

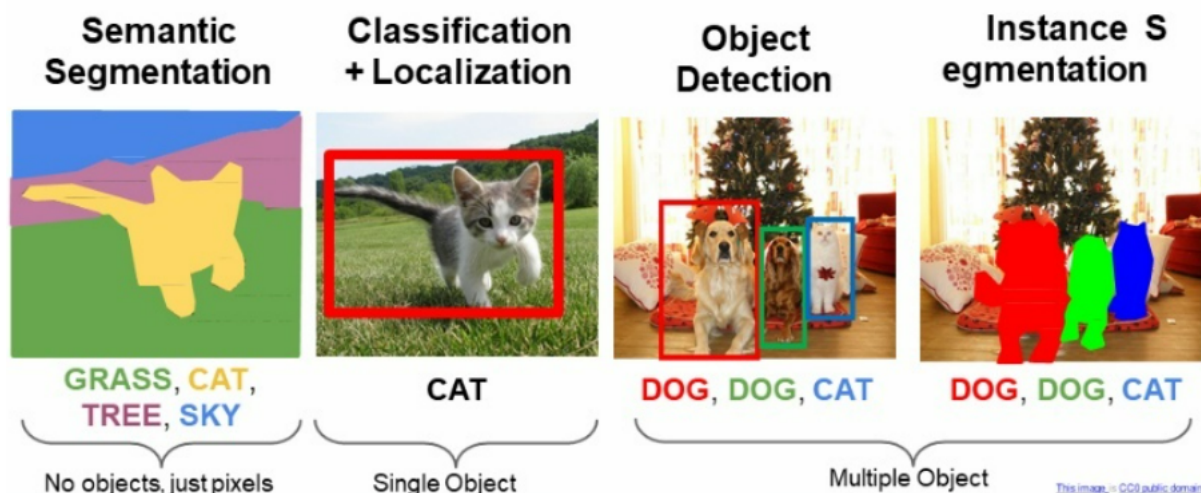
Week13 lec 11Detection and Segmentation

So far: Image Classification

이전 강의에서는 image classification 문제를 주로 다뤘다면 이제부터는 다양한 Computer Vision Task들(Segmentation, Localization, Detection)들을 다룰 것이다.

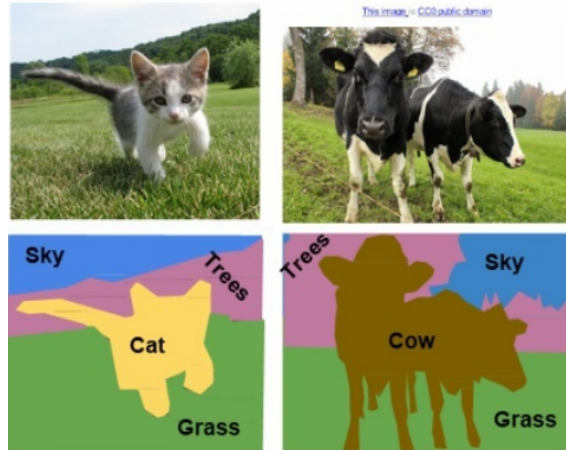
앞선 Image Classification에서는 image가 들어오면 DNN을 통과하여 feature vector가 나오고 (imageNet의 경우 1000 dimension vector인 score) 카테고리가 출력된다.

Other Computer Vision Tasks



이번 강의에서는 Semantic segmentation을 배우고 classification + localization과 object detection, instance segmentation에 대해서 배운다.

Semantic Segmentation



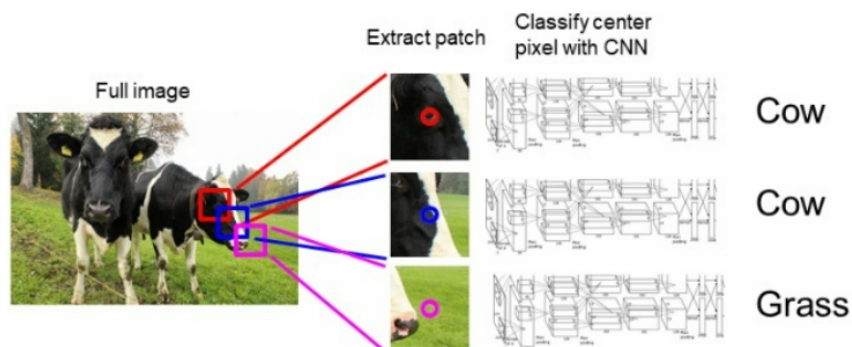
Semantic Segmentation에서는 이미지의 각 pixel들을 카테고리 라벨링한다.

