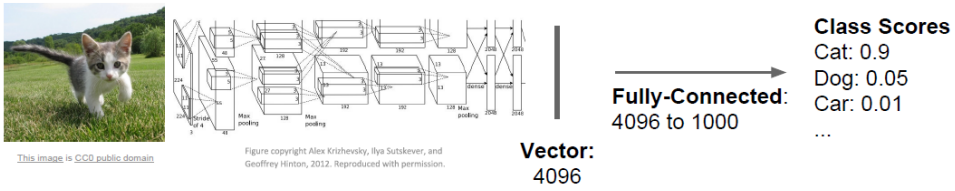
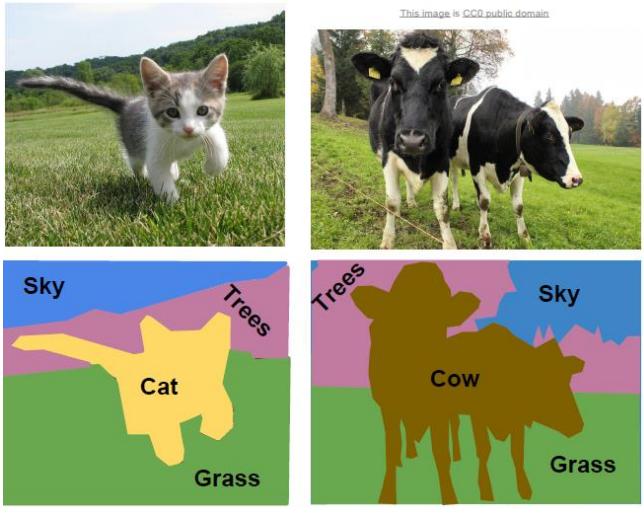
< Segmentation >

- 기존 Computer Vision



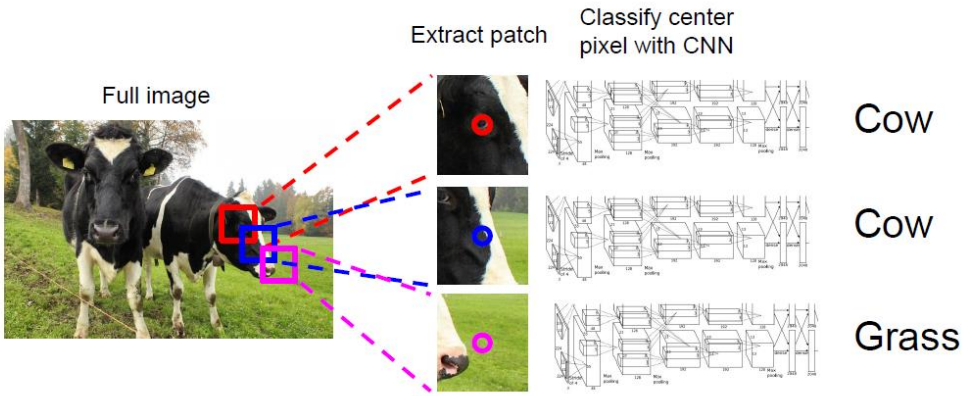
- 이미지 입력 받고 network 통과해서 그 결과로 feature vector 산출하는 방식
- 최종결과로 정해진 개수의 차원이 스코어를 기반으로 나옴
- 즉 이미지 입력하면 그 이미지가 어떤 카테고리에 해당하는지 출력해줌

- 1. Semantic segmentation



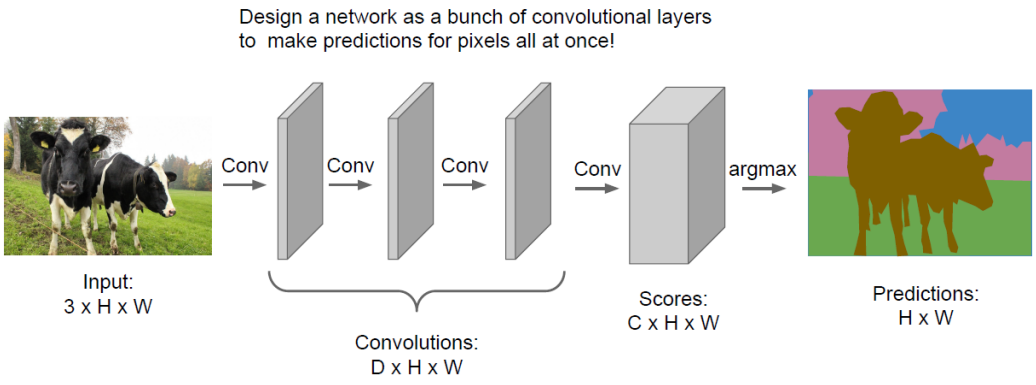
- Semantic : 의미론적
- 이미지 별로 category를 나누는 것이 아니라 '픽셀'별로 category label을 부여하는 개념
- 즉 입력으로 이미지 들어오고 각 이미지의 픽셀에 카테고리가 할당됨
- 문제점 : 붙어있는 소 두마리를 개체별로 인식하지 못하고 하나의 '소'로 인식함

- Instance Segmentation : Sliding Window



- 입력 이미지를 작은 이미지 단위인 '패치'로 나누고 이 패치의 중심픽셀이 어느 카테고리에 해당하는지 분류해줌
- **문제점** : 엄청난 비효율
  - 이미지를 패치로 나누고 각 영역을 forward/backward 해야하기 때문
  - 독립적인 패치를 다루는 접근법이므로 인접한 영역의 특징을 공유할 수 없음

- Fully Convolutional Network



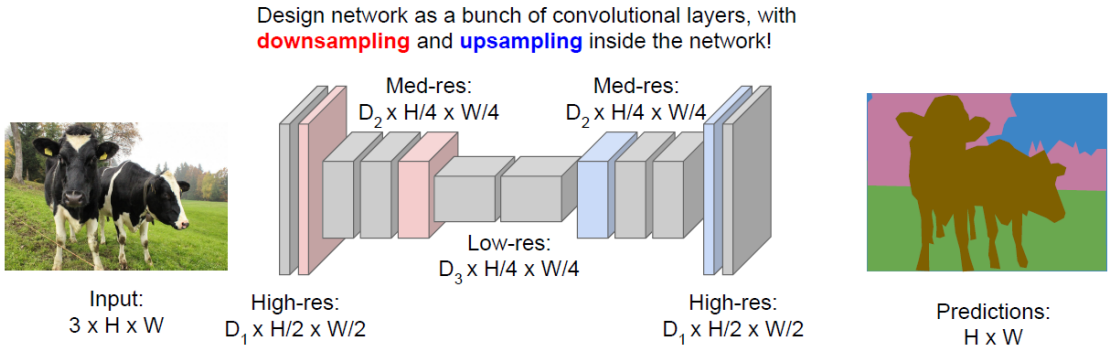
- 모든 픽셀들을 한꺼번에 예측하기 위해 **수 많은 convolution 층**을 쌓음
- Fully connected layer 없이 convolution layer만 쌓음
- Zero padding 과 3×3 filter 활용하여 각 convolution 층의 공간 정보 유지가능
- C × H × W 형태로 최종 tensor 출력 , C는 카테고리 개수

- 최종 tensor는 입력 이미지의 모든 픽셀 값에 대해 score 를 부여한 값
- Training 과정 : output의 모든 픽셀의 classification loss를 구하고 평균내서 일반적인 역전파 과정 거치게 함

- Q : training data는 어떻게 구축하나요?
  - 모든 픽셀의 카테고리를 알고 있다는 가정하에 입력 이미지의 모든 픽셀에 label을부여함
- 비용과 메모리 문제 : 높은 해상도의 이미지를 같은 공간 크기를 유지한 채로

  네트워크 구축해야함 -> 잘 사용하지 않음

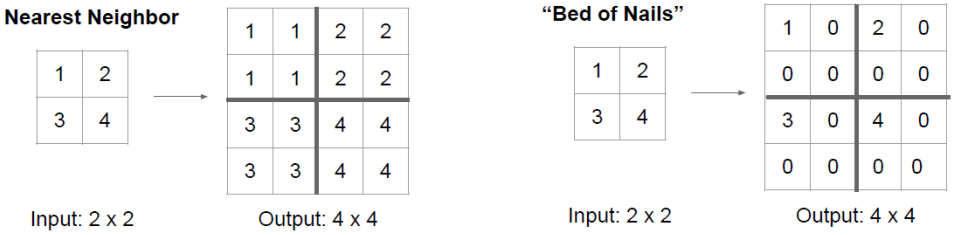
Downsampling & Upsampling



- Convolution layer의 공간크기를 유지하는 것이 아니라

  max pooling이나 stride 기법으로 downsampling 함
- 원래 Fully Connected layer로 넘어가는 단계에서 upsampling 진행하여 다시 spatial resolution을 증가시키고 궁극적으로 입력 이미지의 해상도와 같게 됨.
- 계산적인 효율성 : lower spatial area에서 처리하기 때문

