Lecture 10

저번시간 리뷰(CNN 연구):

VGGNet: 16계층, 19계층 모델

GoogleNet: 22계층 모델

(배치 정규화 발명 전. 그래서 참 대단하다)

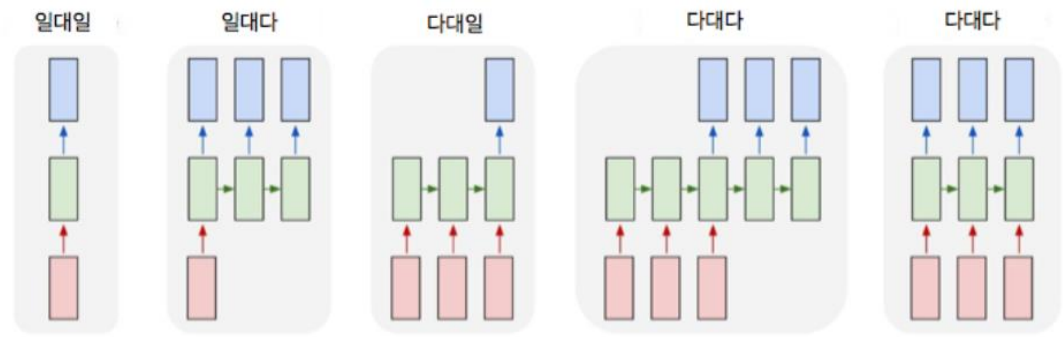
ResNet: 중복 블록(Residual block)갖는 모델.

DenseNet, FractalNet: 모델 내에서 더하기, 항등연결(identity connections)더하기.

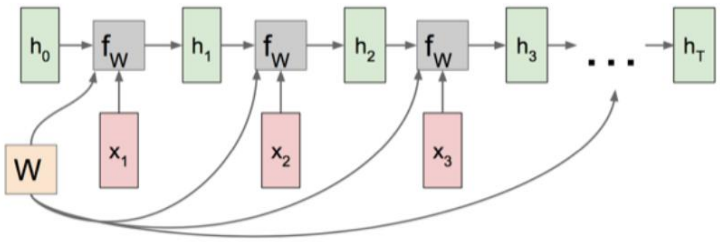
#VGGNet, AlexNet은 FC에 매우 많은 수의 파라미터 갖고 있다. FC계층 줄이고, 파라미터 수 줄이는게 더 좋다.(GoogleNet 이후 연구.)

순환 신경망(Recurrent Neural Networks):

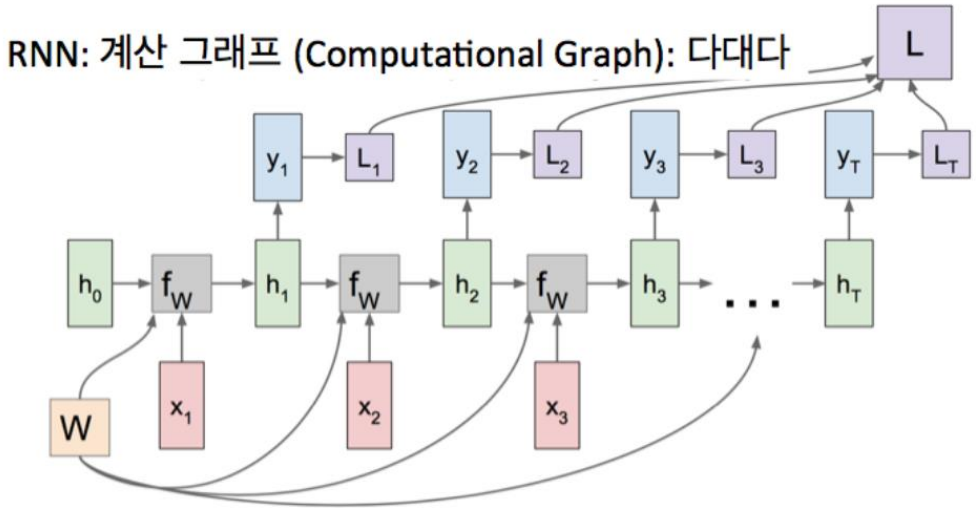
RNN은 이미지의 여기저기 둘러보고 어떤 종류(숫자)인지 결정.




->RNN처리 모식도.

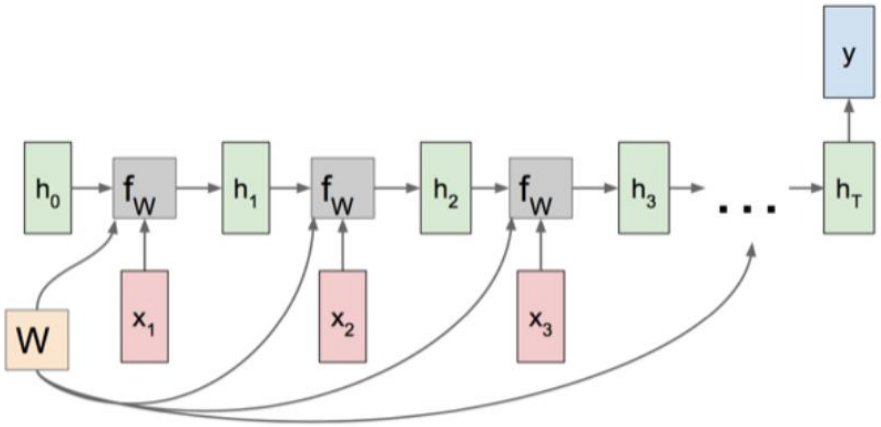



h_0에서 들어오면, x_1, f_w값을 거쳐서 새로운 h_1생성. 그렇게 h_t까지 가는데, W값은 계속 중복된다.

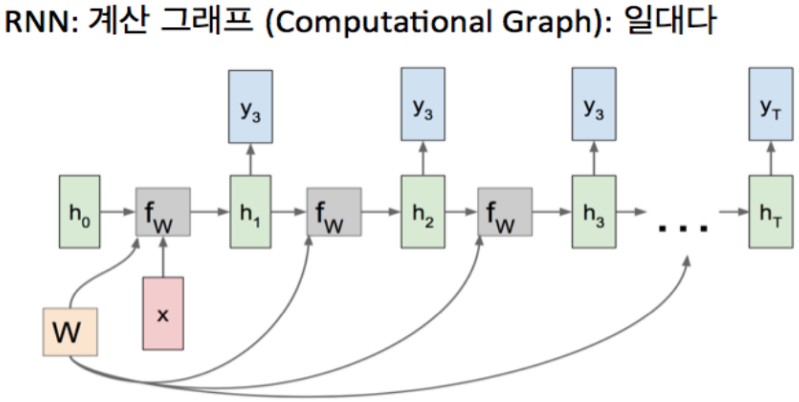



->다대다 과정.