위의 고양이 그림에서는 각 픽셀은 (Sky, Cat, Grass, Trees) 네 카테고리로 라벨링된다. ⇒ 일종의 classification. (단 이미지 전체가 아닌 pixel 기준)

이때 instances들은 절대 구별짓지 않고, 오직 pixel에만 주목한다.

단점) 오른쪽 소 그림을 보면 2마리의 소가 있지만 각각 표시하는 것이 아닌 둘을 합쳐서 cow로 라벨링함 ⇒ semantic segmentation의 문제점

Semantic Segmentation Idea : Sliding Window



위 문제를 해결하기 위해선 **sliding window**를 사용할 수 있다.

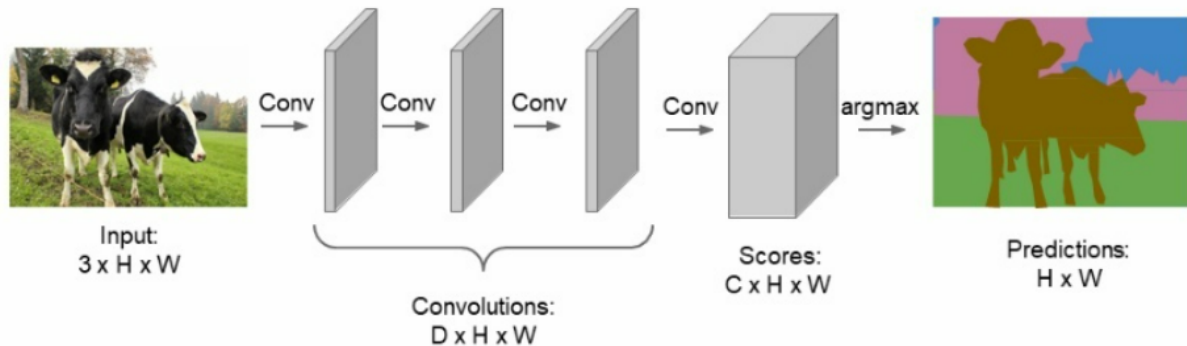
입력 이미지를 작은 단위로 쪼개고, 이 단위 부분이 어떤 분류에 속하는지 classification한다. (해당 그림에서는 머리 부분을 영역 세 개로 추출해서 classification으로 해당 영역이 어떤 카테고리에 속하는지를 결정했다.)

Problem) Very inefficient

작은 영역으로 쪼개고 모든 영역을 forward/backward 하는 것은 비효율적이다.

겹쳐지는 영역에 대해 공통된 특징을 재사용하지 못한다.

Semantic Segmentation Idea : Fully Convolutional



위 문제점을 해결한 것이 Fully Convolutional 방법이다.

FC layer가 없고 Convolution layer로만 구성되어 있다.

3x3 zero padding을 사용하는 Conv layer들을 쌓아올리면 이미지의 공간 정보는 손실되지 않는다.

⇒ 출력 : $C \times H \times W$ (C : 카테고리의 수)

출력은 모든 픽셀 값에 대해 classification score를 매긴 값이다.

⇒ 학습방법

: 모든 픽셀의 classification loss(Cross entropy)를 계산하고 평균값을 취한 후 back prop

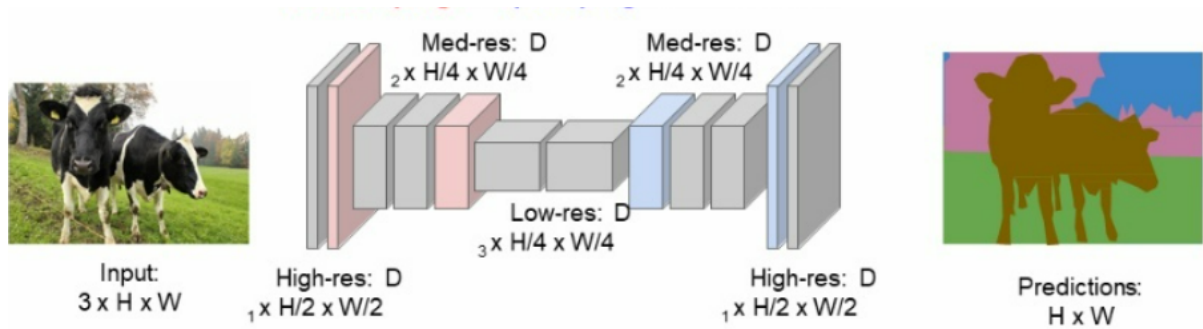
⇒ Training data 만드는 법

: 비용이 큰 작업, 모든 픽셀을 레이블링, 카테고리의 수가 고정되어 있어야함

Problem

: 입력 이미지의 Spatial Size를 유지시켜야 한다. 비용이 매우 크다.