· Upsampling 방법 : Unpooling



·

· **Nearest neighbor**

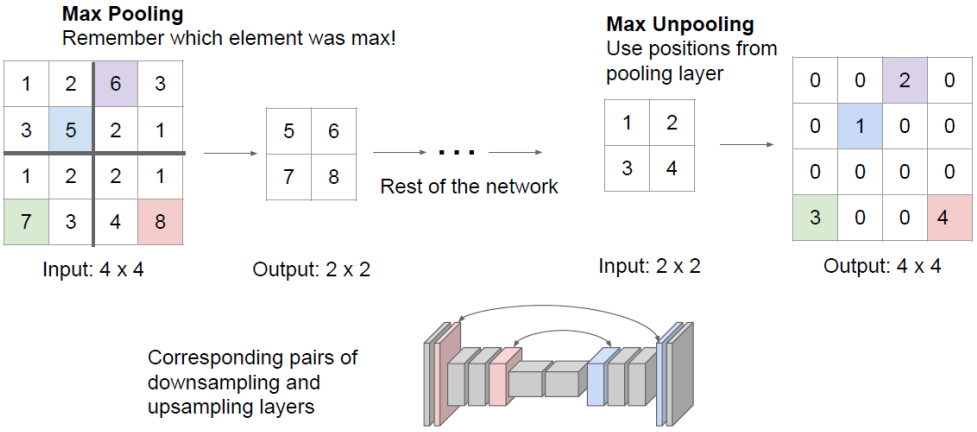
  · 2 ×2 의 Receptive field의 동일한 숫자를 복제하여 사이즈를 키움 방식

· **Bed of nails** (가시방석)

  · 입력 이미지의 숫자를 왼쪽 상단에 배치하고 나머지 0으로 채워줌

  · 0외의 값이 튄다는 의미에서 bed of nails라는 이름 붙임

· Max Unpooling



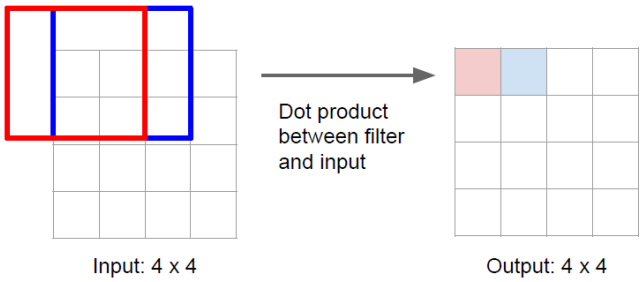
·

· Max pooling 할 때 그 **'위치'**를 기억해서 unpooling 시 그 자리에 값을 배치하고 나머지는 0으로 채워줌

· maxpooling으로 잃어버린 feature map 의 공간정보를 균형있게 유지할 수 있음

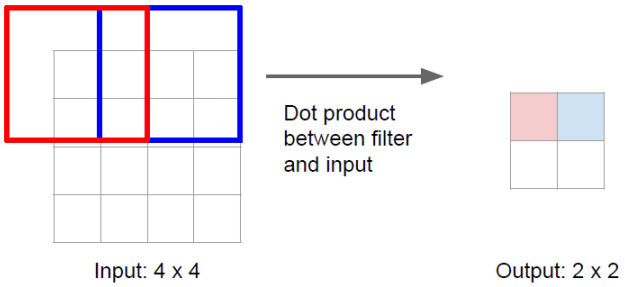
- Transpose Convolution : **학습가능한** upsampling

  - Strided Convolution : 앞에서 나온 학습없이 고정된 함수를 적용하는 방법과 달리 네트워크가 '학습'을 통해 어떤 식으로 dowsampling을 할지 정함
  - 이와 유사하게 upsampling시에도 학습을 하는 방법이 transpose convolution 방법
  - 일반적인 3×3 conv, stride 1, padding 1인 convolution filter 적용시



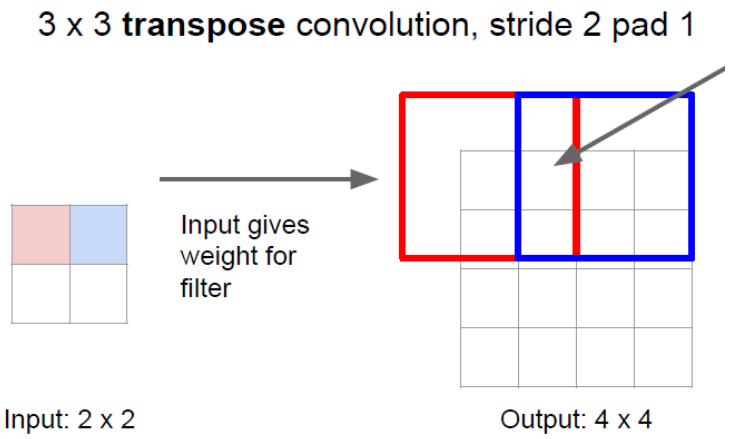
  -
  - Input 4×4 -> output 4×4

  - 3×3 conv, stride **2**, padding 1인 convolution filter 적용시



  -
  - Input 4×4 -> output 2×2 두 픽셀만큼 움직이기 때문에 2분의 1로 다운샘플링 됨

- Transpose Convolution

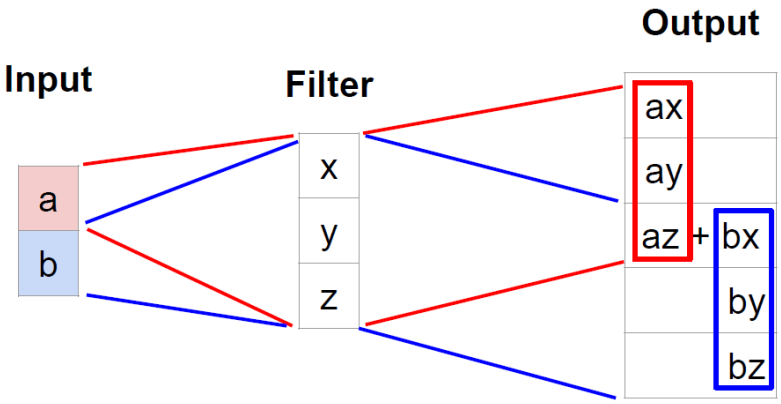


  -
  - Feature map의 분홍색으로 표시된 영역의 스칼라 값을 3×3 filter와 곱함
  - 출력의 3×3영역에 이 값을 넣어줌
  - 필터와 입력을 내적하는 것이 아니라, 입력값이 '가중치' 역할을 하면서 필터에 곱해짐
  - 출력 값 = (3×3)필터 × 입력(가중치)
  - 중첩되는 부분은 더해주면서 upsampling 진행

- Transpose convolution 의 다른 명칭
  - Deconvolution : 혼용되어 쓰이고 있으나, transpose와 달리 deconvolution은 convolution 의 역연산 개념을 의미하므로 주의

    (실제로 transpose는 convolution의 역연산 아님)

- Transpose convolution 예시



-

- 출력값 = 필터 × (가중치 역할을 하는 입력값)
- 중첩되면 더해줌

Source https://lsjsj92.tistory.com/416

https://hoya012.github.io/blog/SIngle-Image-Super-Resolution-Overview/

그렇다면 왜 이 연산을 'transpose convolution'이라고 칭하는가?

- 모든 convolution은 matrix multiplication으로 나타낼 수 있다
  (1 dimensional example; 3 x 1, stride = 1 convolution과 동일한 결과)

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

- $\vec{x}$: weight matrix, $\vec{a}$: input matrix

- 그럼 **transpose** convolution은?

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

- $\vec{x}$ 대신 $\vec{x}^T$

- 위 예시처럼 stride = 1만 봤을 땐 normal convolution과 transpose convolution이 기본적으로 같은 연산같아 보이지만
  stride > 1에서는 transpose convolution은 더 이상 convolution이 아니다

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

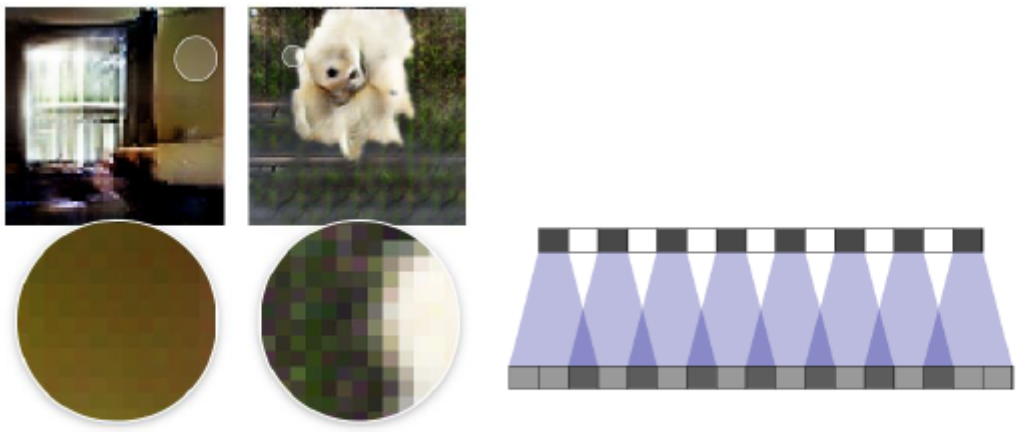
$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

When stride>1, convolution transpose is no longer a normal convolution!