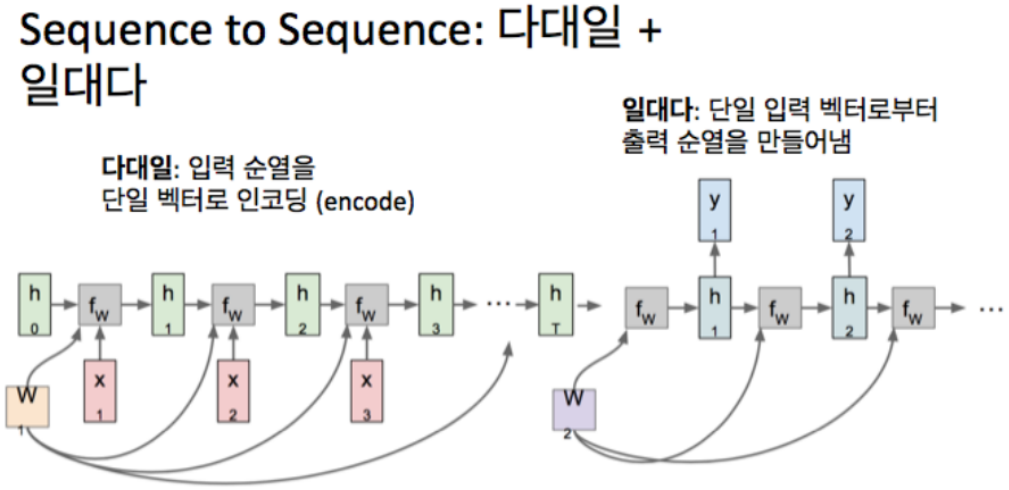
똑같이 위와같이 계산 하고, 대신 중간중간 y_i값이 하나씩 나옴. 전체 손실값(L도 다 더해서 계산)




->다대일 과정. 끝에 y값 하나로 뱉기.




->일대다 과정. x값(입력값?)하나로 들어오고, y값은 여러 개(출력값?)

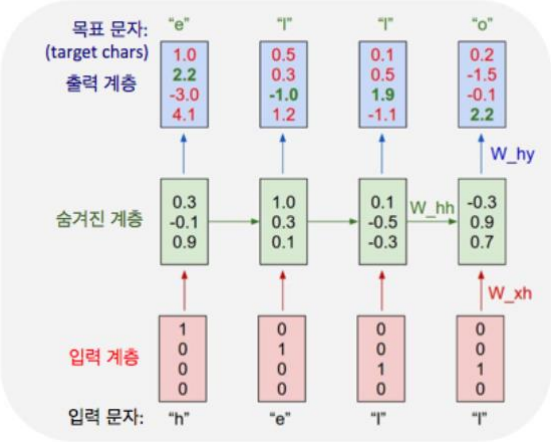



->다대일 과정 거치면서 나온 최종 h값을, 일대다 과정으로

#다대일 거치면서 나온 요약 문을, 일대다 거치면서(unrolling)되는 느낌.

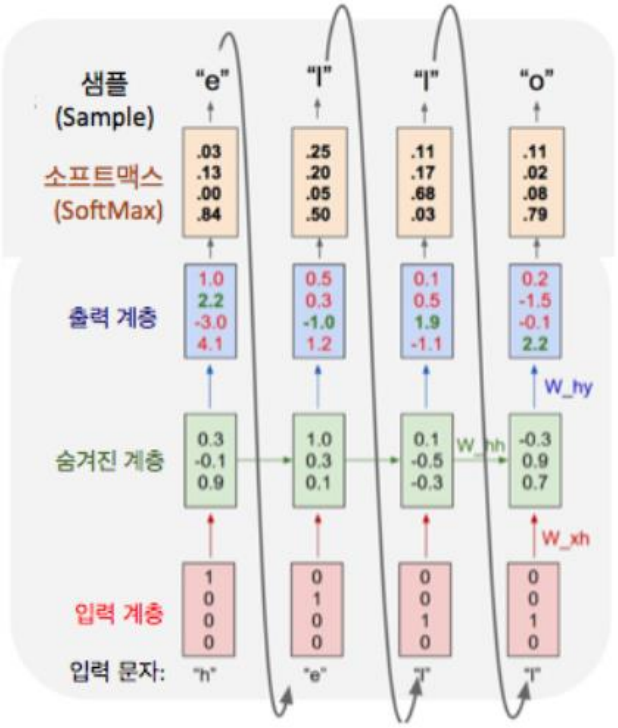
RNN 사용 예시:

언어 모델링.

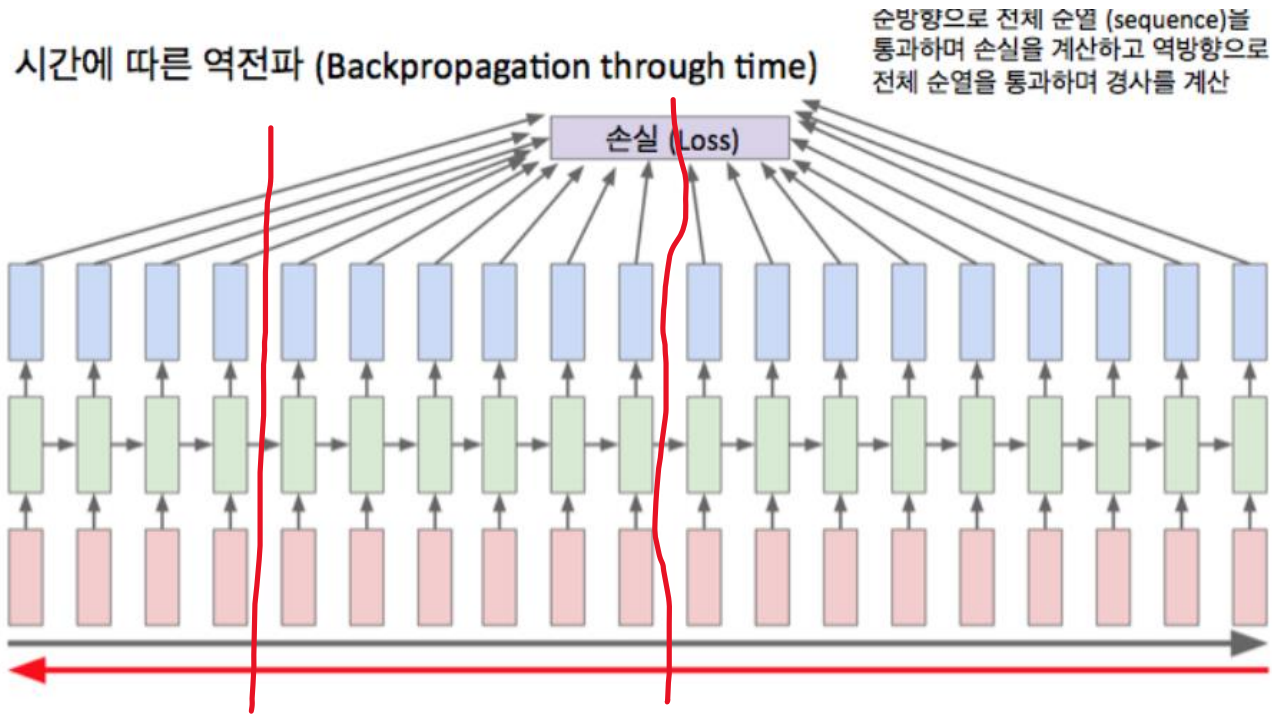



->이건 잘못된 모델.

