Deep Learning Seminar

7. Segmentation

HA SEUNG HYUN

Contents

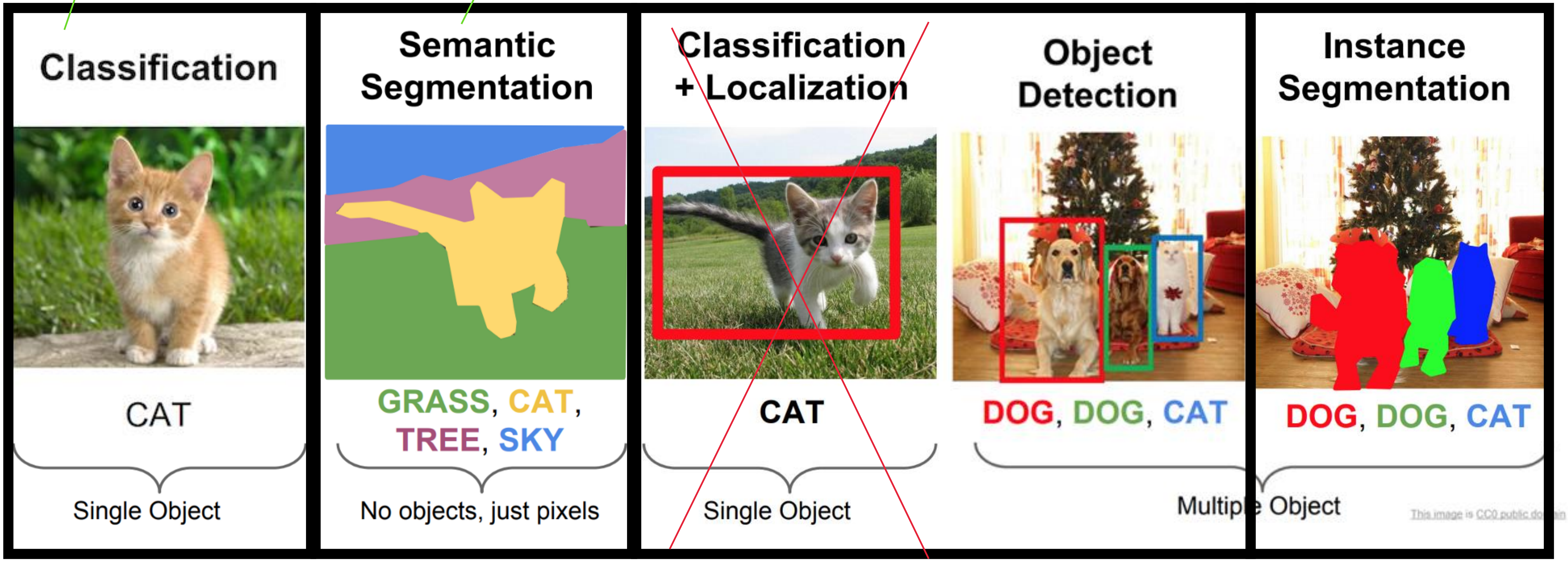
Reference: CS231n lecture note (Fei-Fei Li)
CS229 lecture note (Andrew Ng)
모두를 위한 머신러닝 (Sung kim)

1. Overview

Overview

각 영역의 픽셀값, 만약 해당 그림에 고양이가 겹쳐있으면 세세하게 구분 못하고
두 고양이 모두 고양이 영역에 해당된다고 인식. 픽셀의 카테고리만 나눔

이미지 전체를 보고 하나의 카테고리로 분류



object detection은 single로 하는 경우 없이 multi를 함

1. Computation Cost 3, 2, 1 순서로 적게 듦

드는 비용

Object Detection 1

|

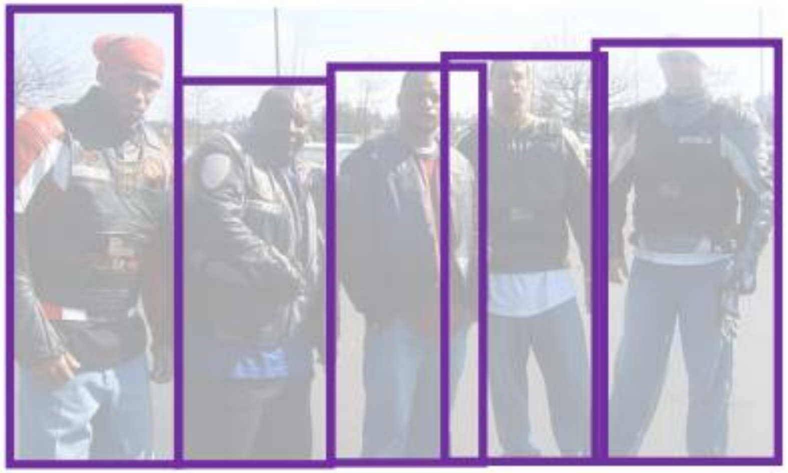
Semantic Segmentation 2 - 3

|

Instance Segmentation 3 - 4 → 비용이 많이 들어 잘 안씀

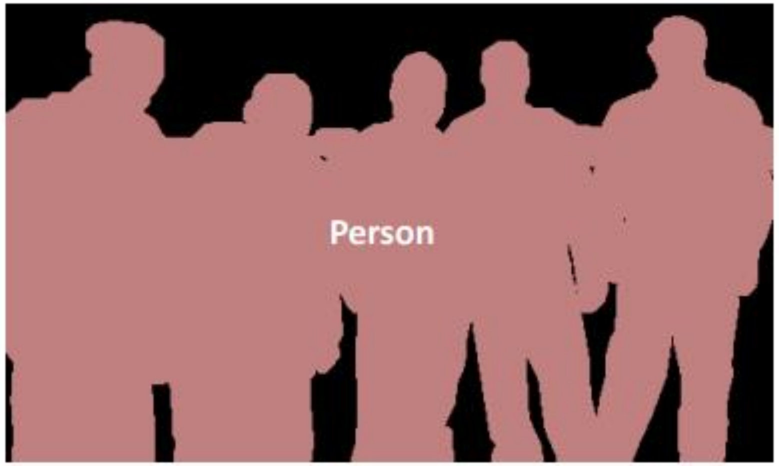
실제로 코딩적 가치로는 Object Detection 이 가장 높다.

각 물체 감지 박스로 구분



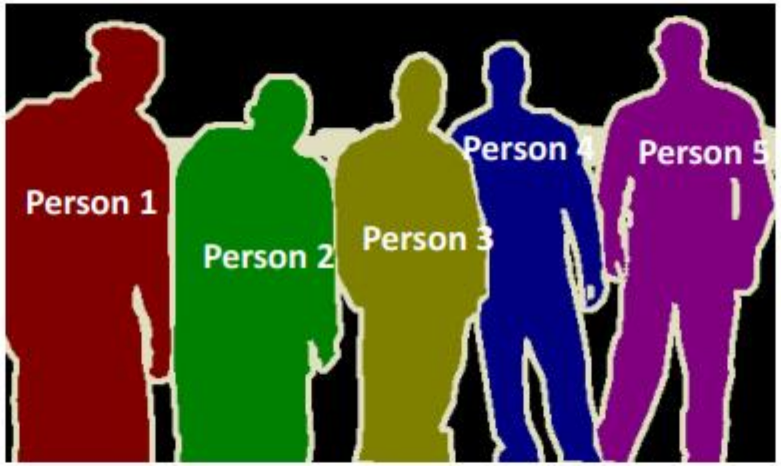
Object Detection

여러 물체가 있더라고 픽셀에 대한 카테고리 분류 세세하게 구분 못함.



Semantic Segmentation

Object Detection에서 각 객체마다 픽셀을 세세하게 분류



Instance Segmentation

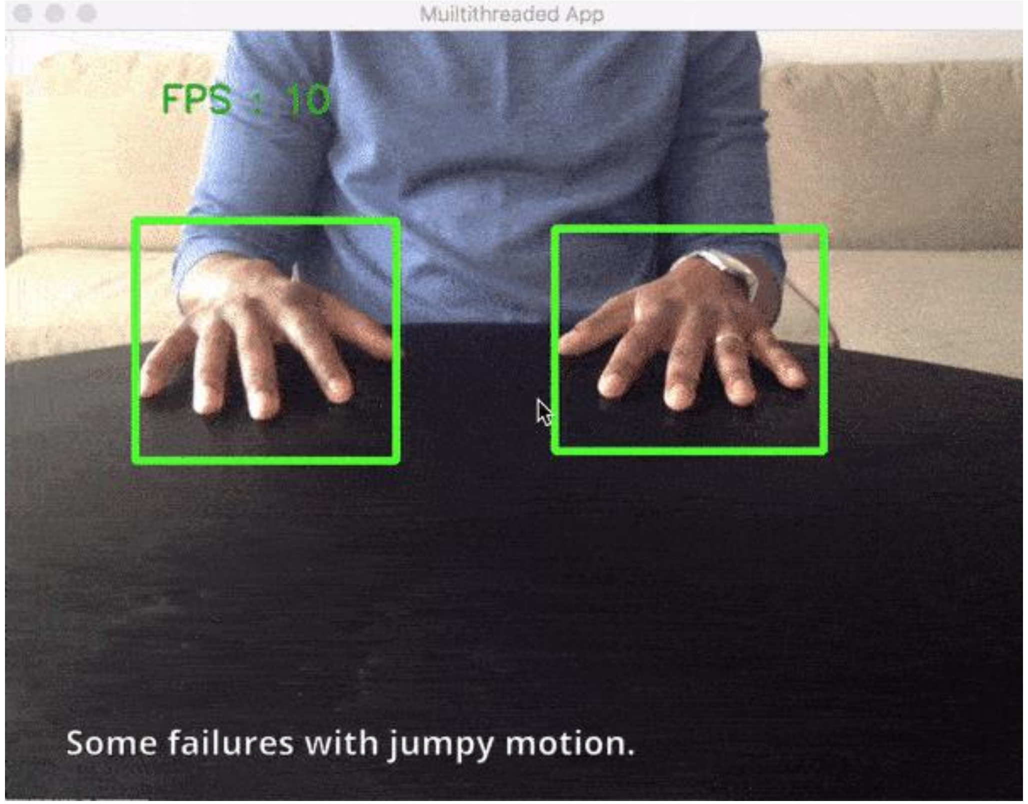
Overview

FPS (Frame Per Second)
초당 프레임 수, 1초당 몇개의 이미지 Frame을 처리할 수 있는가

인간의 눈. 1초당 8개 이미지 이상 처리 → 자연스럽게 느낌

FPS가 높을 수록 속도가 빠르다. 속도가 빠르기 때문에 모델이 가볍고 Computation Cost가 적게 든다.
FPS가 높다라는 얘기는 1초마다 이미지를 처리하는데 걸리는 속도가 빠르다.
이미지 하나 처리하는데 빠르다.
인공지능 모델이 가벼워야 한다. layer가 적고 데이터 로딩 시간이 적고...전처리, 후처리가 짧아야함.
Forwarding도 한번에~

Object Detection Example



모델이 가볍다
- Low And Model

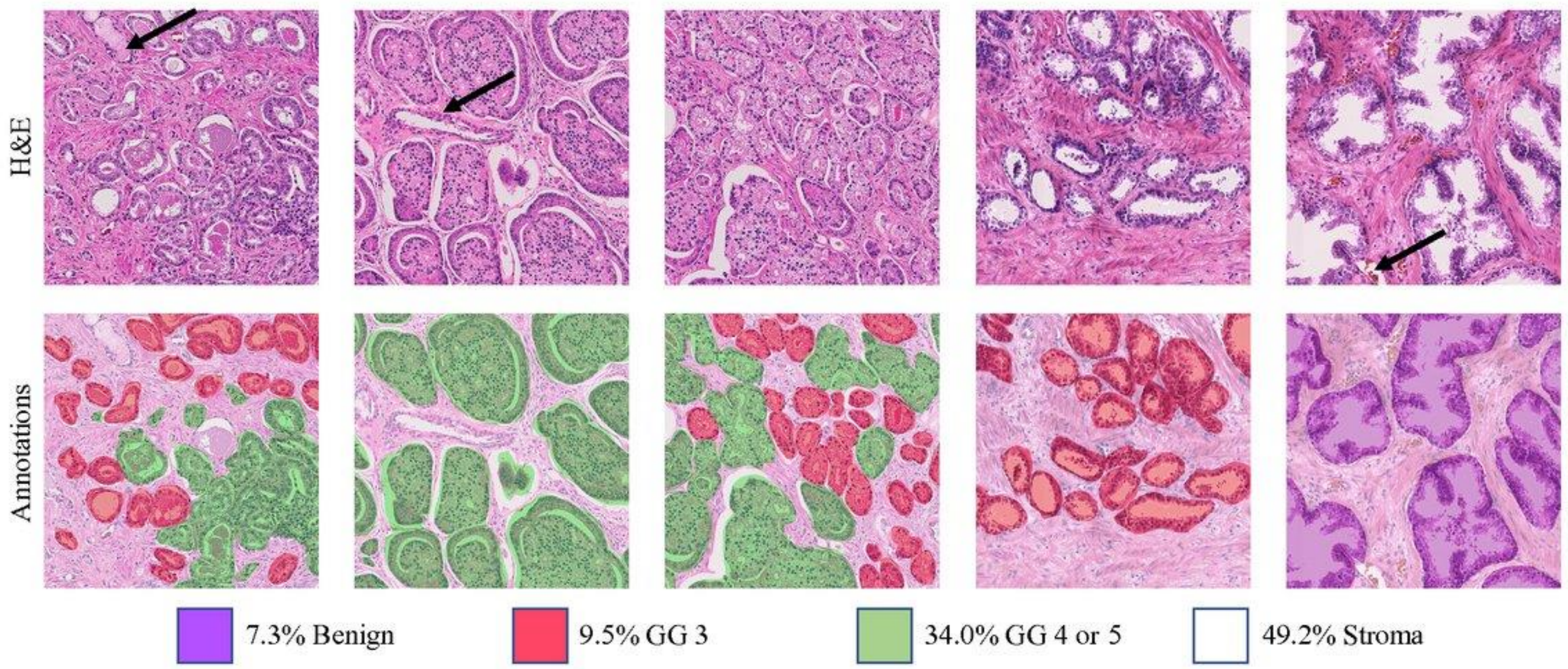
속도는 빠른데 성능은 떨어진다.