- transpose convolution의 **더하는** 방식의 문제점:
  receptive field에 따라 달라지는 magnitude
- solution:
  3 x 3, stride = 2 transpose convolution에서 자주 발생하는 checkerboard artifact를 완화하기 위해 최근에는 4 x 4, stride = 2 / 2 x 2, stride = 2 사용

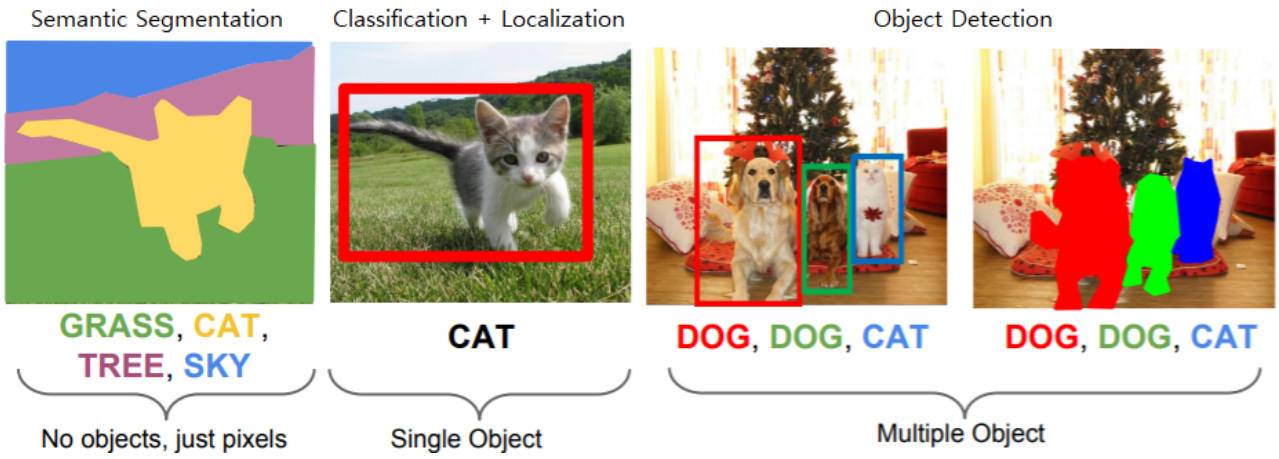
* Checkerboard artifact



- Transpose 연산을 할 때 filter size와 stride에 따라 **불균형한 중첩**(uneven overlap)이 일어날 수 있으며, 이로 인해 'checkerboard artifact' (격자 형태로 깨지는 현상)가 발생

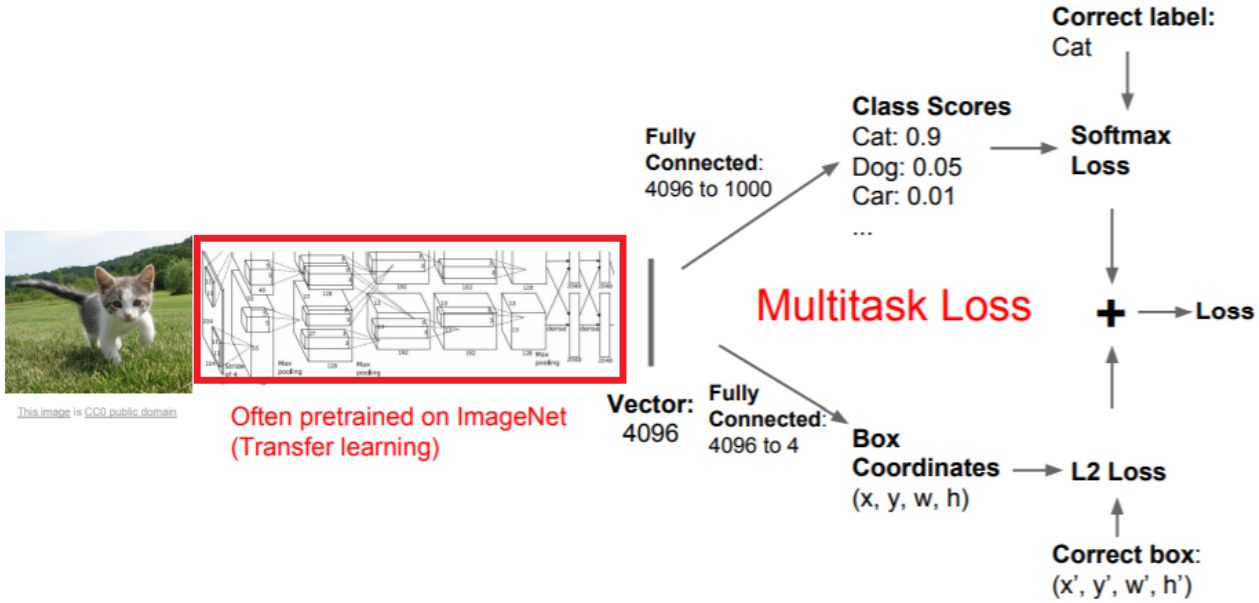
2. Classification + Localization

- 이미지가 어떤 카테고리에 속하는지(Classification) 뿐만 아니라 실제 객체가 어디에 있는지(Localization)



- Object detection과의 차이점: 이미지 내 **한 개의 객체**에만 집중

- 전체적인 구조(AlexNet example)

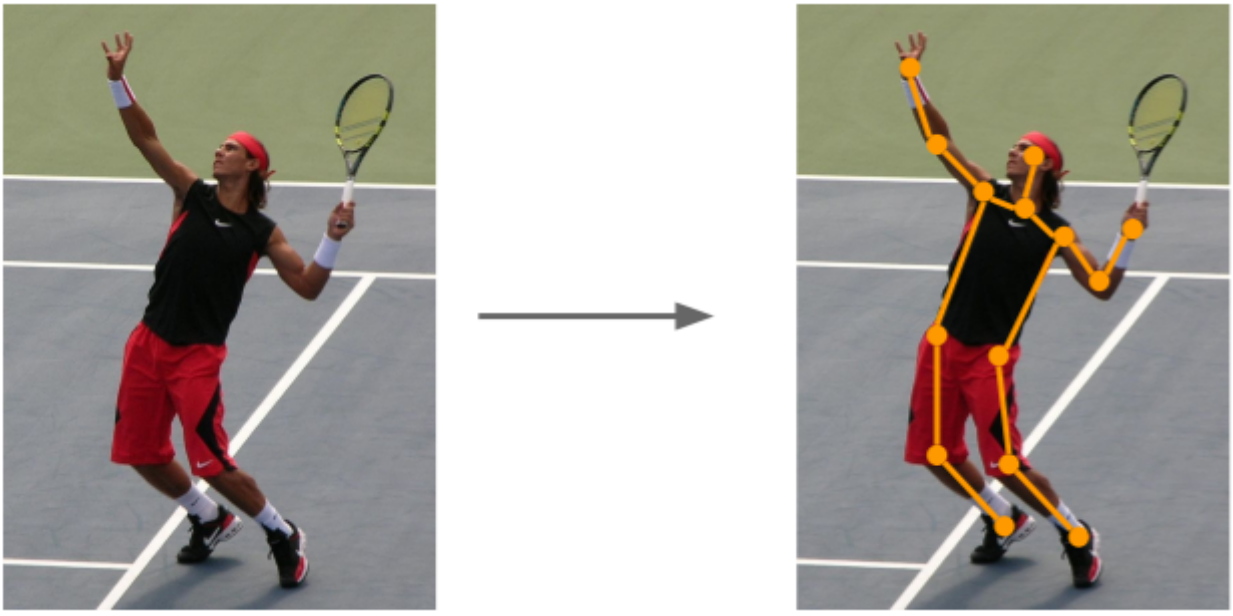


- Semantic segmentation처럼 final vector에서 **class scores**로 보내주는 FC Layer
- Final vector에서 **4개의 숫자**(x, y, w, h)로 보내주는 또 다른 FC Layer
- Loss function도 두 개(Softmax loss, regression loss)
  => 두 loss의 가중합(Multitask Loss)에 대한 gradient 계산
- 이때 가중치를 결정하는 hyperparameter 설정이 상당히 어렵다(hyperparameter 값이 변하면 loss functioin 자체가 변하기 때문)
- 방법은 진리의 케바케,,지만 한 가지 general strategy는 hyperparameter 값에 따라 달라지는 loss 값 자체를 비교하기보다는 다른 performance metric을 두고 비교하는 것