흐름은 RNN써서 훈련된 것과 유사해 보이는 텍스트 합성.

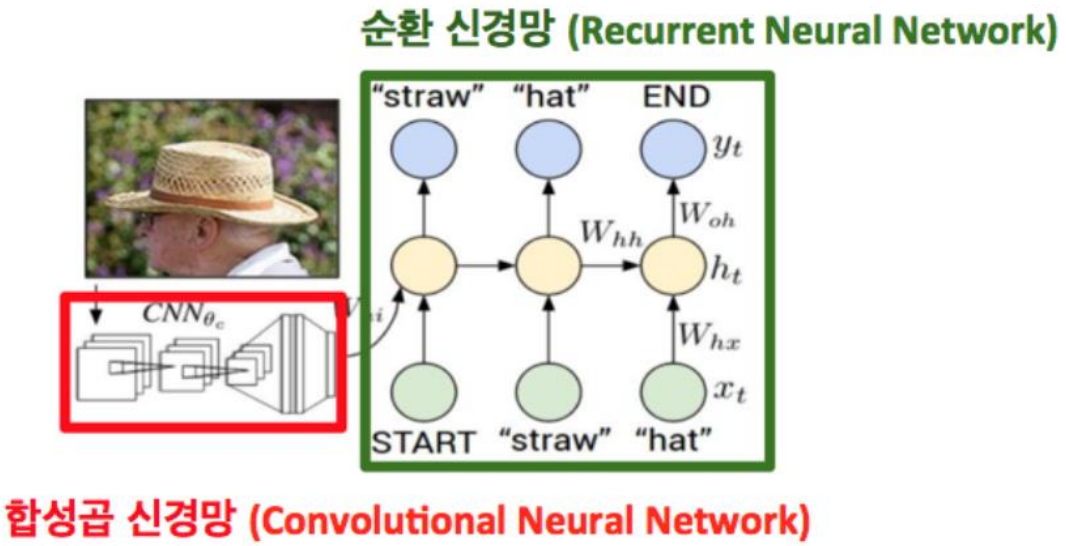



-> 이런식으로 한번에 한글자씩 순열 합성.

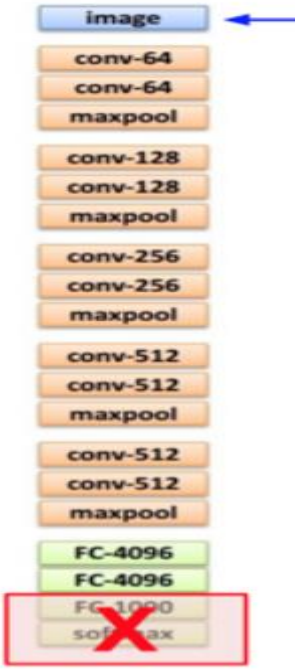



순방향(검은 화살표)으로 손실 계산, 역방향(빨간 화살표)로 경사 계산.

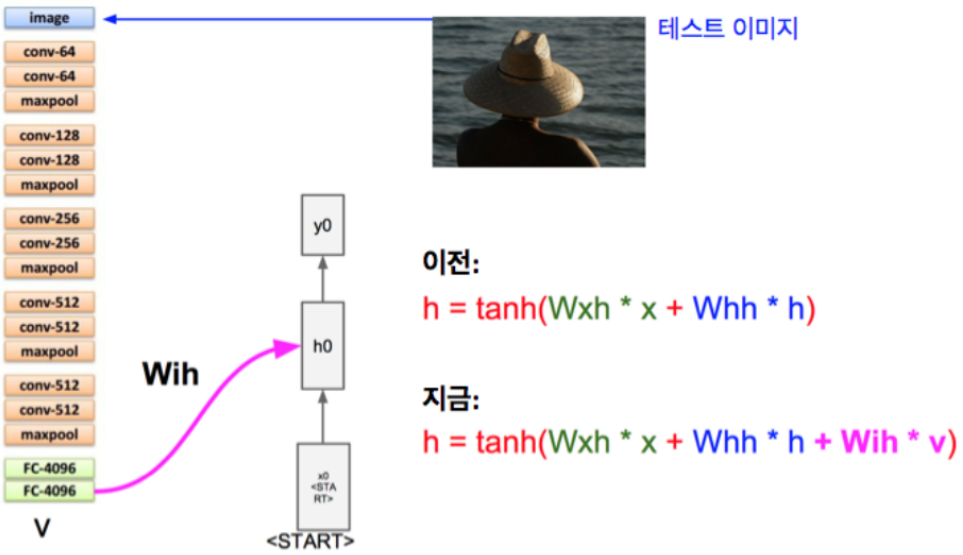
짤라서 계산도 가능.(전체 다 한번에 계산하면, 메모리 터져서)



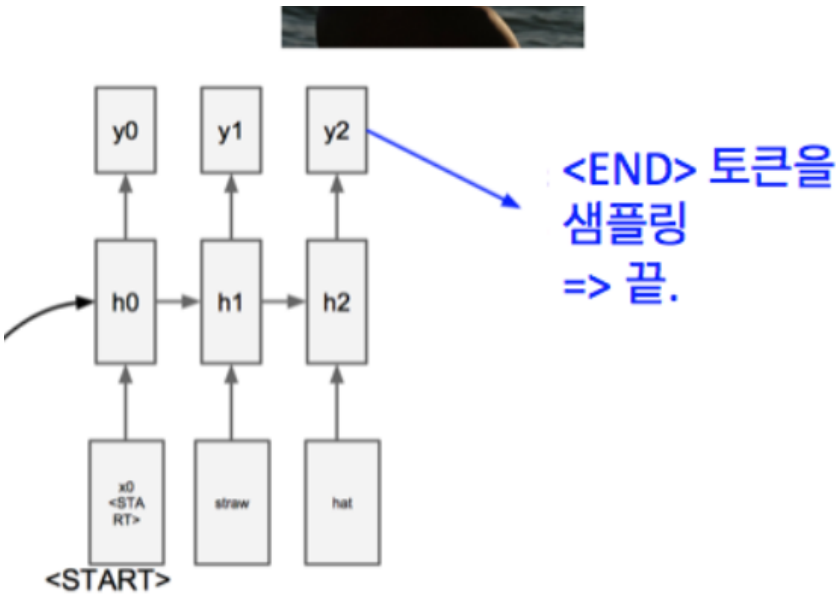

#입력 이미지 -> CNN을 거치고 나온 것 -> RNN에 넣기 -> 이미지 관련된 텍스트.