Fully Convolution : Down sampling and Upsampling

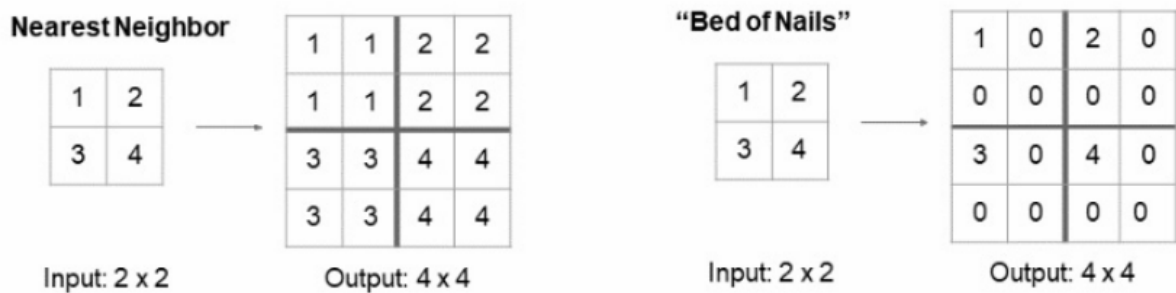


처음에는 maxpooling, stride 를 이용하여 downsampling

기존에 fc layer가 있던 곳에서 upsampling 진행 \Rightarrow spatial resolution 키워 입력 해상도와 일치하게

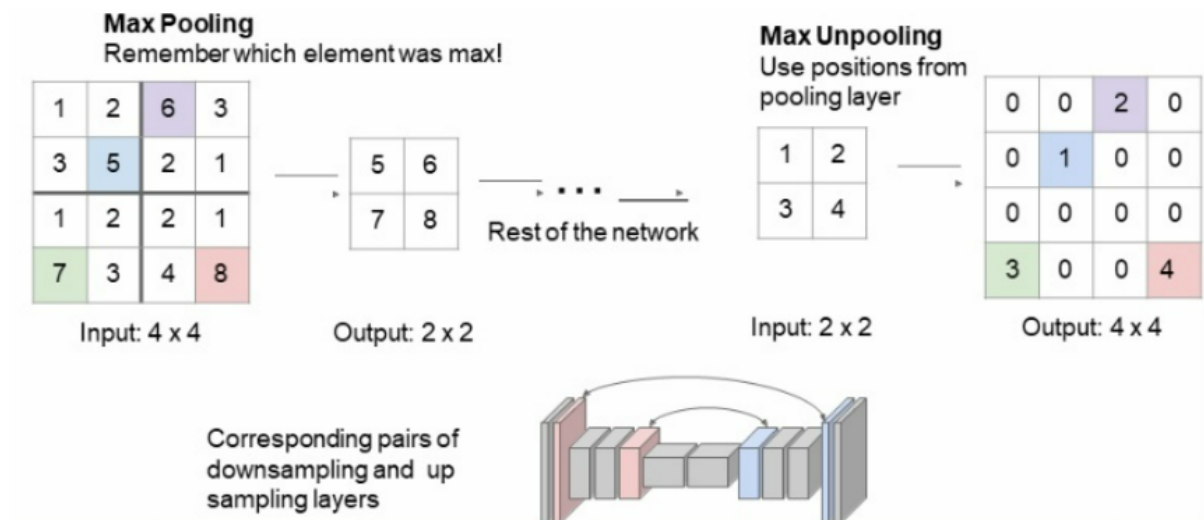
Upsampling을 하는 방법을 알아보자

In-Network upsampling : “Unpooling”



- Nearest Neighbor : 인접한 방법으로 키우기
- Bed of Nails : 1번째 위치에 input값을 넣고 나머지는 0으로 padding

In-Network upsampling : “Max Unpooling”