High And Model
속도는 느리지만 성능은 좋다.

Trade - Off 관계

Overview

Semantic Segmentation Example

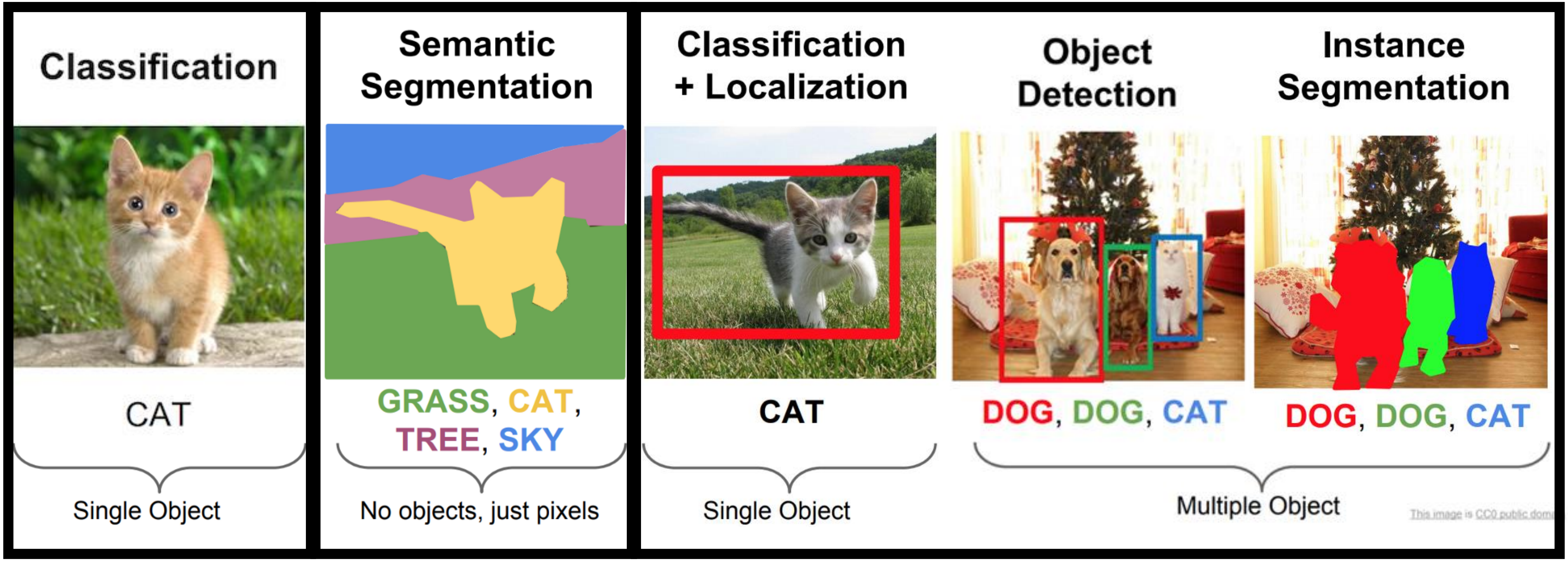


Overview

Instance Segmentation Example

2. Classification

Classification



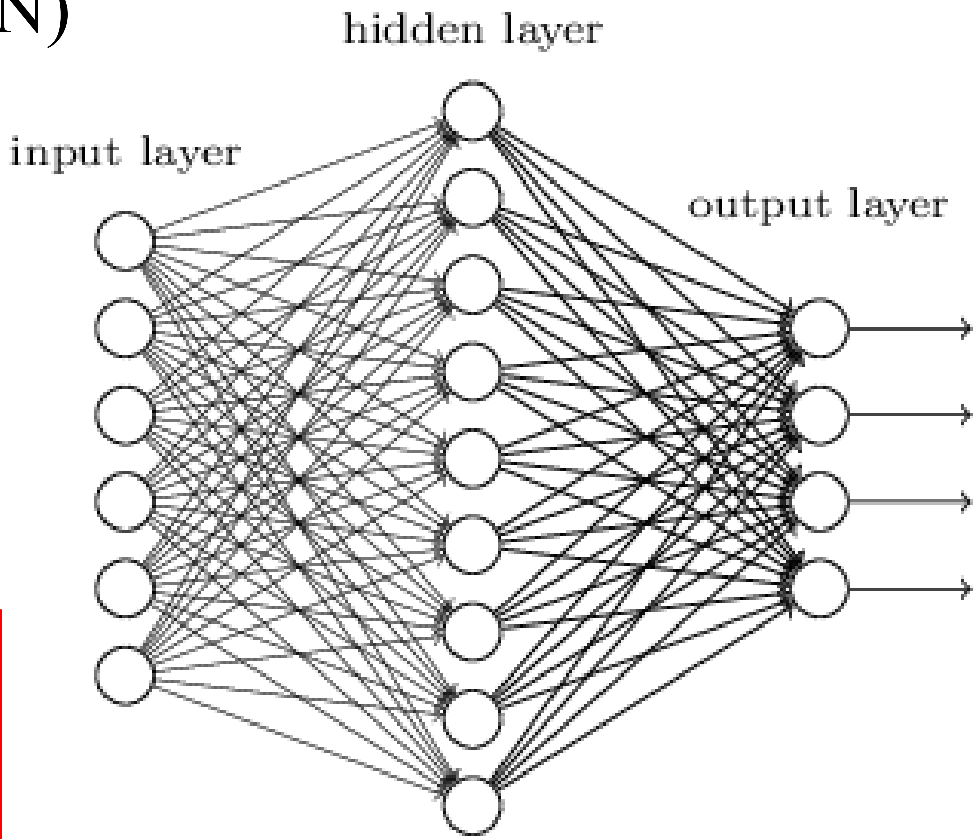
Classification

- Fully Connected Network (FCN)

1) 1차원으로 늘려야 하기 때문에 공간 정보 사라짐

1) Disappear spatial information
2) Computationally Expensive

Input



Fully Connected Network

Classification

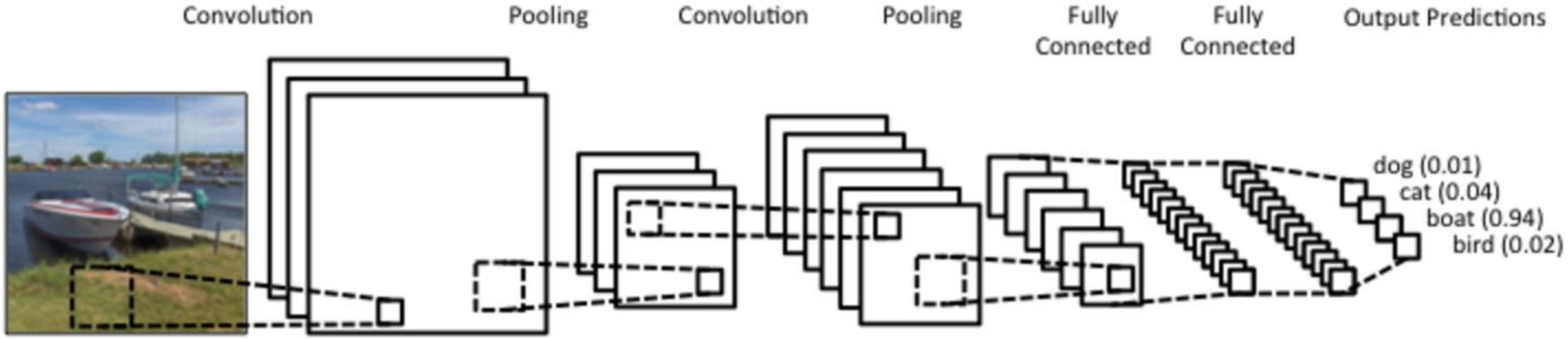
- Convolutional Neural Network (CNN)

사실상 Fully Connected Network 하고 유사.

FCN의 상위호환 Network가 CNN
상위호환이란 이전 네트워크 구조를 그대로 가져왔다.

FCN은 이미지 사이즈와 동일한 필터를 이미 사용.
CNN은 부분적인 값 뽑는데 굳이 filter 크기를 FCN처럼 크게 할 필요가 있을까? 라는 의문에서 도출된 것이 CNN의 Filter.
전체적인 특징을 추출하는 방식을 부분적인 특징을 추출하는 방식으로 만든것이 CNN
filter의 크기를 줄이면 filter 안에 들어가는 w 값이 작아지기 때문에 연산량이 줄어든다. 즉, computation cost도 적어진다.

단, 이미지 데이터 일때 한정



Convolutional Neural Network

Classification

- Popular Model

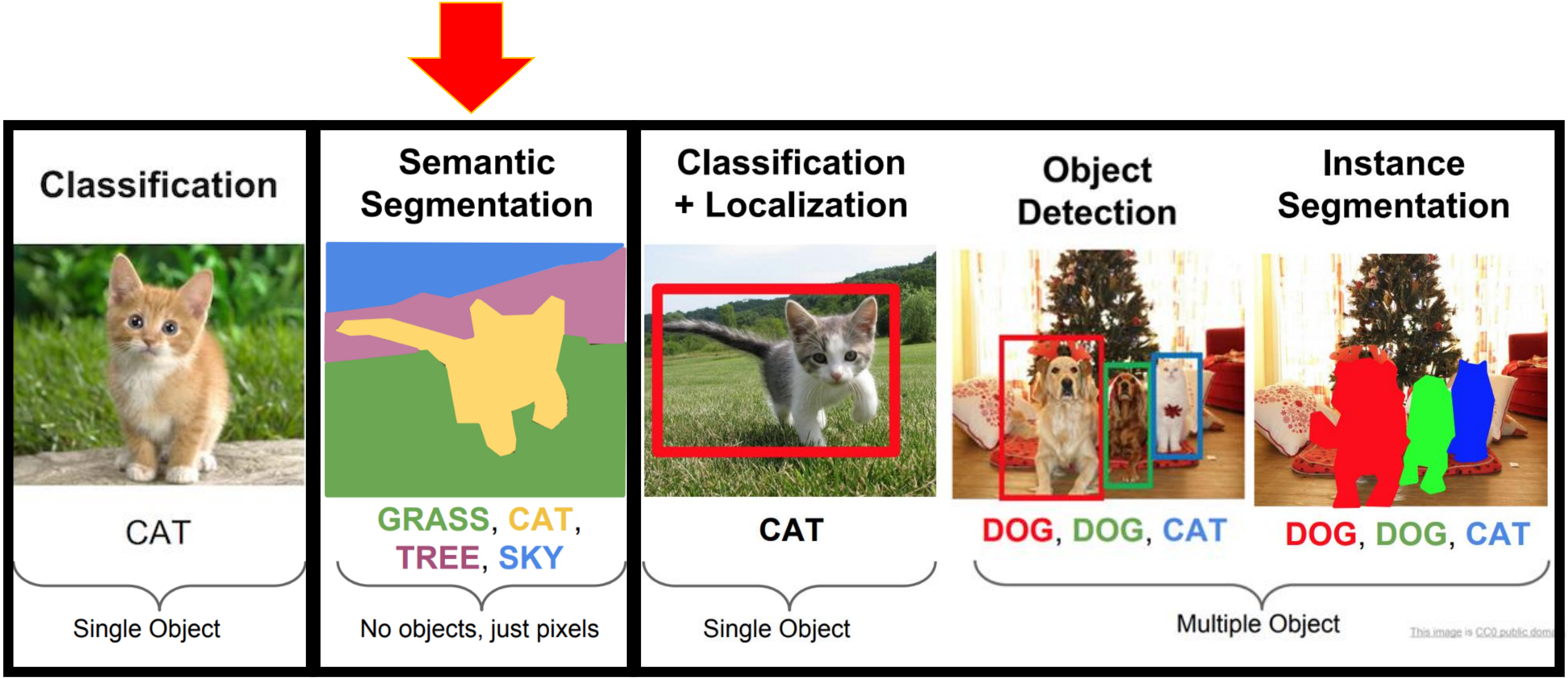
1) VGG
2) GoogLeNet (Inception)
3) ResNet
4) DenseNet