다만, CNN에 넣는 것은 똑같은데, 끝에 Softmax 점수를 얻는 게 아니라, 그 전 단계 FC계층까지만 거치고, 전체 문맥 요약 후, RNN에 넣기.



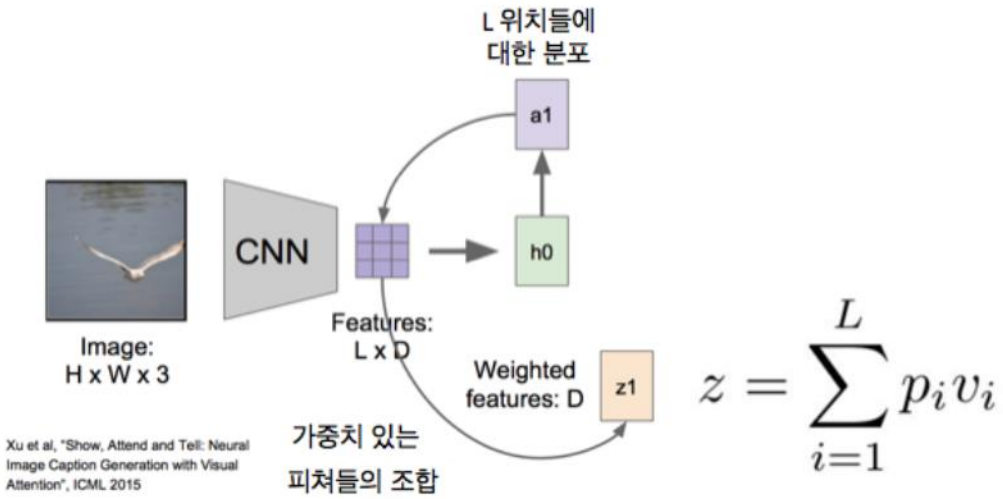

CNN계층을 다 거치고 난 후에, V값으로 나왔다.




그러고 이렇게 RNN계층을 거친다.

잘못된 결과를 출력하는 경우가 더러 있어서, 그래서 만들어진게, Attention.

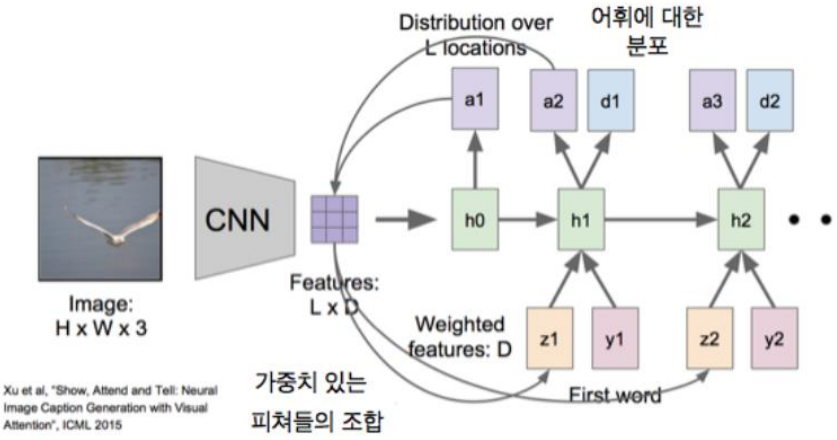
Attention: 연관된 것 중에서, 어떤 부분에 집중해야 하는지를 결정하는 것.




구성은 이렇게 된다.

a1에서 다시 Features로 들어가는데, 매 시점마다 어휘를 샘플링 한다는 것을 뜻함.

## 이미지 캡셔닝과 어텐션 (Attention)

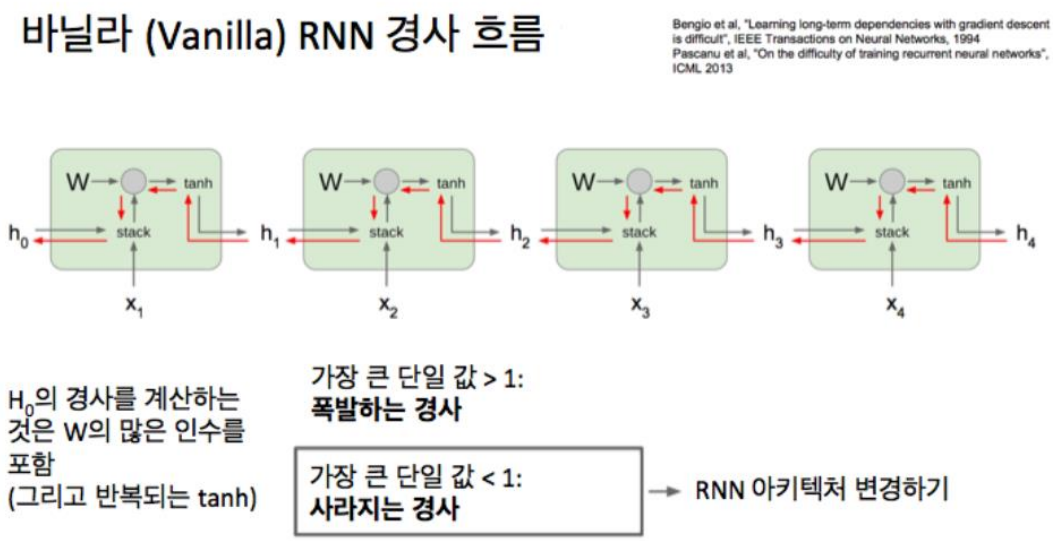



그렇게 또 다른 입력이 들어오면, 2개의 출력 값을. (a2,d1). 또 a2, a3를 또 Features로 거친다.

어텐션은 '이미지 캡셔닝' 뿐만 아니라 질의 응답에도 사용됨.

#어텐션 이란게 각 단계의 계산값을 기억해두는 느낌.

다계층 RNN에서 '바닐라 RNN', 'LSTM', 'GRU' 사용.



