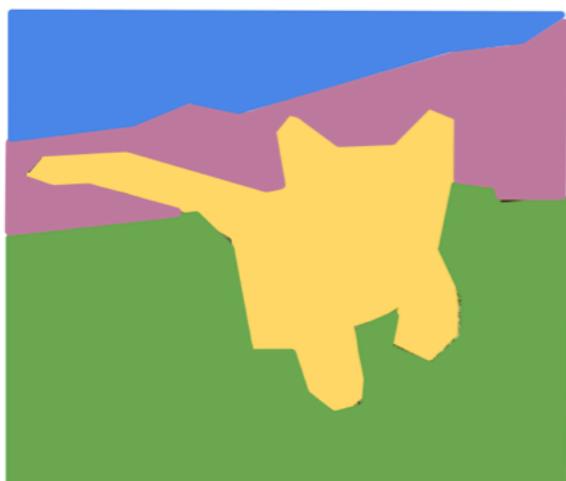


# CNN 응용 사례

안남혁

# 영상 인식

Segmentation



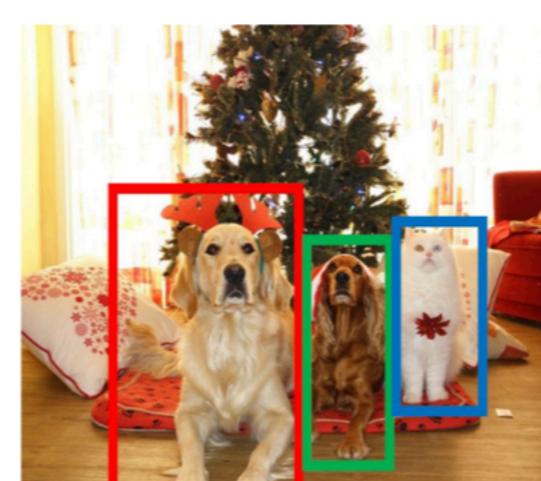
GRASS, CAT,  
TREE, SKY

Localization



CAT

Detection



DOG, DOG, CAT

Instance  
Segmentation

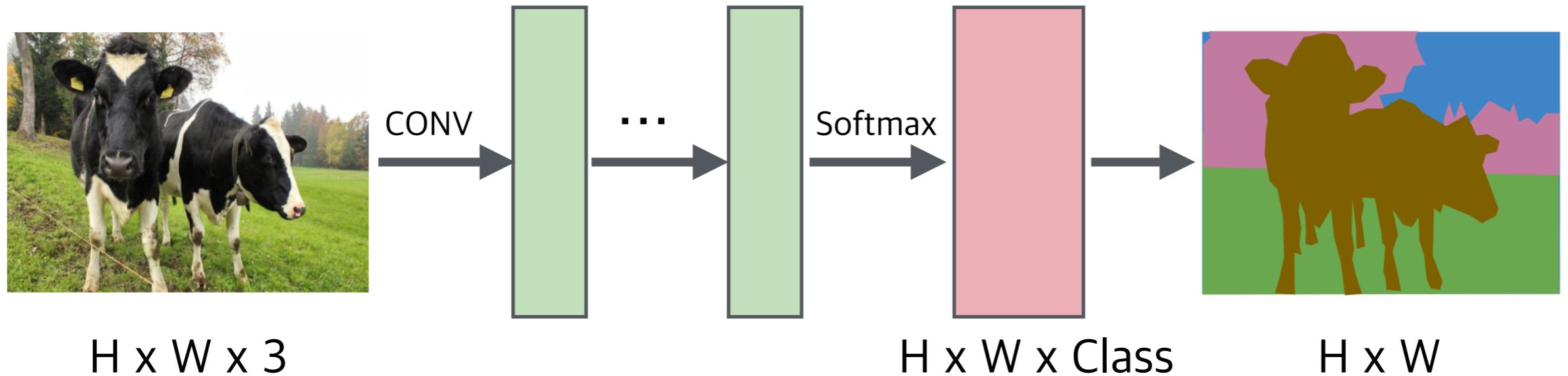


DOG, DOG, CAT

# Segmentation

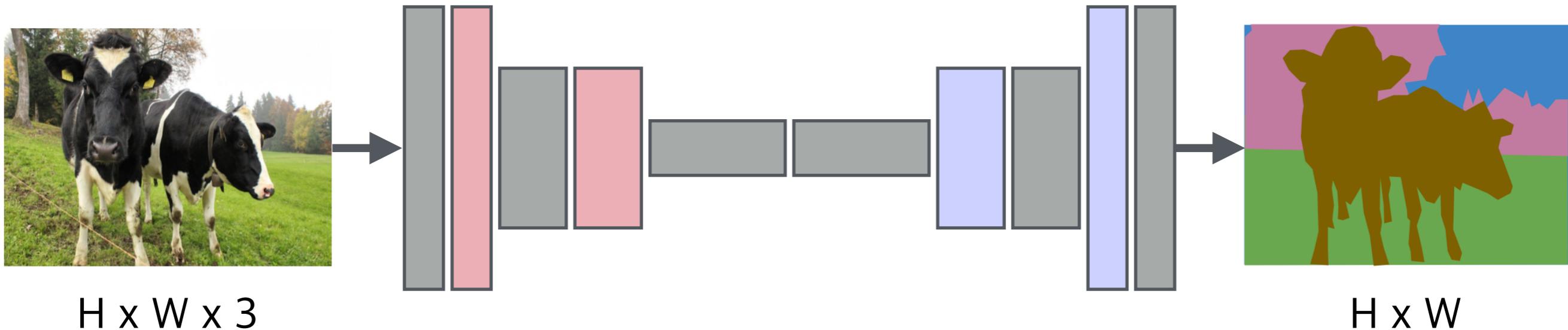
- 이미지의 모든 픽셀을 정답 카테고리로 분류
- 여러 물체에 대해서 상관하지 않음
  - 여러 물체에 대한 segmentation은 instance segmentation
- 데이터셋
  - PASCAL VOC, MS COCO가 가장 많이 쓰이는 데이터셋

# Fully Convolutional Network



- 아이디어:
  - 기존 FC 레이어를 사용하면 공간 정보가 모두 소실될 우려가 생김
  - FC 레이어를 빼고 **fully-convolutional**하게 네트워크 구성
  - 다양한 이미지 크기를 입력으로 받을 수 있음
- 단점:
  - 이미지 크기가 크면 메모리 요구량 / 연산량이 매우 커짐

# Fully Convolutional Network



- 아이디어:
  - 모델 내부에서 이미지를 다운샘플링 -> 업샘플링 하도록 구성하여 해결
  - 다운샘플링: **Pooling** 혹은 (strided) **Convolution**
  - 업샘플링: **Unpooling** 혹은 **Transpose convolution** (deconvolution)

# Unpooling

Max Pooling

1	2	3	4
5	6	7	8
9	8	7	6
5	4	3	2



6	8
9	7

Max Unpooling

3	4
5	6



0	0	0	0
0	3	0	4
5	0	6	0
0	0	0	0

- Pooling 시 최대값의 위치를 저장해놓고 unpooling 시 저장해놓은 최대값 위치로 값을 전파
- 따라서 pooling과 unpooling은 짹이 맞아야함

# Transpose Convolution

3x3 transpose convolution, stride 2 pad 1

1	-1
0	1

X

-1	1	-1
1	0	1
-1	1	-1

=

0	1		
1	-1		

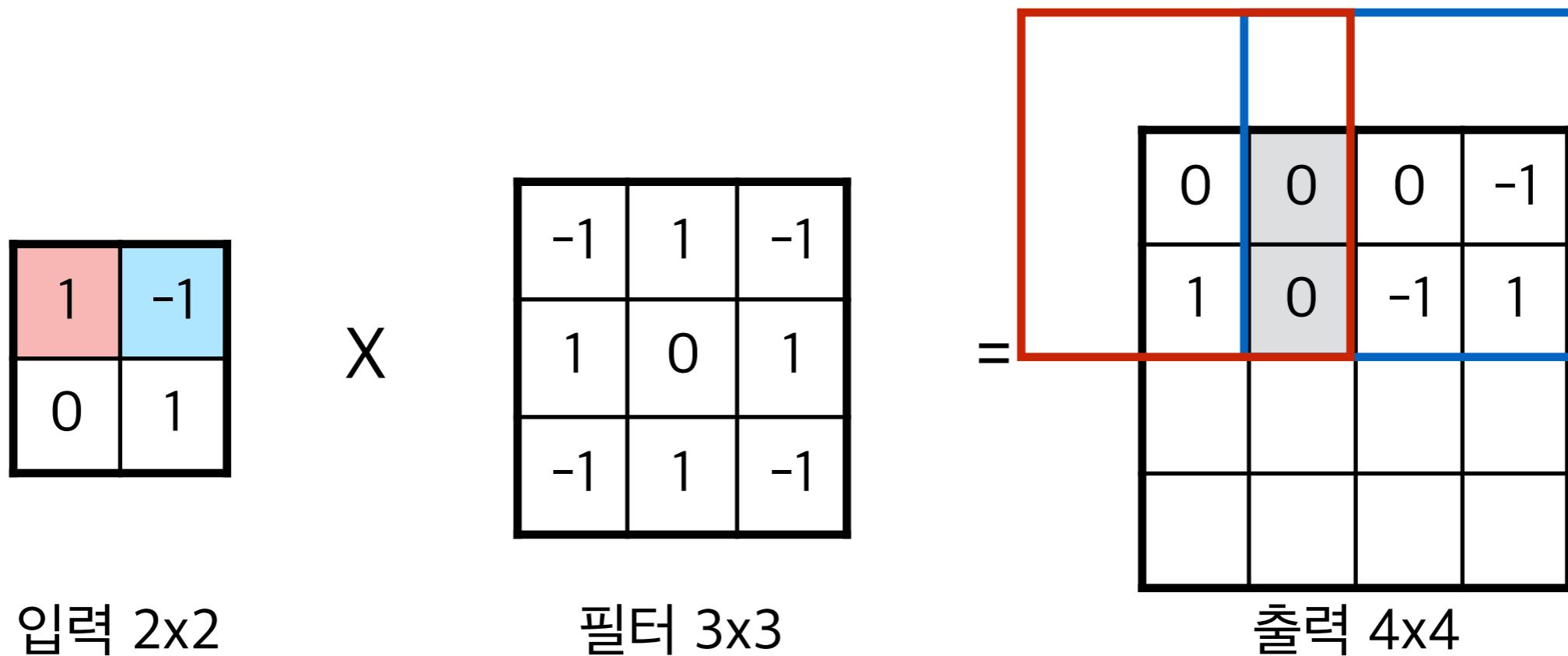
입력 2x2

필터 3x3

출력 4x4

# Transpose Convolution

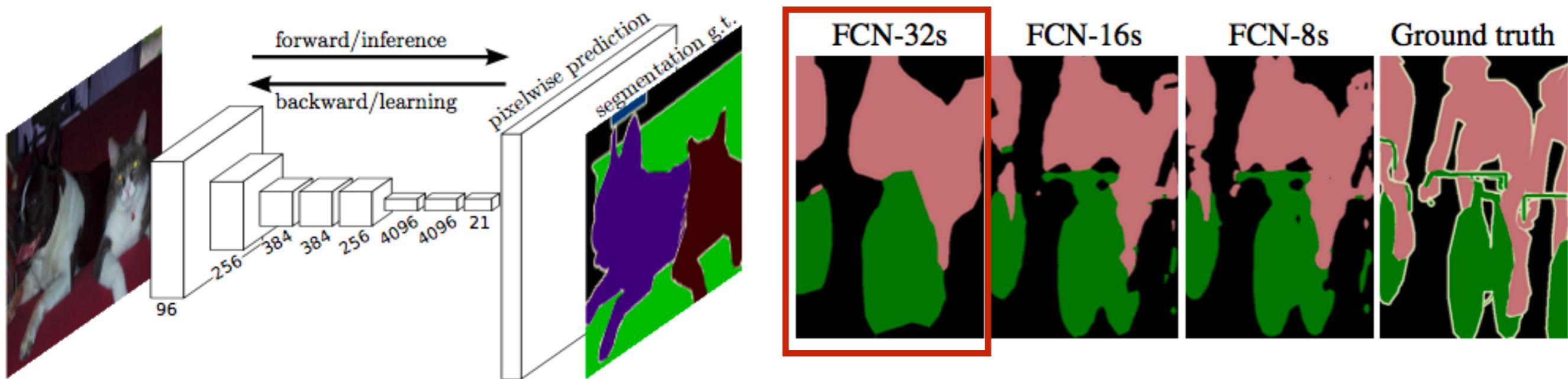
3x3 transpose convolution, stride 2 pad 1



- 입력 픽셀은 항상 1칸씩 움직이고 출력 픽셀을 stride 만큼 움직여서 진행
- 출력값이 겹치는 부분은 모든 출력값을 더해줌

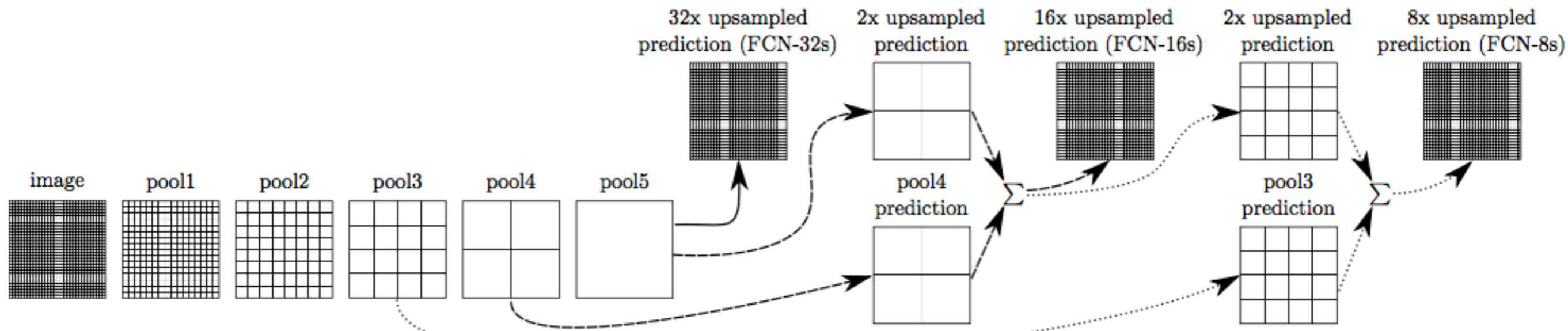
# FCN

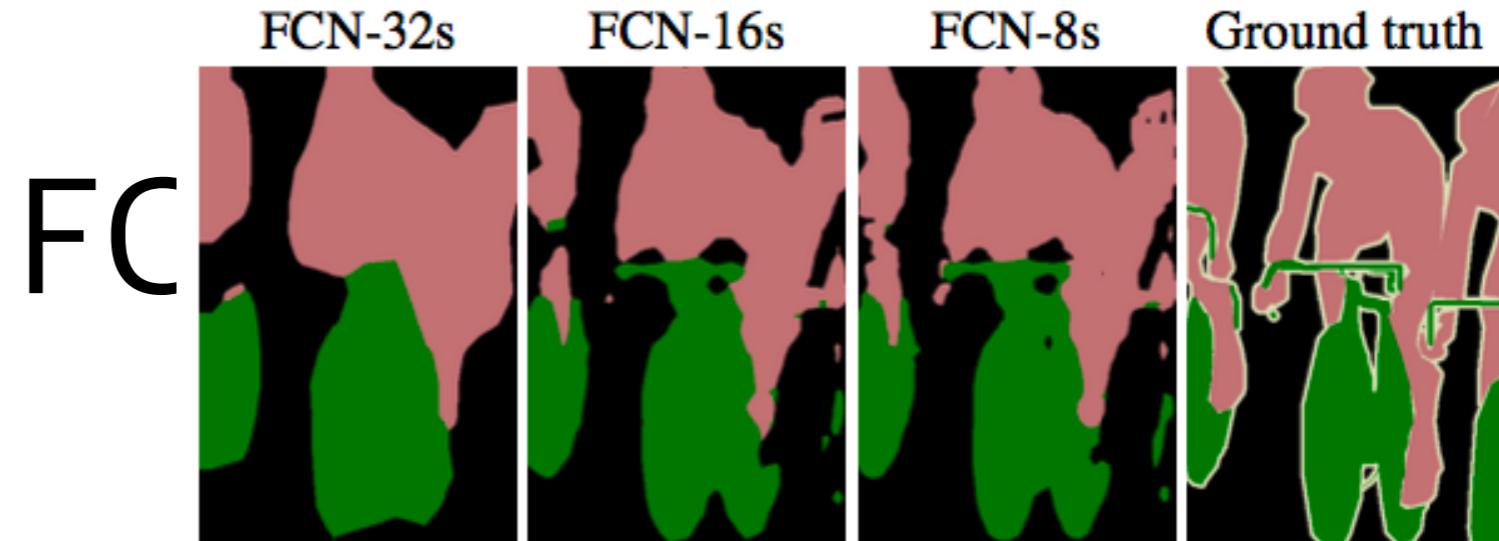
- Fully-convolutional 개념을 segmentation에 처음 적용
  - 마지막 conv에서 얻은 히트맵을 원 이미지 크기만큼 확대
  - 확대 레이어는 unpooling 혹은 transpose convolution으로 사용 가능
  - **디테일한 부분은 정보의 소실 우려**



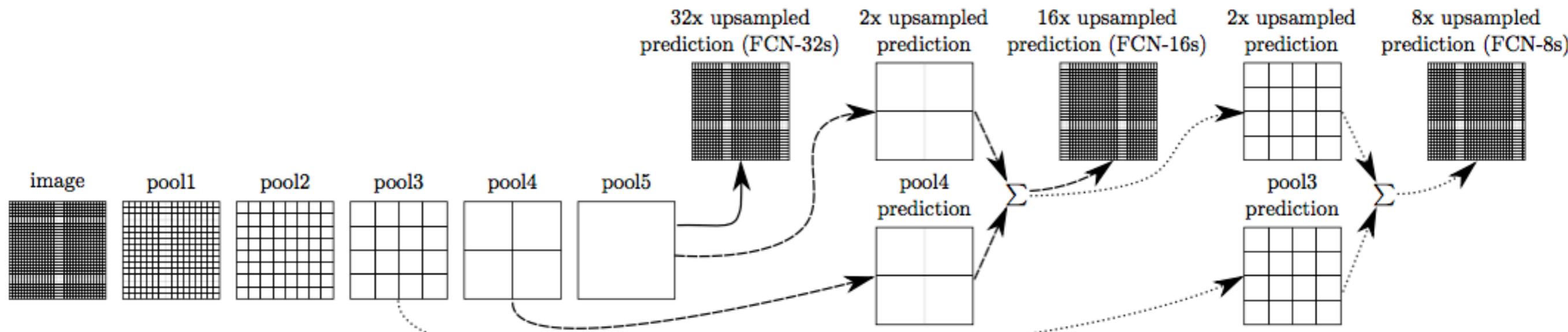
# FCN

- 여러 레이어의 정보를 혼합하여 디테일 문제를 해결
  - 얕은 레이어는 물체의 디테일한 정보를, 깊은 레이어는 전체적인 구조 정보를 갖고 있음
  - 이는 깊은 레이어에서 특징 맵의 크기가 매우 작아지기 때문
  - 여러 풀링 레이어의 결과를 업샘플과 덧셈 연산으로 정보를 합치는 과정



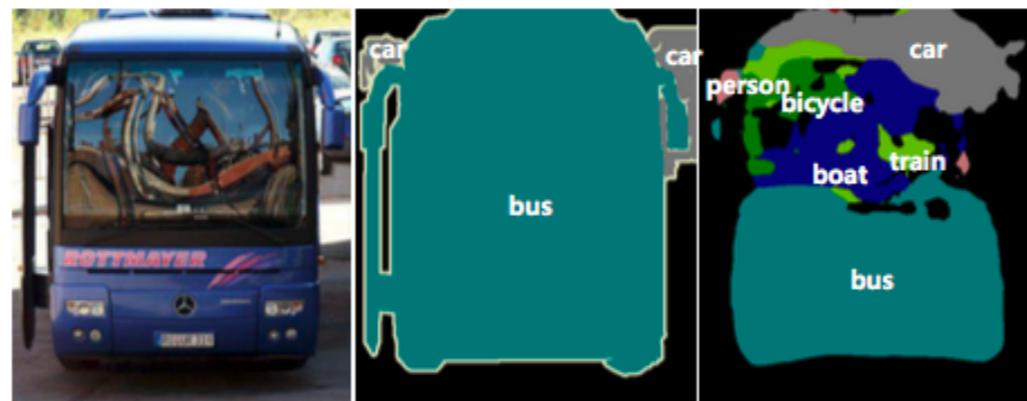


- 여러 레이어의 정보를 혼합하여 디테일 문제를 해결
  - 얕은 레이어는 물체의 디테일한 정보를, 깊은 레이어는 전체적인 구조 정보를 갖고 있음
  - 이는 깊은 레이어에서 특징 맵의 크기가 매우 작아지기 때문
  - 여러 풀링 레이어의 결과를 업샘플과 덧셈 연산으로 정보를 합치는 과정