3. Semantic Segmentation

FPS
→
Object Detection
Low And Model
속도 ↔ 성능

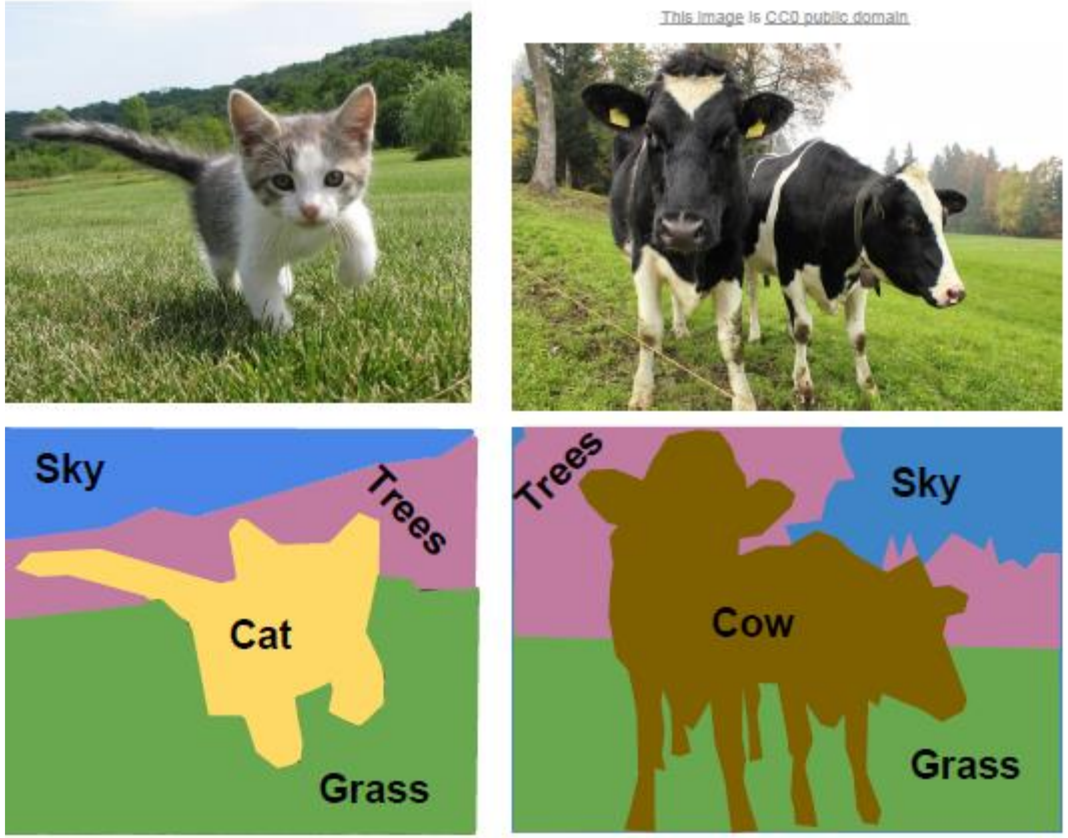
3-1. Overview of semantic segmentation
3-2. Upsampling & Convolutions
3-3. U-Net
3-4. Evaluation Matrix

Semantic Segmentation



Semantic Segmentation

- Label each pixel in the image with category label



이부분 나중에 강의 다시 듣기

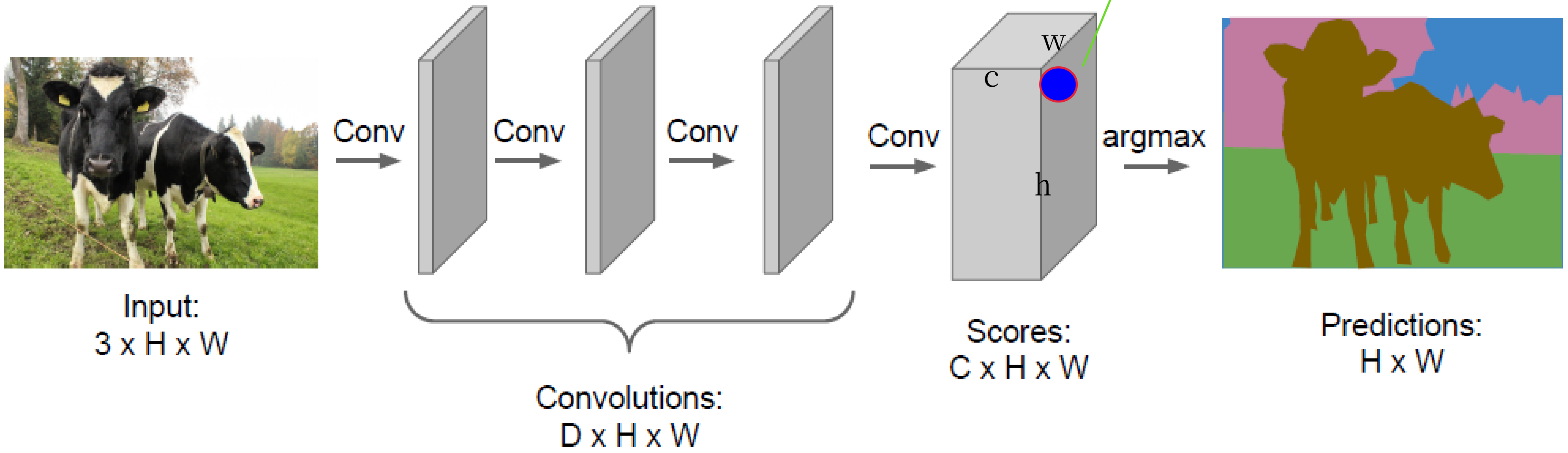
Semantic Segmentation

CNN만 가지고 Semantic Segmentation 가능하다는 것을 보여주는 그림

픽셀 값들을 Classification 하는 것

Design a network as a bunch of convolutional layers
to make predictions for pixels all at once!

512



ConV를 통해 사이즈를 동일하게 유지. filter 3x3 채널 stride=1, padding=1 → 원본 이미지와 동일한 사이즈 유지(CNN)

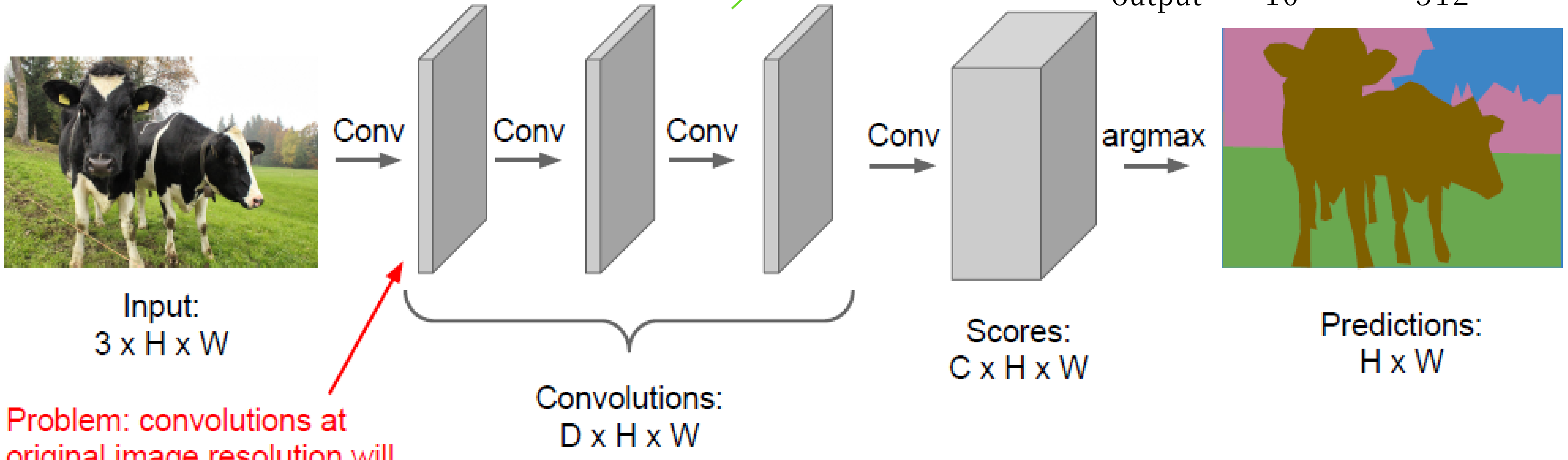
결과적으로 computation cost ↑
- 불가능한 모델, 사이즈는 동일한데 채널만 늘리기 때문에
연산량이 기하급수적으로 증가함

Semantic Segmentation

	C	H	W
input	3	512	512
Filter 3x3x3 padding 1 stride 1 16개			
layer1	16	512	512
Filter 3x3x3 padding 1 stride 1 32개			
layer2	32	512	512
Filter 3x3x3 padding 1 stride 1 64개			
layer3	64	512	512
Filter 3x3x3 padding 1 stride 1 128개			
layer4	128	512	512
layer5	64	512	512
layer6	32	512	512
output	10	512	512

특징 추출(채널 수)
연산량이 높아지는 것을 막아야한다.
→ 사이즈를 줄였다가 다시 늘린다.

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



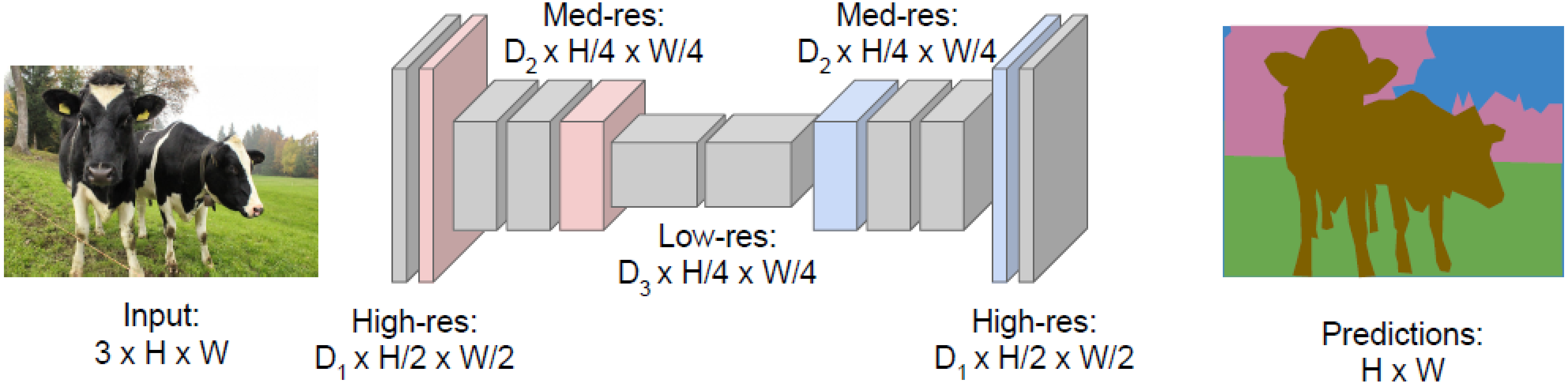
Problem: convolutions at original image resolution will be very expensive ...

vgg 모델은 size는 2배로 줄어들고 channels 2배로 증가한다.
특징을 추출하면서도 사이즈를 줄이고 있다.
채널을 늘리는 것보다. 사이즈를 줄이면 전체적으로 1/2 감소, 연산량 감소
즉, 연산량 감소에는 size 줄이는 것이 엄청 큰 영향을 끼친다.

특징을 추출하는 것에 사이즈는 중요하지 않다.
픽셀값이 다 필요하지도 않고 큰 특징만 추출하면 된다.
CIFAR10 의 경우 32x32인데도 충분히 특징 추출하는 것처럼

Semantic Segmentation

Design network as a bunch of convolutional layers, with downsampling and upsampling inside the network!



1. 원본, output 사이즈 동일
2. 사이즈를 줄었다가(downsampling) 늘린다(upsampling)

Semantic Segmentation

사이즈를 늘리는 작업

Downsampling: Pooling, strided convolution

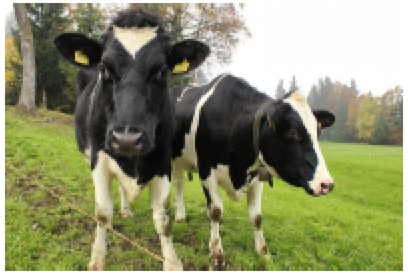
동일한 용어 : convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

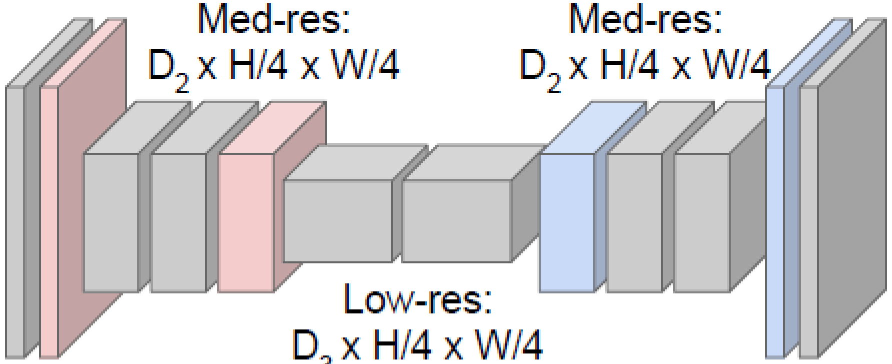
Upsampling: ???

채널이랑 상관이 없다!!