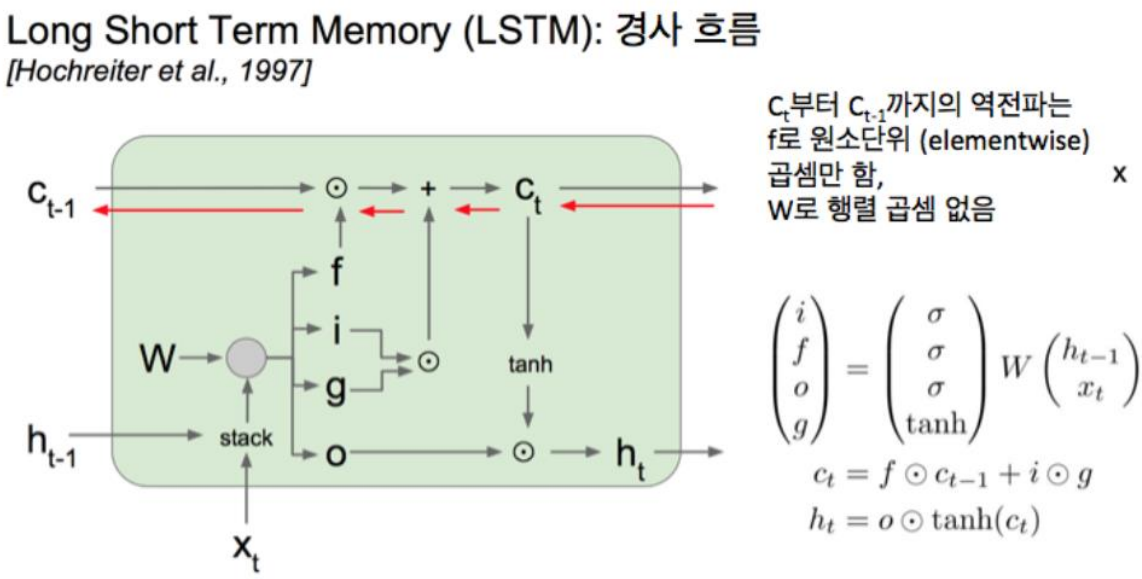

-> 바닐라 RNN.

빨간 화살표는 역전파 과정(경사계산: h에서 h_0까지 W를 다 곱하기)

exploding gradient(폭발하는 경사): 역전파 과정중에서, 값이 1보다 큰 값을 역전파 한다고 할 때, 최종 경사(h_0)값은 매우 커진다. -> gradient clipping(경사 잘라내기) 사용해서, 경사 비율을 조정해서 해결.

vanishing gradient(사라지는 경사): 값이 1보다 작은 값을 역전파 한다고 하면, 경사는 계속 줄어든다.

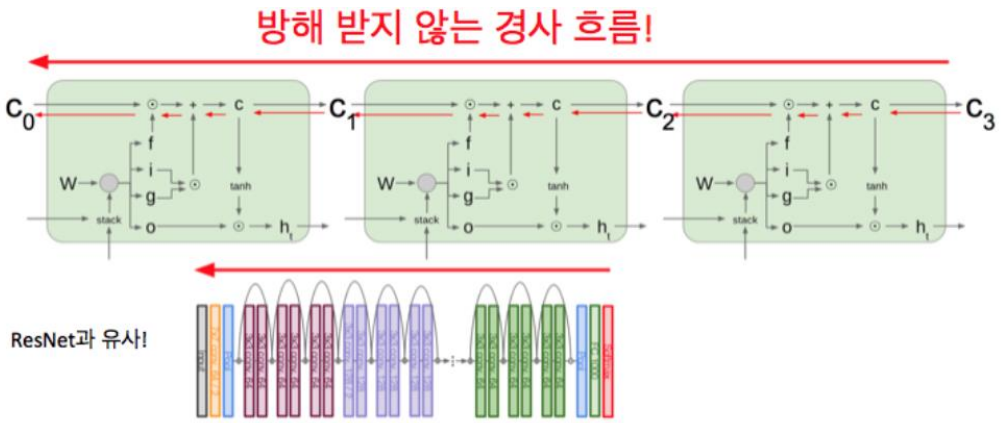
-> RNN구조 자체를 바꿔야 한다.




->LSTM(Long Short Term Memory).

역전파 할 때, 중간에 '+'가  2가지로 복사된다. 그래서 '원소마다의' 곱하기를 통해 역전파가 된다.

중간에 잊기 게이트 덕분에 폭발경사, 사라지는 경사가 없어진다.(전에서의 가중치 반복해서 곱하는거 사라져서).




그래서 서로 이렇게 역전파에 방해받지 않는다.

**GRU** [*Learning phrase representations using rnn encoder-decoder for statistical machine translation,* Cho et al. 2014]

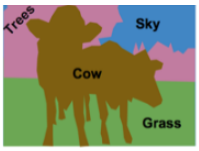
$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

또 GRU라고 또 다른 '사라지는 경사'문제를 해결하려 함.

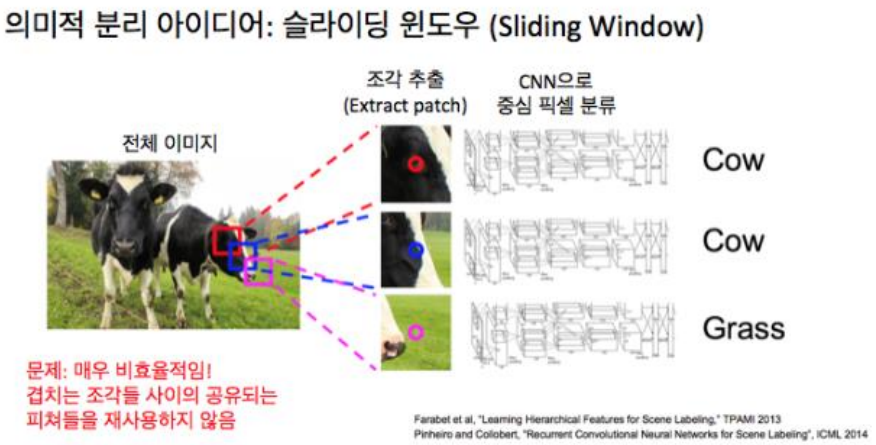
Lecture 11

이미지 분리(Segmentation), 위치 찾기(Localization), 탐지(Detection)

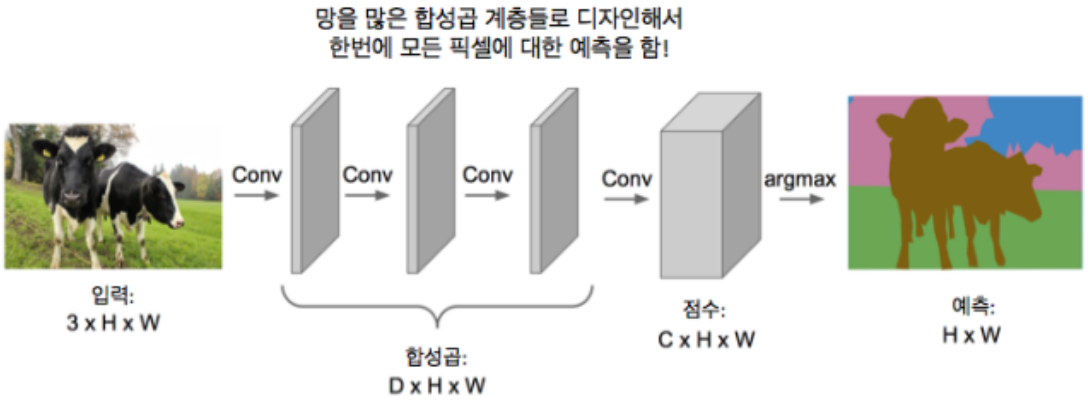

의미적 분리(Segmantic, segmentation):

-> 이미지에서 각 '픽셀'을 레이블로 구분. '픽셀'만 신경써서, 

이렇게 소 두마리 있는걸 구별 못함.