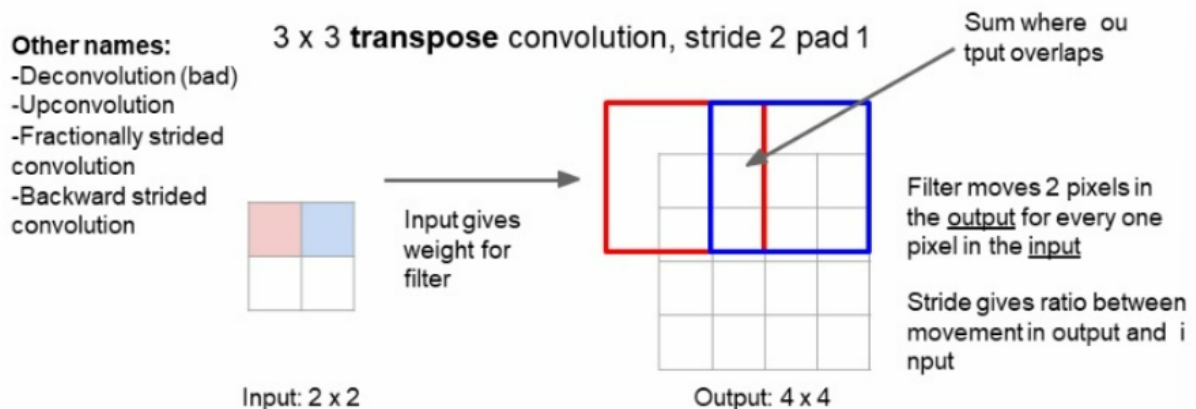


- Max Unpooling : Max pooling을 한 위치를 기억해두고 그 선택된 자리에 값을 넣고 나머지는 0으로 채운다. ⇒ maxpooling때 잃었던 공간 정보를 좀 더 균형있게 유지할 수 있는 방법.

Learnable upsampling : Transpose Convolution

- Transpose Convolution

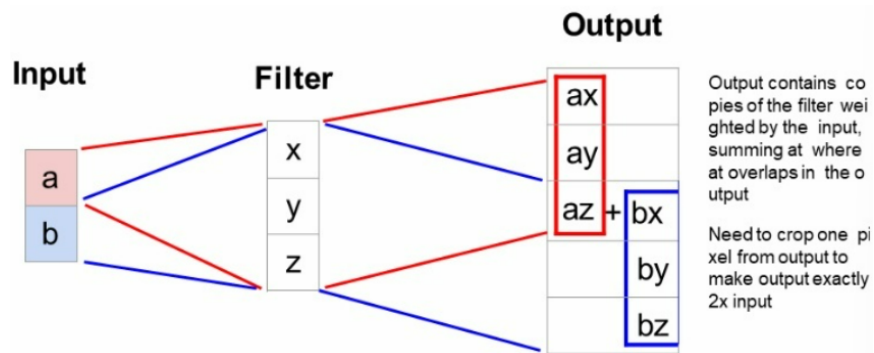
Strided convolution의 경우 어떤 식으로 downsampling을 할지 네트워크가 학습을 할 수 있는데 이와 유사하게 upsampling에서도 학습을 할 수 있다.



특징맵에서 분홍색 값을 선택해서 3x3 filter와 곱하고 그 출력의 3x3영역에 그 값을 넣는다.
⇒ 필터와 입력의 내적을 계산하는 것이 아니라 입력 값이 필터에 곱해지는 “가중치” 역할을 한다.

Stride = 2 이기 때문에 두칸 움직여서 같은 행위 반복, 겹치는 부분은 단순히 더해준다.

Transpose Convolution : 1D Example

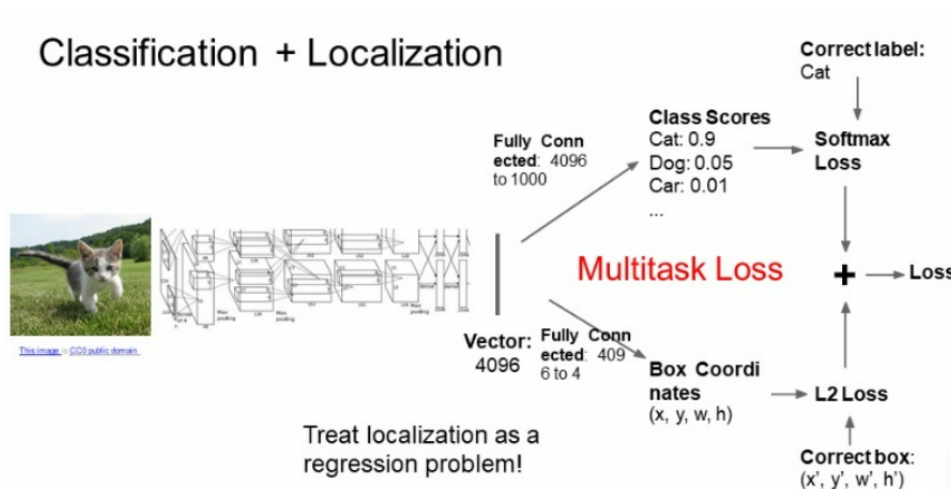


다른 방법으로 도식화한 것, 겹쳐지는 부분을 단순히 더하는 것을 확인할 수 있다.