동일한 용어 : DeConvolution



Input: 3 x H x W



High-res: D_1 x H/2 x W/2

Med-res: D_2 x H/4 x W/4

Low-res: D_3 x H/4 x W/4

Med-res: D_2 x H/4 x W/4

High-res: D_1 x H/2 x W/2

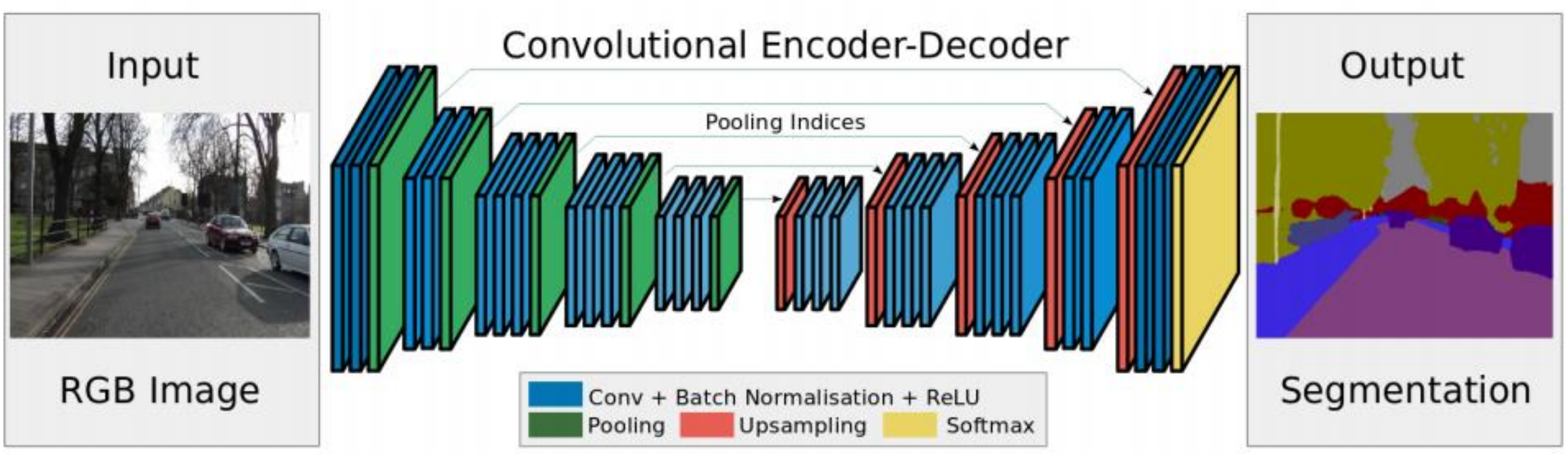
Predictions: H x W

사이즈가 절반씩 줄어든다. (down sampling)

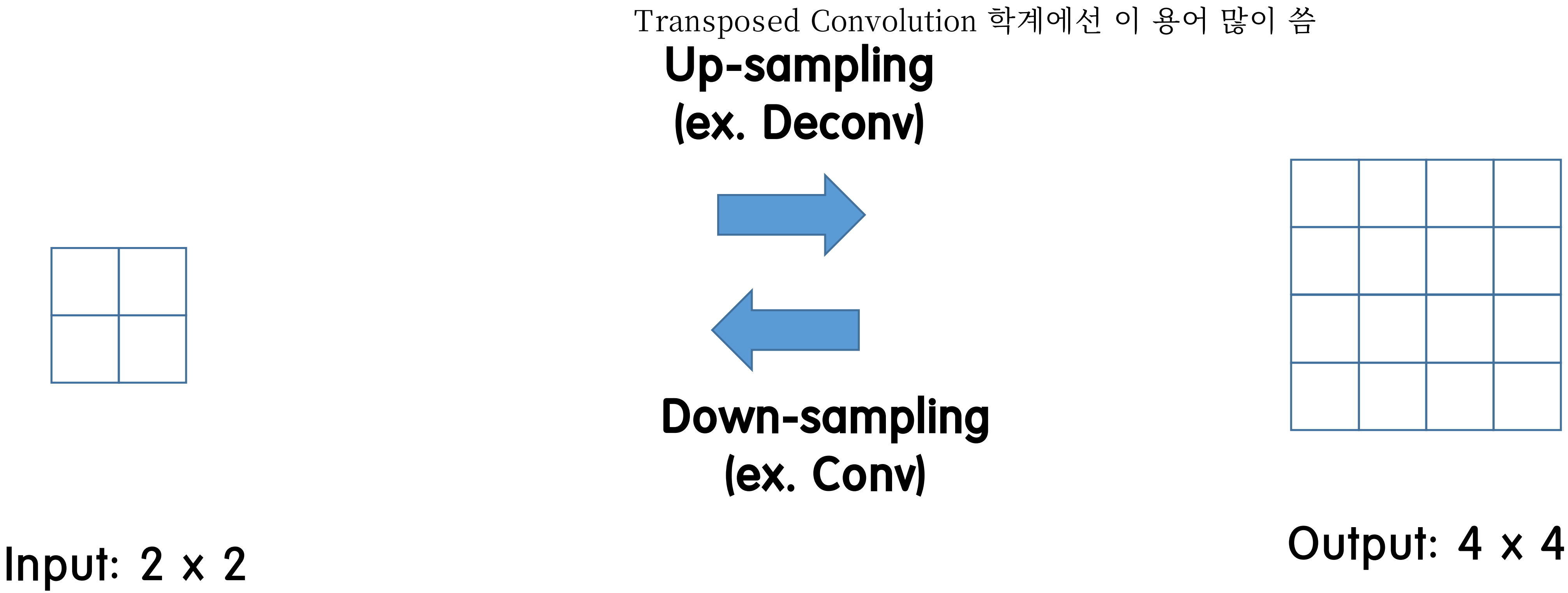
1. stride = 2 → 학습을 진행하기 때문에 더 좋다.
2. max pooling → 학습 진행 x. 하지만 연산을 건너뛰기 때문에 속도에 좋다.

Semantic Segmentation

Batch Normalization 연산을 할때마다 w, b 값이 작아질건 작아지고 커질건 커져서 이를 줄여주는 역할이 필요함. batch size마다 값의 분산이 커지는데 분산을 줄여주는 역할



Upsampling



Transposed Convolution 학계에선 이 용어 많이 씀
Up-sampling
(ex. Deconv)
Down-sampling
(ex. Conv)
Input: 2 x 2
Output: 4 x 4

Unpooling (Upsampling)

1. 의미 없는 값 0 으로 채워줌.

하지만 이 방법은 사용안함. 무식한 방법

- Train 되지 않는다.
- 0이라는 값은 검정색이므로 최종 이미지에 검정 줄무늬가 생김

"Bed of Nails"

1	2
3	4

Input: 2 x 2

→

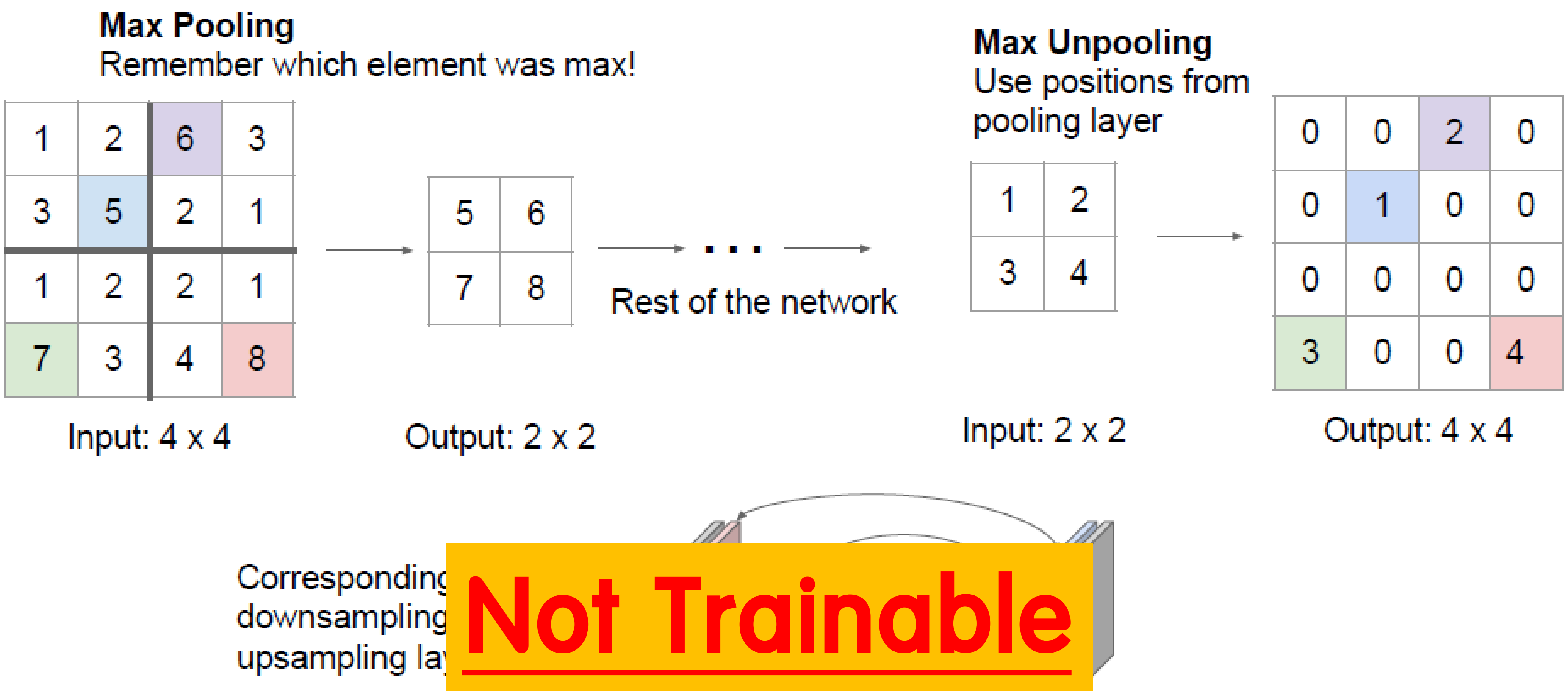
1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Output: 4 x 4

Not Trainable

Unpooling (Upsampling)

해당 방법도 사용 안하고 있음



이제부터 사용하는 방법

Interpolation (Upsampling)

보간법 - 가장 옆에 있는 이웃으로 채운다.

값을 채울때
조금 더 유연하고 부드럽게 출력되도록 채우는
방법이 다양하게 존재
ex) 1, 2 → 1 1 2 2
ex) 1, 2 → 1 1.5 2 2.5

근사치 값으로 채운다.

Nearest Neighbor

1	2
3	4

Input: 2 x 2

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Output: 4 x 4

1. 나름 의미 있는 값으로 채움.
2. 화질은 좀 깨지더라도 Garbage 값으로 채우는 것은 아니다.

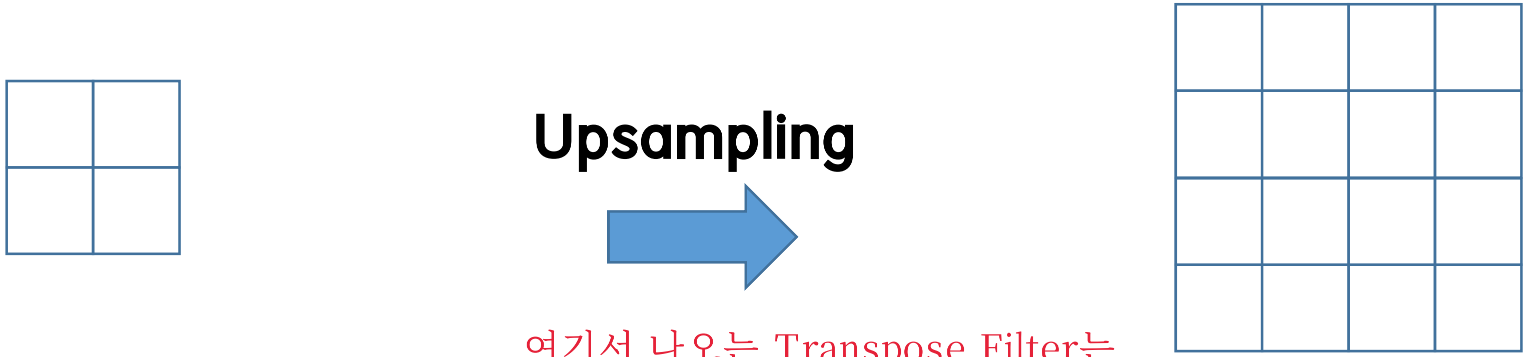
3. 하지만 Not Trainable
- 학습할 수 없다는 것은 filter 안의 값들이 W값이 아니기 때문에 backward가 안된다.

Filter

Not Trainable

Transpose Convolution (Upsampling)

= Deconvolution



여기서 나오는 Transpose Filter는
Weight 값으로 채워짐.

Input: 2 x 2

Output: 4 x 4

Trainable

Transpose Convolution (Upsampling)

= Deconvolution

사이즈를 늘릴 때 논란 Hot

Interpolation VS Deconvolution

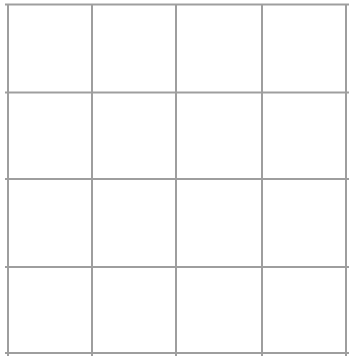
"Trainable"

"deconvolution checker border"
겹쳐지는 일정한 패턴이 생긴다.