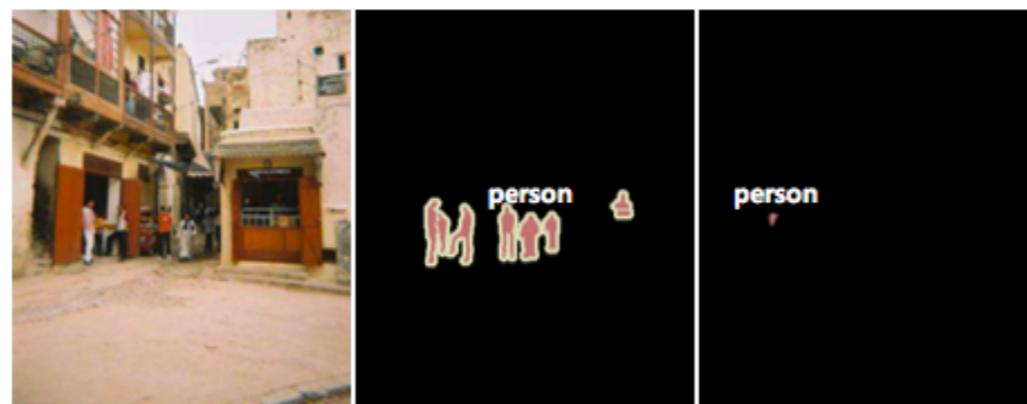


# FCN의 단점

- 물체가 너무 크거나 작으면 잘 탐지하지 못함
  - 마지막 레이어의 특징맵을 (16x16) 확대해서 결과를 내놓기 때문
  - 여러 정보를 혼합하는 과정도 완벽한 해결책은 아님



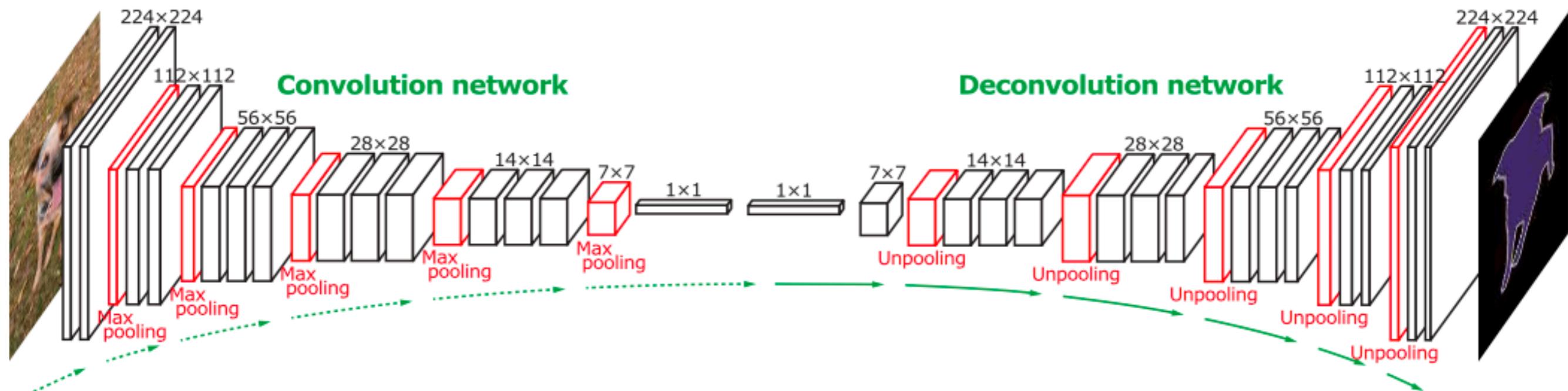
(a) Inconsistent labels due to large object size



(b) Missing labels due to small object size

# DeconvNet

- FCN의 단점을 보완하기 위한 네트워크 구조
  - VGG-16 네트워크를 모래시계처럼 쌓아놓은 구조
  - **Pooling-unpooling, convolution-deconvolution**이 짹지어 연결
  - (Transpose convolution == deconvolution)



# DeconvNet 시각화

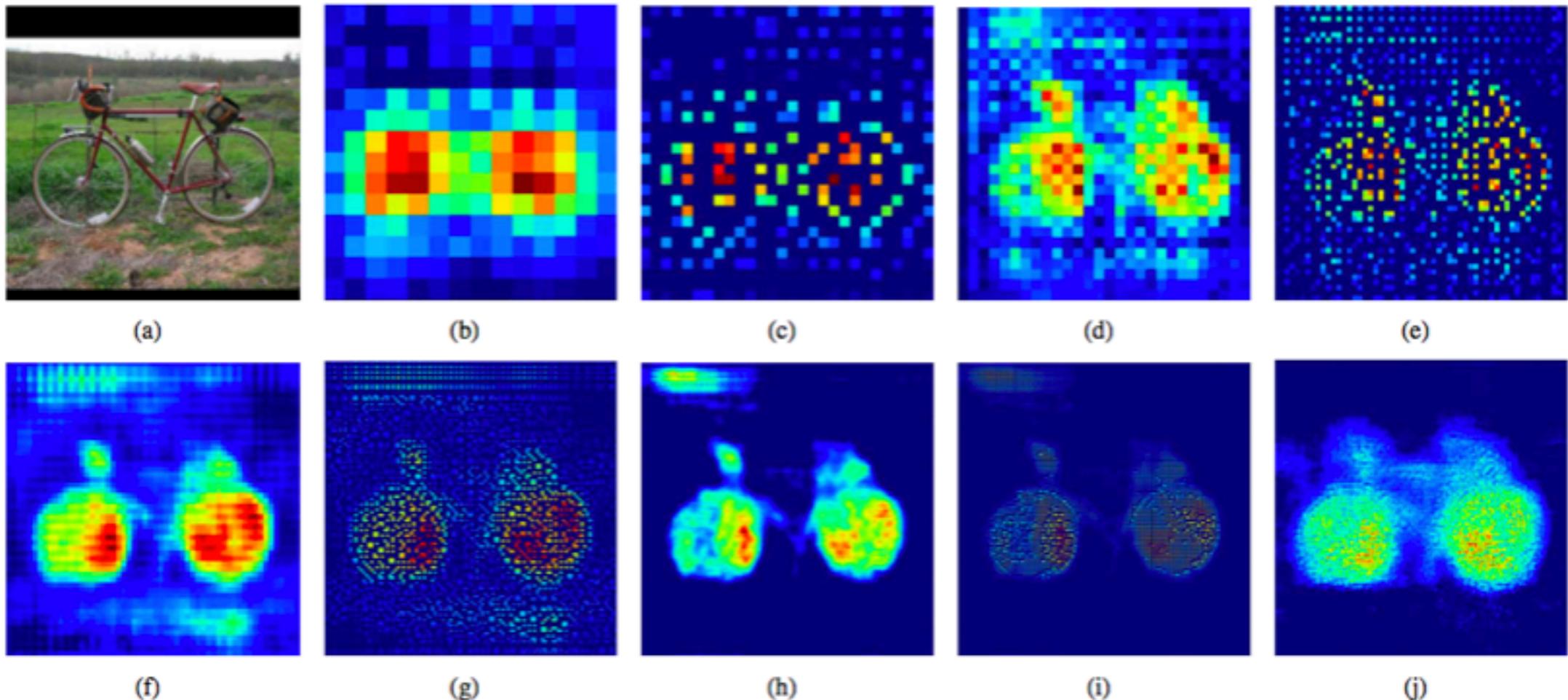
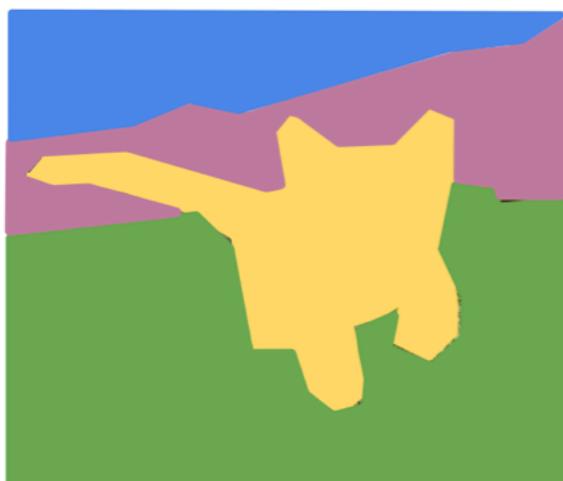


Figure 4. Visualization of activations in our deconvolution network. The activation maps from (b) to (j) correspond to the output maps from lower to higher layers in the deconvolution network. We select the most representative activation in each layer for effective visualization. The image in (a) is an input, and the rest are the outputs from (b) the last  $14 \times 14$  deconvolutional layer, (c) the  $28 \times 28$  unpooling layer, (d) the last  $28 \times 28$  deconvolutional layer, (e) the  $56 \times 56$  unpooling layer, (f) the last  $56 \times 56$  deconvolutional layer, (g) the  $112 \times 112$  unpooling layer, (h) the last  $112 \times 112$  deconvolutional layer, (i) the  $224 \times 224$  unpooling layer and (j) the last  $224 \times 224$  deconvolutional layer. The finer details of the object are revealed, as the features are forward-propagated through the layers in the deconvolution network. Note that noisy activations from background are suppressed through propagation while the activations closely related to the target classes are amplified. It shows that the learned filters in higher deconvolutional layers tend to capture class-specific shape information.

# Detection

Segmentation



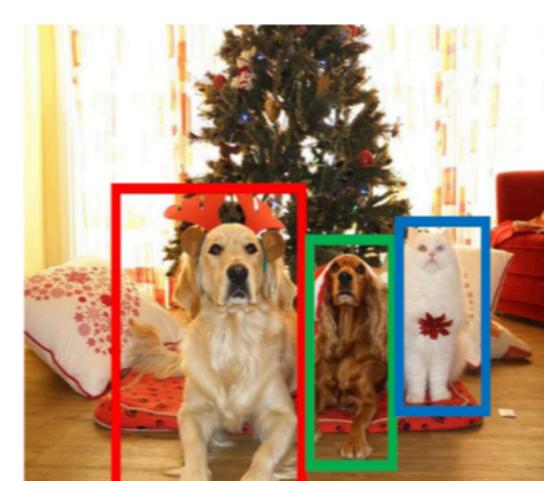
**GRASS, CAT,  
TREE, SKY**

Localization



**CAT**

Detection



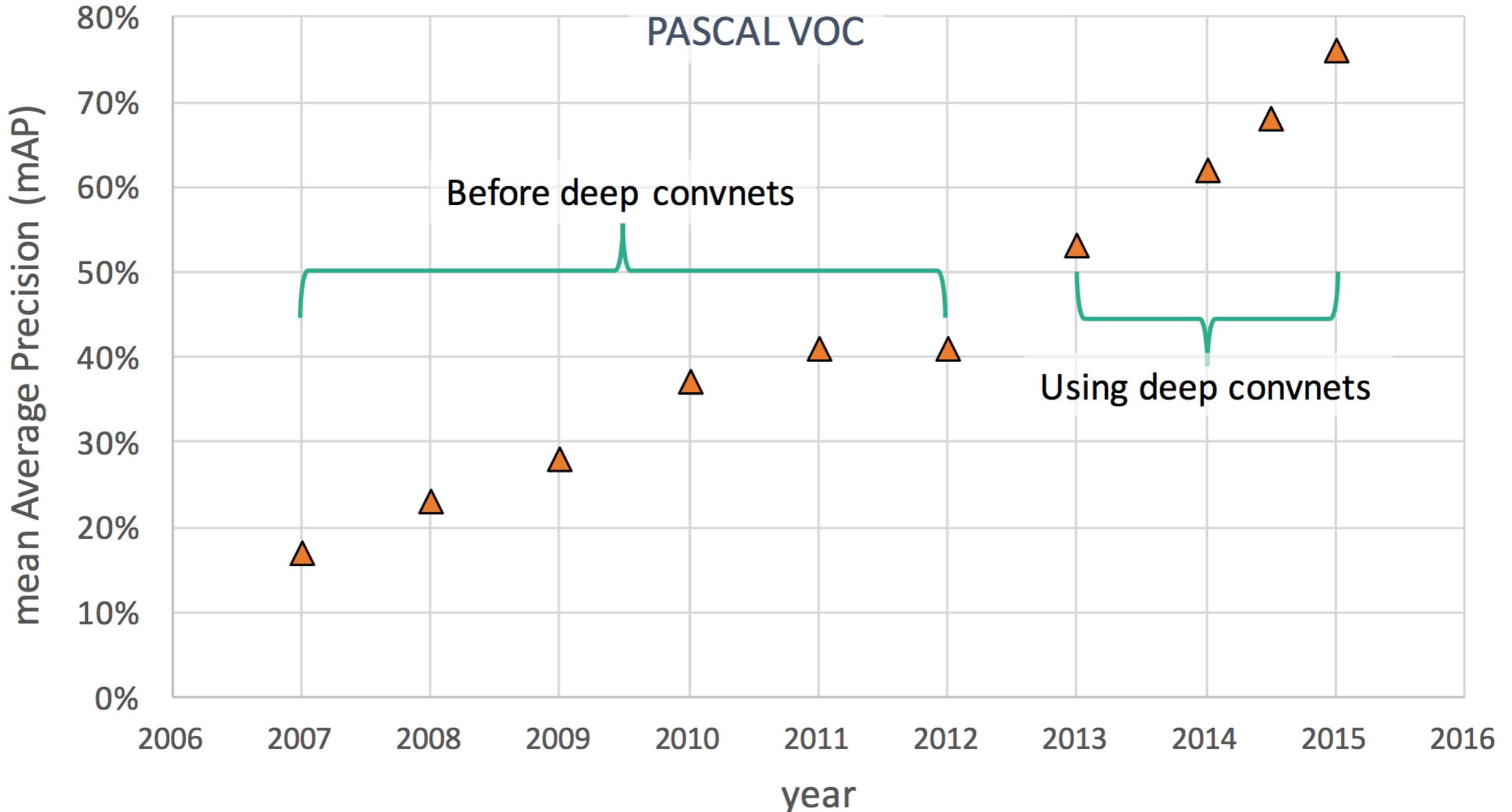
**DOG, DOG, CAT**

Instance  
Segmentation



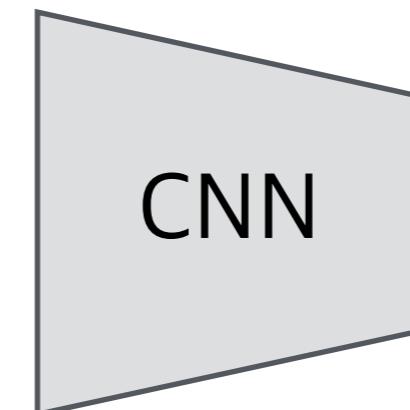
**DOG, DOG, CAT**

# Detection



# Detection

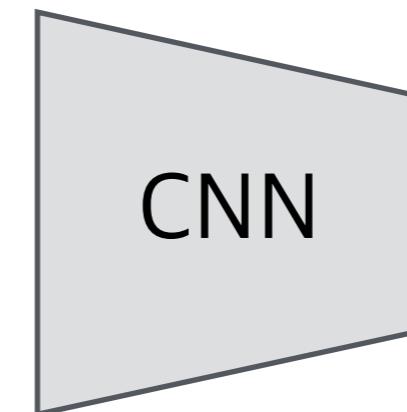
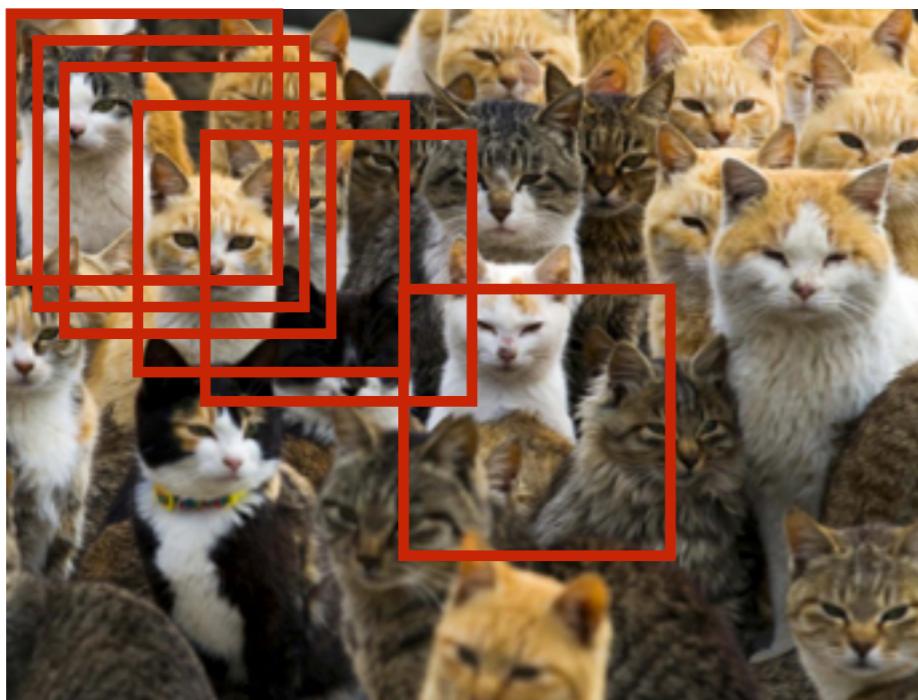
- Sliding window:
  - 이미지 부분 영역을 움직이면서 물체를 분류하는 방법
  - 부분 영역을 잘라내고 이를 CNN의 입력으로 넣어 물체를 판별하는 방식
  - 단순하지만 모든 부분 영역을 CNN으로 판별해야 하기 때문에 **매우 느림**



고양이 0.7  
강아지 0.3  
...

# Detection

- Sliding window:
  - 이미지 부분 영역을 움직이면서 물체를 분류하는 방법
  - 부분 영역을 잘라내고 이를 CNN의 입력으로 넣어 물체를 판별하는 방식
  - 단순하지만 모든 부분 영역을 CNN으로 판별해야 하기 때문에 **매우 느림**



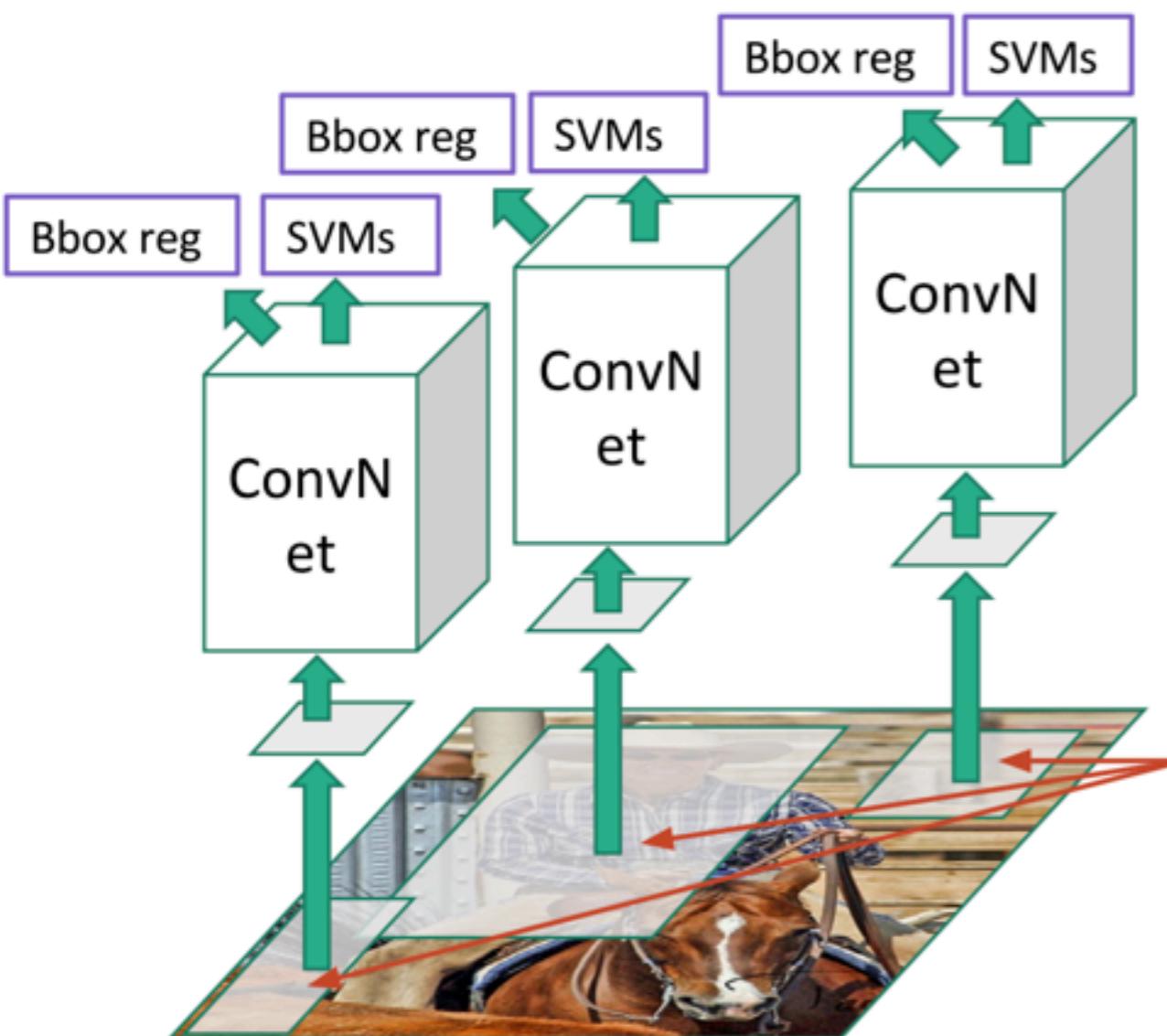
고양이 0.7  
강아지 0.3  
...