Classification + Localization

이미지가 어떤 카테고리에 속하는지 + 실제 객체가 어디에 위치하는지 boxing

(Object detection과 다른 것은 객체가 오직 하나 뿐이라 가정)



총 2개의 fc layer가 있는데 하나는 class score를, 다른 하나는 box coordinate(bounding box의 좌표)를 한다. (box coordinate의 경우 width, height, x, y 값을 가지고 있다.)

⇒ 두 개의 loss가 존재한다.

Supervised learning : 학습 이미지는 카테고리 레이블과 해당 객체의 bounding box ground truth (GT) 를 동시에 가지고 있어야 한다.

각각 Softmax loss와 L2 loss를 사용한다. ⇒ multitask loss

Gradient를 구하기 위해서는 두 loss모두를 최소화시켜야한다. ⇒ hyperparameter조절이 까다롭다.

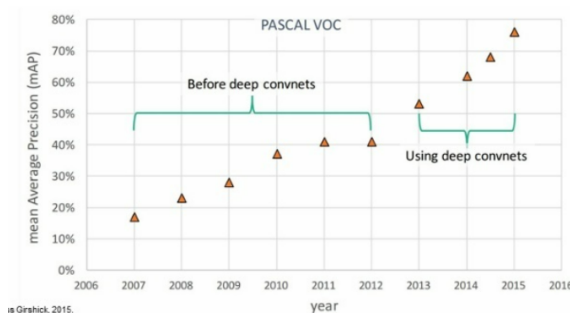
Pretrained 된 모델을 사용하면 더 좋고 빠르다.

Object Detection

고정된 카테고리 존재,

이미지가 주어진다면 이미지가 나타나는 객체들의 Bbox와 해당하는 카테고리를 출력한다.

⇒ 예측해야하는 Bbox의 수가 입력 이미지에 따라 달라진다.



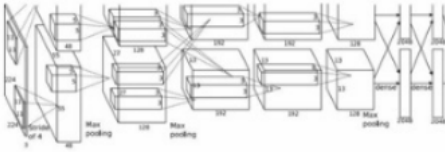

딥러닝을 사용하고 나서 성능이 급격히 좋아지는 것을 확인할 수 있다.

Object Detection as Regression?

⇒ 앞선 localization 과 다르게 Detection은 이미지가 몇 개인지 예측을 할 수 없기 때문에 regression을 사용하기 어렵다.

Object Detection as Classification : Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO
Cat? NO
Background? YES

Sliding Window를 사용해서 어떤 카테고리에 속하는지를 묻는다.

여기서 background는 어느 카테고리에도 속하지 않는 경우이다.

Problem) 어떻게 영역을 추출할지와 object가 몇개인지, 어디에 존재하는지를 몰라 크기를 조절할 수 없다. 중형비를 갖춘 것을 표현해야 할 수 도 있다.

Region Proposals

Sliding Window대신 region proposal을 사용한다. (딥러닝 아님)

객체가 있을 법한 후보를 찾아내는 것.