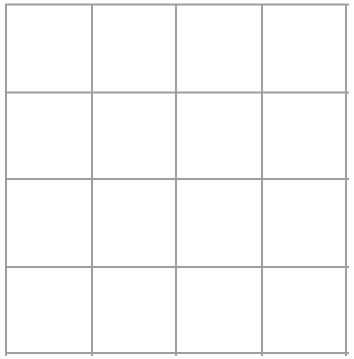
"학습된다 → 연산량이 많다
→ 성능 향상에 크게 개선되는 부분도 없다."

결론적으로 대부분의 코드에서
Interpolation 방법을 사용

Recall: Typical 3 x 3 convolution, stride 1 pad 1



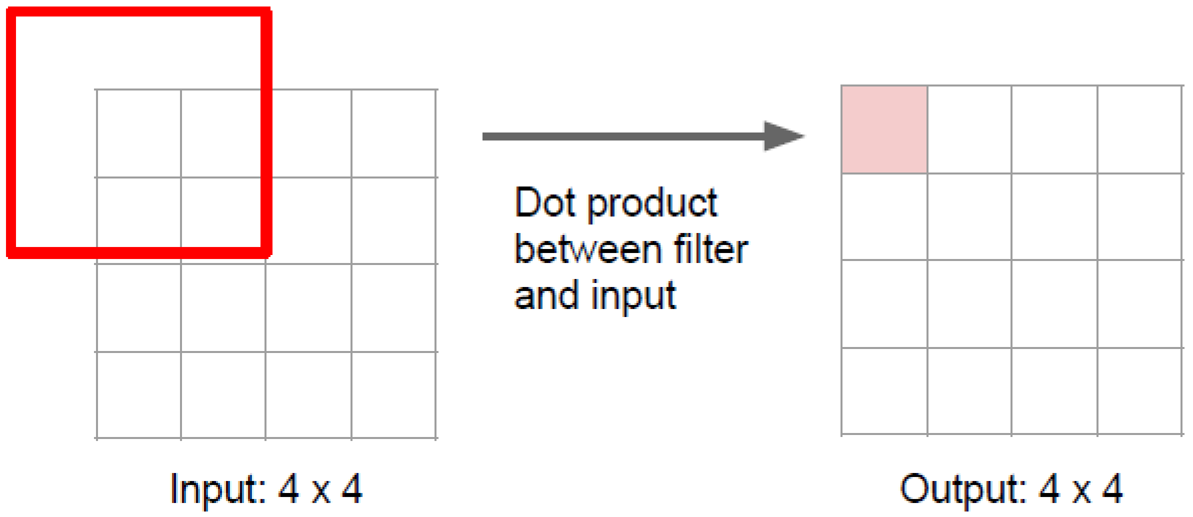
Input: 4 x 4



Output: 4 x 4

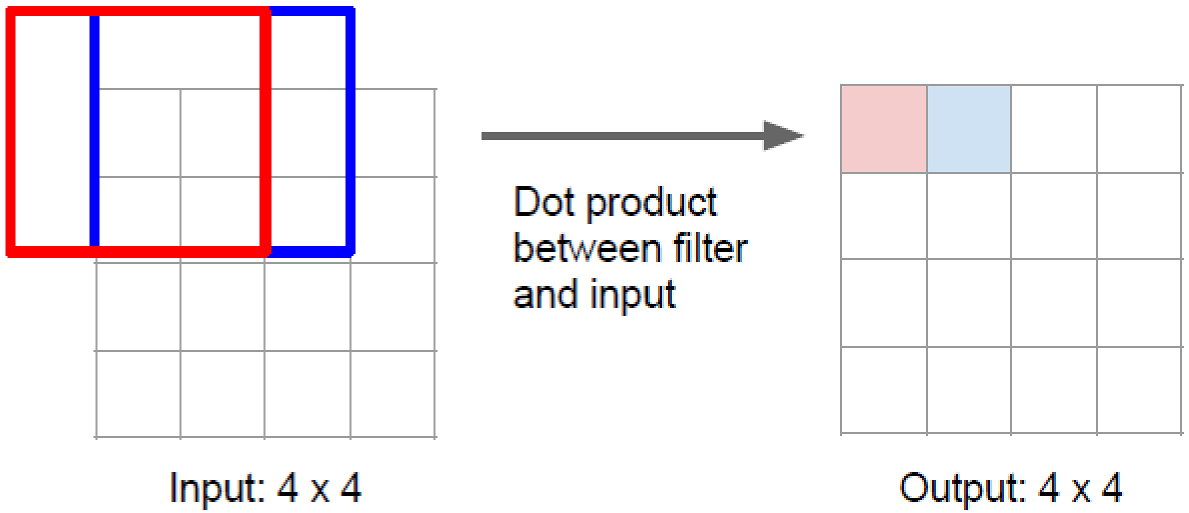
Transpose Convolution (Upsampling)

Recall: Normal 3 x 3 convolution, stride 1 pad 1



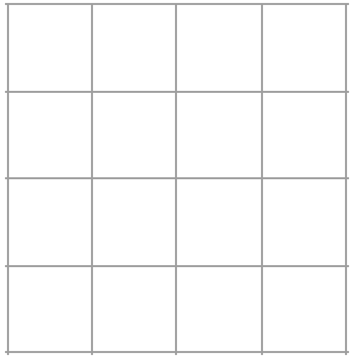
Transpose Convolution (Upsampling)

Recall: Normal 3 x 3 convolution, stride 1 pad 1

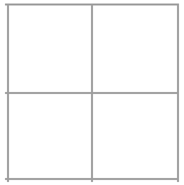


Transpose Convolution (Upsampling)

Recall: Normal 3 x 3 convolution, stride 2 pad 1



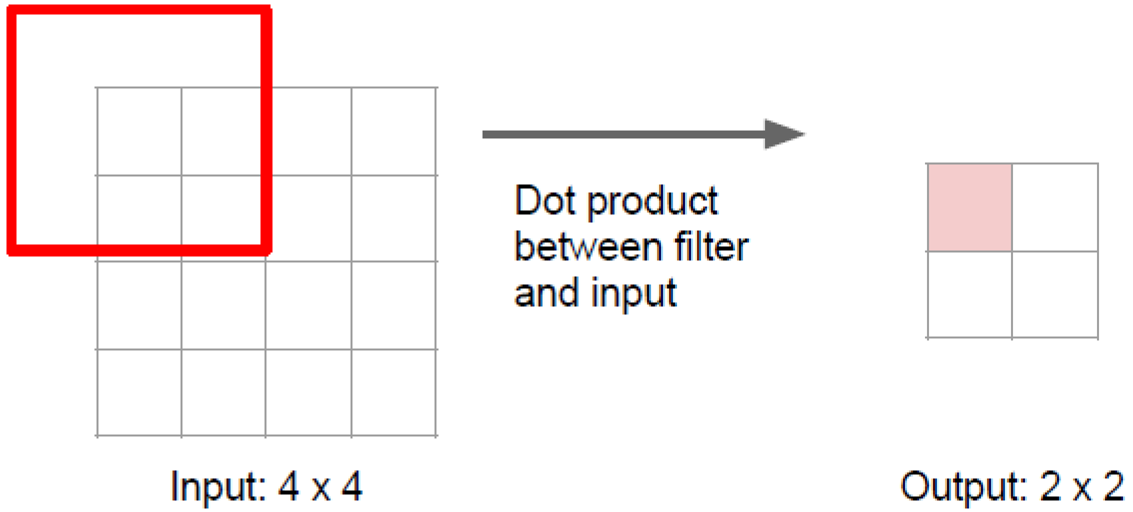
Input: 4 x 4



Output: 2 x 2

Transpose Convolution (Upsampling)

Recall: Normal 3 x 3 convolution, stride 2 pad 1

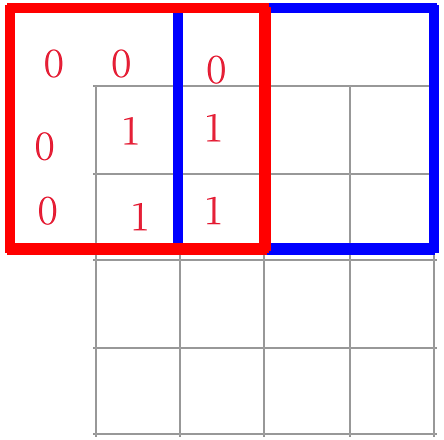


Transpose Convolution (Upsampling)

filter

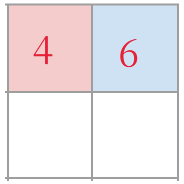
$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Recall: Normal 3 x 3 convolution, <u>stride 2</u> pad 1



Input: 4 x 4

Dot product between filter and input



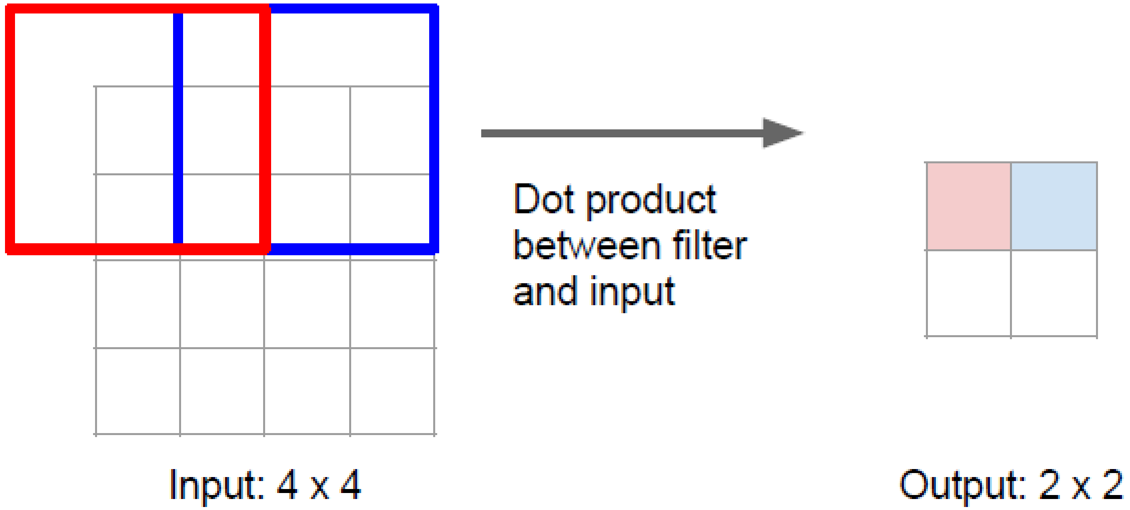
Output: 2 x 2

Filter moves 2 pixels in the input for every one pixel in the output

Stride gives ratio between movement in input and output

Transpose Convolution (Upsampling)

Recall: Normal 3 x 3 convolution, <u>stride 2</u> pad 1

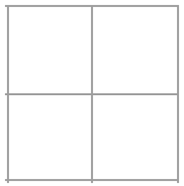


Filter moves 2 pixels in the input for every one pixel in the output

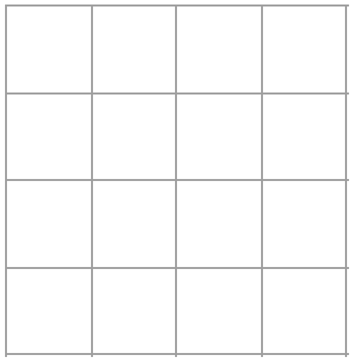
Stride gives ratio between movement in input and output

Transpose Convolution (Upsampling)

3 x 3 **transpose** convolution, stride 2 pad 1

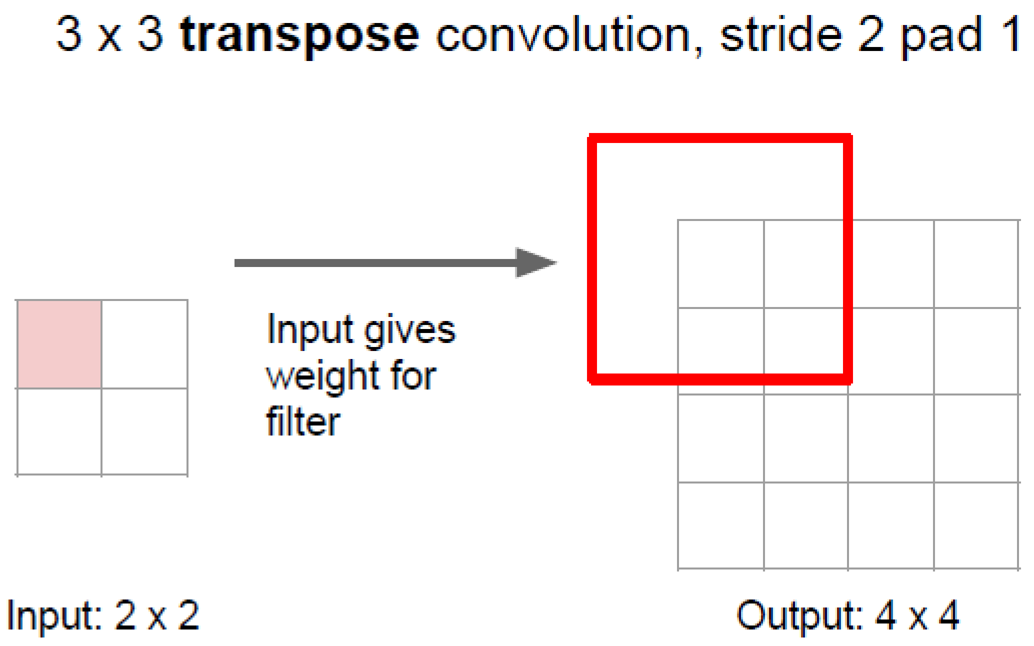


Input: 2 x 2



Output: 4 x 4

Transpose Convolution (Upsampling)



Trainable 하게 사이즈를 키워보자

Transpose Convolution (Upsampling)

DeConvolution Filter
사이즈를 늘려주는 filter

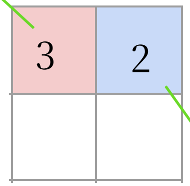
filter

1 1 1
1 1 1
1 1 1

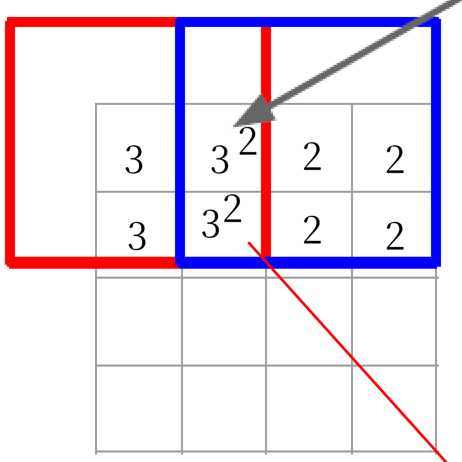
3 x 3 transpose convolution, stride 2 pad 1

3 3 3
3 3 3
3 3 3

Sum where output overlaps



Input gives weight for filter



Filter moves 2 pixels in the output for every one pixel in the input

Filter안에서 Weight 값들이 modefy 됨.