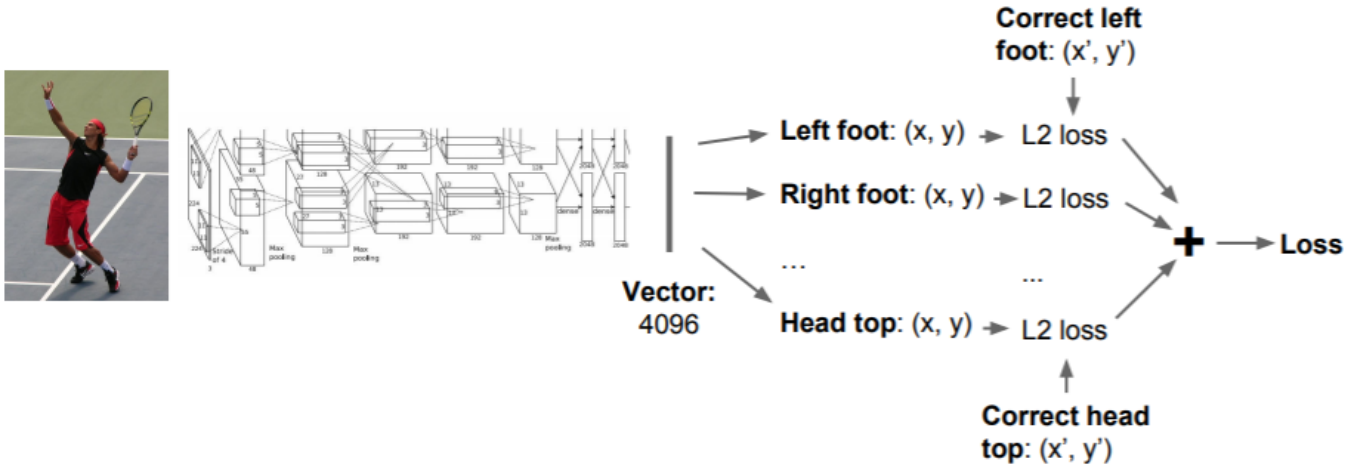
* Classification loss와 Regression loss
Classification loss(Cross entropy, Softmax): Categorical output
Regression loss(L2 Euclidean, L1, Smooth L1): Continuous ouptput

Human Pose Estimation

(another example of predicting a fixed number of positions in the image)

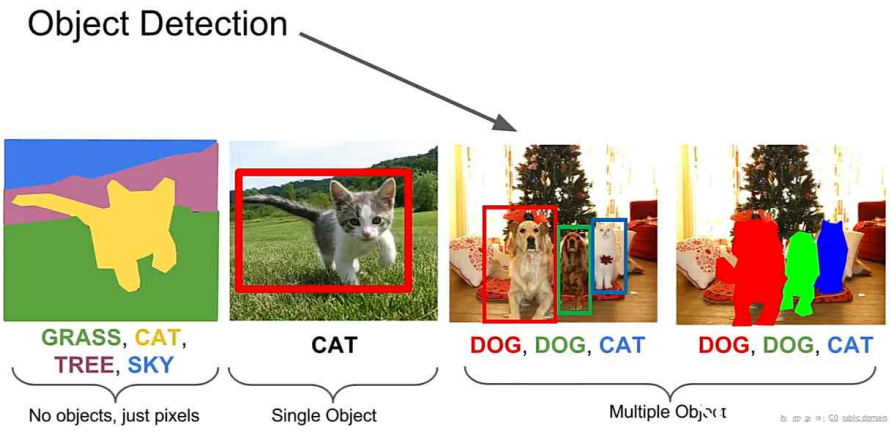


input image -> 14개 관절의 위치 출력

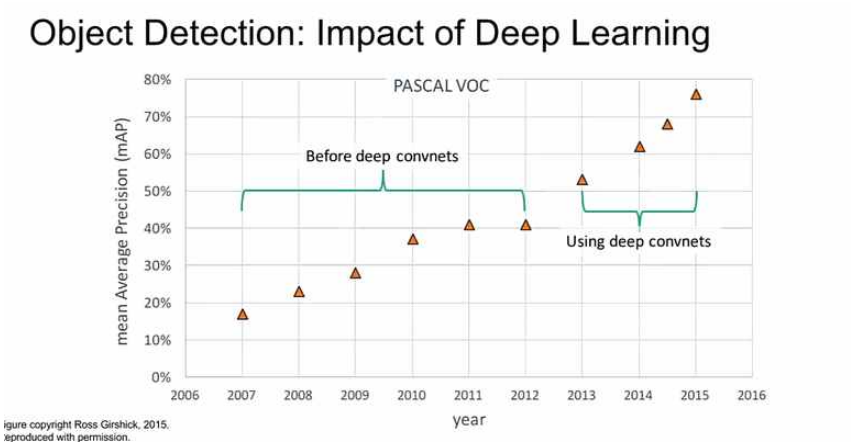


- 출력해야 할 bounding box의 개수가 고정되어 있는 경우엔 언제든 classfication + localization framework 사용 가능

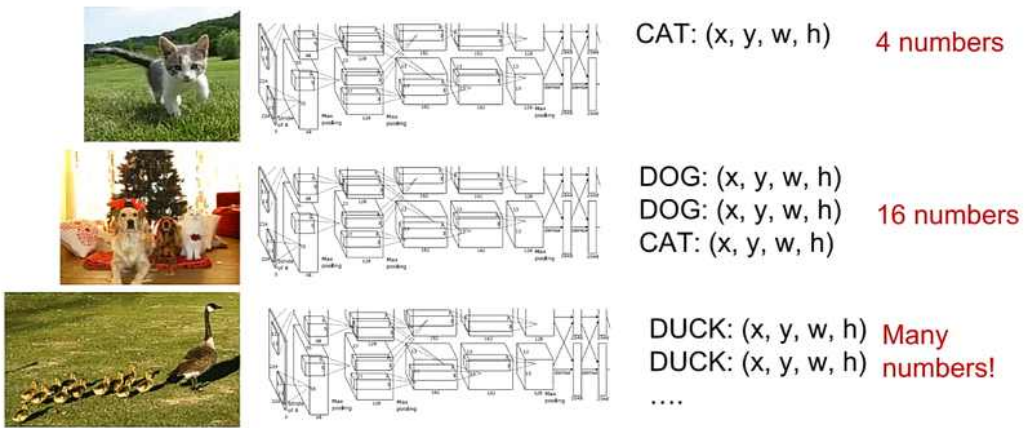
[Object detection]



- object detection : 입력 이미지가 주어지면 이미지가 나타나는 객체들의 box와 카테고리를 예측
- 예측해야하는 box의 수가 입력 이미지마다 달라진다. (매우 어렵다)



- 해마다 object detection의 정확성은 상승하고 있다.
- 2013년 딥러닝이 도입된 이후 매우 상승하고 있으며, 2016년 이후에도 매우 큰 향상을 보여주고 있다.



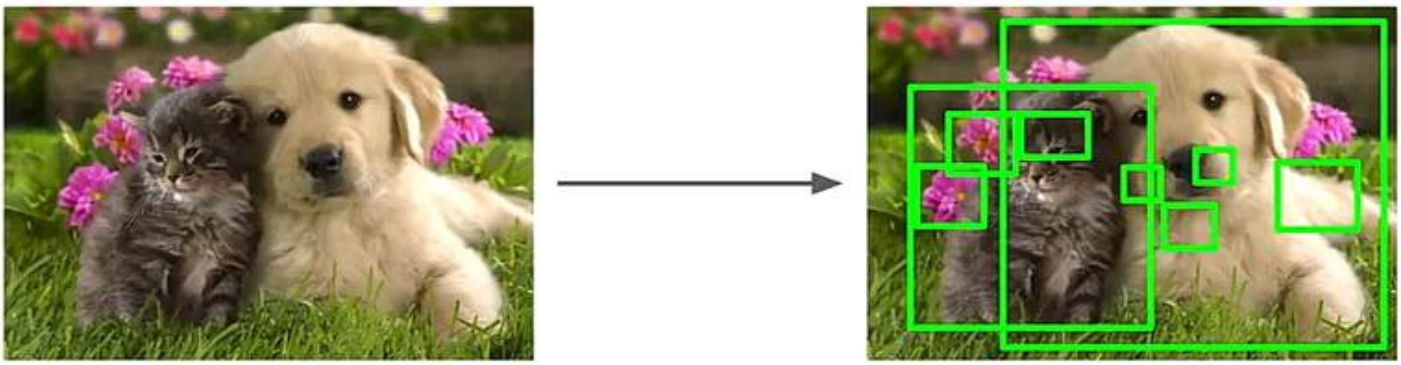
- Object Detection은 Localization과는 다르게 객체의 수가 이미지마다 다르다.