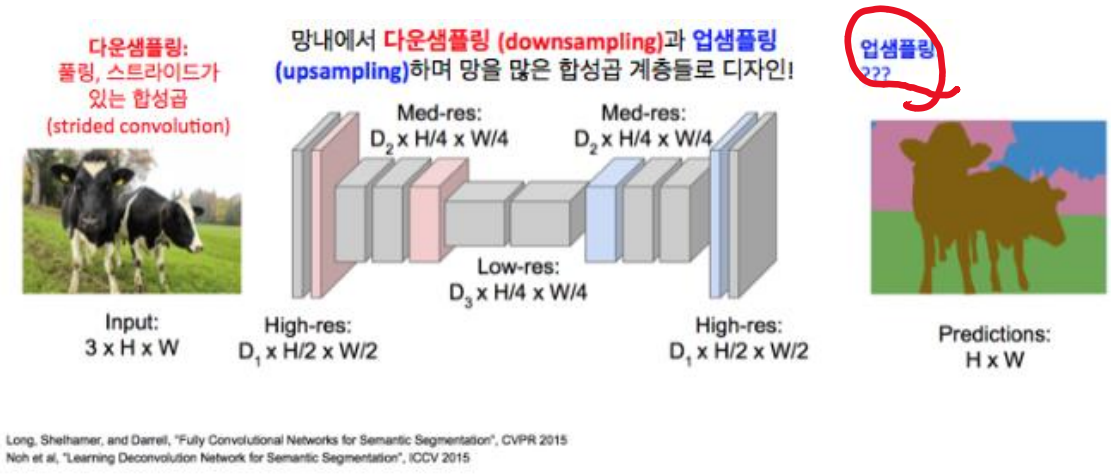



위와 같은 문제를 해결하기 위해, '슬라이딩 윈도우 방식'(구역별로 잘라내서 그걸 CNN거치게 하는 것)을 쓰는 것인데, 전체 이미지에 대한 것이 아니라 잘라낸 것에만 적용하는 거라서 별로다.



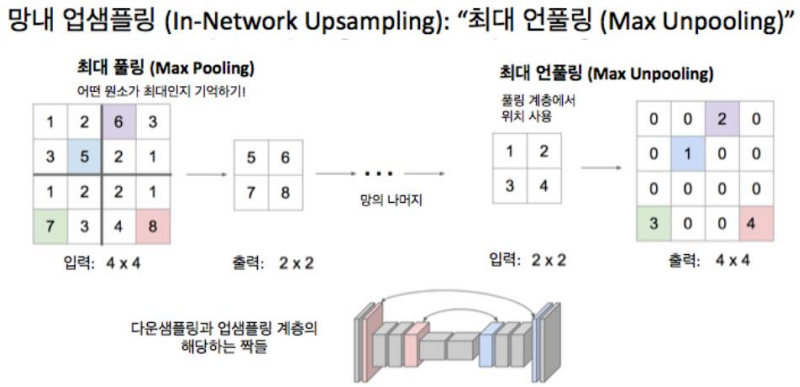

-> 'fully convolutional Network'아이디어. 독립적 조각 분류 느낌이 아닌, . CNN망 쌓은 느낌.

너무 많은 메모리를 차지해서, 잘 사용 안 하는 방식.

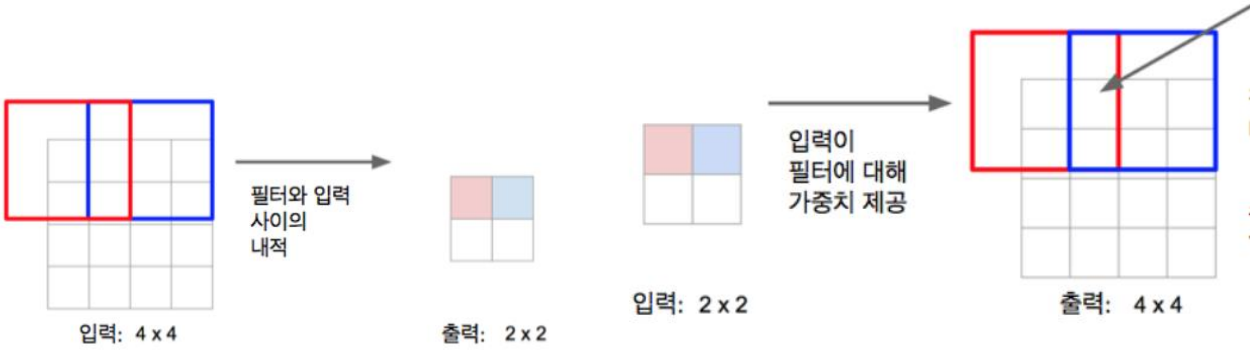



Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

->다운 샘플링, 업샘플링 방식. 위 fully convolutional network방식이랑 비슷한데, 처음 '다운 샘플링'으로 작은 수의 합성곱 거치고, '업 샘플링'으로 고해상도 이미지 출력.




->'업 샘플링' 방식으로는 최근접 이웃 언풀링(같은 수 복제 방식), 바늘방석(bed of nalls:0값 채워넣기)방식 존재. 위 그림은 첫 다운 샘플링으로 맥스 풀링 방식 사용. 후에 바늘방석 방식 사용 업 샘플링 사용.




->다운 샘플링, 업 샘플링(스트라이드 2, 패딩 1 방식)

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

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

예: 1D 합성곱, 커널 크기=3,
스트라이드=2, 패딩=1

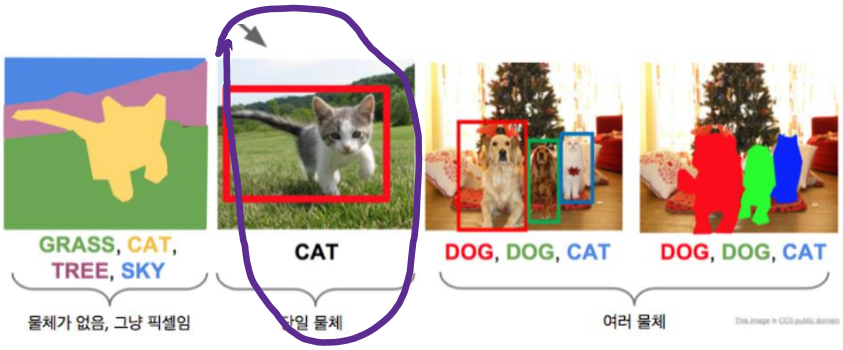
$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