Stride gives ratio between movement in output and input

Input: 2 x 2

원본이미지 2x2

Output: 4 x 4

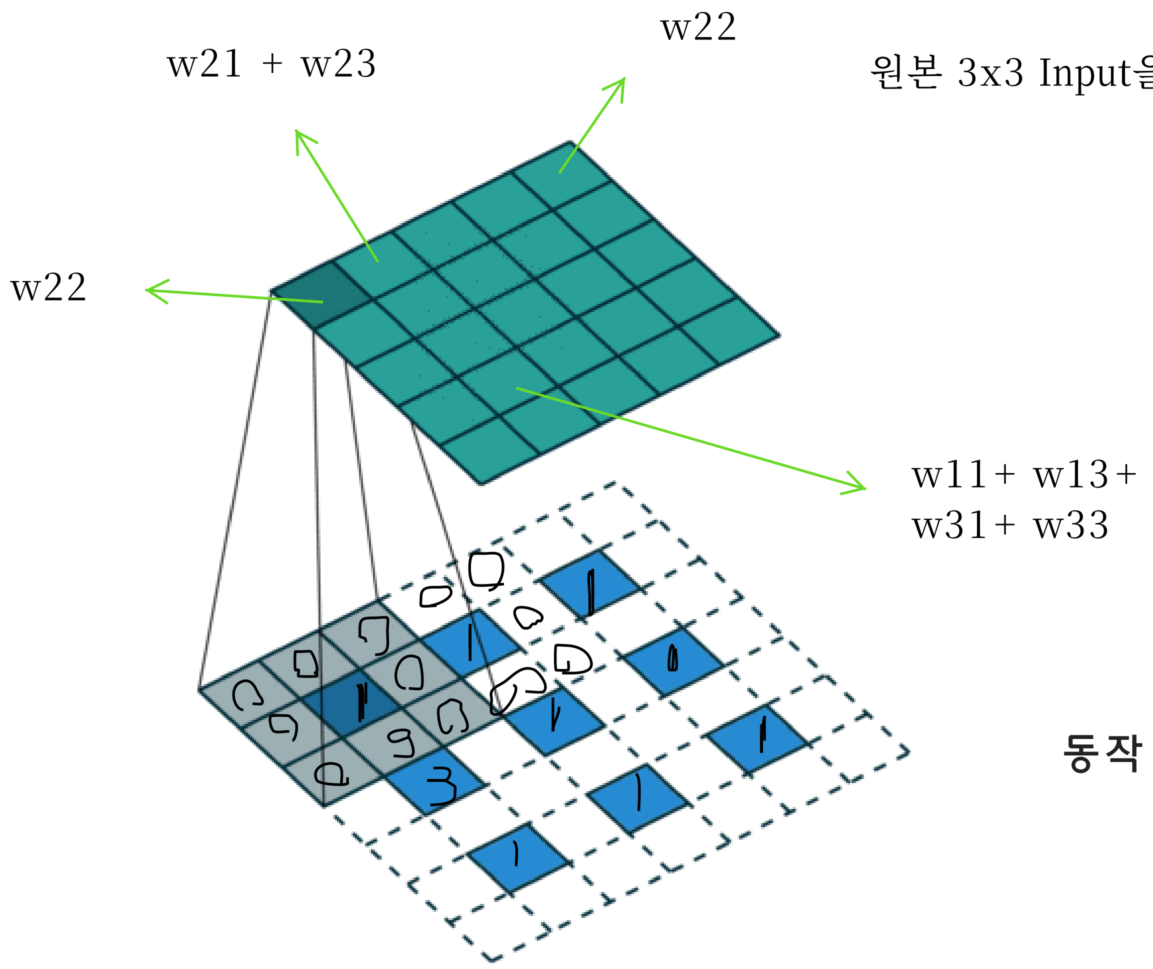
1 1 1
1 1 1
1 1 1

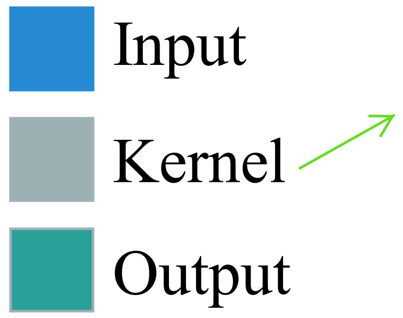
2 2 2
2 2 2
2 2 2

겹치는 값은 더해짐
결국
3 5 2 2
3 5 2 2
이렇게 됨

Transpose Convolution (Upsampling)

원본 3x3 Input을 밑의 그림처럼 찢는다.





w11 w12 w13
w21 w22 w23
w31 w32 w33

이라고 했을때
또 filter가 찢은 input에서 한칸씩 이동한다고 하자

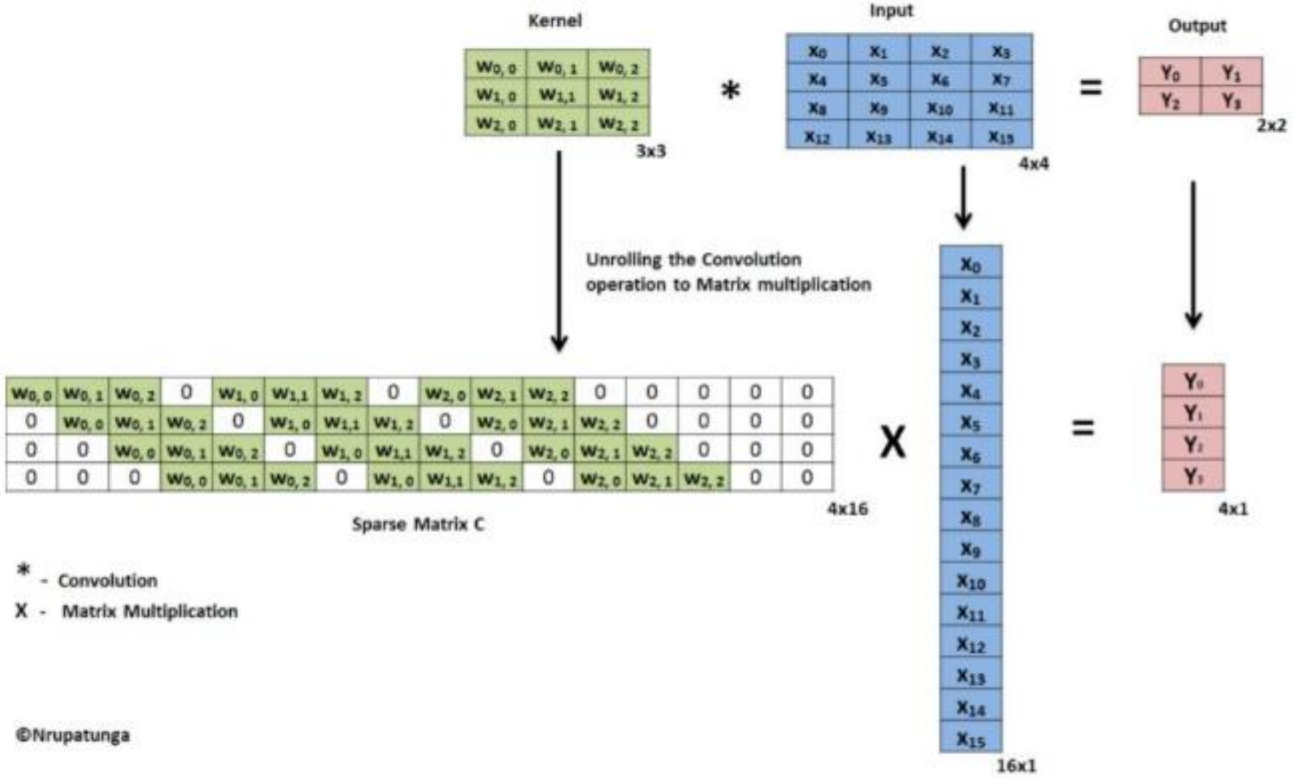
filter가 이동하면서 filter에서 input에 위치한 w값만 더해서 output추출

동작 방식

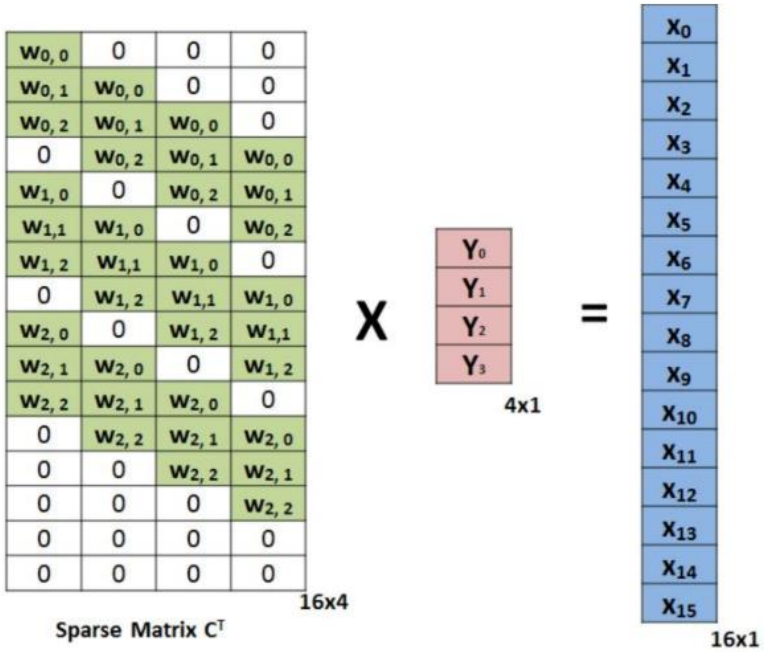
1) Input 내 각 픽셀 주위에 zero-padding
2) Zero-padding된 input에 convolution 연산

Transpose Convolution (Upsampling)

- Convolution



- Deconvolution

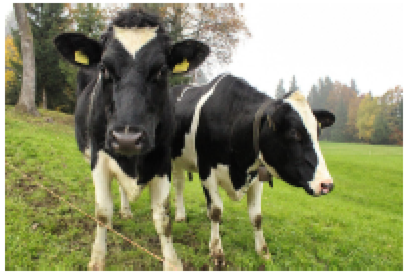


Semantic Segmentation

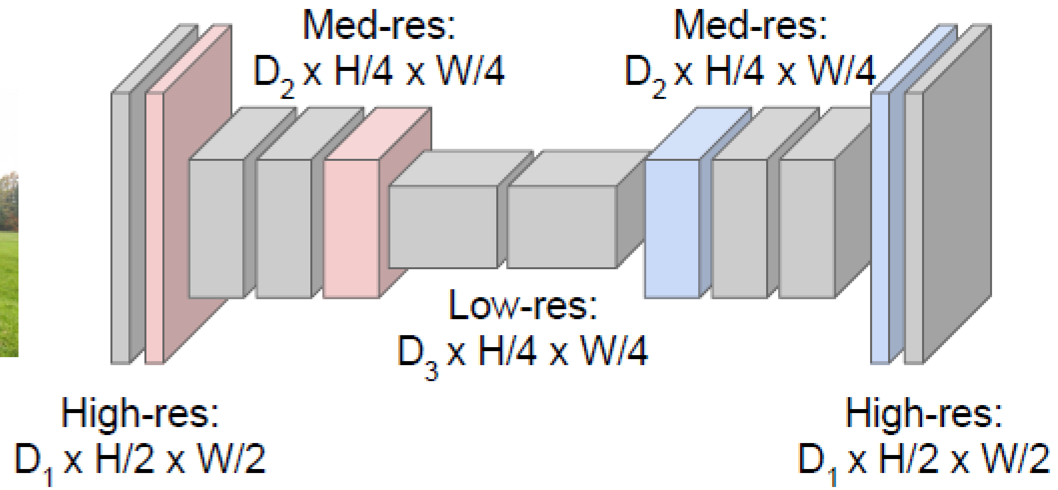
Downsampling: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: Unpooling or strided transpose convolution



