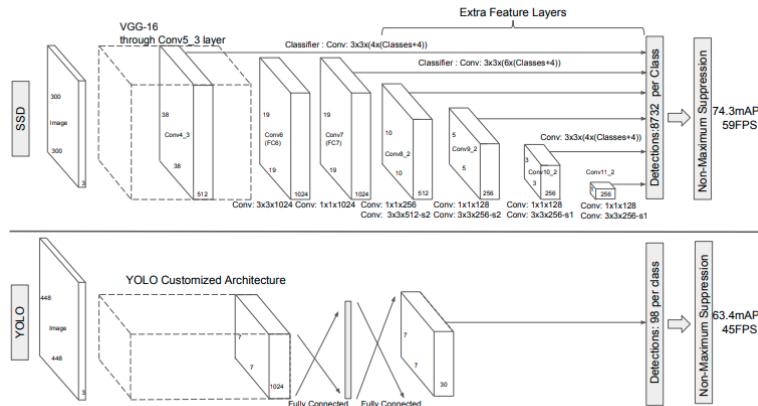
- 기존 region proposal methods는 기존에 1) region proposal 2) classification/regression 이렇게 단계를 나눠서 진행하는 방식이었다. YOLO는 region proposal 단계를 제거하여 빠른 속도를 가진다.
- 입력 이미지, feature map을  $7 \times 7$  grid로 나누고 각 cell마다 특정 개수의 anchor box를 만들어서, class score(probability), box confidence를 계산한다.
- output:  $7 \times 7 \times (5 \times B + C)$   
 $= (7 \times 7 \text{ grid}) * ((x, y, w, h), \text{confidence}) * \text{bbox 수} + \text{classification score}$



- 참고: <https://yeomko.tistory.com/19>,  
<https://techblog-history.younghunjo1.tistory.com/186?category=1031745>

### SSD

- 작은 물체를 잘 잡아내지 못하는 YOLO의 단점을 보완하기 위해 나왔다.
- CNN(VGG16)을 통과하여 추출된 feature map을 convolution layer을 거쳐 그 다음 층에 넘겨주는 동시에 object detection을 수행한다. 이전 fully convolution network에서 convolution layer를 거치면서 디테일한 정보들이 사라지는 문제점을 앞단의 feature map들을 끌어오는 방식으로 해결했다.



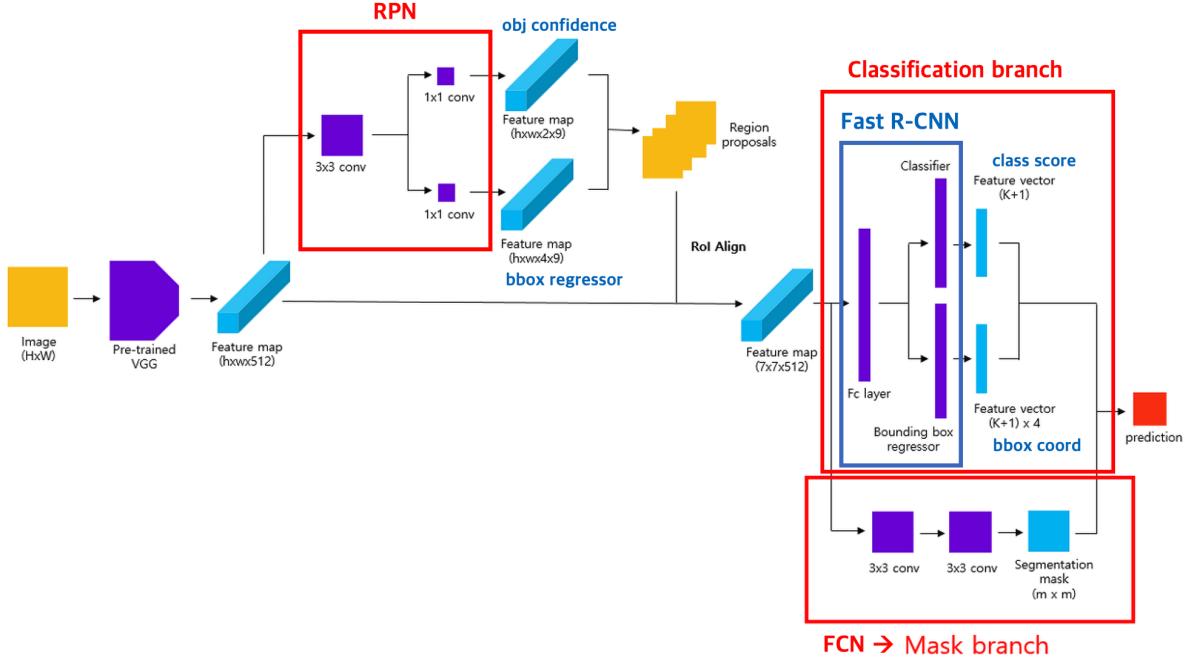
- 참고: <https://yeomko.tistory.com/20>
- Faster R-CNN은 더 느리지만 더 정확하다. SSD는 빠르지만, 그리 정확하지는 않다.

## Instance Segmentation

- 각 픽셀이 어떤 object instance인지 알아내는 task
- Semantic Segmentation + Object Detection

### Mask R-CNN

- Faster R-CNN에 기반하여 제작되었다.
- mask branch는 각각의 RoI에 대하여 class별로 binary mask를 출력한다.  
(classification branch와 분리되어 수행된다.)



- 출처: <https://herbwood.tistory.com/20?category=856250>