# Detection

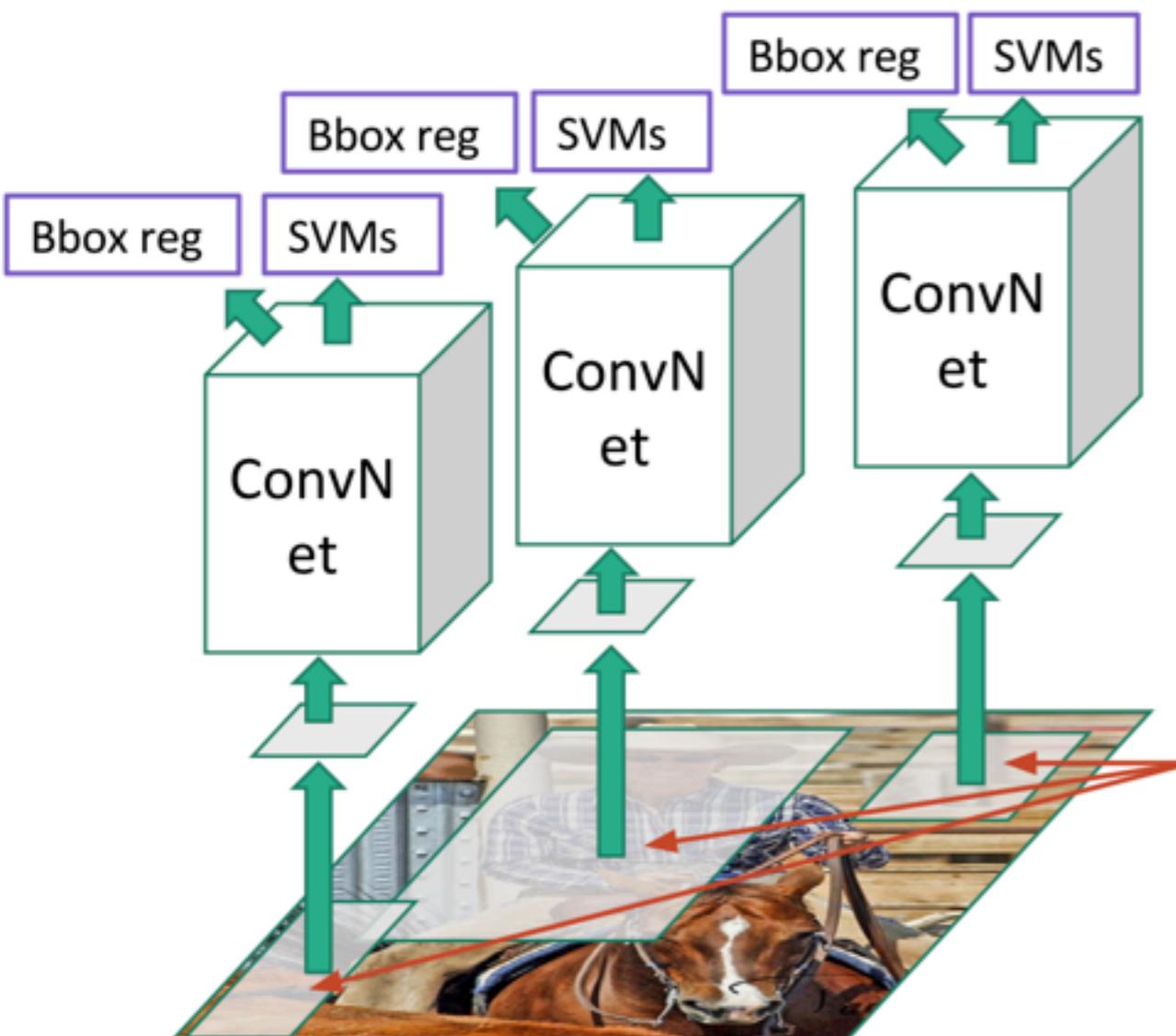
- Region proposal:
  - 모든 부분 영역을 탐색하는 것은 매우 비 효율적
  - 비교적 빠른 알고리즘을 사용해서 탐색 영역의 수를 줄이자
  - Selective search가 가장 많이 쓰이는 region proposal 알고리즘
  - (2000개 영역을 뽑아내는데 CPU에서 2~3초 걸림)



# R-CNN

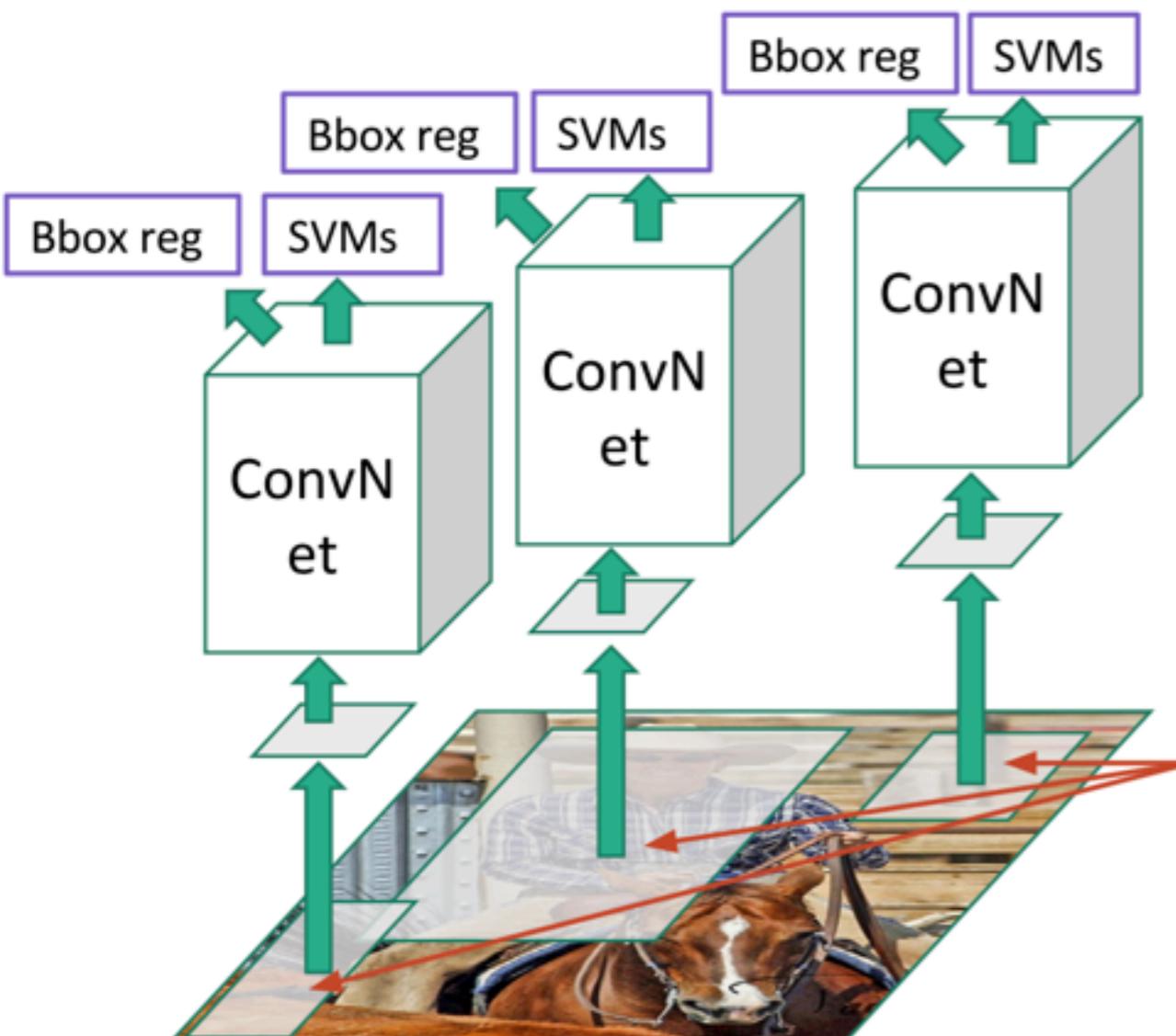


# R-CNN



Region proposal 알고리즘을 통해  
관심 영역 (Region of Interest) 추출  
(대략 2천개)

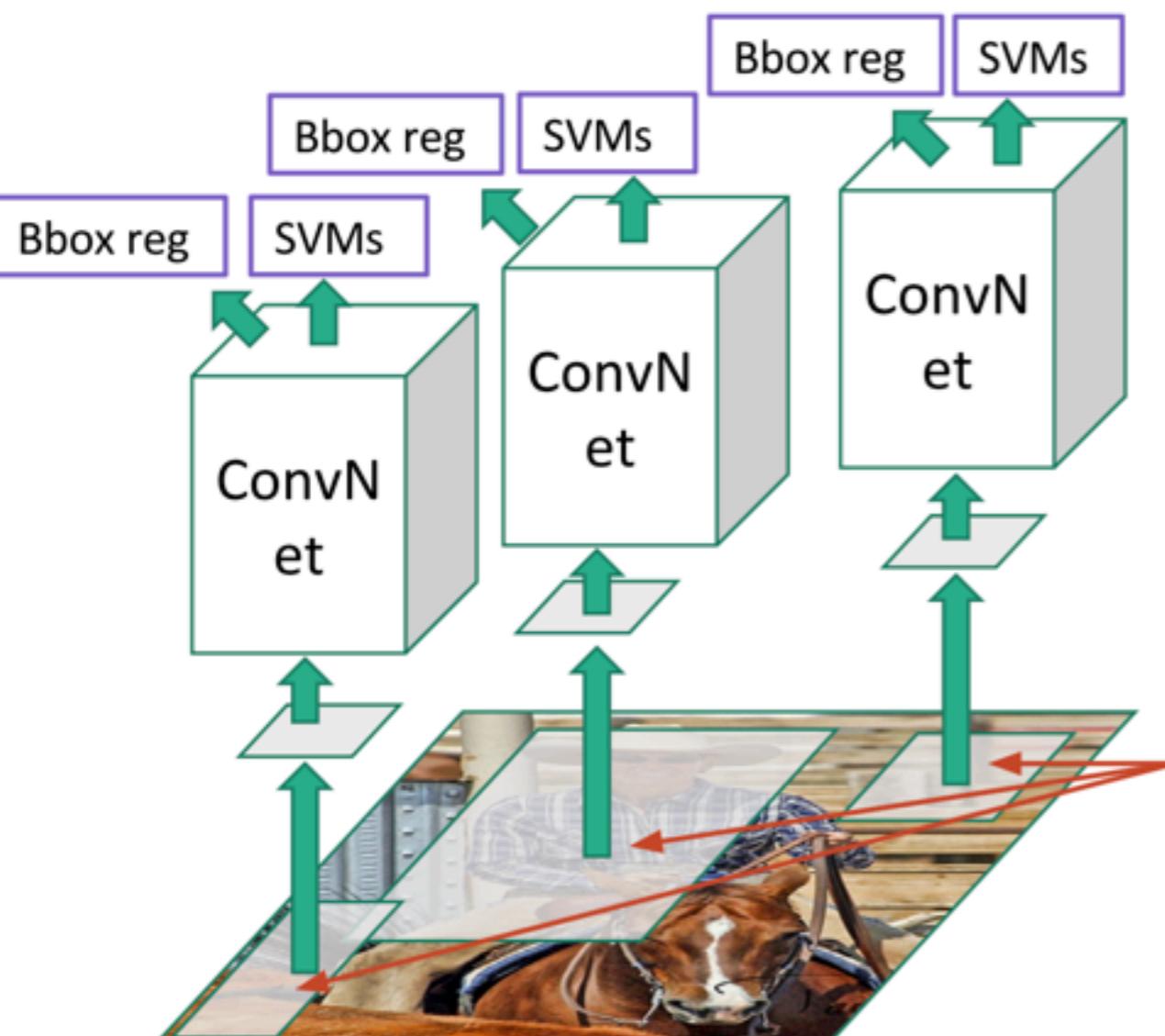
# R-CNN



이미지 리사이즈 (wrapping):  
CNN 입력 사이즈에 맞추기 위해

Region proposal 알고리즘을 통해  
관심 영역 (Region on Interest) 추출  
(대략 2천개)

# R-CNN



SVM: 현재 영역의 물체 분류

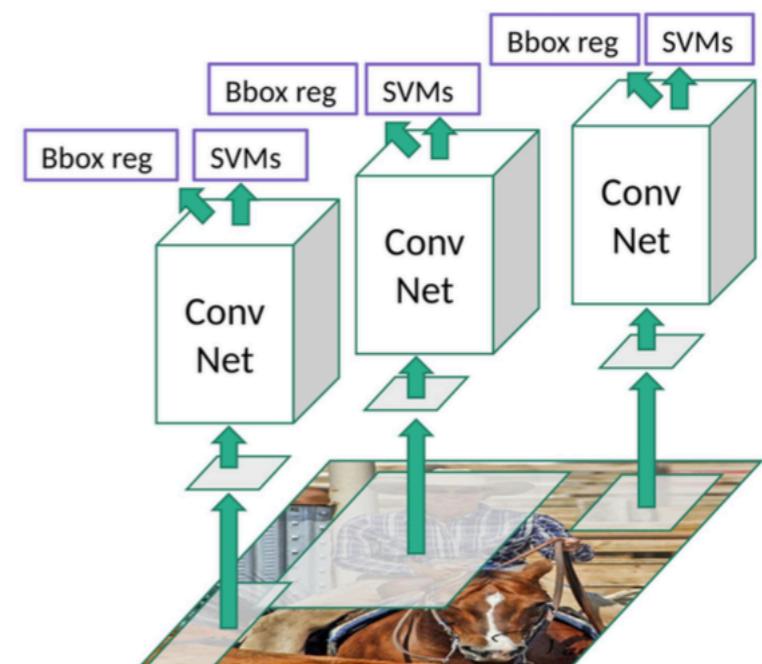
Bbox reg: 선형 회귀를 통해 박스의 세밀한 위치 조정

이미지 리사이즈 (wrapping):  
CNN 입력 사이즈에 맞추기 위해

Region proposal 알고리즘을 통해  
관심 영역 (Region of Interest) 추출  
(대략 2천개)

# R-CNN의 문제점

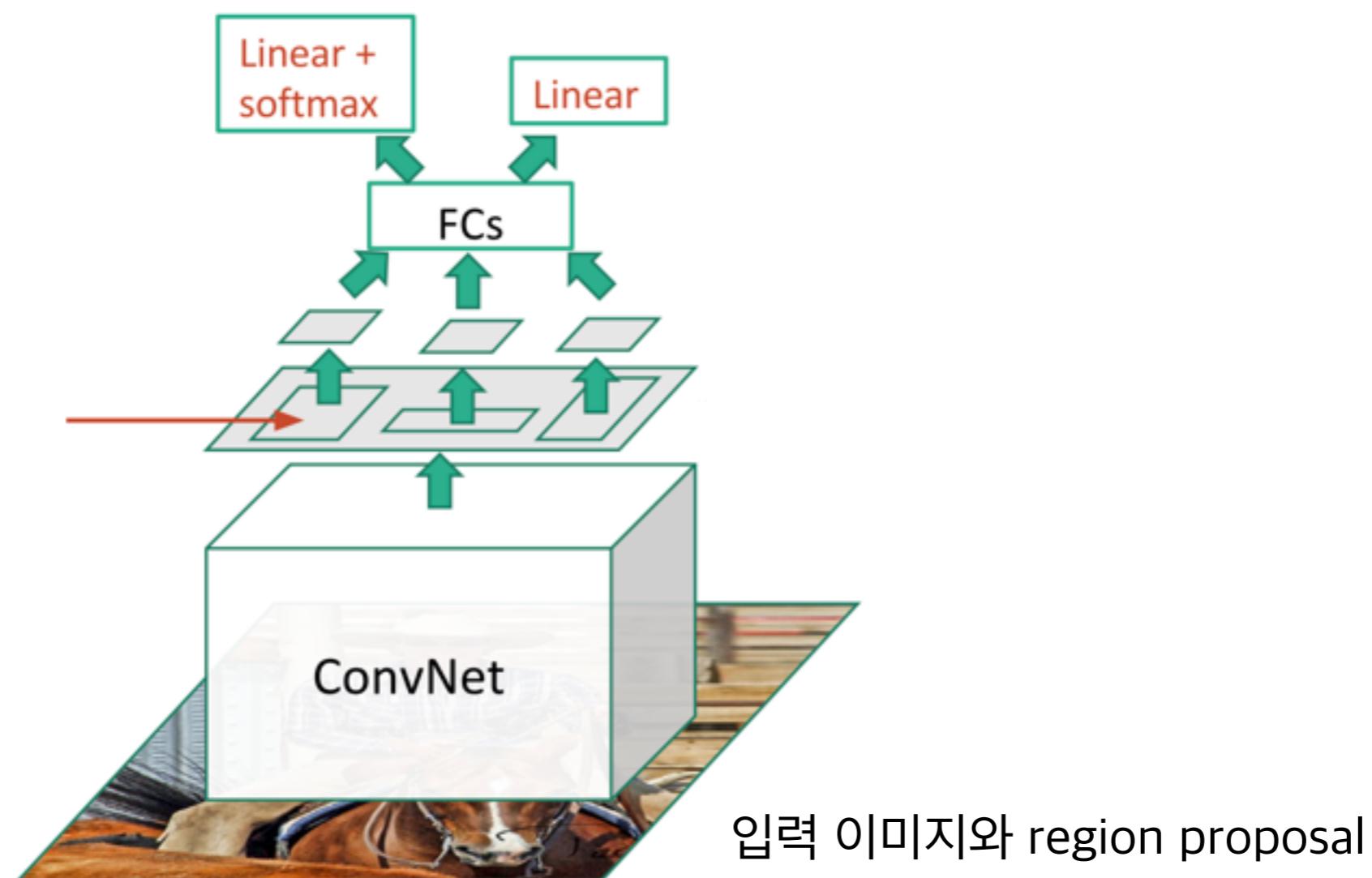
- 복잡한 네트워크 학습 단계
  1. 새로운 데이터셋에 네트워크 fine-tune (softmax 분류)
  2. 부분 영역 이미지 분류를 위한 SVM 학습
  3. 바운딩 박스 미세 조정을 위한 선형 회귀 학습
- 학습 / 테스트가 매우 느림
  - 학습하는데 84시간
  - 2, 3번 학습 단계를 위해 모든 부분 영역의 특징 맵을 디스크에 저장해야함 (수백 GB)
  - 이미지 한 장 테스트하는데 47초



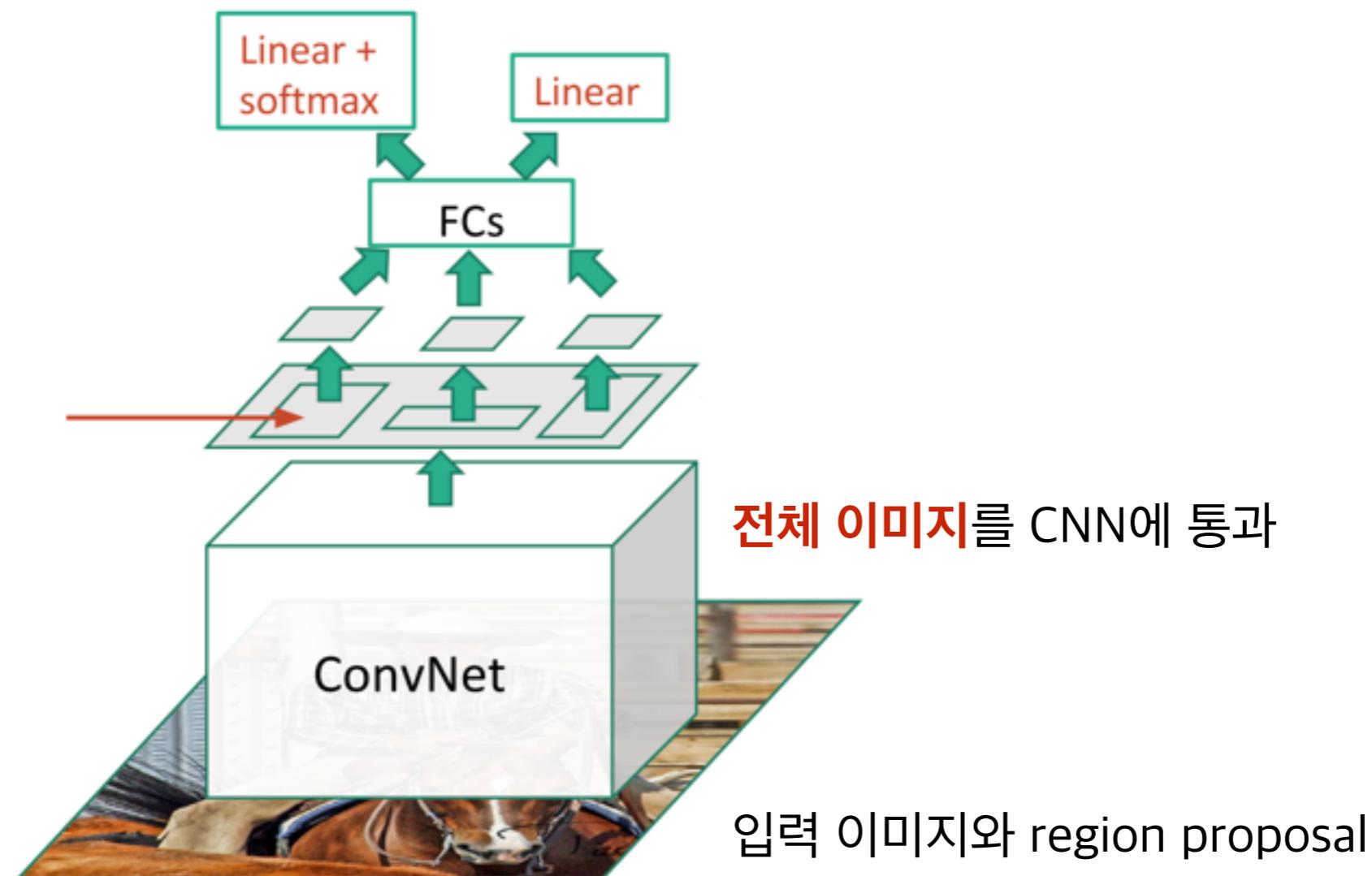
# Fast R-CNN

- ROI pooling
  - R-CNN은 제안된 모든 영역을 전부 CNN에 돌리기 때문에 느린 구조
  - -> 이미지를 CNN에 **한 번만 통과** 시킨 뒤 관심 영역으로 쪼개자
  - **ROI pooling** 레이어의 도입
- End-to-end 학습
  - R-CNN은 SVM, 선형 회귀를 학습하기 위해 다단계의 학습법이 필요
  - 이를 한 번에 학습하기 위해 크로스엔트로피 + L1 loss를 결합