Input:
3 x H x W



Predictions:
H x W

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

Several types of convolution

1) Convolution

2) Transpose Convolution

3) Atrous Convolution

4) Separable Convolution

1, 3, 4는 동일한 역할
사이즈를 줄이거나 동일하게

사이즈를 늘리는 역할

성능이 굉장히 좋다

속도가 빠르다. 모델이 가볍다.

ex) mobileNet - 기존 속도 100배

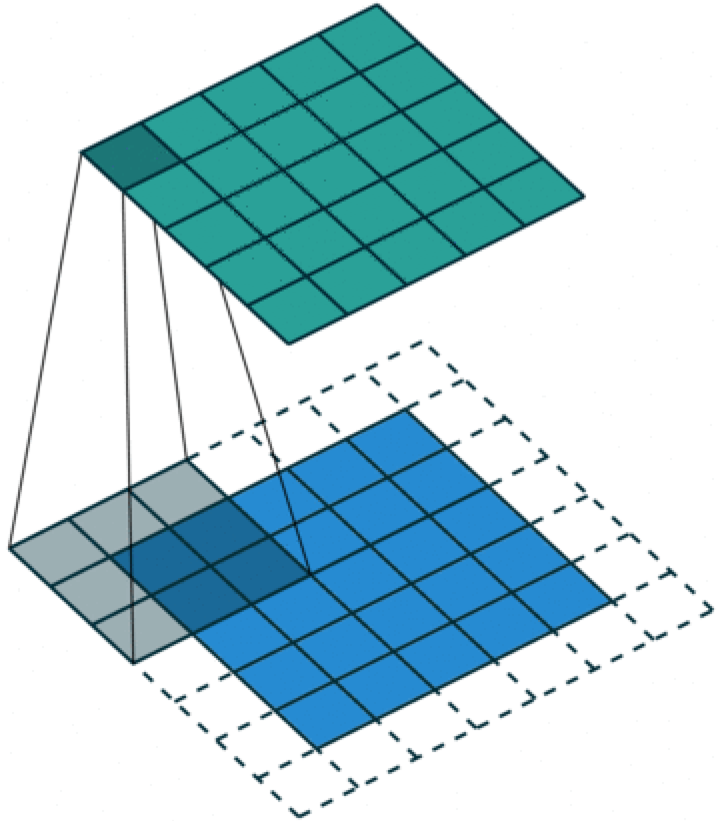
※ 실습 코드가 매우 중요
main.py - 네이버 개발자가 만듦
train.py - github + 강사 수정
Object Detection - Apple AI 개발자

Several types of convolution

1) Convolution

DeConvolution 구글링하면 애니메이션으로 이해 가능.

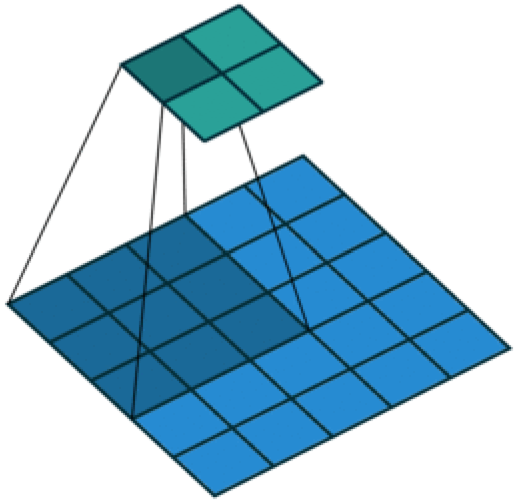
- Down-sampling



Several types of convolution

2) Transpose Convolution

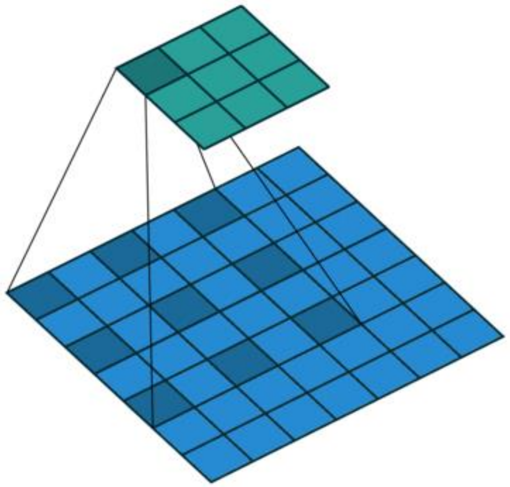
- Up-sampling
- Checker boarder Issue



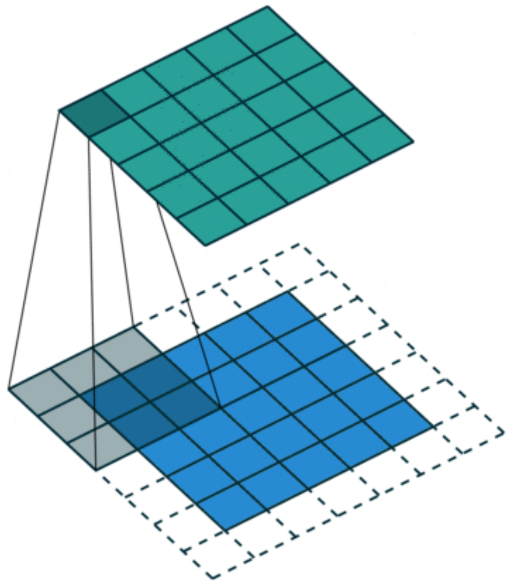
Several types of convolution

3) Atrous Convolution

- Down-sampling
- Wider Field of view at the same computational cost
- Use it when you need a wide field of view (Not good if field of view is too small)



Atrous convolution



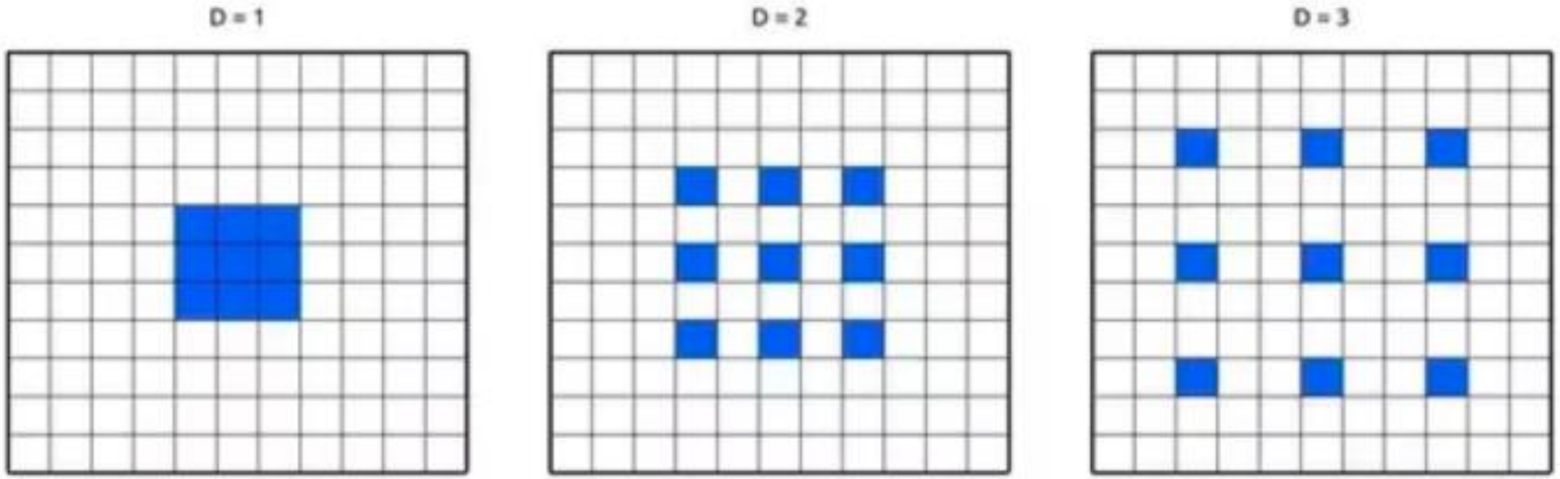
Convolution

Several types of convolution

3) Atrous Convolution

다양한 Dilated rate를 이용하여 병렬적으로 사용해서 더 많은 특징 추출가능

(ex. Deeplab v3+)



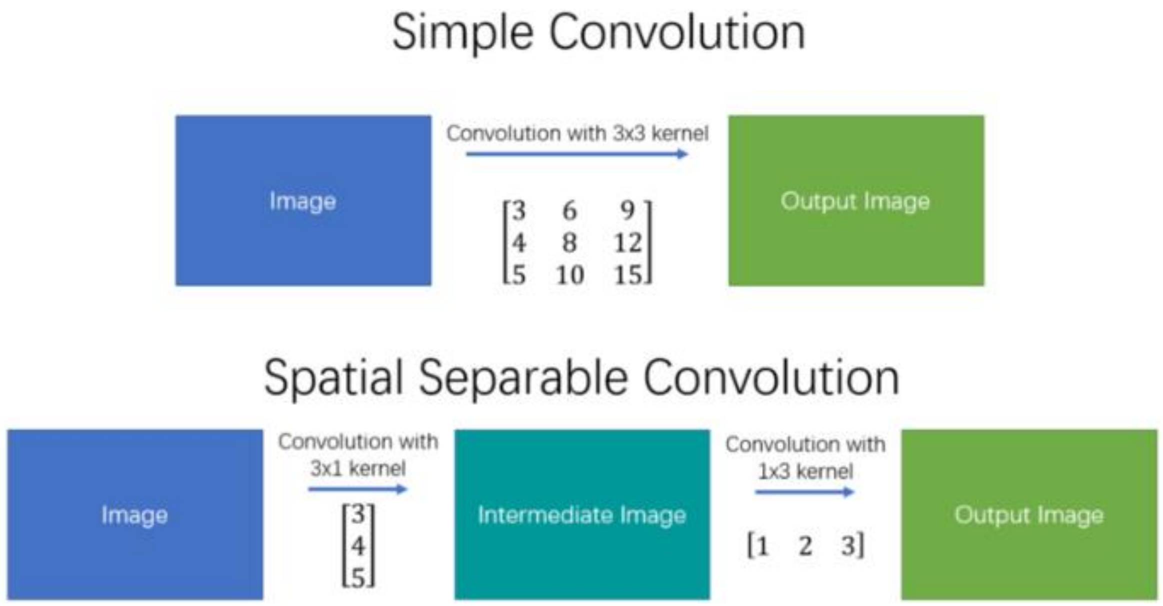
Atrous Convolution. 왼쪽부터 dilation rate: 1, 2, 3

Several types of convolution

4) Separable Convolution

기존의 Convolution와 다르게 공간과 관련된 Convolution와 채널과 관련된 Convolution을 따로 적용하여
<u>**기존 Convolution을 표현할 수 있으면서도 파라미터 수를 낮추는 Convolution**</u>

(Guo et al. Network Decoupling: From Regular to Depthwise Separable Convolutions)