- 이미지마다 객체의 수가 달라지고 이를 알 수 없어서, 이를 다루기 위해 새로운 방법이 필요하다.

[Sliding window]

- 입력 이미지를 매우 다양한 영역으로 나누어서 CNN 이 각각의 영역에서 학습 및 예측하게 한다.
- 파란색 박스만 추출해 CNN 의 입력으로 넣는다고 하면 첫 사진의 경우 배경, 두 번째 사진의 경우 개라고 예측
- 하지만 어떻게 영역을 추출해야 할까? object가 어디 있는지, 얼마나 큰지 알 수 없어 어려운 문제이다.
- 모두 다 해보는 방식의 brute force 방식의 sliding window를 하려면 경우의 수가 너무 많고 계산량이 엄청나다.

[Region proposal network]



- Object가 있을법한 box 들을(예를 들면 1000개) 만들어내는 방법

Selective search (Box generating 방법 중 하나)

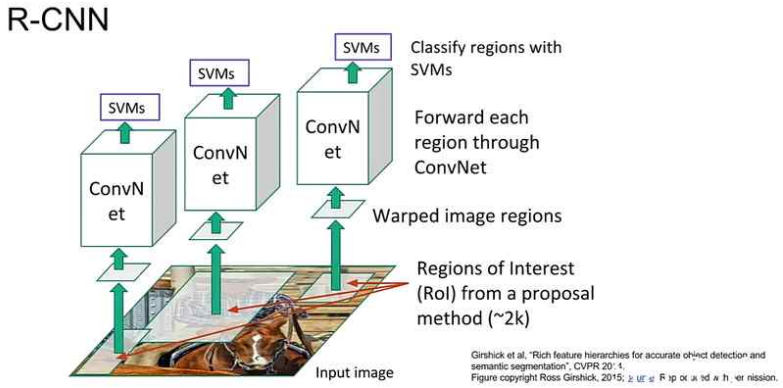
- 색상이나 강도 패턴 등이 비슷한 인접한 픽셀을 합치는 방식이다.
- 이 방법은 noise가 커서 대부분은 실제 객체가 아니겠지만 recall이 높다. (즉 이미지에 객체가 존재한다면 selective search의 Region proposal 안에 속할 가능성이 큼)

$$(Recall) = \frac{TP}{TP + FN}$$

(실제 True인 것 중에서 모델이 True라고 예측한 것의 비율)

- 이제는 무식하게 이미지의 모든 위치와 스케일을 고려하지 않고 우선 region proposal networks를 적용하고 객체가 있을법한 region proposal을 얻는다. 그리고 이 region proposal을 CNN의 입력으로 한다.
- 이 방법은 앞의, 모든 window를 brute force 방식으로 고려하는 거보다 훨씬 계산량이 감소한다.

[R-CNN]



[작동 과정]

1. 이미지가 주어지면 Region proposal(Region of interest) 을 얻기 위해 Region proposal network를 수행한다.
   - selective search를 통해 2000개의 ROI(Region of interest)를 형성

2. CNN classification은 입력사이즈가 동일해야 하는데 각 ROI의 사이즈가 달라서, 사이즈를 같게 변형시킨다.

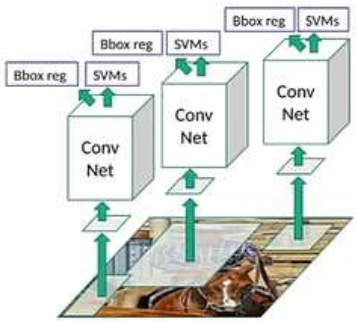
3. 사이즈가 맞추어진 ROI들을 CNN의 입력으로 보낸다.

4. ROI 들의 최종 Classification 으로 SVM을 사용한다.
   - 이때 R-CNN 은 selective search 의 region proposal이 정확하지 못한 경우도 있기 때문에 Region proposal을 보정하기 위한 regression 과정도 거친다.
   - 그에 따라 R-CNN는 BOX의 카테고리 예측 및, BBOX (Bounding Box)를 보정해줄 offset 값 4개도 예측한다.

5. 위 값들에 대해 multi-task loss로 두고 한번에 학습한다.

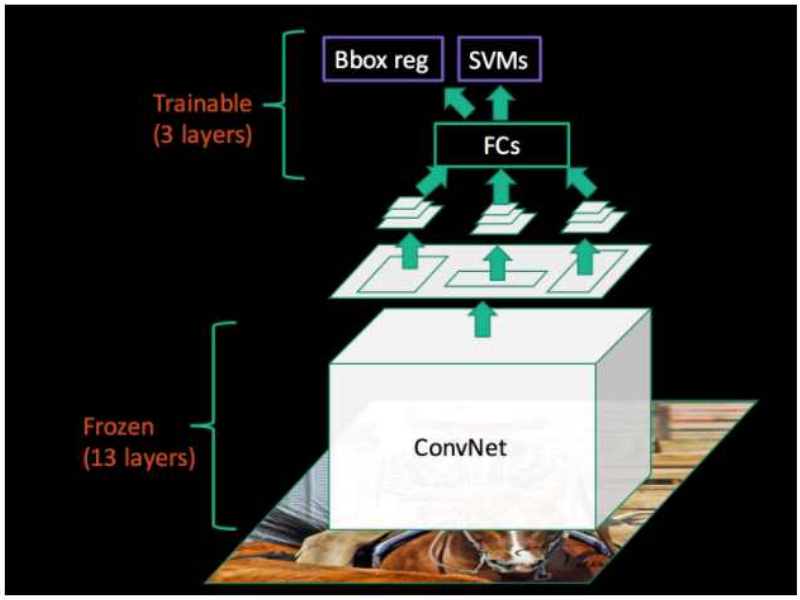
[note]
- 대게 ROI로는 사각형을 쓴다. 물론 다른 모양도 있다.
- 여기서 Region proposal 방법을 학습하지는 않는다. 고정적인 selevtive search를 쓴다.
- training data 에는 이미지 모든 객체에 대한 bbox 가 있어야 한다. train 시에 fully supervision이 있다고 가정

- Ad hoc training objectives
  - Fine-tune network with softmax classifier (log loss)
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressions (least squares)
- Training is slow (84h), takes a lot of disk space
- Inference (detection) is slow
  - 47s / image with VGG16 [Simonyan & Zisserman. ICLR15]
  - Fixed by SPP-net [He et al. ECCV14]



[R-CNN의 단점]
- R-CNN 여전히 계산비용이 높다. R-CNN 은 2000개의 Region Proposal 각각이 독립적으로 CNN 입력으로 들어가기 때문이다.
- 그리고 학습이 되지 않는 Region proposal 은 문제가 될 수 있다.
- 학습과정 자체가 매우 오래걸린다. (논문에 84시간 걸렸다고 한다.)
- Test time 도 이미지 한 장당 30초가 걸린다고 한다.

[SPPnet]

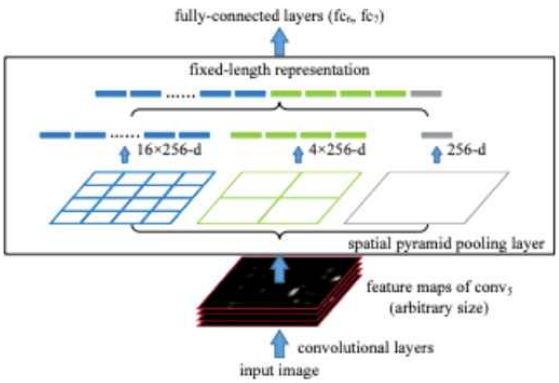


- R CNN에서 가장 크게 나타는 속도 저하의 원인인 region proposal마다의 CNN feature map 생성을 보완함

- 이를 통해 학습시 3배, 실제 사용 시 10~100배의 속도 개선을 이뤄냄

-> Region Proposal에 CNN을 바로 적용하는 것이 아닌, **이미지에 우선 CNN을 적용하여 생성한 Feature map을 Reigon Proposal에 사용함**