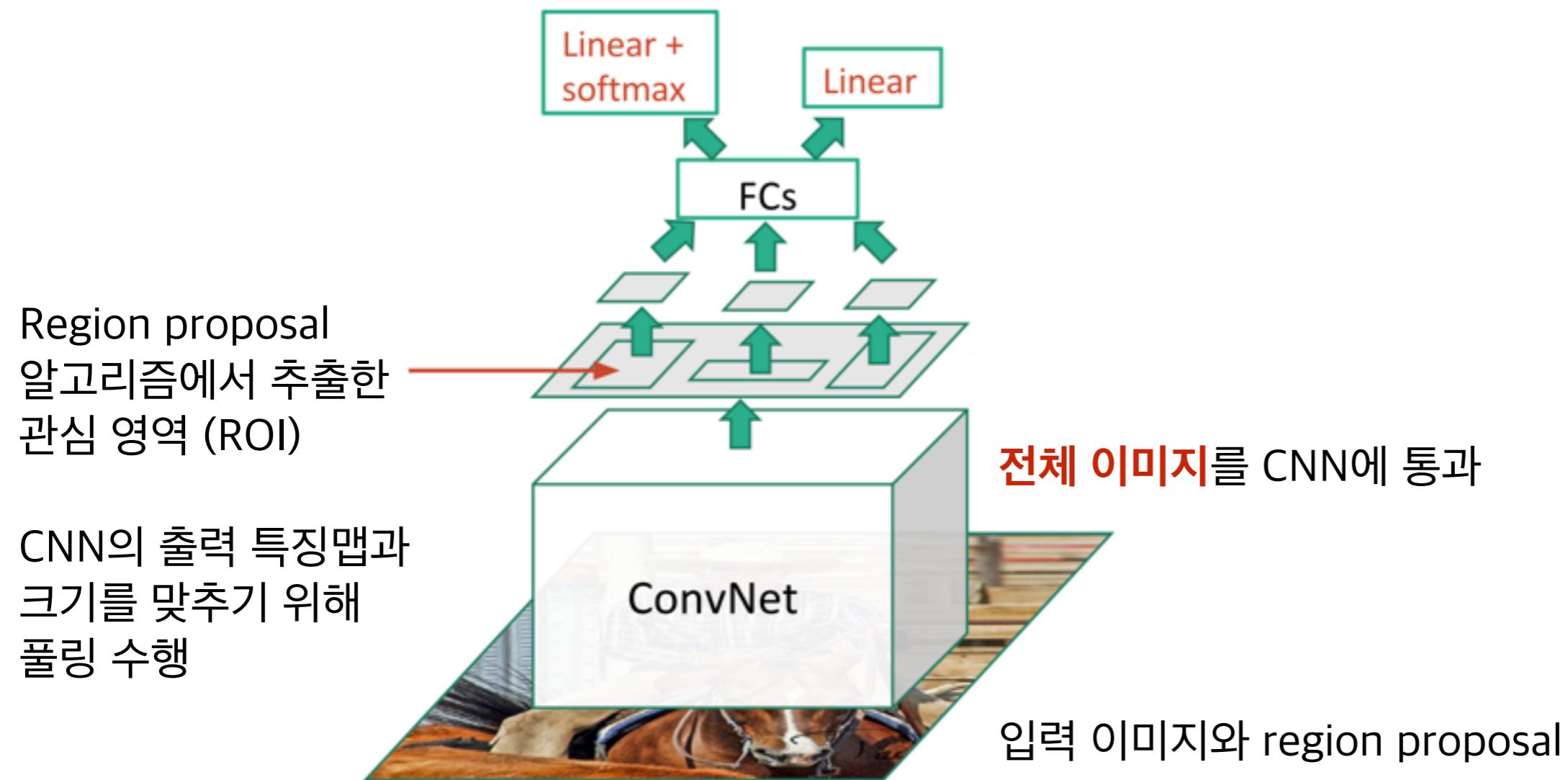
# Fast R-CNN



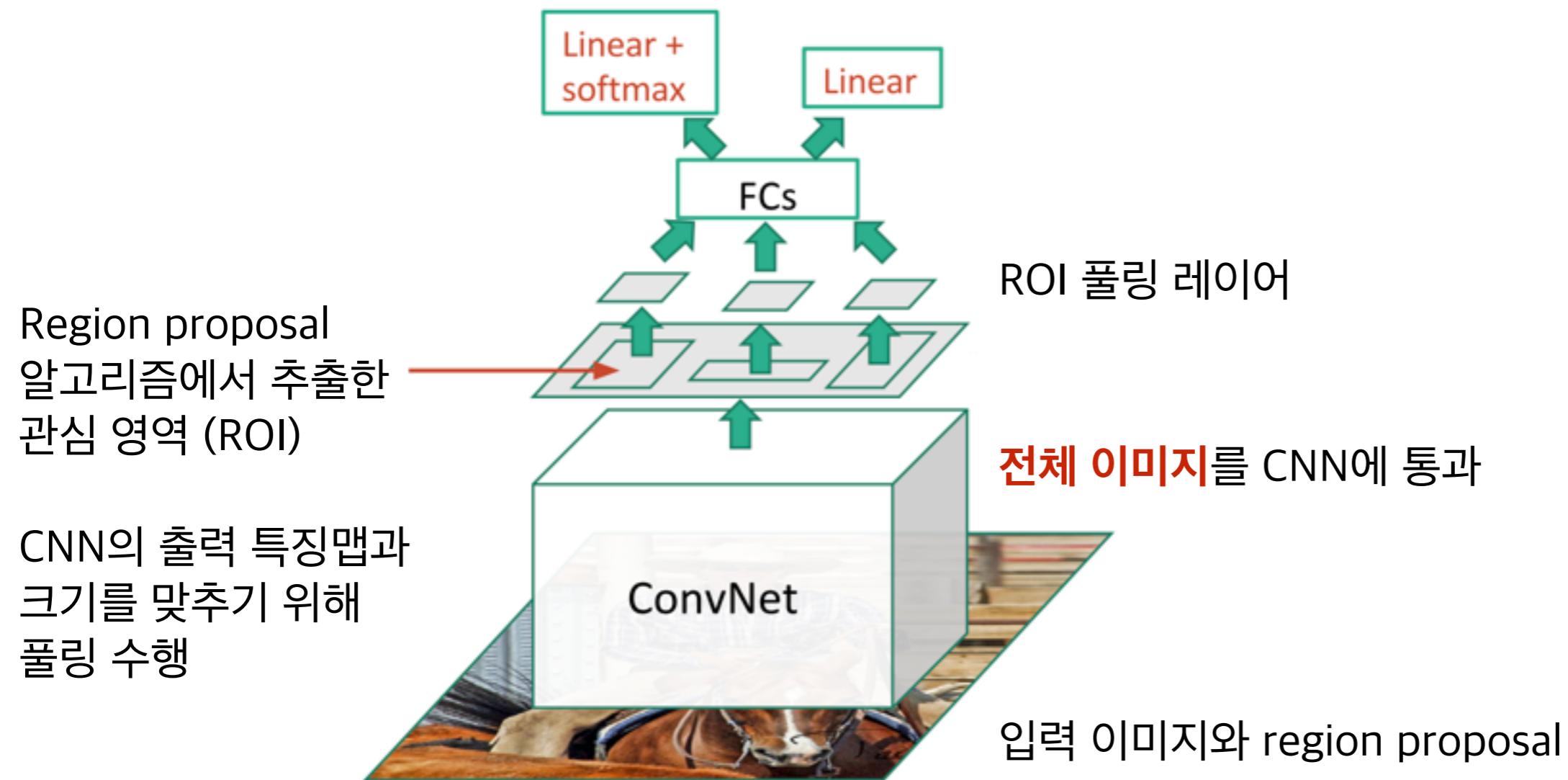
# Fast R-CNN



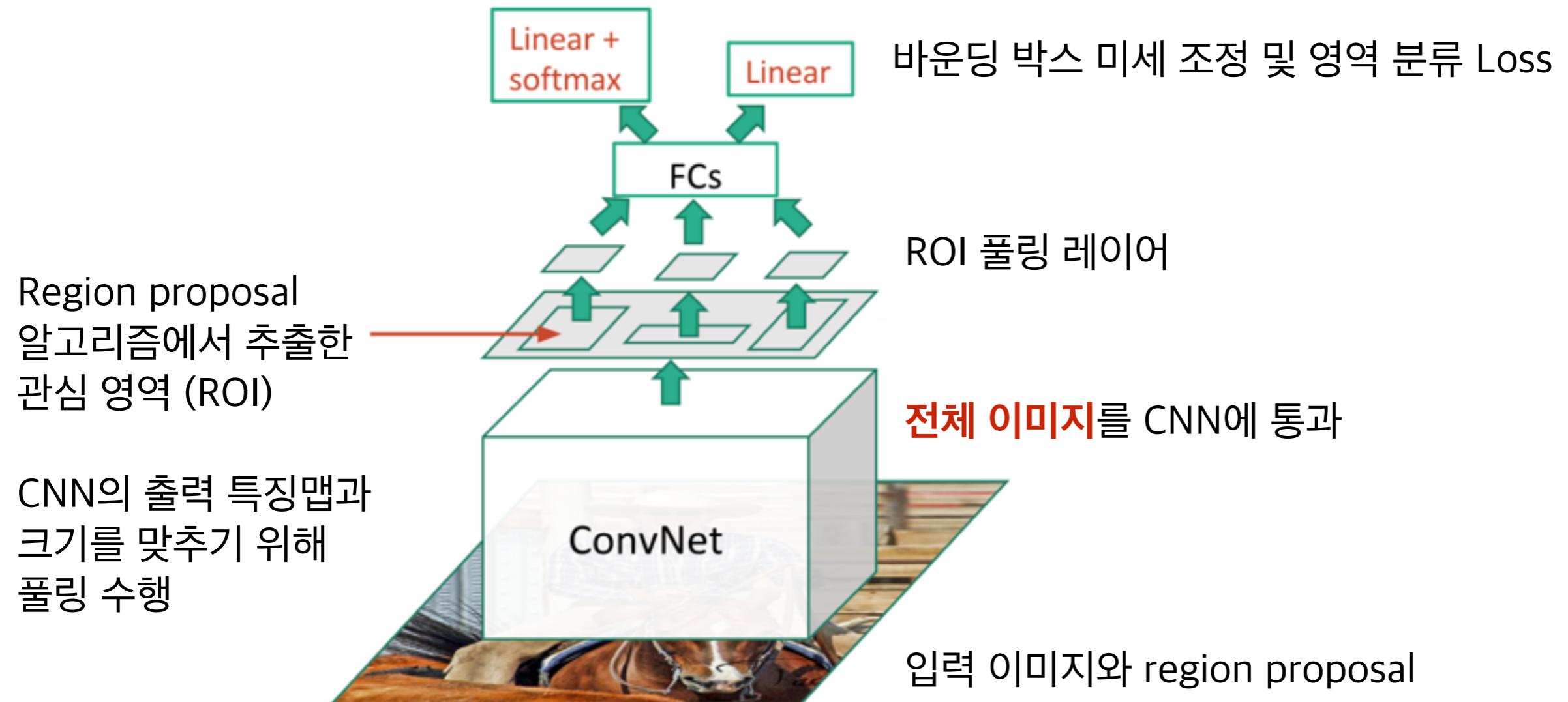
# Fast R-CNN



# Fast R-CNN

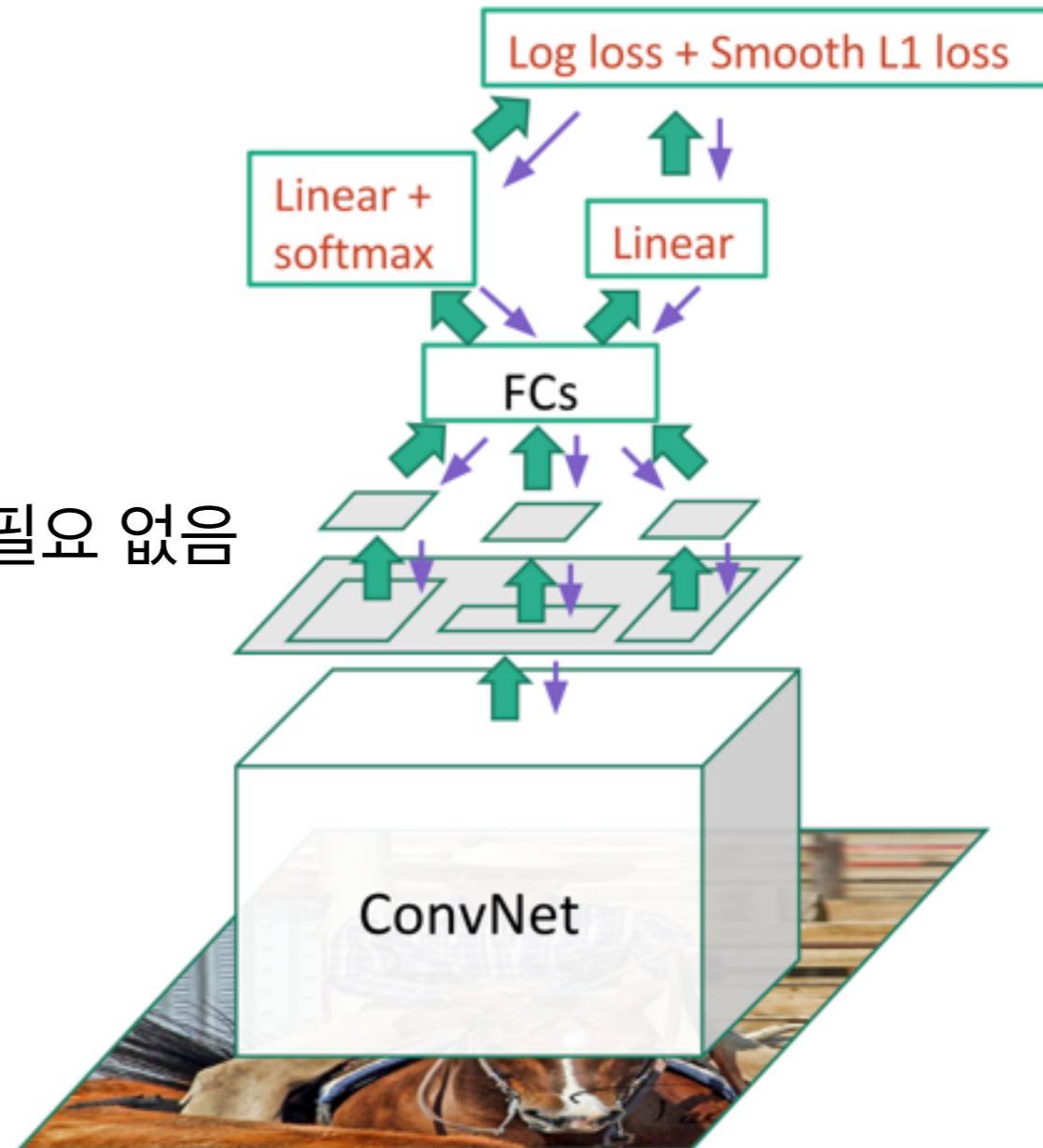


# Fast R-CNN

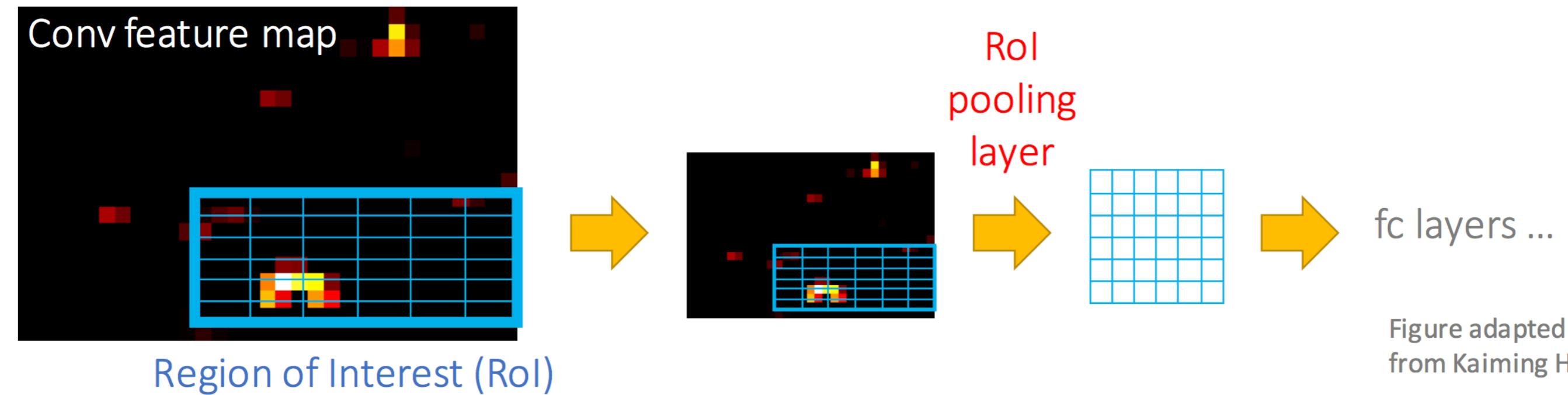


# Fast R-CNN

- End-to-end 방식으로 학습 가능
  - Loss = cls loss + bbox loss
  - 추출한 특징맵을 디스크에 별도로 저장할 필요 없음
- CNN은 한 번만 활용
  - ROI 풀링 레이어가 이미지 리사이즈