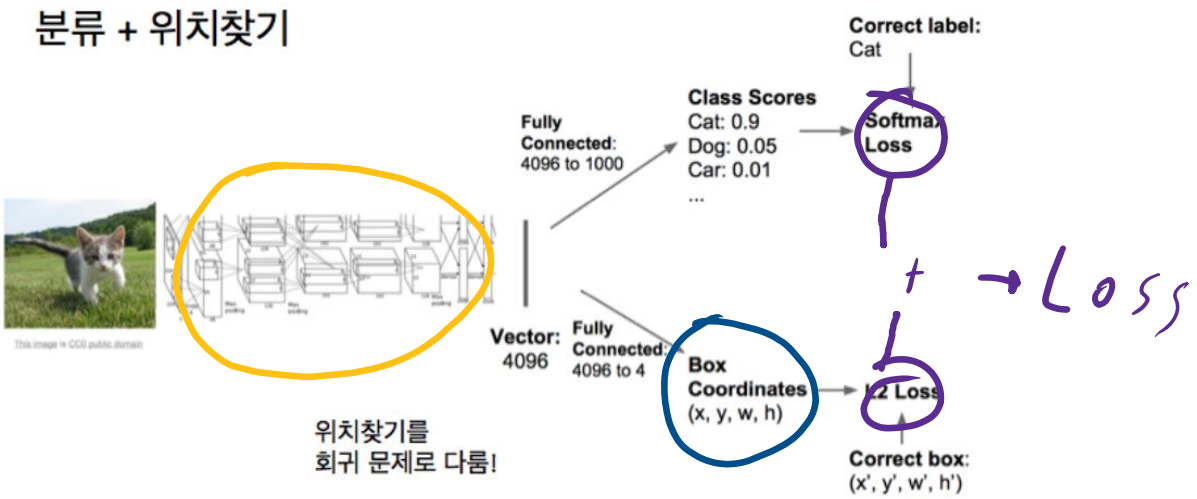
$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

스트라이드>1일때, 합성곱 전치는
더 이상 정상적인 합성곱이 아님!

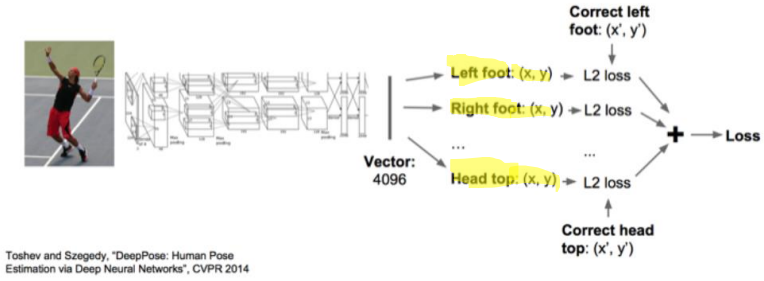
->행렬 계산.

분류 + 위치찾기(Classification + Localization):

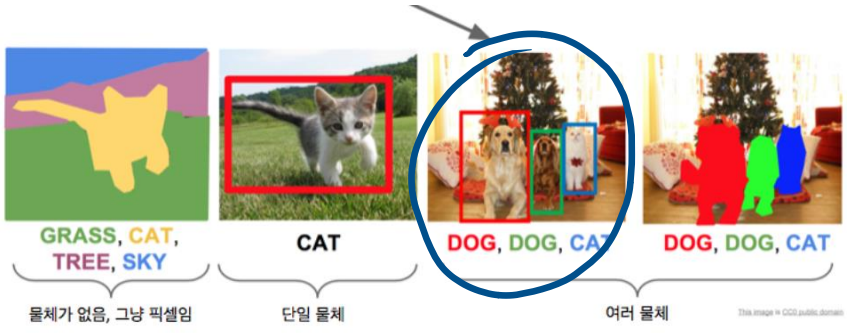





->노란색 신경망(AlexNet)거치고 나온 이미지 요약 내용. FC로 연결하고, 좌표 알려주는 값 뱉고, 나온 각 Loss값 더해서 최종 Loss값.



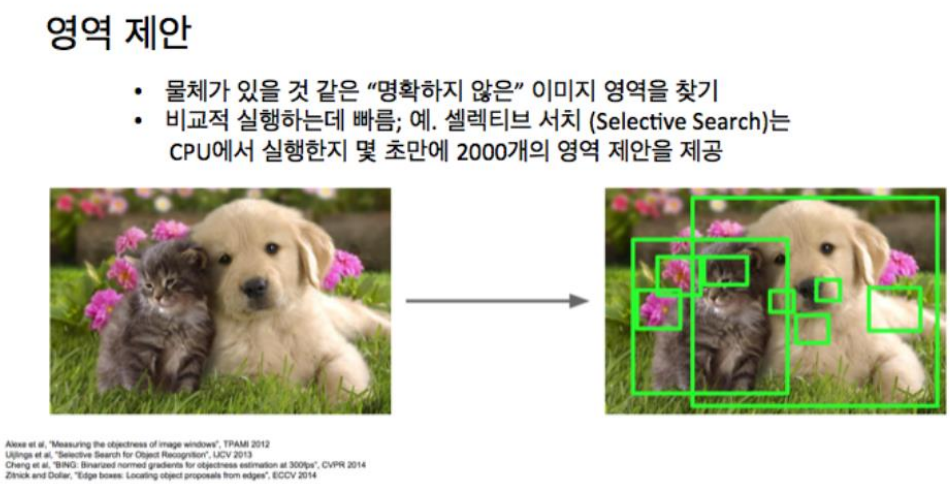

이런식으로, 왼발 오른발, 머리, 다리 나누어서 손실값 내고, 이런식으로.




->물체 탐지.

각 물체마다 (x,y,w,h)값 4개씩 예측(좌표찾기)해야 하는데, 회귀 문제로 풀면 안된다.

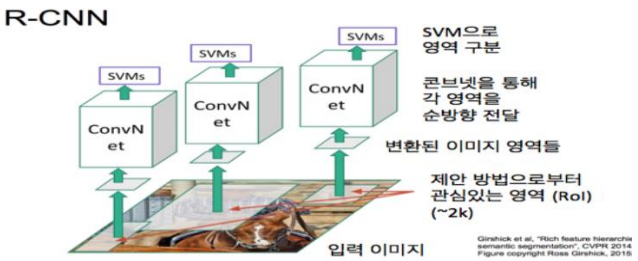

->그래서 여러 조각 나눈 CNN적용 하는 방식. #너무 많은 계산이 들어감



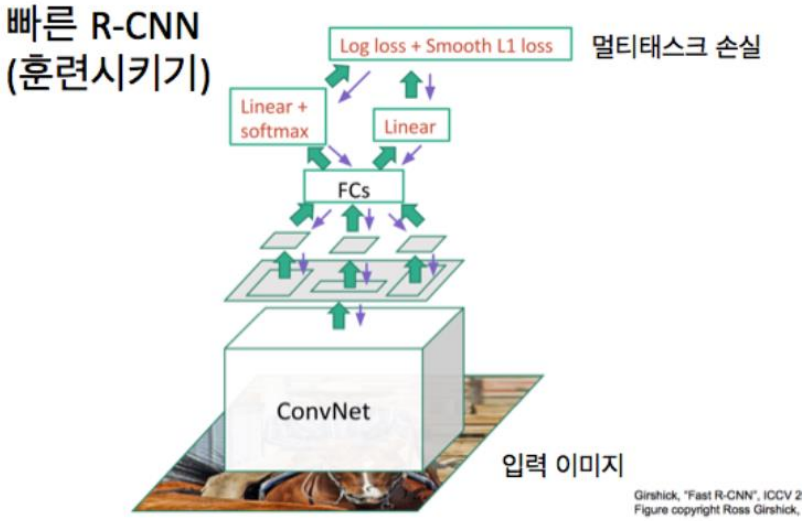

->그래서 나온 'region proposals'(영역제안 방식)