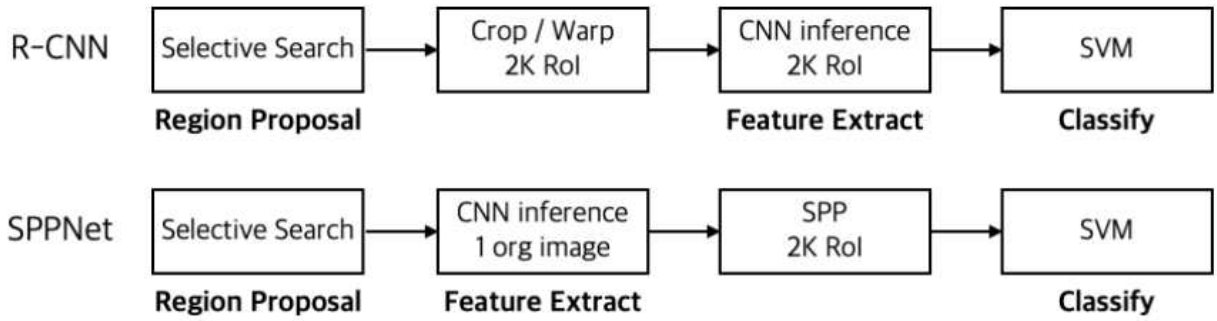
- SPPnet(Spatial Pyramid Pooling) : Spatial Pyramid Pooling이라는 특징적 구조를 활용하여 임의의 사이즈를 가진 이미지를 모두 활용할 수 있게 함.
- SPP layer: 이미지의 사이즈에 관계없이 특징을 잘 반영할 수 있도록 여러 크기의 bin을 만들고 그것을 활용하는 구조



Spatial Pyramid Pooling Structure

위의 4*4, 2*2, 1*1, 세 가지의 영역이 하나의 '피라미드'
bin: 피라미드의 한 칸을 의미함 (ex: 64*64*256의 input이 들어온다면 4*4 피라미드의 bin 크기는 16*16)

각 bin에서 maxpooling을 수행하고 그 결과를 이어 붙여줌. 즉 입력 피쳐맵의 채널크기를 k, bin의 개수를 M이라 하면 SPP의 최종 output은 k*M 차원의 벡터가 된다. 즉 미리 설정한 bin의 개수와 CNN의 채널 값으로 output의 크기가 고정됨!



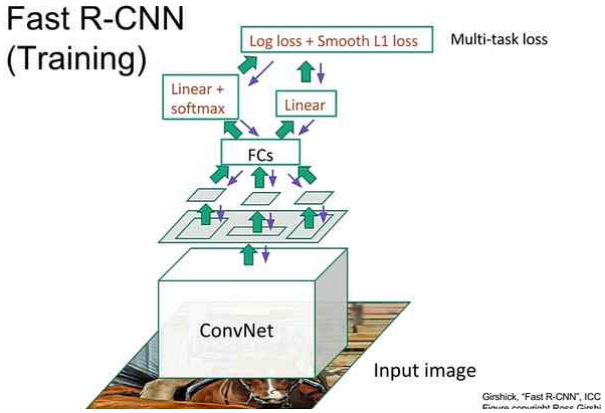
- 한계점

1. R CNN과 동일한 학습 파이프라인을 가져 multi-stage로 학습이 진행되어 저장 공간을 요구하고 학습의 속도는 개선이 어려움

2. CNN의 Parameter가 학습되지 못하기 때문에 Task에 맞는 fine-tuning이 어려움

3. 여전히 최종 classification은 SVM, Region Proposal은 Selective Search를 활용함

[Fast-R-CNN]



[작동 과정]

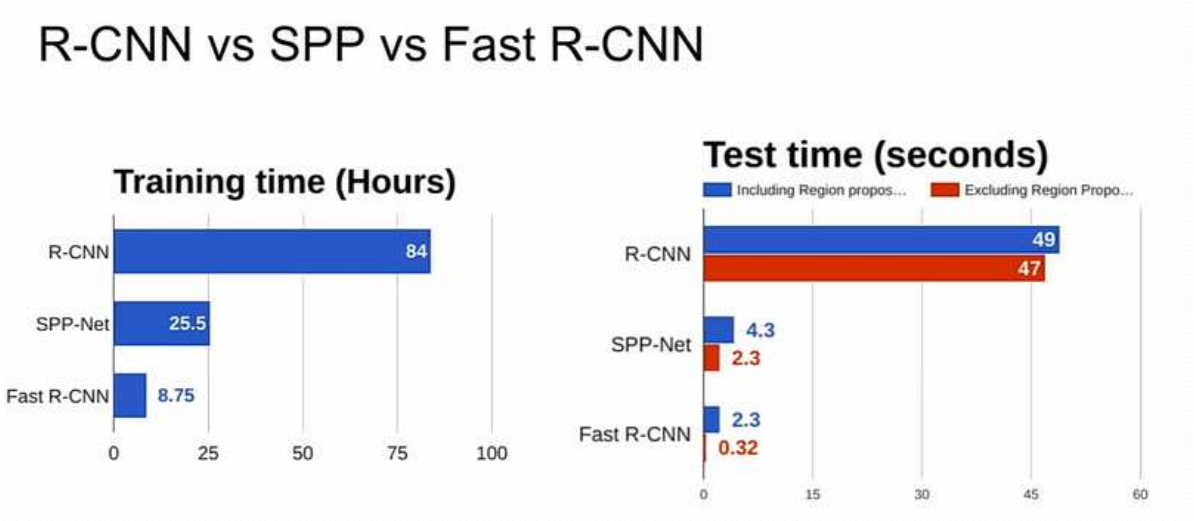
0. 입력은 전체 이미지와 object proposal 두 가지

1. 전체 이미지에 대해 CNN을 수행하여 전체 이미지에 대한 Feature map을 얻는다.

2. CNN Feature map에서 ROI를 뽑아낸다.
- 즉 CNN Feature를 여러 ROI가 공유한다.
- 뽑아내는 방식은 여전히 Selective search의 방법으로 Region proposal을 계산한다.

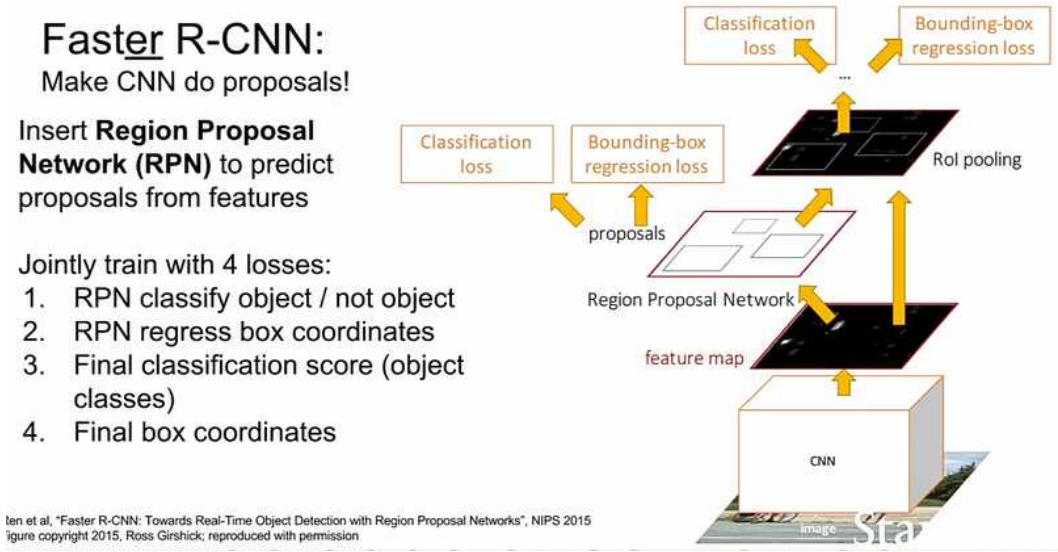
3. CNN의 Feature map에서 가져온 ROI를 fc layer의 입력에 맞게 크기를 바꾸어준다.
- fc-layer 는 고정된 크기를 받아야 하기 때문.

4. 크기 조절한 ROI를 fc layer의 입력으로 넣어 classification score, linear regression offset을 계산
- 학습 시에는 두 loss를 합쳐 multi task loss로 Back prop를 진행.



- Train time은 Featrue map을 공유하기 때문에, Fast R-CNN이 매우 빠르다.
- Region proposal을 계산한 이후 CNN을 거치는 과정도 모든 Region proposal이 공유하기 때문에 빠르다.
- 결국 Region proposal을 계산하는 시간이 대부분이 된다.(2000개의 region proposal을 selective 로 계산하는데 2초 걸린다. 즉, 이 구간이 병목구간!)