# ROI 풀링 레이어



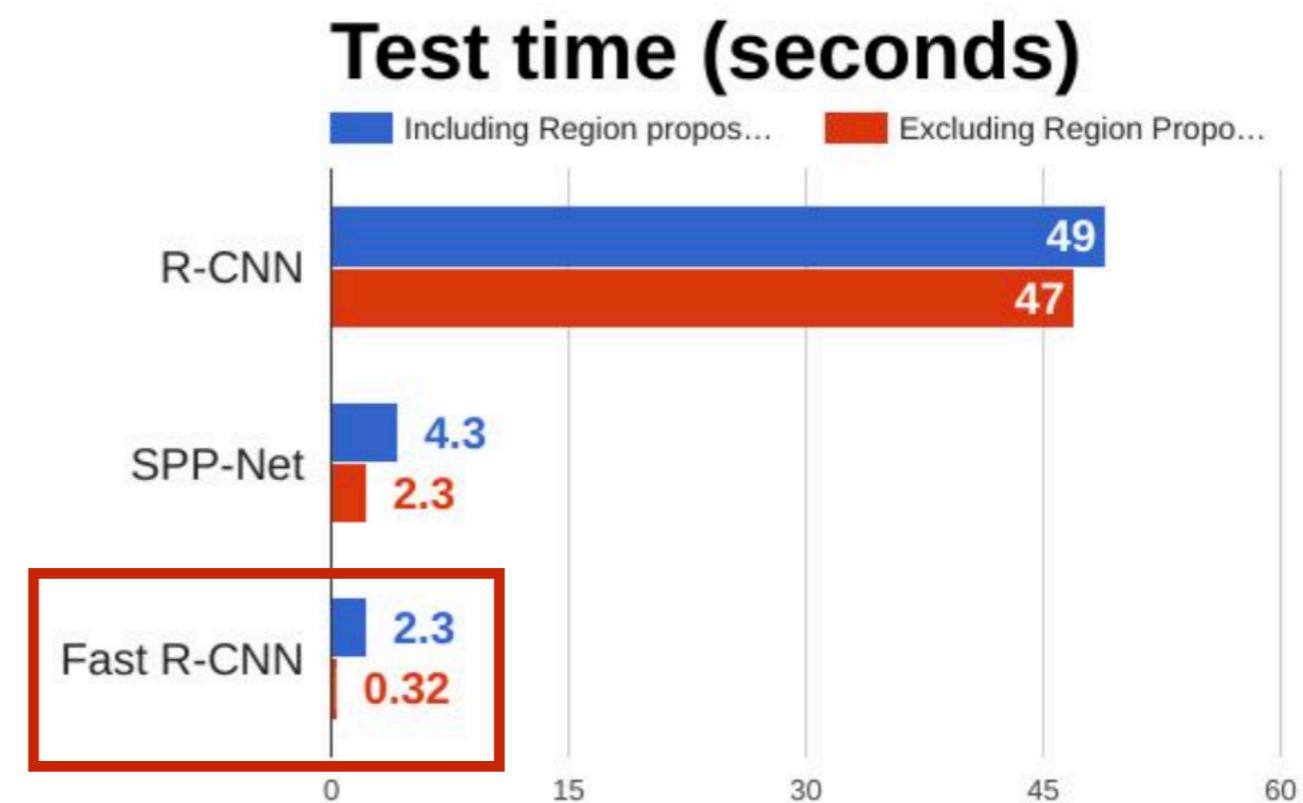
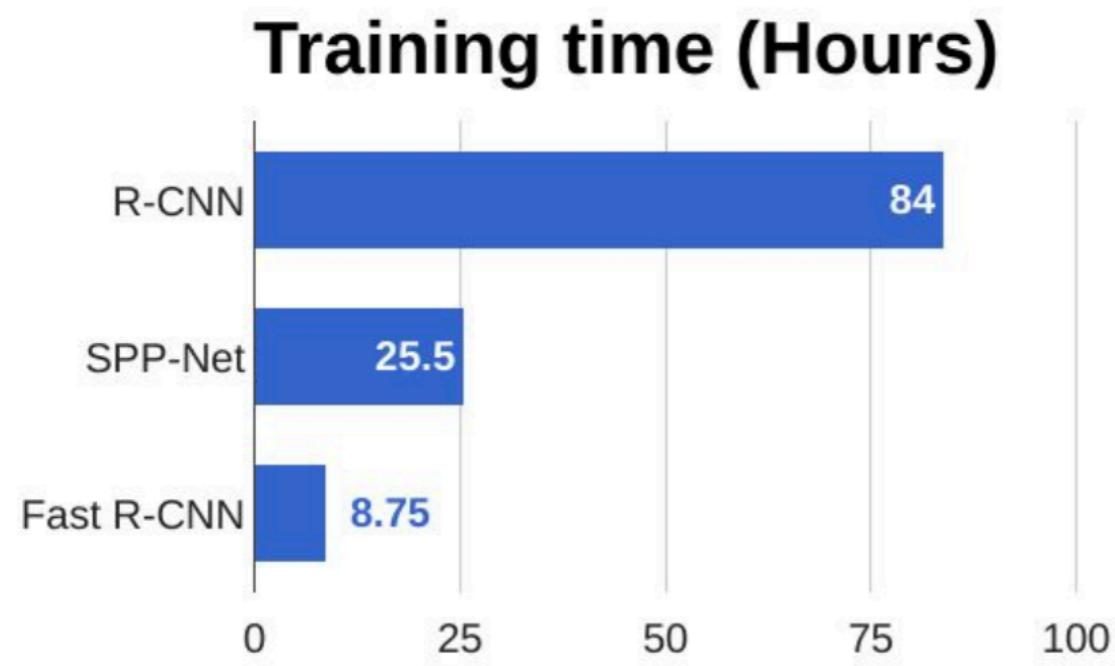
- FC 레이어에 입력되는 **정해진 이미지 크기와 맞추기 위해** 풀링을 통해 동적으로 크기를 변환하는 과정
- 예시)
- ROI 특징맵  $21 \times 14 \rightarrow 3 \times 2$  맥스 풀링, stride(3, 2)  $\rightarrow 7 \times 7$  출력 특징맵
- ROI 특징맵  $35 \times 42 \rightarrow 5 \times 6$  맥스 풀링, stride(5, 6)  $\rightarrow 7 \times 7$  출력 특징맵

Girshick, Ross. "Fast r-cnn." ICCV. 2015.

Girshick, Ross. "Training R-CNNs of various velocities" ICCV Tutorial 2015.

Jinwon Lee. "PR-012: Faster R-CNN" <https://youtu.be/kcPAGIgBGRs>

# R-CNN vs Fast R-CNN



이제 region proposal 알고리즘에 병목이 걸림

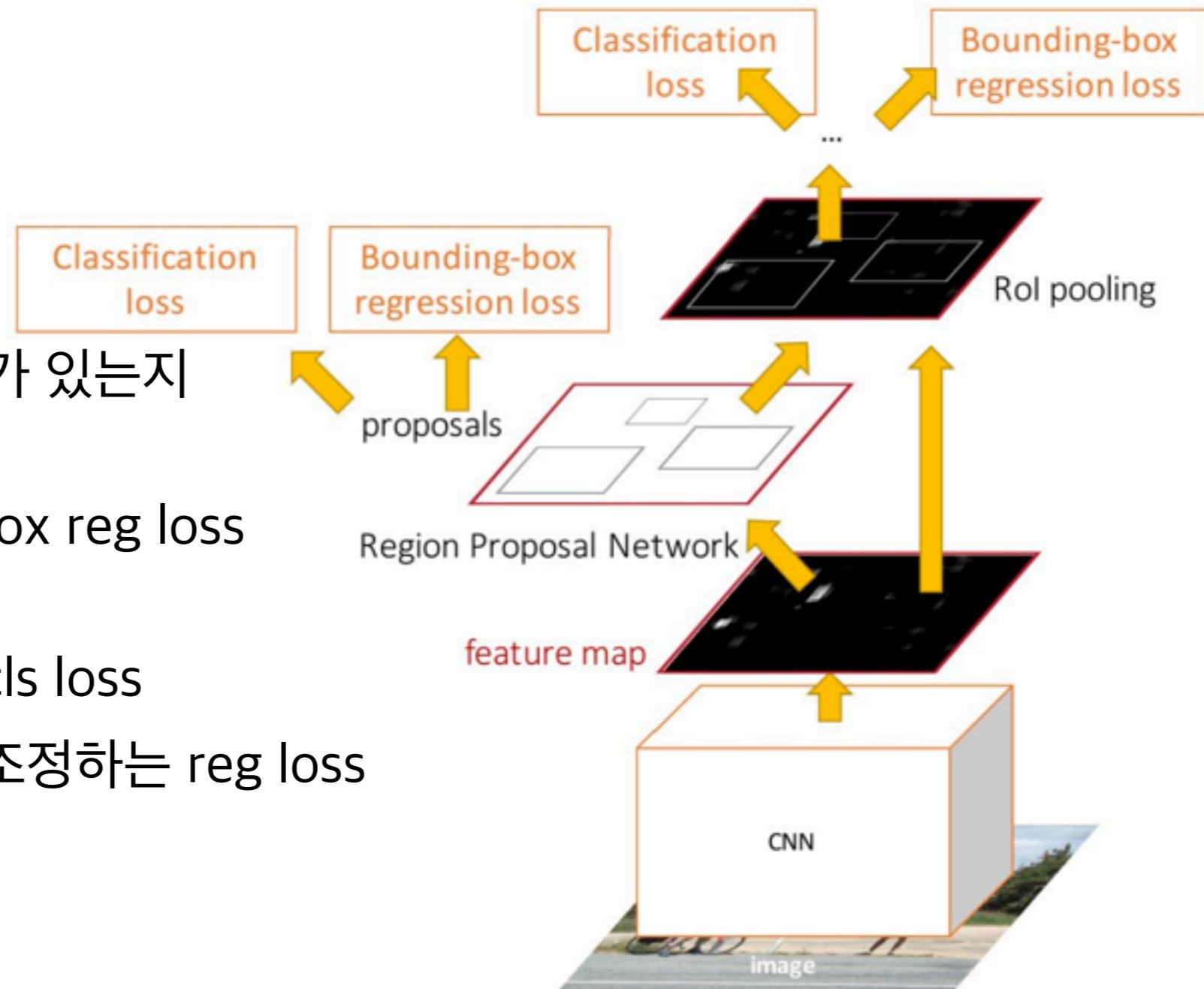
# Faster R-CNN

- Fast R-CNN의 한계
  - Faster R-CNN은 이제 외부 region proposal 알고리즘이 병목이 됨
  - 또한 region proposal 알고리즘은 CPU에서만 동작
- Region Proposal Network (RPN)
  - ROI를 예측하는 새로운 네트워크를 만들어 GPU에서 동작하게 구성
  - RPN은 직접적으로 ROI를 예측하도록 학습하기 때문에 별도의 외부 region proposal 알고리즘이 필요하지 않음
  - CNN을 통과한 특징맵과 RPN의 결과를 결합하여 ROI 풀링 레이어에 입력

# Faster R-CNN

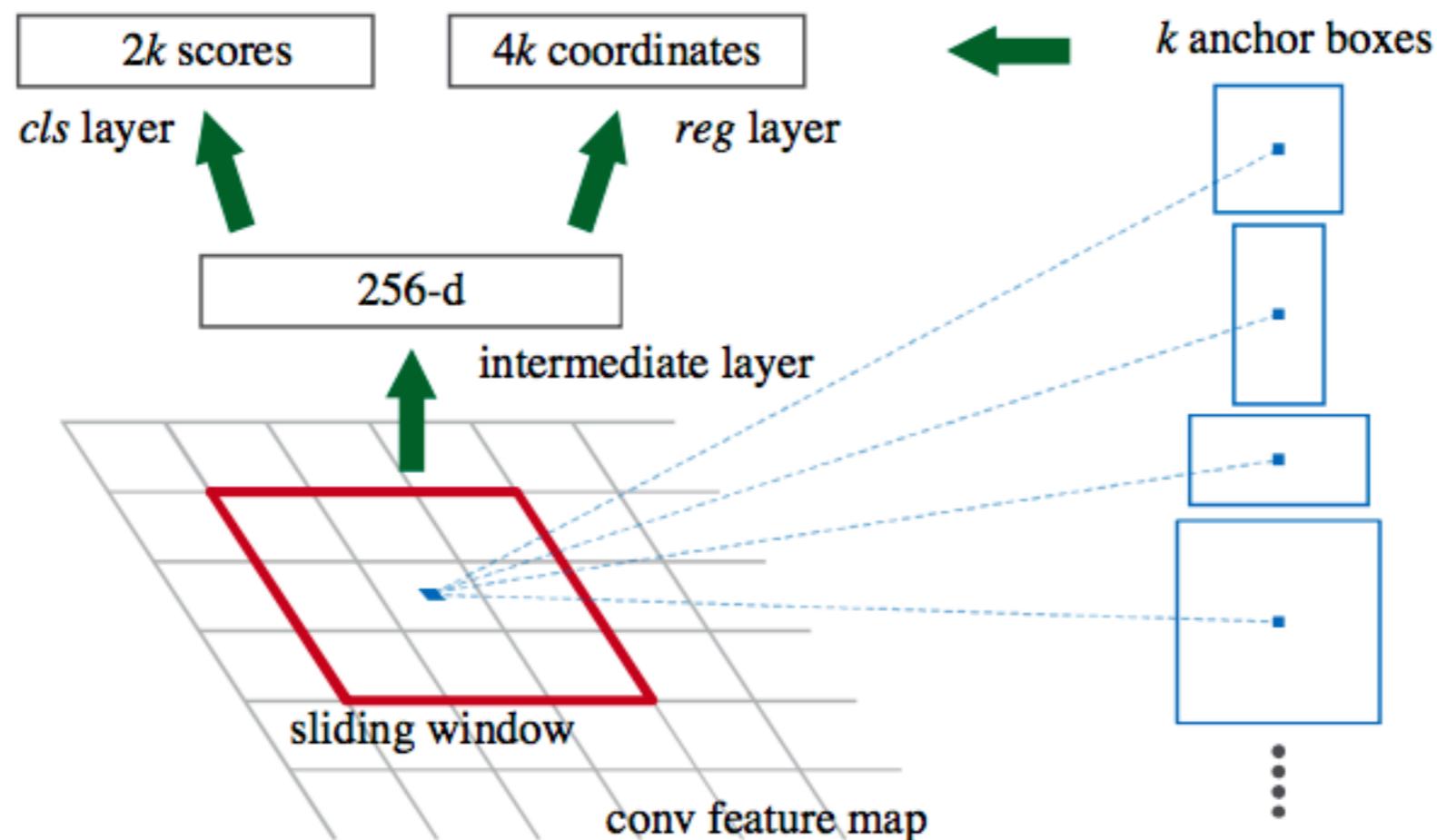
- 학습할 LOSS:

1. RPN에서 현재 영역에 물체가 있는지 없는지 판단하는 cls loss
2. RPN의 위치를 결정하는 bbox reg loss (간략한 위치를 결정)
3. 물체의 클래스를 판단하는 cls loss
4. 바운딩 박스의 위치를 미세조정하는 reg loss

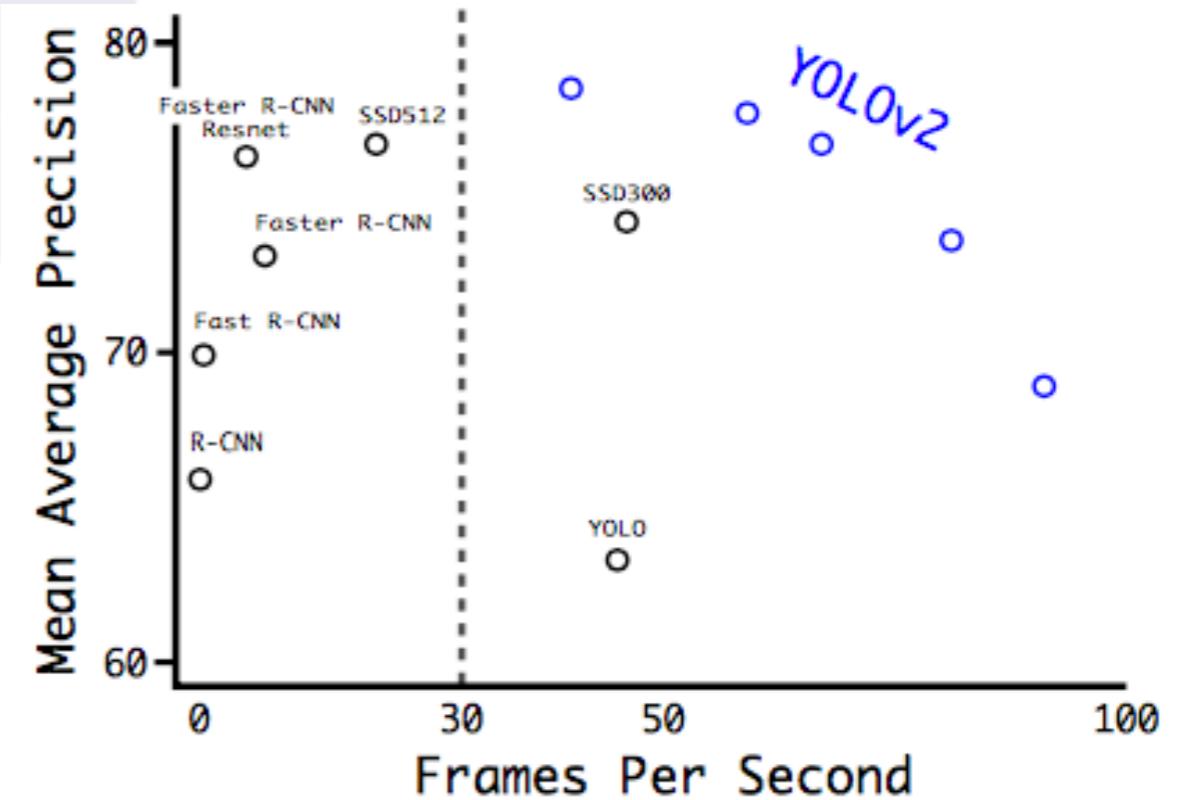


# RPN

- 각 셀은  $k$  개의 디폴트 앵커 박스를 갖고 있음
  - 개략적인 바운딩 박스의 모양을 미리 초기 박스로 설정 해놓음
  - 디폴트 앵커 박스를 미세 조정하는 방식으로 학습



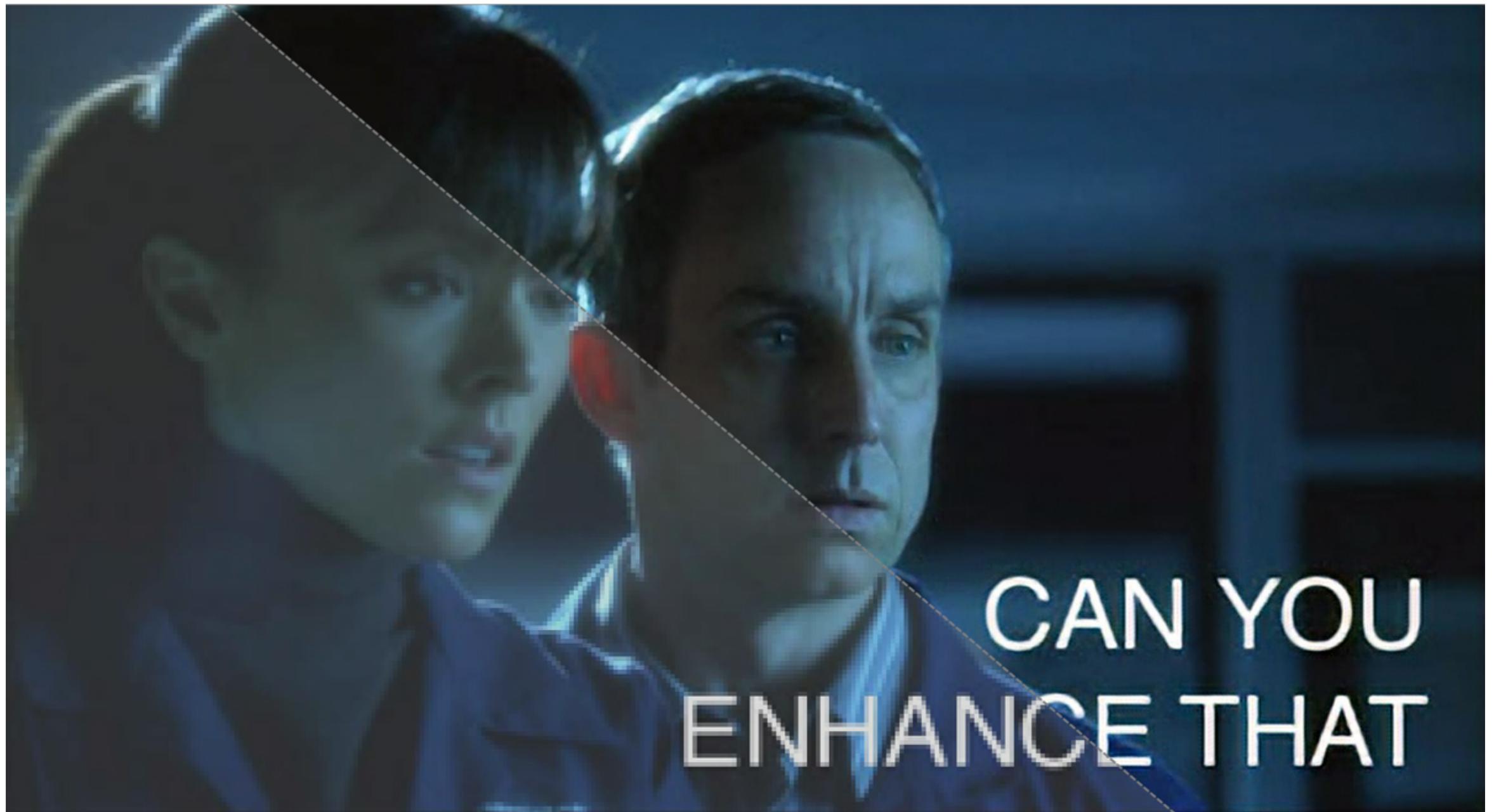
# Detection 성능 그래프



Huang, Jonathan, et al. "Speed/accuracy trade-offs for modern convolutional object detectors." arXiv preprint arXiv:1611.10012 (2016).

Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." arXiv preprint arXiv:1612.08242 (2016).