셀렉티브 서치(Selective Search)라는 방식 써서, cpu에서 2초동안 실행하면 2000개의 영역 뱉는다.

노이즈 껴있긴 한데, 그래도 위에꺼 보다 더 좋음.

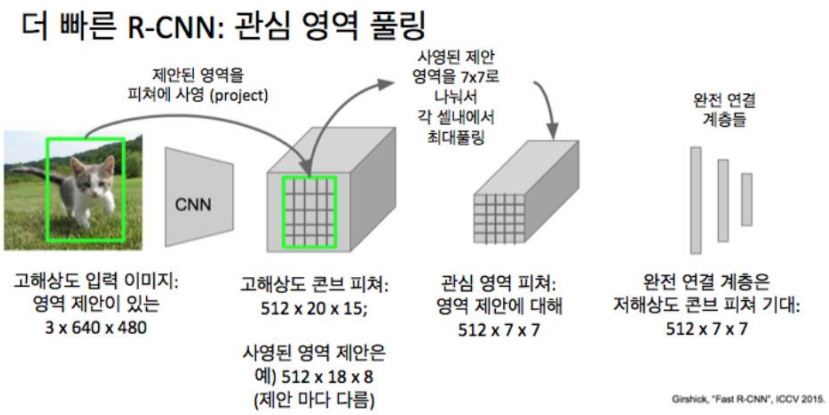



-> R-CNN방식. 각 영역마다 구역 나눠서 CNN통과. 각 경계 나누는 걸 많이 한다면, 계산이 너무 많이 들어감. 시간도 당연히 많이 걸림.



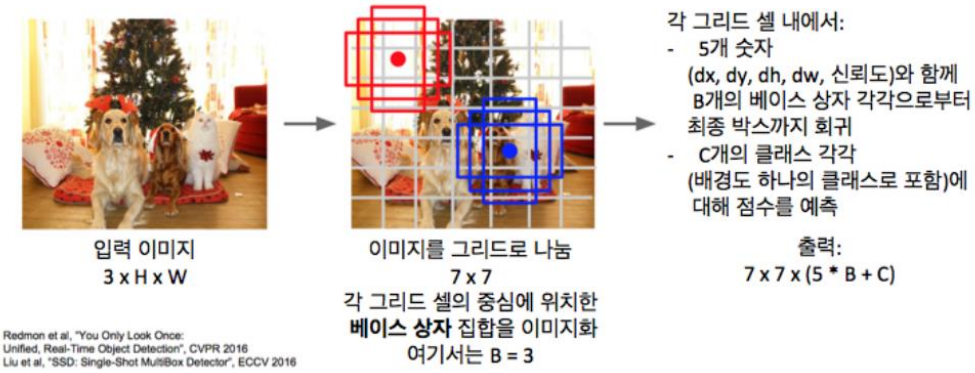

-> 그래서 나온 빠른 R-CNN 방식. 영역 구분 안하고 전체 이미지 한 번에 CNN계층 통과.

FC 거치고 손실값 두개 계산.

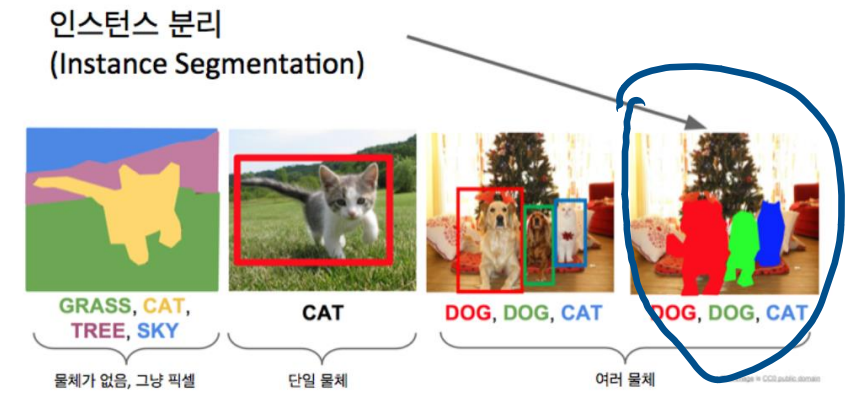


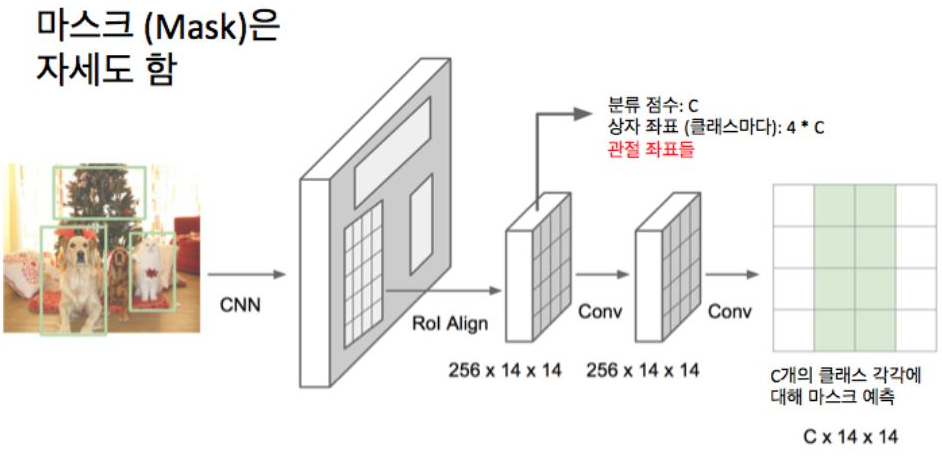

-> 더 빠른 R-CNN. MaxPooling 거친 방식.

# 테스트 시간: R-CNN -> SPP-Net -> Fast R-CNN -> Faster R-CNN.




-> YOLO / SSD방식. 이미지를 그리드 영역으로 나눔.




->Mask R-CNN방식.

단순 분류, 예측이 아닌, 영역 각각을 분리. (마스크)