[Faster R-CNN]



[작동 과정]

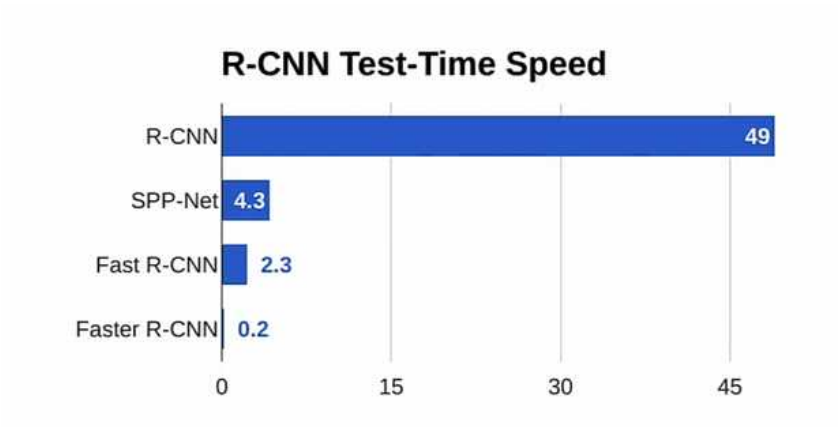
1. 입력 이미지 전체가 CNN 네트워크로 들어가 feature map을 형성한다.

2. Faster R-CNN 은 별도의 Region proposal network 가 있다. 이 RPN 네트워크가 Feature map을 가지고 Region proposal을 계산한다.

3. 여기서 얻은 ROI에서 loss를 얻고, multi task loss를 이용해 여러 가지 loss를 한번에 계산
• loss는 총 4가지 이다. 이들의 균형을 맞추는건 어렵다

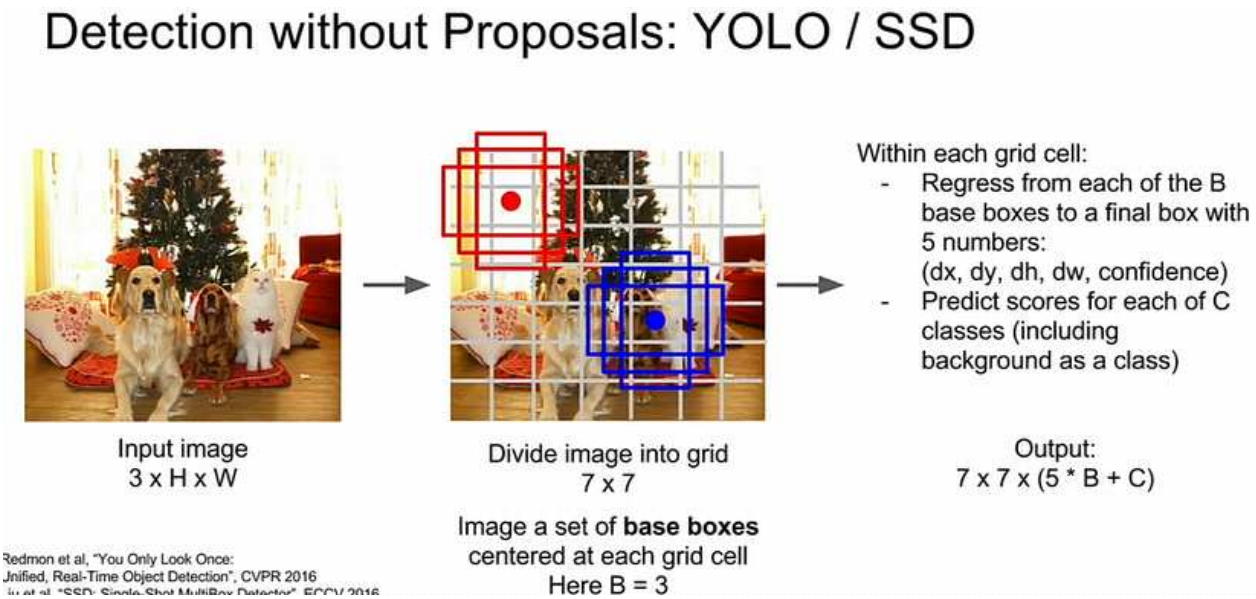
[Loss]

- RPN 에 관한 loss1 : 이곳에 객체가 있는지 없는지에 대한 classification loss
- RPN 에 관한 loss2 : 예측한 BBOX regression 에 관한 loss
- 마지막 부분의 loss1 : classsification loss
- 마지막 부분의 loss2 : BBox Regression에 관한 loss



- Faster R CNN 은 네트워크 밖에서 계산되던 region proposal의 병목을 없애서 엄청 빨라졌다.
- R-CNN 계열 네트워크는 각 후보 ROI마다 독립적으로 연산을 수행했다.
- R-CNN 계열 네트워크를 Region based method 이라고 한다.
- 다른 방법도 있는데 feed forward를 오로지 한 방향으로 수행하는 네트워크이다. (아래에서 알아보자)

[Yolo(You only look once) / SSD(Single shot detection)]



[아이디어]

- 각 task를 따로 계산하지 말고 regression 문제로 풀어보자는 아이디어
- 거대한 cnn을 통과하면 모든 것을 담은 예측값이 한번에 나온다.
- 위 방법들을 Single Shot methods 라 한다.

[작동 과정]

1. 입력 이미지가 있으면 이미지를 크게 나눈다.(ex 7*7 grid)
- 각 grid cell 내부에는 base bbox 존재, 위 예시의 경우 3가지(길쭉 /정사각형 / 넙죽)
- 이 각 grid cell 에 대해서 bboxes 가 있고 이를 기반으로 예측 수행

2. bbox 의 offset을 예측하고,각 bbox 에 대해서 classification scores를 계산
- offset : 실제 위치가 되려면 base box를 얼만큼 옮겨야하는지.
- classification scores : bbox 안에 이 카테고리에 속한 객체가 속할 가능성.

3. 이에 따라 네트워크에 입력 이미지가 들어오면 7*7 grid 마다 (5B+C)개의 tensor를 가진다.
- B는 base bbox 의 offset(dx,dy,dh,dw) 와 confidence score 로 구성된다.
- C는 C 개의 카테고리에 대한 Classification score 이다.

4. 3 dim tensor 출력을(ouput = 7*7*(5*B+C)이용해 이를 CNN으로 한번에 학습한다.

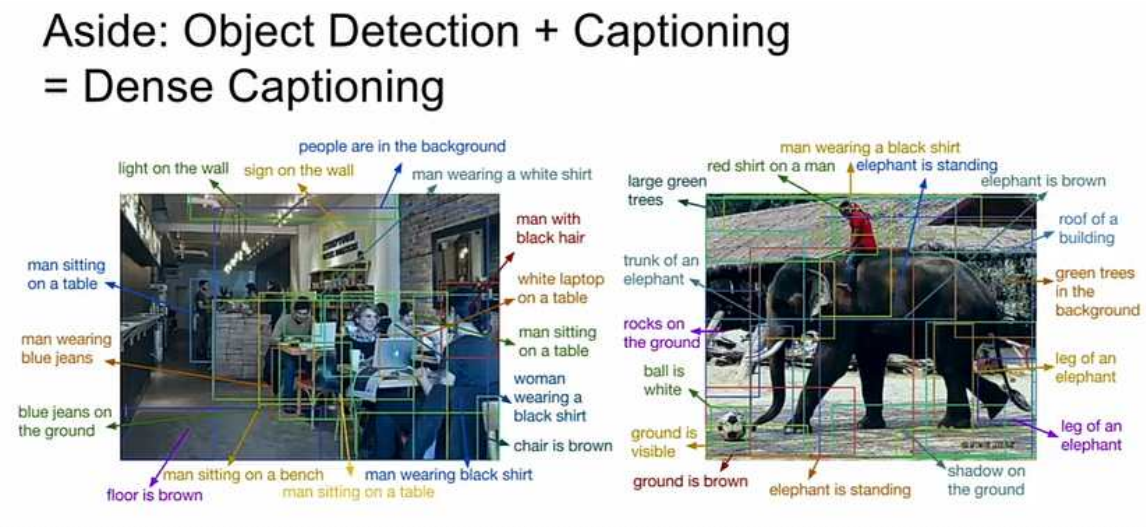
[기존 방식과 비교]

- Faster R-CNN : RPN 으로 먼저 Regression 문제를 풀고 ROI 단위로 classification
- singshot 방법은 한번에 forward pass 만으로 끝나버린다.

[Object dection 의 다양한 요소들]

- base network
- 다양한 architecture
- Hyperparamter(이미지 사이즈 …)
- Faster R-CNN 계열이 정확도는 높으나 느리다
- Single shot method는 빠르나 정확도가 약간 떨어진다.