# Super-resolution



CAN YOU  
ENHANCE THAT

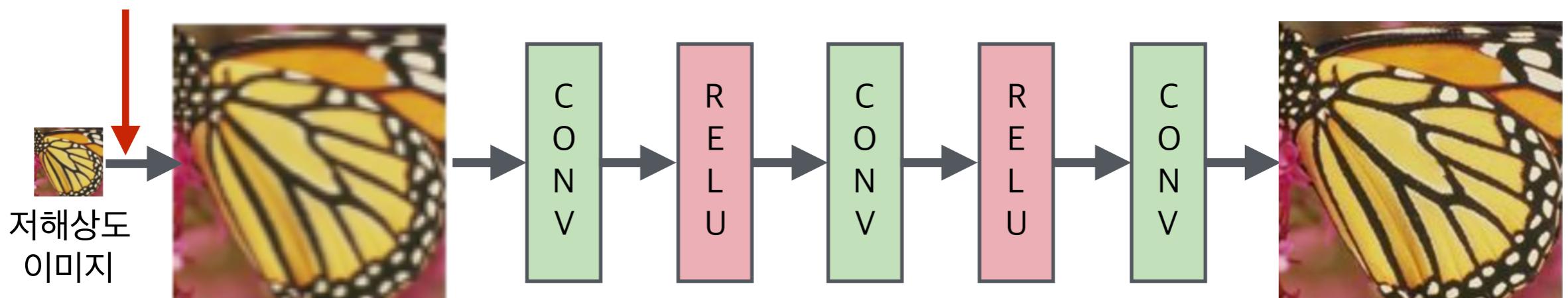
# Super-resolution

- 저해상도 이미지를 고해상도로 복원하는 작업
- 흔히 사용하는 SR 접근 방식:
  - 단일 이미지 SR: 저해상도 이미지 한장을 통해 고해상도 복원
  - 멀티 이미지 SR: 여러 장의 이미지를 사용해 복원 (서로 다른 뷰 등등)
- 최근 딥러닝을 적용한 SR 방식은 대부분 **단일 이미지 SR**

# SRCNN

Super Resolution CNN

bicubic 방법으로 이미지 확대  
(일반적인 이미지 리사이즈 방법)



- Loss 함수

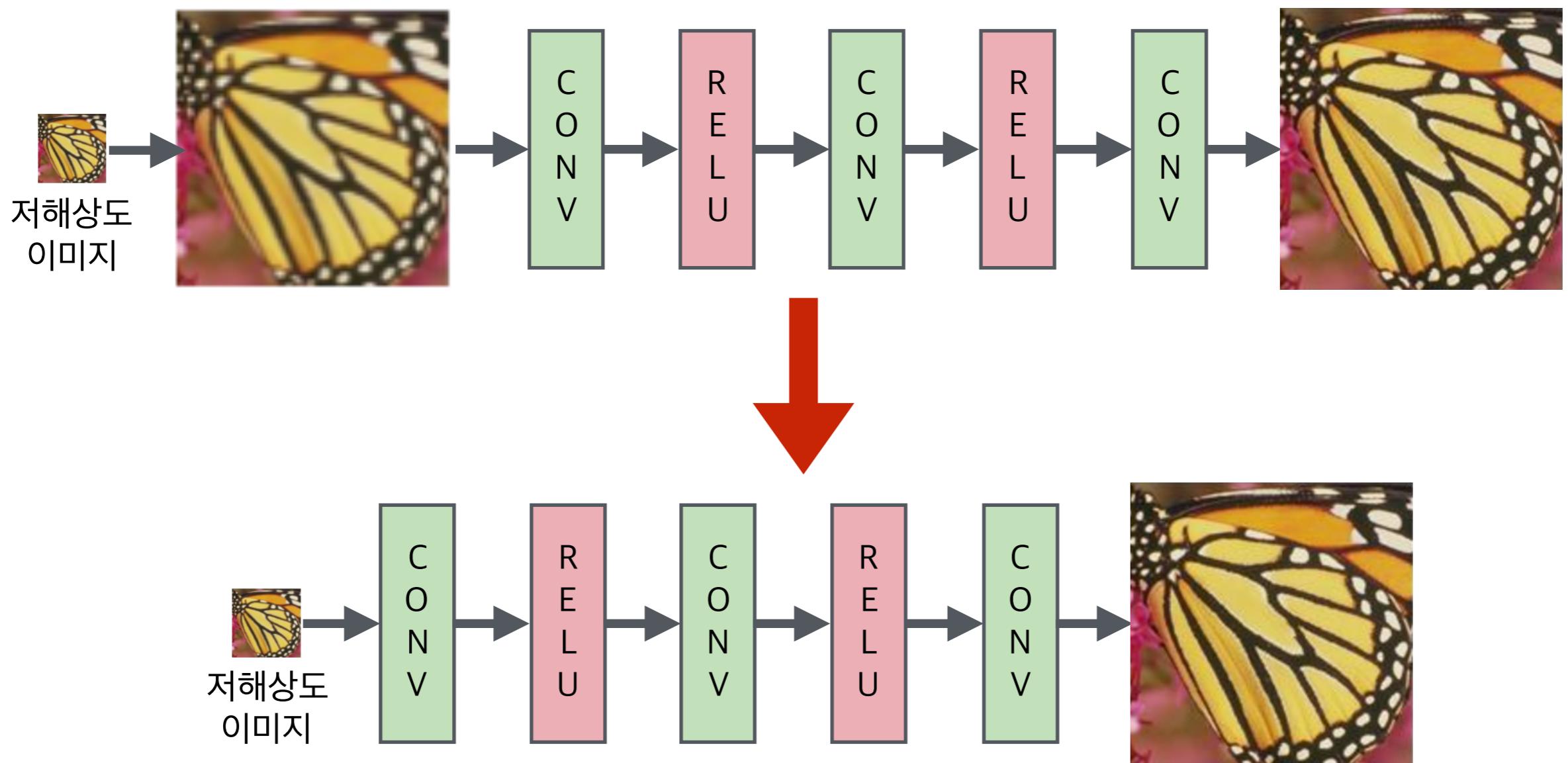
- 복원 이미지와 원본 이미지의 차이를 MSE를 통해 계산
- SGD를 통해 네트워크 학습

$$L(\Theta) = \frac{1}{n} \sum_{i=1}^n \|F(\mathbf{Y}_i; \Theta) - \mathbf{X}_i\|^2,$$

# SRCNN

- 큰 크기의 필터가 더 좋은 성능을 보임
  - [9x9 -> 5x5 -> 5x5] 네트워크가 [9x9 -> 1x1 -> 5x5] 보다 좋은 성능
  - 큰 크기의 필터는 더 넓은 **receptive field**를 가지며,  
넓은 receptive field는 이미지 해상도 복원에서 매우 중요
  - 이미지를 복원할 때 주변 픽셀들을 함께 고려해야 하기 때문
- 네트워크를 깊게 쌓는 것이 큰 효과가 없음
  - 3 레이어 네트워크가 가장 좋은 성능

# FAST-SRCNN



# FAST-SRCNN

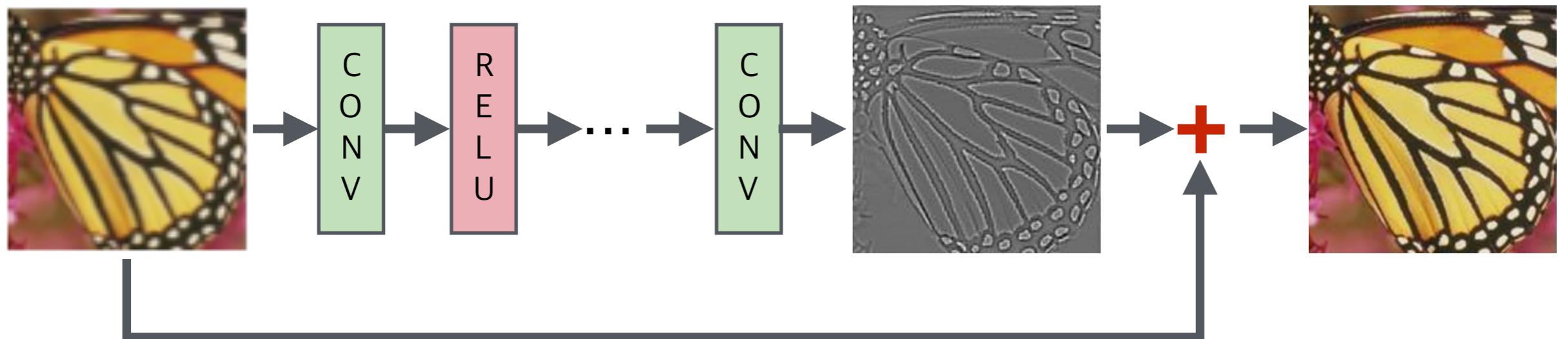
- 입력 이미지는 저해상도 이미지 (1번 모델)
  - 연산 속도가 기존 SRCNN 모델 대비 8배 빠름
  - 마지막 레이어를 deconv로 구성하여 원본 해상도로 크기를 키움
- 파라미터 수 줄이기 (2번 모델, 3번 모델)
  - Bottleneck 방식을 이용하여 파라미터와 연산 속도를 대폭 감소
  - 그에 따라 더 깊은 모델을 구성하여 성능 증가

	SRCNN-Ex	Transition State 1	Transition State 2	FSRCNN (56,12,4)
First part	Conv(9,64,1)	Conv(9,64,1)	Conv(9,64,1)	<b>Conv(5,56,1)</b>
Mid part	Conv(5,32,64)	Conv(5,32,64)	<b>Conv(1,12,64)- 4Conv(3,12,12) -Conv(1,64,12)</b>	<b>Conv(1,12,56)- 4Conv(3,12,12) -Conv(1,56,12)</b>
Last part	Conv(5,1,32)	<b>DeConv(9,1,32)</b>	<b>DeConv(9,1,64)</b>	<b>DeConv(9,1,56)</b>
Input size	$S_{HR}$	$S_{LR}$	$S_{LR}$	$S_{LR}$
Parameters	57184	58976	17088	12464
Speedup	$1 \times$	$8.7 \times$	$30.1 \times$	$41.3 \times$
PSNR (Set5)	32.83 dB	32.95 dB	33.01 dB	33.06 dB

# VDSR

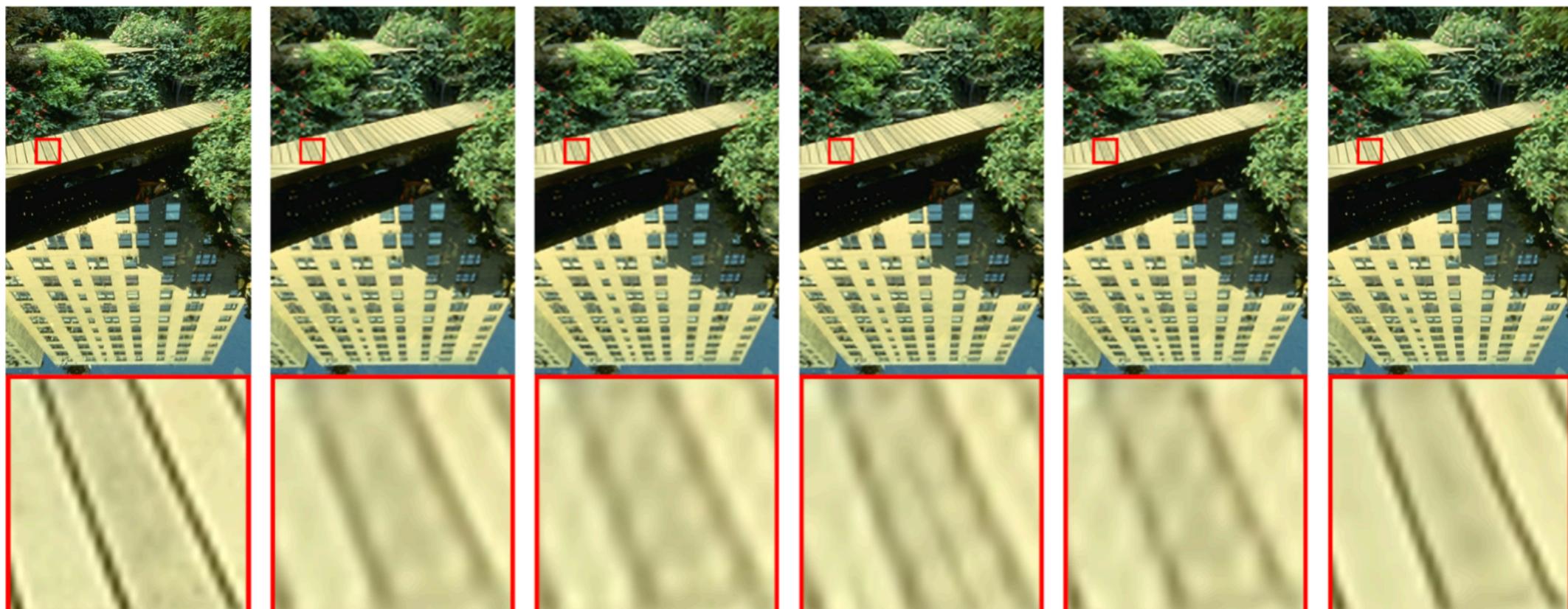
Very Deep Super Resolution

Residual 이미지



# VDSR

- Residual 이미지
  - 이미지를 바닥부터 생성하는 일은 어렵고 수렴도 잘 안됨
  - 원본과 저해상도 이미지의 차이인 **residual 이미지**를 생성한 뒤, residual 이미지를 저해상도 이미지와 합하여 이미지 복원
  - 간단하지만 깊은 네트워크에서도 효과적으로 수렴이 가능
- 깊은 레이어 사용
  - VGGNet 스타일의 20 레이어의 모델 사용
  - 각 필터의 receptive field는 작지만 전체 receptive field는 넓음
  - SRCNN은 13x13, VDSR은 41x41



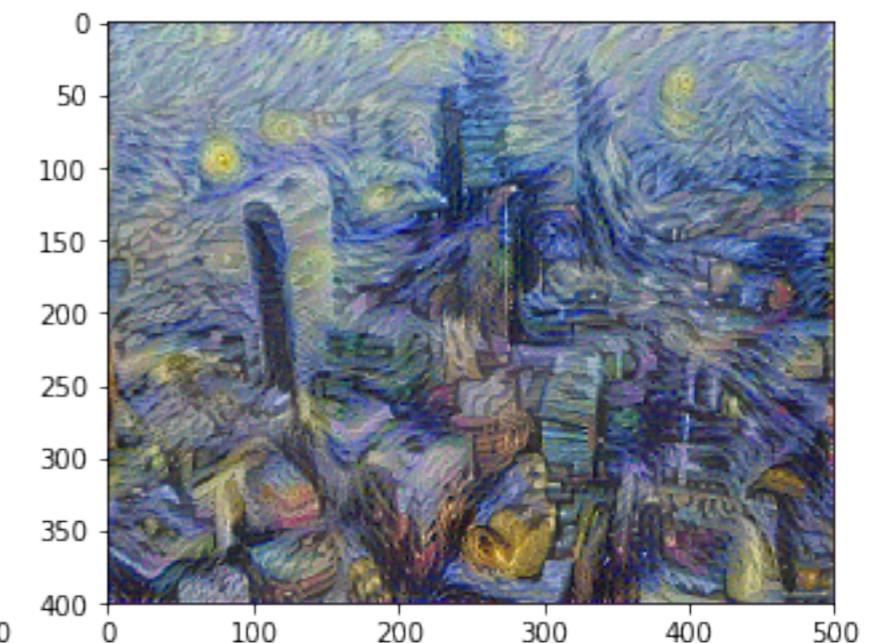
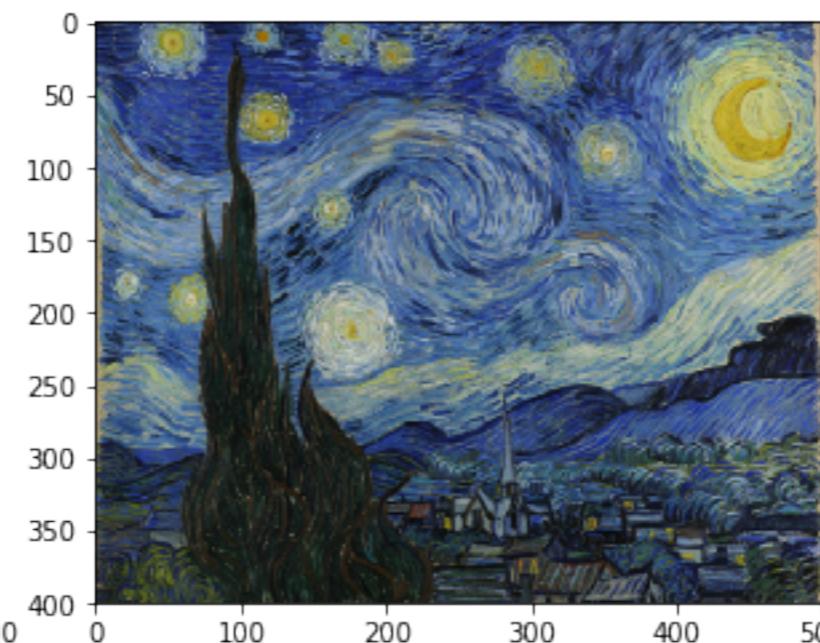
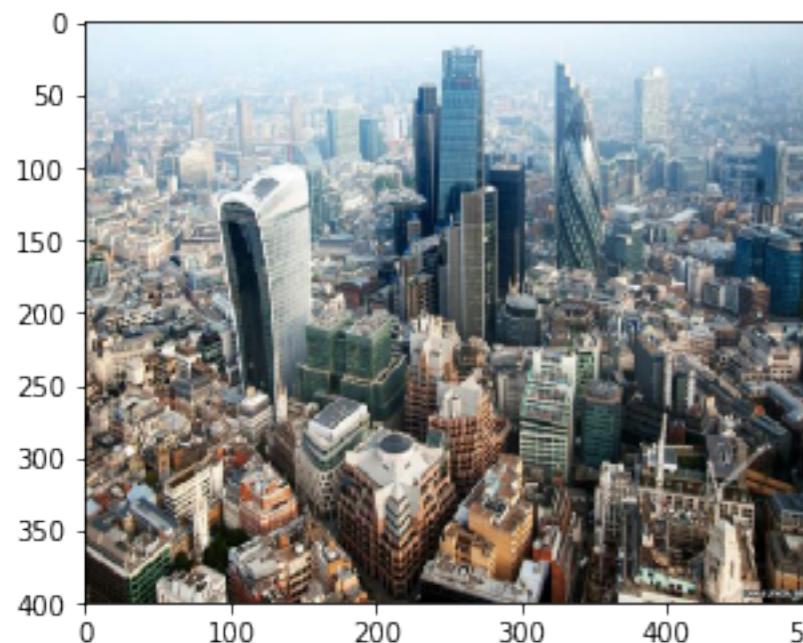
Ground Truth (PSNR, SSIM)	A+ [22] (22.92, 0.7379)	RFL [18] (22.90, 0.7332)	SelfEx [11] (23.00, 0.7439)	SRCNN [5] (23.15, 0.7487)	VDSR (Ours) (23.50, 0.7777)
------------------------------	----------------------------	-----------------------------	--------------------------------	------------------------------	--------------------------------



n Ground Truth (PSNR, SSIM)	A+ [22] (27.08, 0.7514)	RFL [18] (27.08, 0.7508)	SelfEx [11] (27.02, 0.7513)	SRCNN [5] (27.16, 0.7545)	VDSR (Ours) (27.32, 0.7606)
--------------------------------	----------------------------	-----------------------------	--------------------------------	------------------------------	--------------------------------

# Style transfer

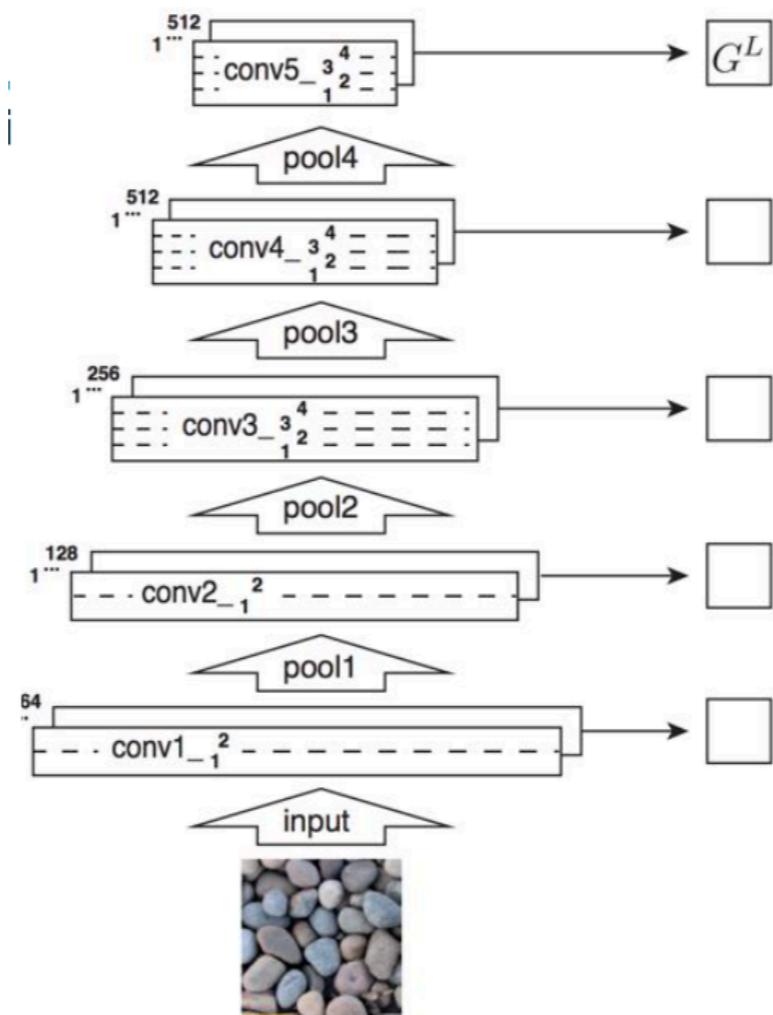
- Style 이미지와 Content 이미지를 통해 새로운 이미지 생성
  - Style 이미지와 생성된 이미지 간의 **style loss** 계산
  - Content 이미지와 생성된 이미지 간의 **content loss** 계산
  - 전체 loss는 style\_loss와 content\_loss의 혼합