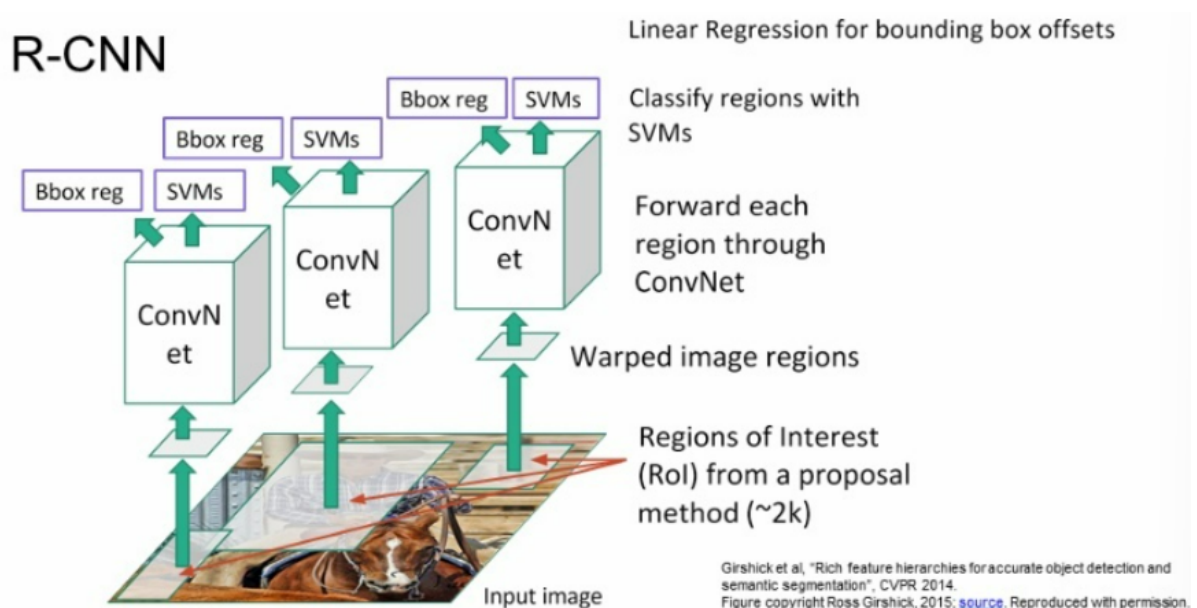
1000개의 region 을 selective search를 통해 찾아내고 이미지에 객체가 존재한다면 selective search의 region proposal안에 속할 가능성이 높다.

⇒ 있을법한 region proposal을 얻어내고 이 region proposal를 CNN의 입력으로 해서 추출한다.

R-CNN

위의 아이디어가 R-CNN이다.

이미지가 주어진다면 region proposal을 얻기 위해 region proposal network(RPN)를 수행한다.



Selection search를 통해서 2K의 ROI(region proposal of interest)를 얻어낸다.

추출된 ROI로 CNN classification을 수행하기 위해 고정된 사이즈로 맞춰준다.

(warped image region)

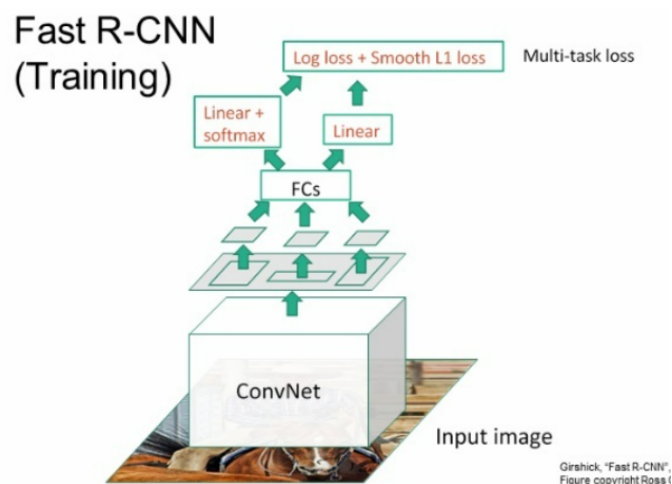
CNN을 통과시킨다.

최종 분류에 SVM을 사용한다.

Bbox reg는 regression(Selection search의 region proposal이 정확하지 않을 수 있기 때문에 보정)

Problem) 계산비용이 많고 느리다. 학습되지 않은 region proposal 때문에 문제가 될 가능성이 있다.

Fast R-CNN



R-CNN의 단점들을 보완시켰다.

R-CNN과 다르게 각 ROI마다 CNN을 수행하지 않고 전체 이미지에 대해 CNN을 수행한다.

⇒ 전체 이미지에 대한 고해상도 feature map을 얻을 수 있다.

Selective search 방법으로 region proposal 을 계산하는데 ROI를 뜯어내지 않고 CNN에서 나온 feature map 에다가 ROI를 뜯어낸다.

⇒ 하나의 CNN에서 나온 feature를 여러 ROI 가 공유할 수 있게 된다.

FC-layer 에서 ROI의 크기가 조정된다. ⇒ ROI Pooling layer

⇒ classification score와 linear regression의 값을 얻을 수 있다.

⇒ backpropagation은 multi-task loss로 계산된다.

Problem) Region Proposal을 계산하는데 시간이 오래 걸린다.

⇒ bottle neck

Faster R-CNN