[Dense Captioning]

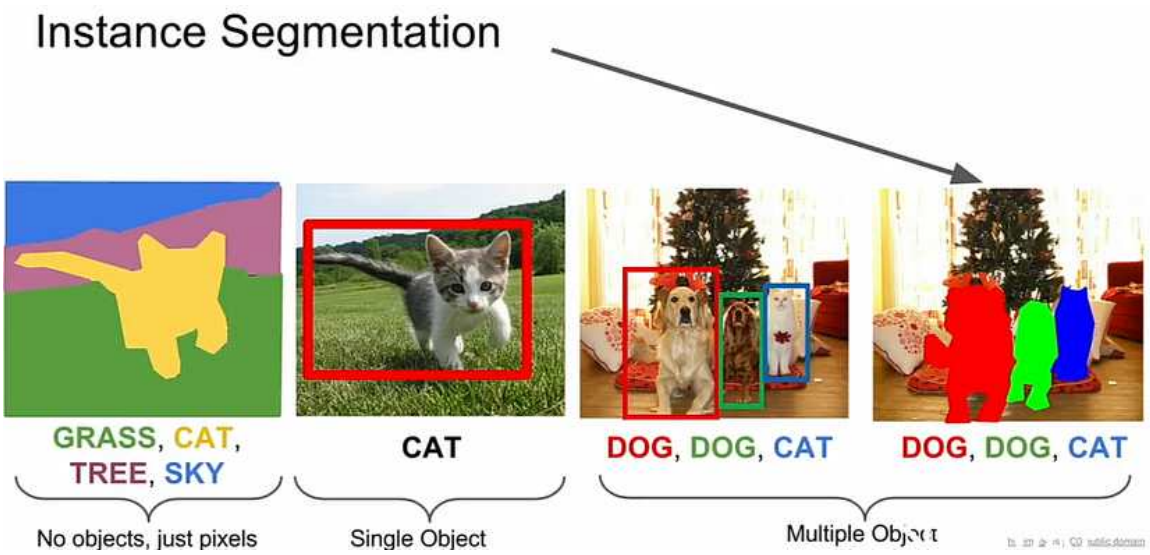


- 각 Region에 대해서 카테고리를 예측하는 것이 아니라 각 Region의 Caption을 예측해야 한다.
- 이 모델을 end to end로 학습시켜 모든 걸 동시에 예측하게 하였다.
- 이 때, Region에 caption 이 있는 데이터 셋이 있어야 한다.
- Faster R CNN과 방식이 유사. 네트워크에는 Region proposal stage가 있다.
- Caption을 예측해야 하므로 RNN language model을 도입

- 위를 보여준 이유는 문제들을 서로 엮어서 생각할 수 있다는 것.
- 위 예시처럼 새로운 문제가 있을 때, object detection, image captioning 들의 요소를 합쳐서 재활용 할 수 있다는 것이다. 합친 네트워크를 End To End 로 학습시킴으로써 문제를 해결 가능할 것이다.

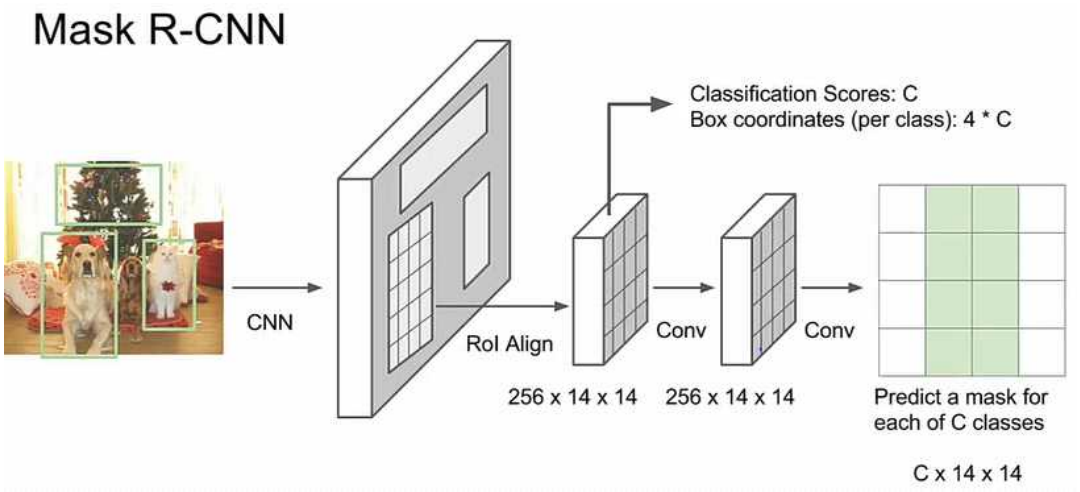
[Image segmentation]

• Object detection 과 비슷하지만, 객체별로 bbox를 예측하는 것이 아니라 객체별 segmentation mask를 예측하는 것이다. (각 객체에 해당하는 픽셀)



• 위의 경우 개 고양이들을 각각 구분해야 하고 각 픽셀이 어떤 객체에 속하는지를 모두 결정해주어야 한다.

[Mask R-CNN]



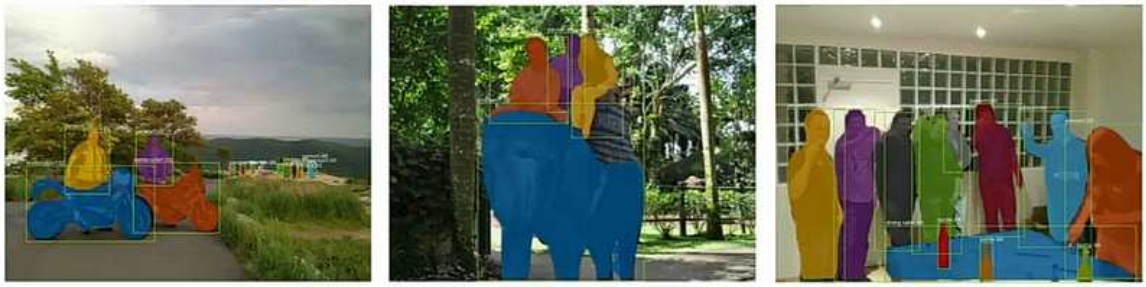
1. 처음 입력 이미지가 CNN 을 거친다.

2. Feature map 에서 RPN 이 ROI를 뽑아낸뒤 ROI pooling을 수행한다. 그 이후 두 갈래로 나뉜다.

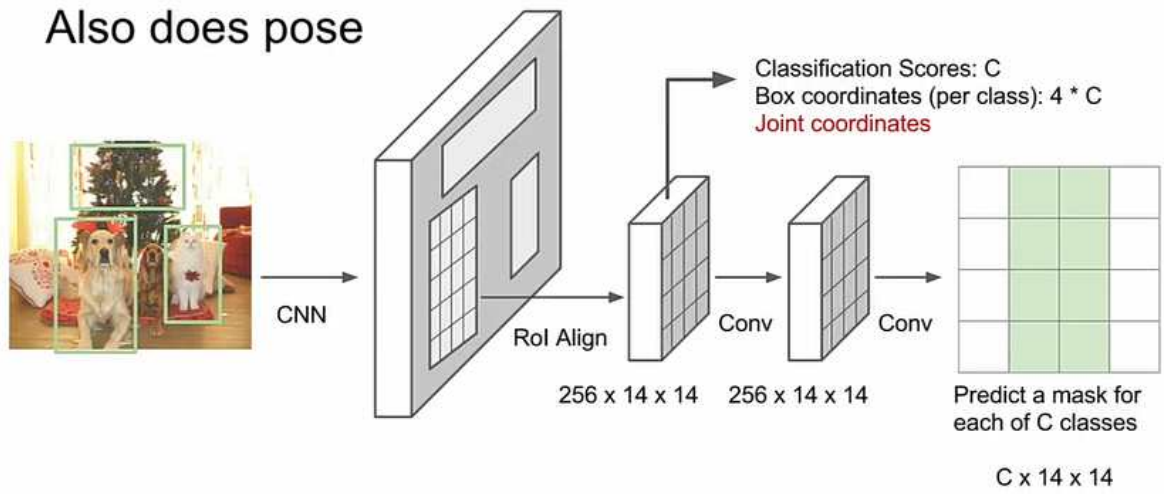
2-1. 상단의 갈래는 faster R-cnn 과 유사하다. 각 region proposal이 어떤 카테고리에 속하는지 예측하고, region proposal의 좌표를 보정해주는 bbox regression 도 예측한다.

2-2. 하단의 갈래는 각 픽셀마다 객체인지 아닌지를 분류한다.

※ Mask R CNN 은 오늘 배운 모든 방법을 통합했다고 볼 수 있다.



- 예시를 보면 매우 잘 작동하고 있음을 알 수 있다.
- Mask는 pose estimation도 가능하다.



- Region proposal에 갈래 하나를 추가해서 region proposal 관절에 해당하는 좌표를 예측하게 하면 된다.
- loss와 layer를 하나 더 추가하는 셈이다. 이에 따라 multi task loss에 loss가 추가된다.