# Style 합성

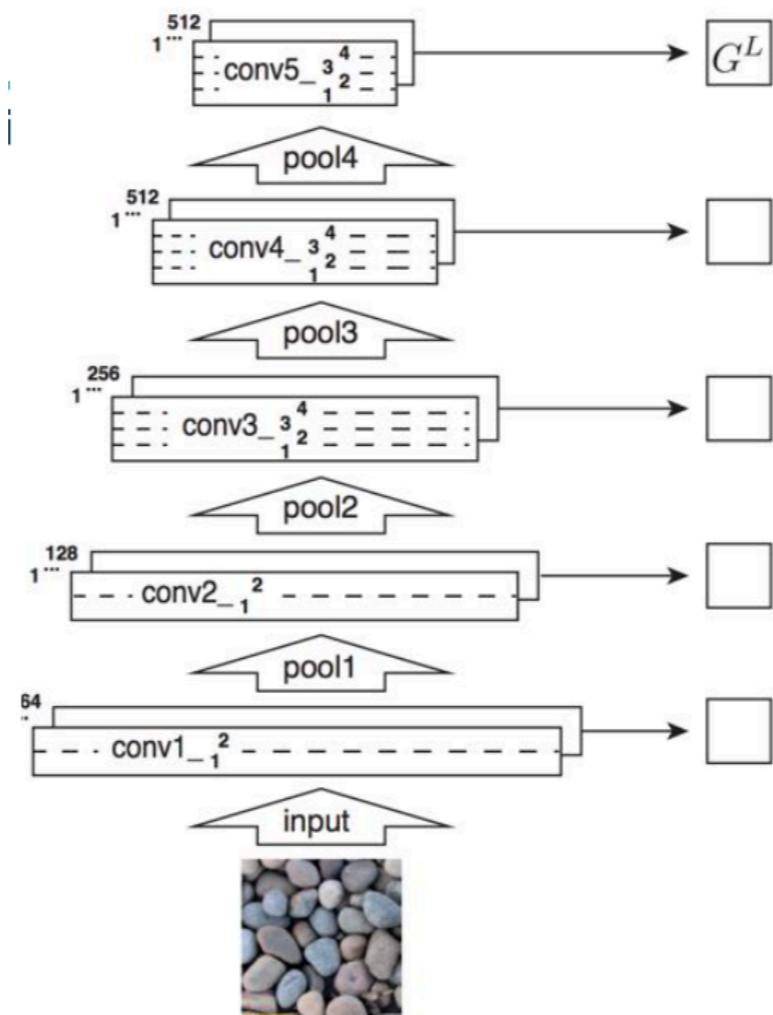
# Style 합성

1. 사전학습된 VGG-19
2. 입력 이미지를 CNN에 전파시킨 후, 모든 레이어에서 레이어의 결과값을 뽑음



# Style 합성

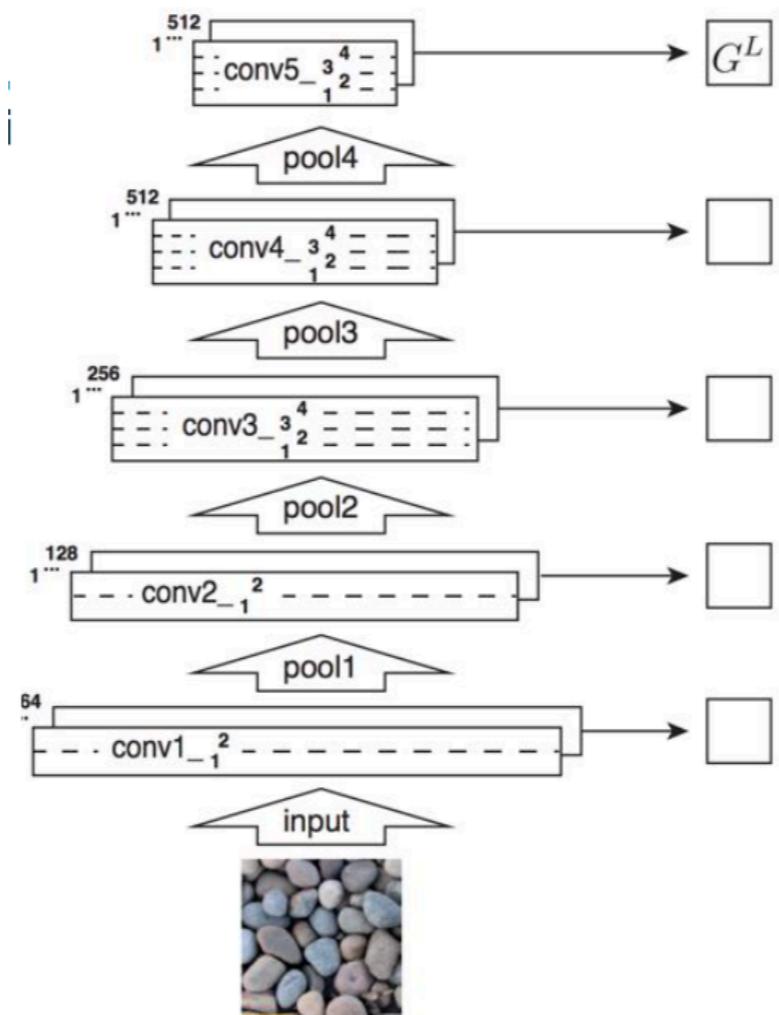
1. 사전학습된 VGG-19
2. 입력 이미지를 CNN에 전파시킨 후, 모든 레이어에서 레이어의 결과값을 뽑음
3. 뽑혀진 결과값을 통해 **gram 행렬**을 계산



# Style 합성

1. 사전학습된 VGG-19
2. 입력 이미지를 CNN에 전파시킨 후, 모든 레이어에서 레이어의 결과값을 뽑음
3. 뽑혀진 결과값을 통해 **gram 행렬**을 계산

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l (\text{shape } C_i \times H_i)$$

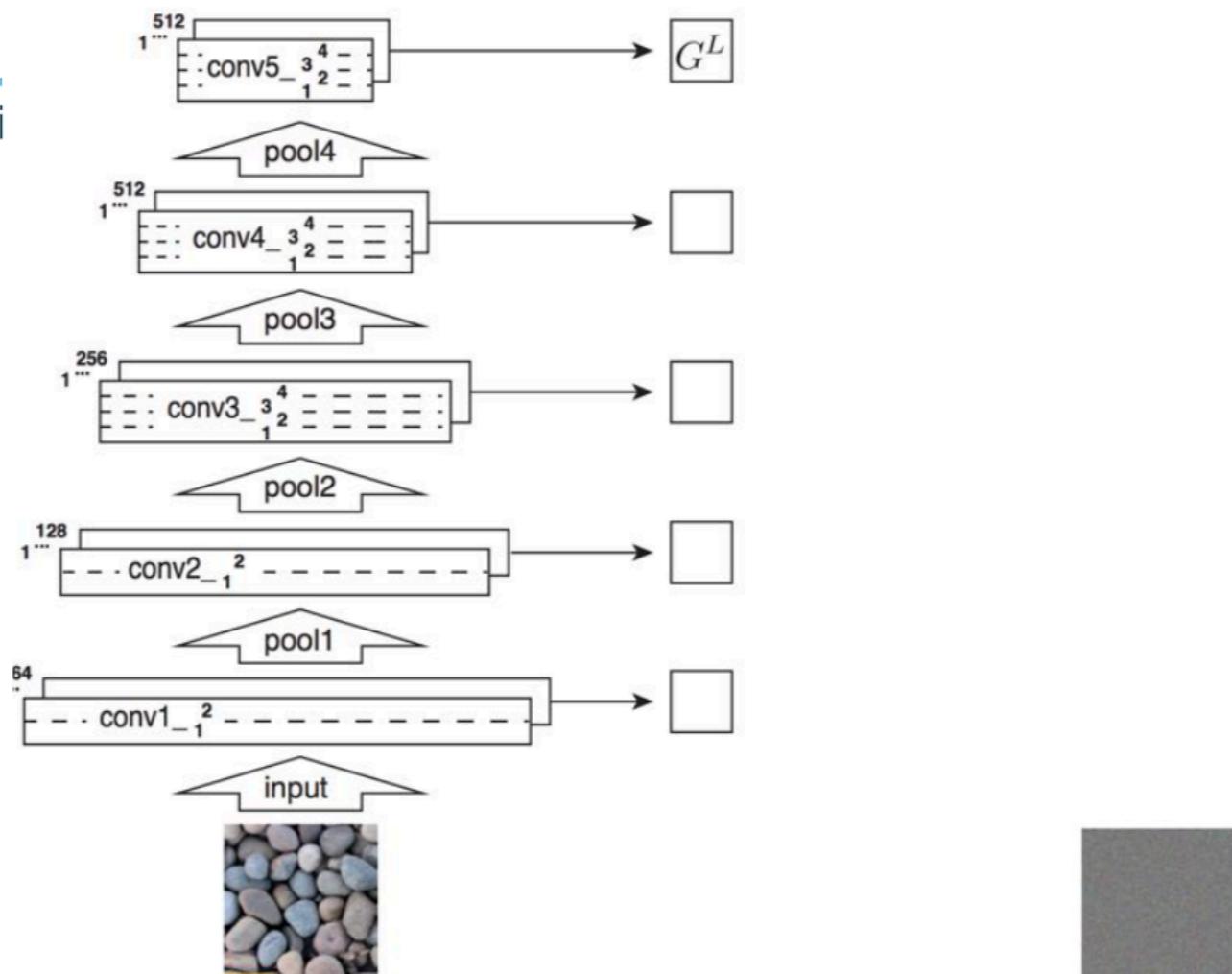


# Style 합성

1. 사전학습된 VGG-19
2. 입력 이미지를 CNN에 전파시킨 후, 모든 레이어에서 레이어의 결과값을 뽑음
3. 뽑혀진 결과값을 통해 **gram 행렬**을 계산

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l (\text{shape } C_i \times H_i)$$

4. 노이즈로 부터 생성할 이미지 생성

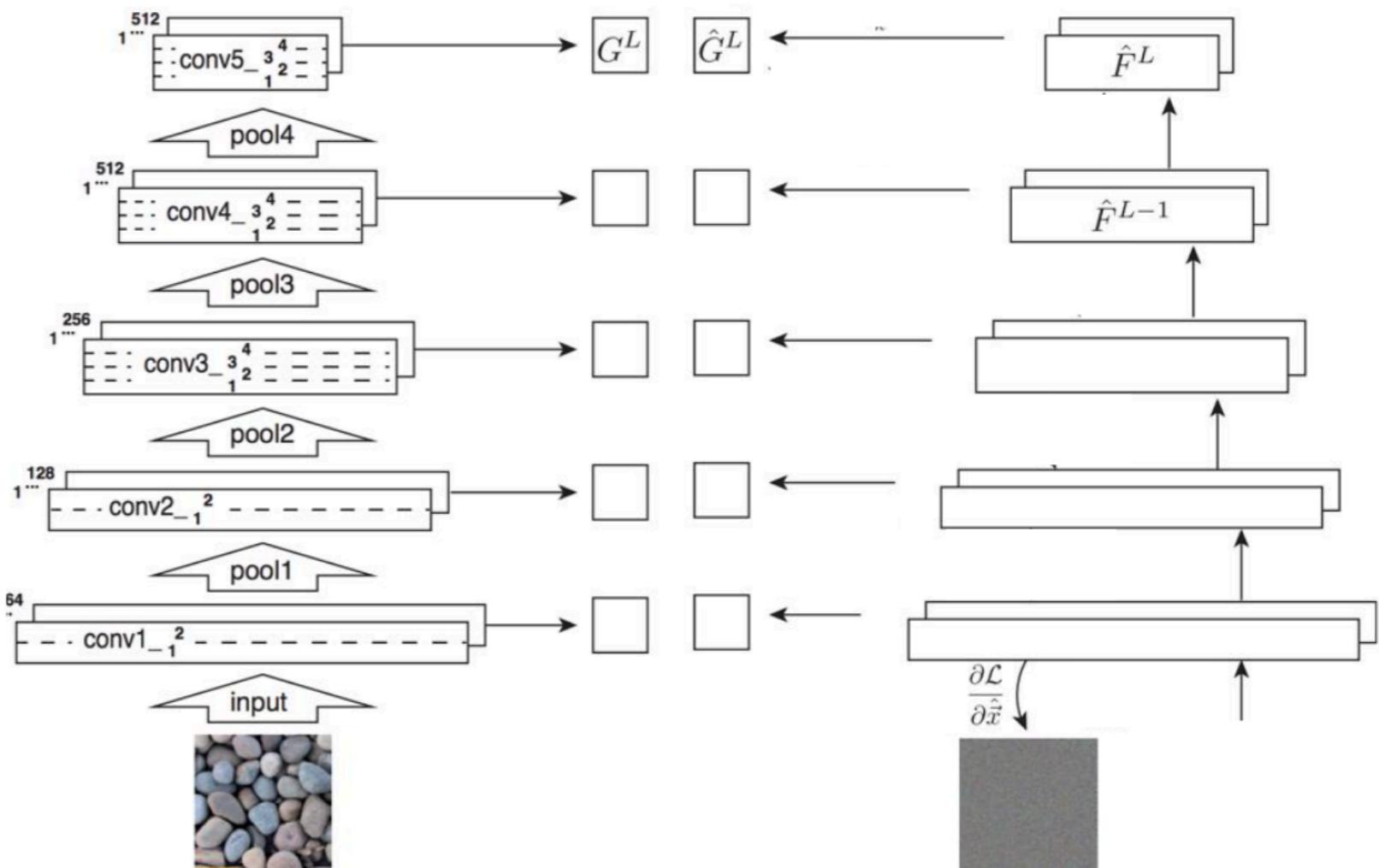


# Style 합성

1. 사전학습된 VGG-19
2. 입력 이미지를 CNN에 전파시킨 후, 모든 레이어에서 레이어의 결과값을 뽑음
3. 뽑혀진 결과값을 통해 **gram 행렬**을 계산

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l (\text{shape } C_i \times H_i)$$

4. 노이즈로 부터 생성할 이미지 생성
5. 생성 이미지로 gram 행렬 계산



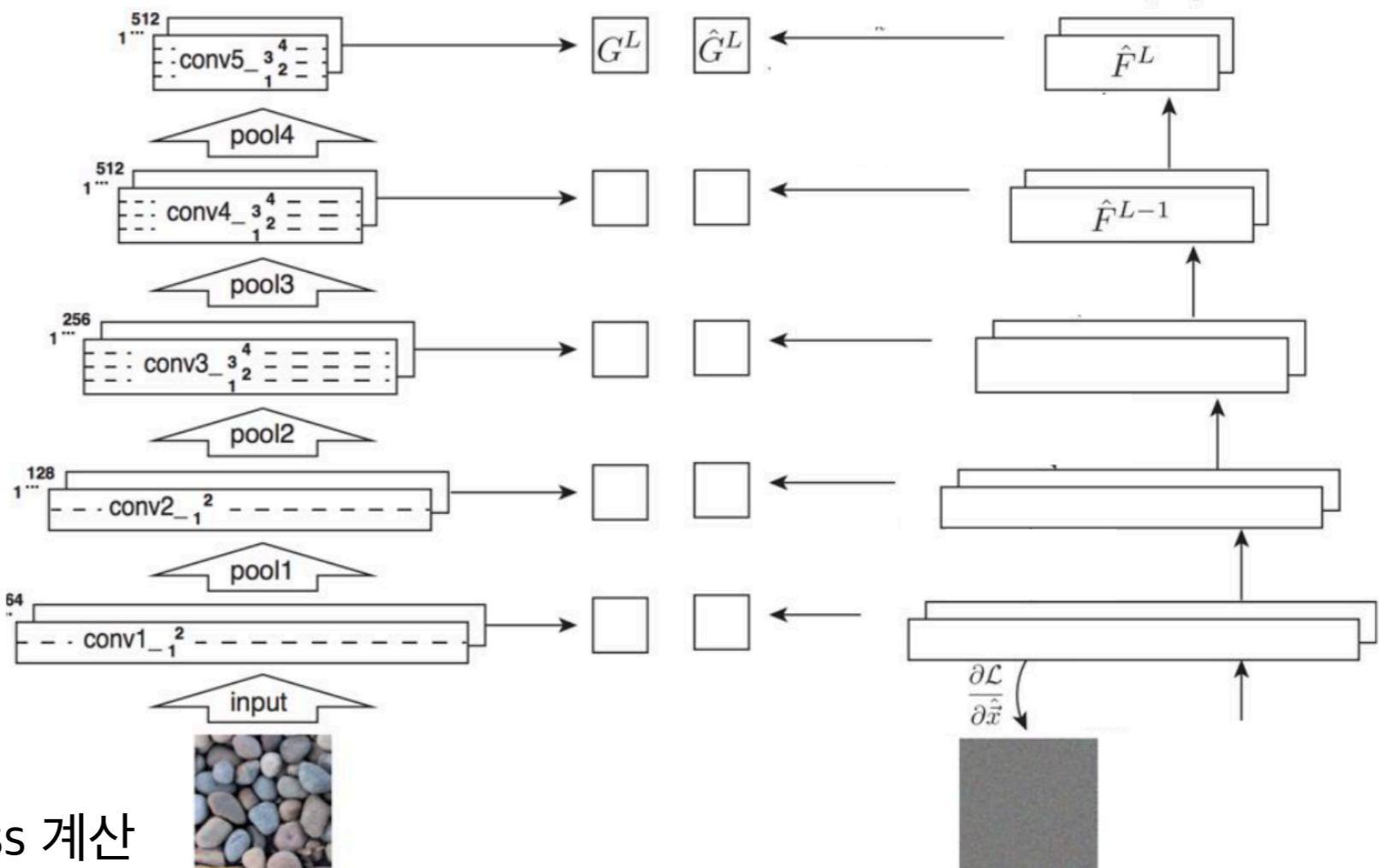
# Style 합성

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - \hat{G}_{ij}^l)^2 \quad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^L w_l E_l$$

1. 사전학습된 VGG-19
2. 입력 이미지를 CNN에 전파시킨 후, 모든 레이어에서 레이어의 결과값을 뽑음
3. 뽑혀진 결과값을 통해 **gram 행렬**을 계산

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l (\text{shape } C_i \times H_i)$$

4. 노이즈로 부터 생성할 이미지 생성
5. 생성 이미지로 gram 행렬 계산
6. 두 gram 행렬간의 거리를 통한 loss 계산



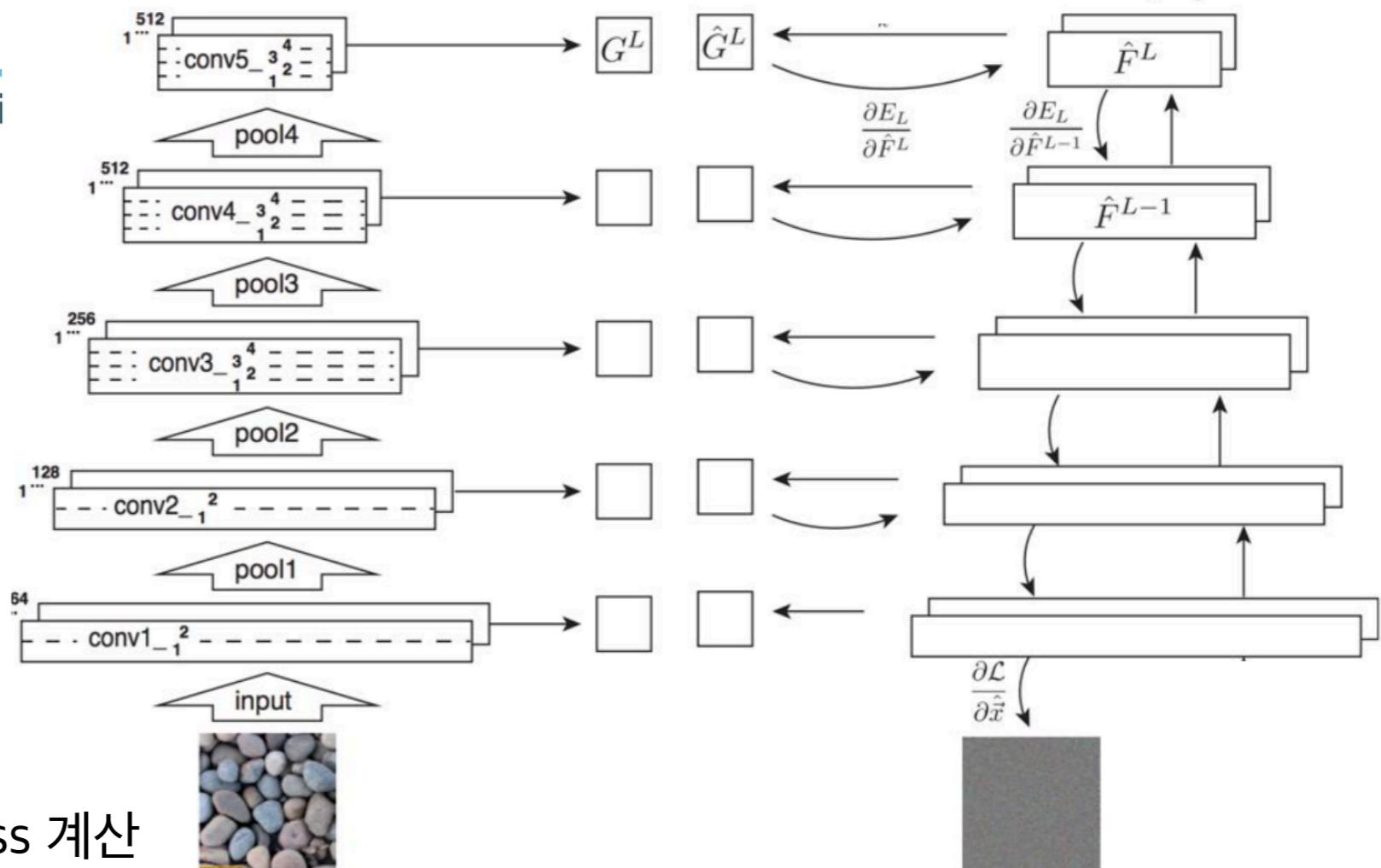
# Style 합성

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - \hat{G}_{ij}^l)^2 \quad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^L w_l E_l$$

1. 사전학습된 VGG-19
2. 입력 이미지를 CNN에 전파시킨 후, 모든 레이어에서 레이어의 결과값을 뽑음
3. 뽑혀진 결과값을 통해 **gram 행렬**을 계산

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l (\text{shape } C_i \times H_i)$$

4. 노이즈로 부터 생성할 이미지 생성
5. 생성 이미지로 gram 행렬 계산
6. 두 gram 행렬간의 거리를 통한 loss 계산
7. 생성 이미지까지 back-prop 후 이미지 업데이트



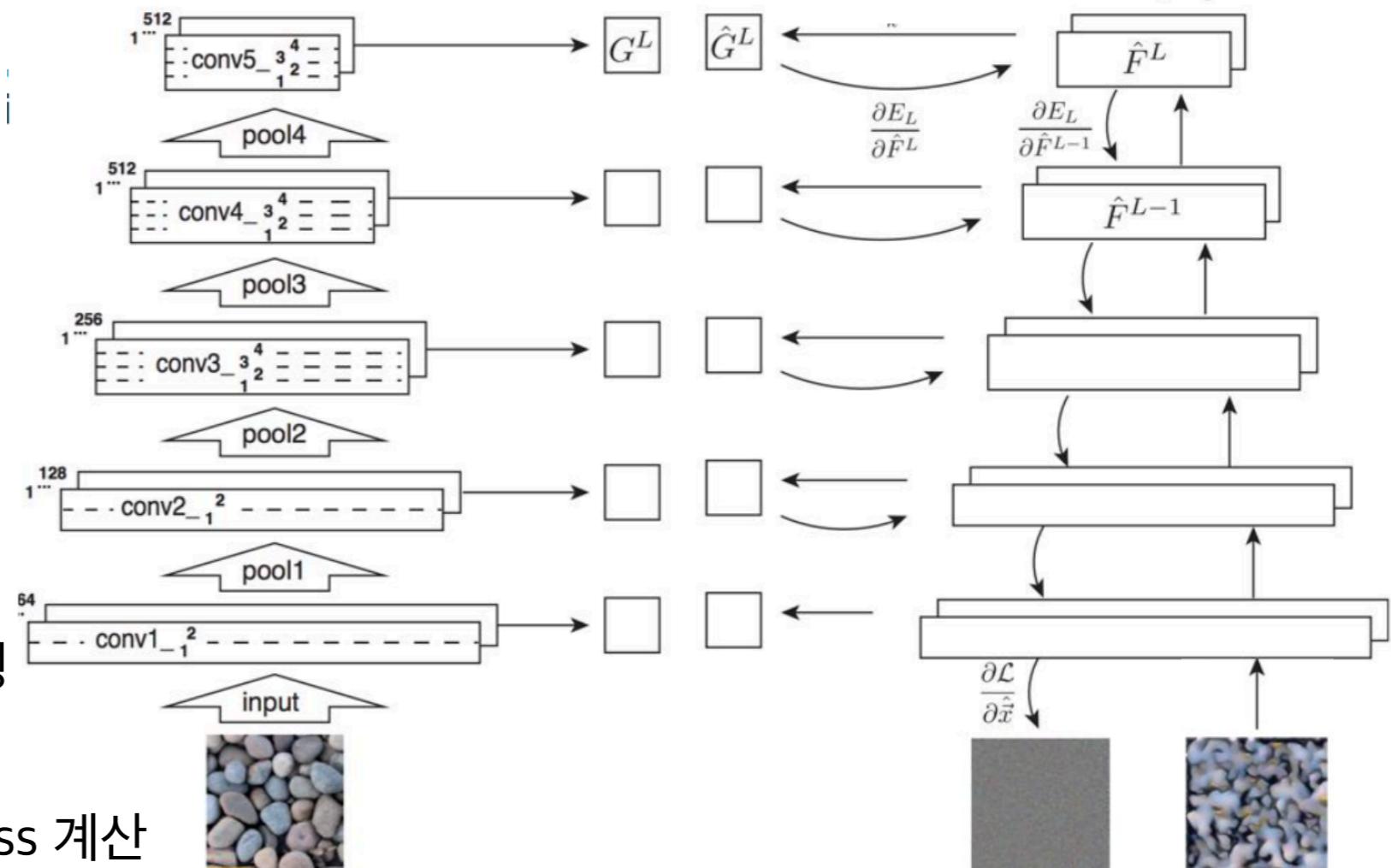
# Style 합성

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - \hat{G}_{ij}^l)^2 \quad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^L w_l E_l$$

1. 사전학습된 VGG-19
2. 입력 이미지를 CNN에 전파시킨 후, 모든 레이어에서 레이어의 결과값을 뽑음
3. 뽑혀진 결과값을 통해 **gram 행렬**을 계산

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l (\text{shape } C_i \times H_i)$$

4. 노이즈로 부터 생성할 이미지 생성
5. 생성 이미지로 gram 행렬 계산
6. 두 gram 행렬간의 거리를 통한 loss 계산
7. 생성 이미지까지 back-prop 후 이미지 업데이트



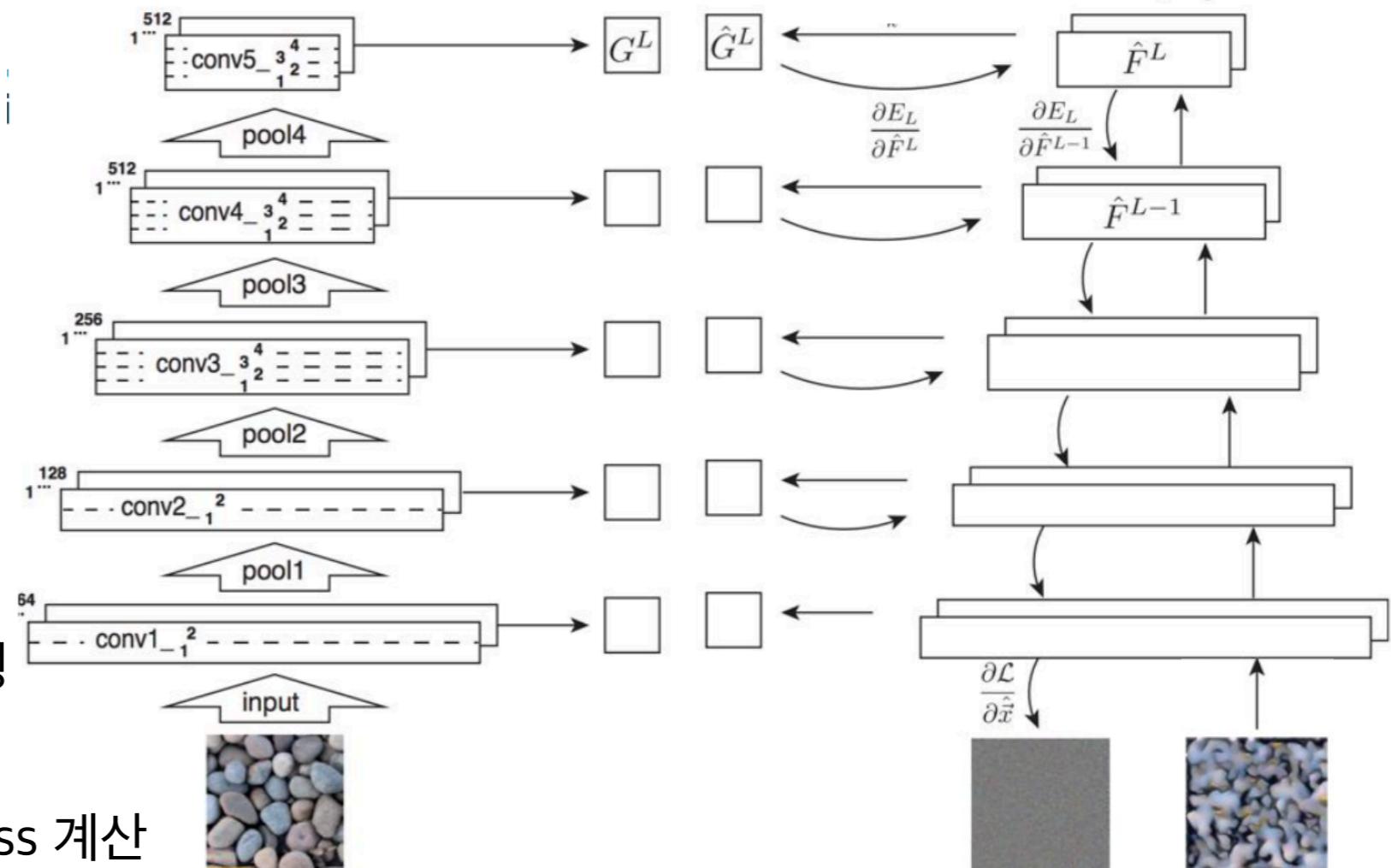
# Style 합성

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - \hat{G}_{ij}^l)^2 \quad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^L w_l E_l$$

1. 사전학습된 VGG-19
2. 입력 이미지를 CNN에 전파시킨 후, 모든 레이어에서 레이어의 결과값을 뽑음
3. 뽑혀진 결과값을 통해 **gram 행렬**을 계산

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l (\text{shape } C_i \times H_i)$$

4. 노이즈로 부터 생성할 이미지 생성
5. 생성 이미지로 gram 행렬 계산
6. 두 gram 행렬간의 거리를 통한 loss 계산
7. 생성 이미지까지 back-prop 후 이미지 업데이트
8. 5-7 반복

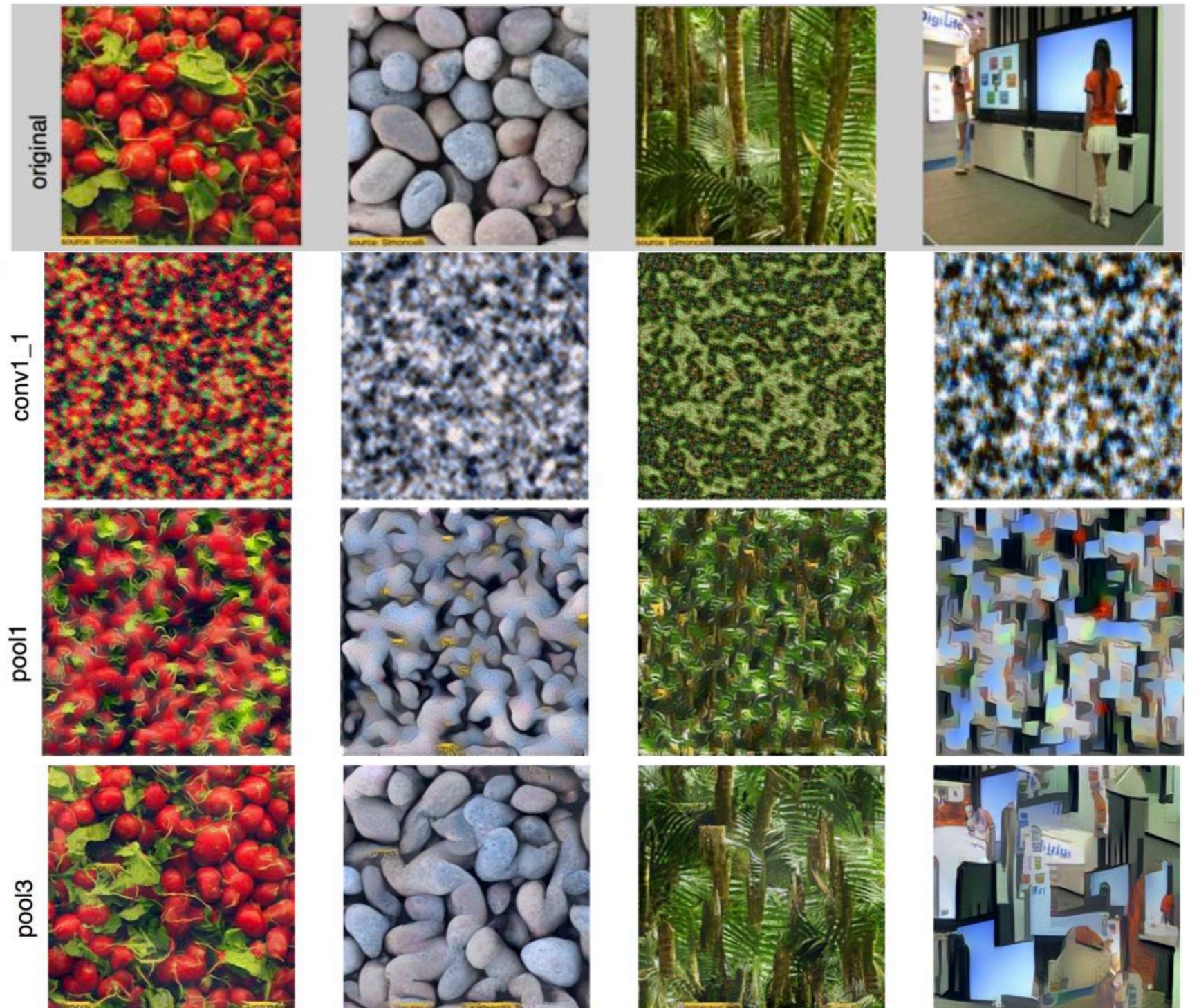


# Style loss

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - \hat{G}_{ij}^l)^2 \quad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^L w_l E_l$$

- 두 gram 행렬간의 차이를 **style loss**로 정의
- 전체 style loss는 각 레이어에서 구해진 style loss ( $E_l$ ) 와 레이어별 가중치 ( $w_l$ ) 를 곱한 뒤 합해서 계산
  - 주로 사용되는 레이어는 [conv1\_1, conv2\_1, conv3\_1, conv4\_1, conv5\_1]
  - 가중치는 깊은 레이어에 더 높게

# Style loss



깊은 레이어에서 Style을  
잘 복원하기 때문

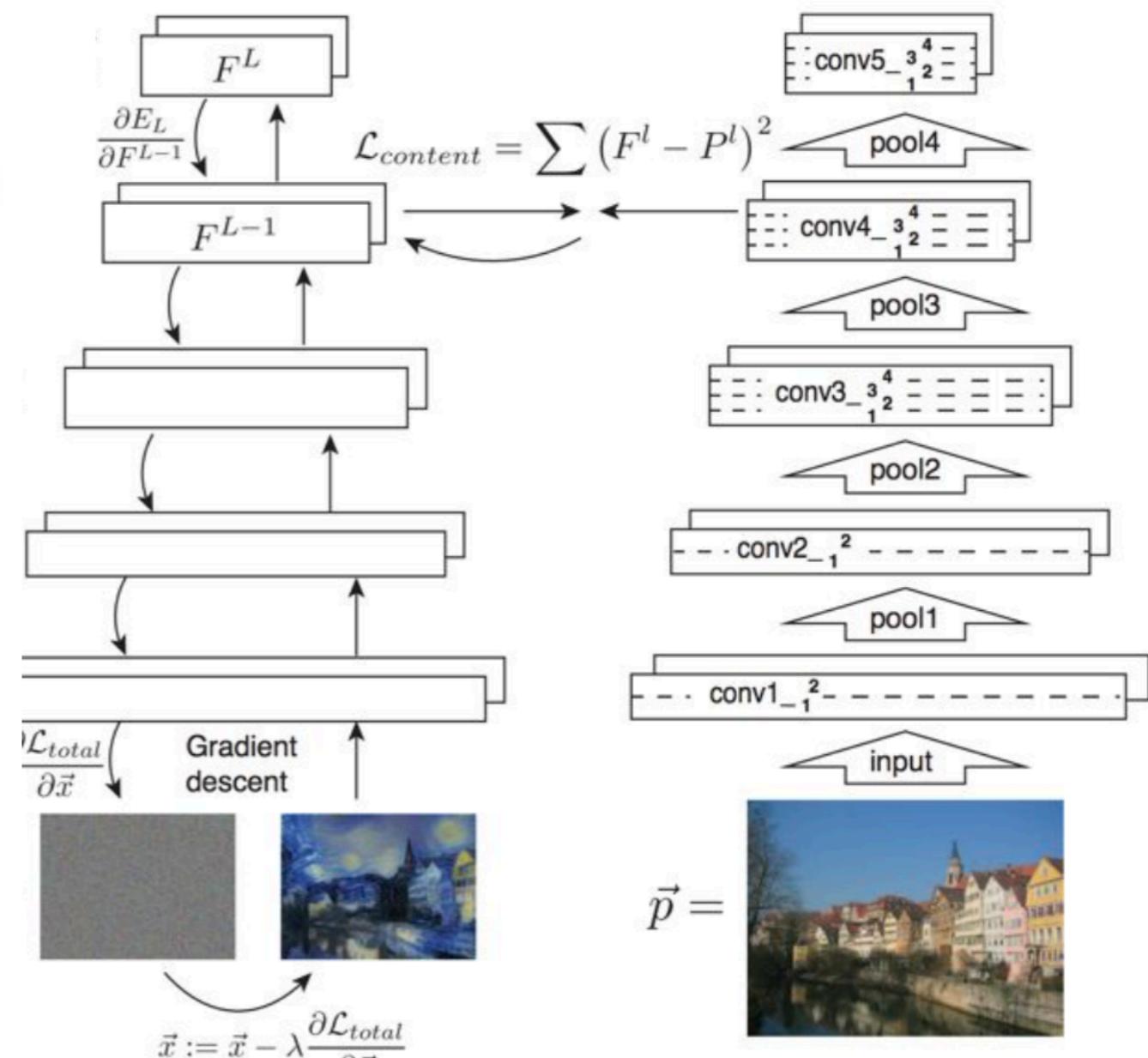
# Content loss

- Content 이미지와 생성된 이미지의 특징 맵의 차이

- conv4\_2에서 추출한 특징맵 사용

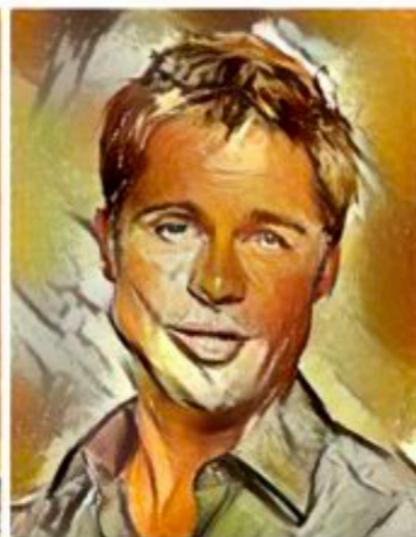
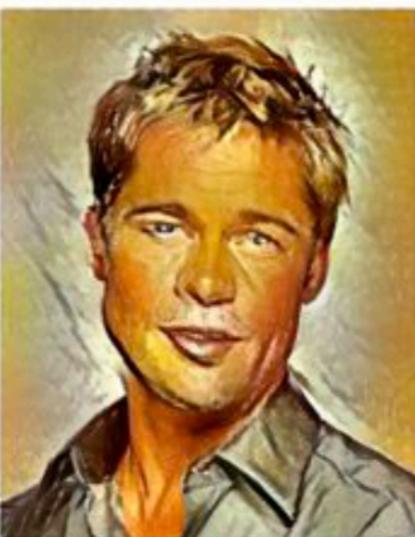
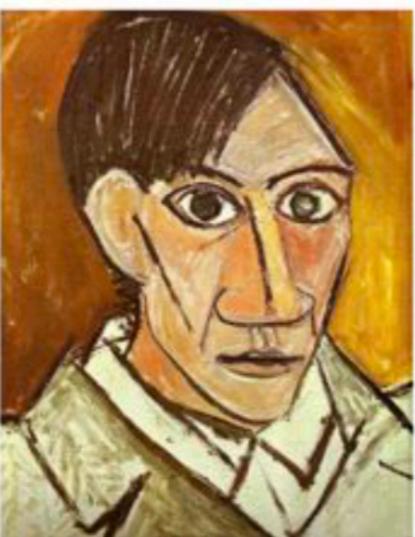
$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

- 계수를 1/2 대신  $1/(4*p.size)$ 로 바꿀 경우 더 빠르게 수렴한다고 알려짐



# 전체 loss

- $\text{loss} = \text{alpha} * \text{style\_loss} + \text{beta} * \text{content\_loss}$



Content loss에 더 강한 가중치

Style loss에 더 강한 가중치