Several types of convolution

4) Separable Convolution

- Normal Convolution

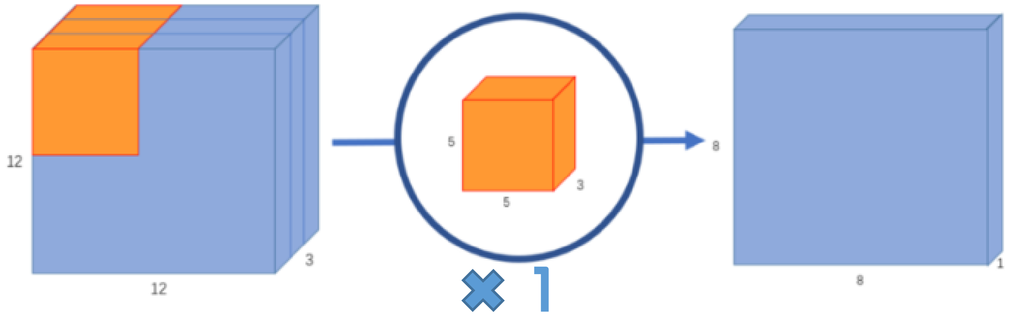


Image 4: A normal convolution with 8x8x1 output

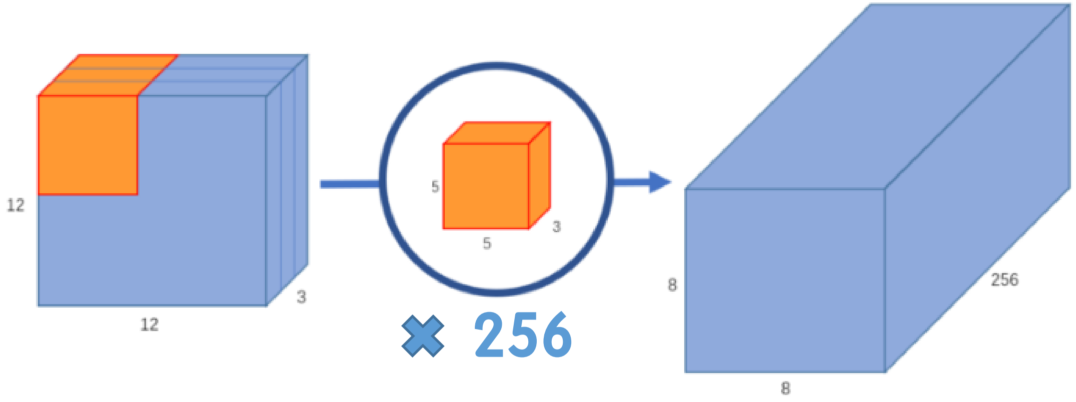


Image 5: A normal convolution with 8x8x256 output

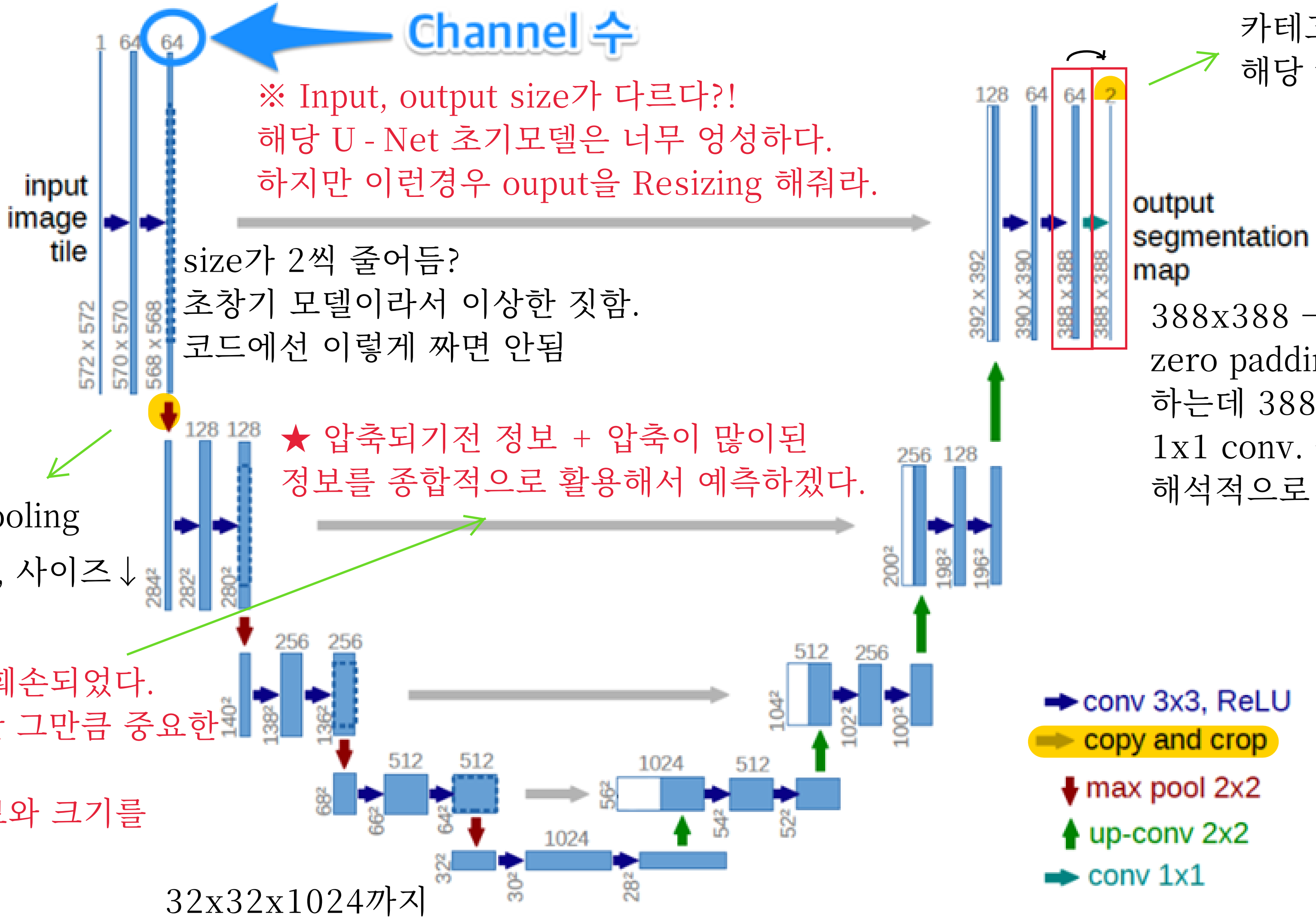
U-Net

고해상도 input의 경우
decoder 도중 더 이상 못 올라오고 학습을 못하는 경우가 생긴다.
이때는 결과물은 작게 Resizing 해줘서
학습이 가능하도록 해주는 경우가 있다. 자율주행이 보통 그럼!!

DownSampling
Convolution

Encoder 압축(사이즈 ↓, 채널↑)

Decoder 압축(사이즈 ↑, 채널↓)



※ Input, output size가 다르다?!
해당 U - Net 초기모델은 너무 엉성하다.
하지만 이런경우 ouput을 Resizing 해줘라.

size가 2씩 줄어듬?
초창기 모델이라서 이상한 짓함.
코드에선 이렇게 짜면 안됨

카테고리 2개
해당 u - net은 cell detection모델

388x388 → 388x388
zero padding을 안해서 원래 386이여야
하는데 388로 나옴.
1x1 conv. 을 사용해서 그럼!!
해석적으로 중요!코드에 영향 미침.

★ 압축되기전 정보 + 압축이 많이된
정보를 종합적으로 활용해서 예측하겠다.

maxpooling
채널↑, 사이즈↓

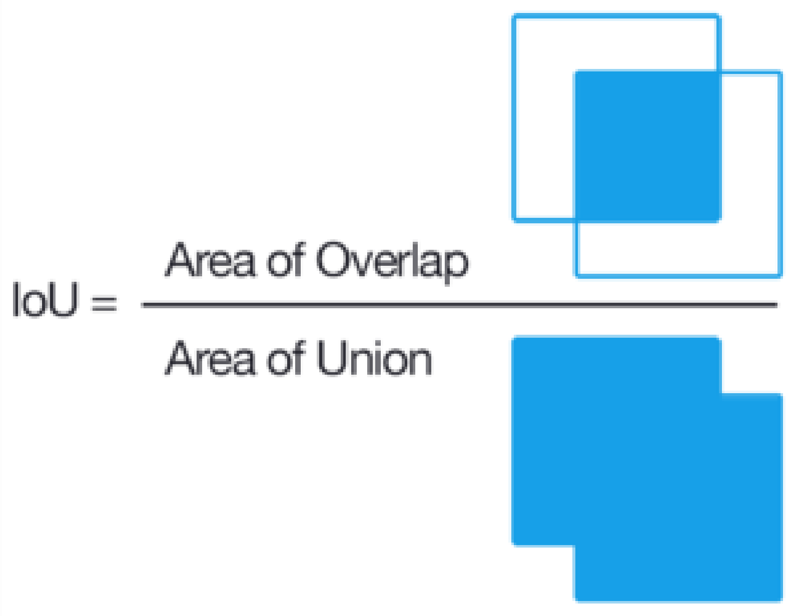
압축을 많이 했다.
→ 원본 이미지 정보가 많이 훼손되었다.
정보 손실이 많아졌다. 하지만 그만큼 중요한
정보를 갖고 있긴하다.
concatenation은 이러한 정보와 크기를
늘린 정보를 붙인것

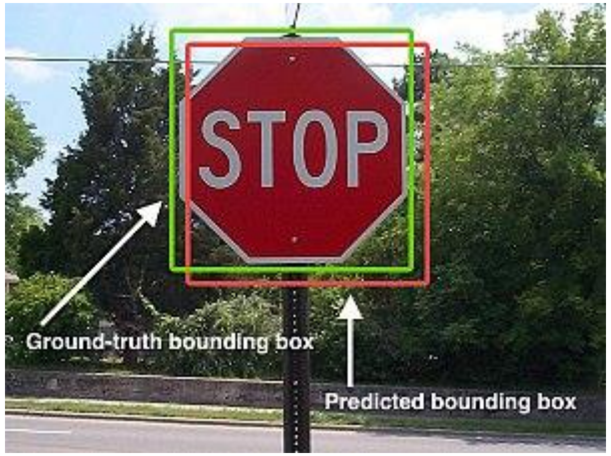
concatenation
채널 방향으로 붙음.
size가 동일해야 한다.
채널은 상관없다.

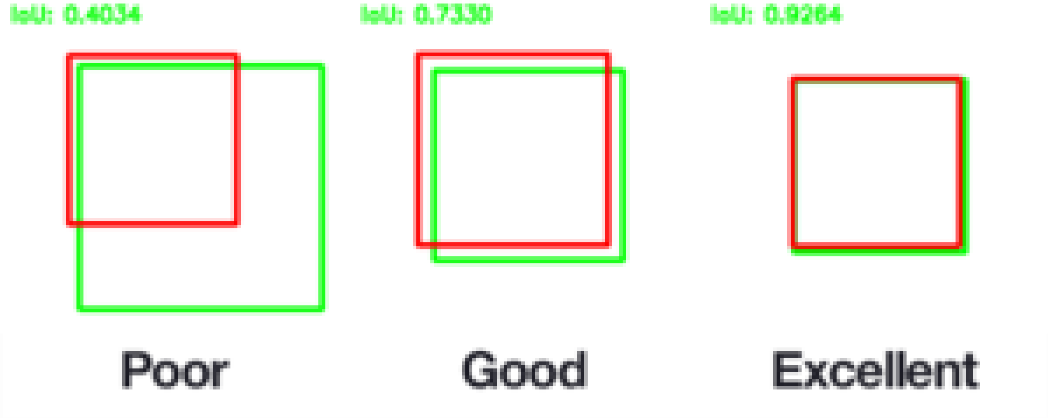
32x32x1024까지

U-Net

- IOU (Intersection over union)







U-Net

- Checkerboard Artifacts on deconvolution

https://distill.pub/2016/deconv-checkerboard/

U-Net

- Deconv vs. Interpolation

Using deconvolution.
Heavy checkerboard artifacts.

deconvolution

Using resize-convolution.
No checkerboard artifacts.

Interpolation

Evaluation Matrix

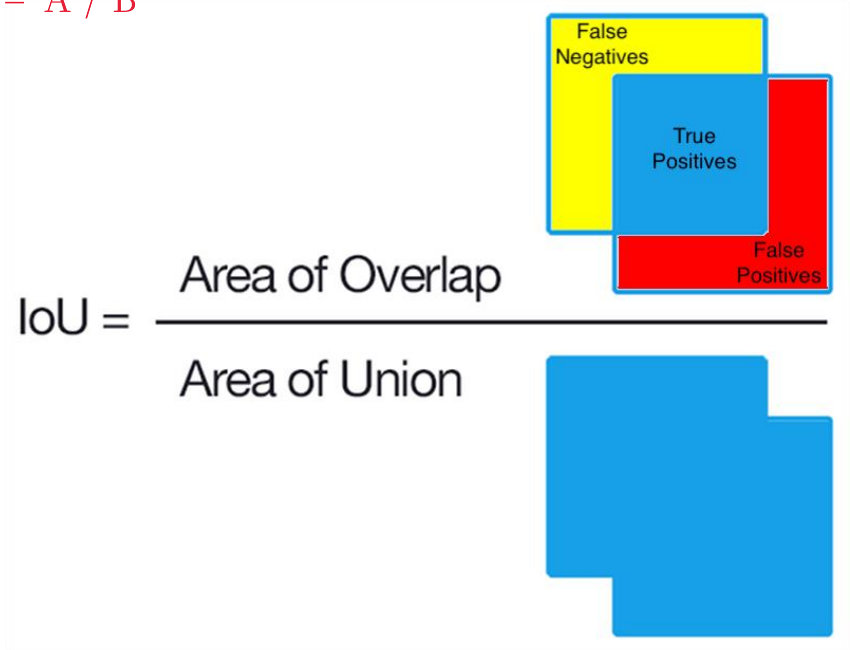
기존과 다른 성능 지표가 필요함.

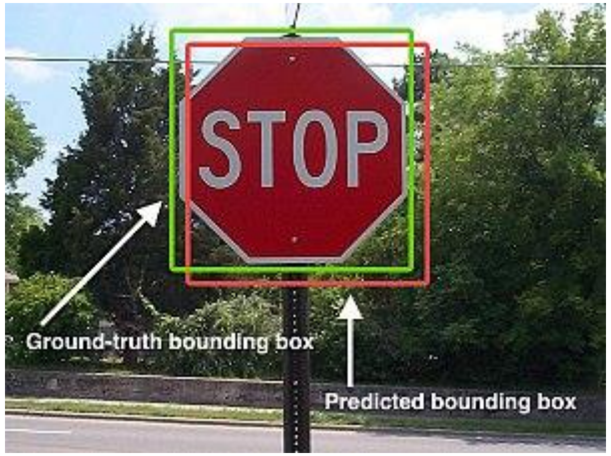
예측이랑 정답이랑 겹쳐서 그 차이에 따른 성능 평가

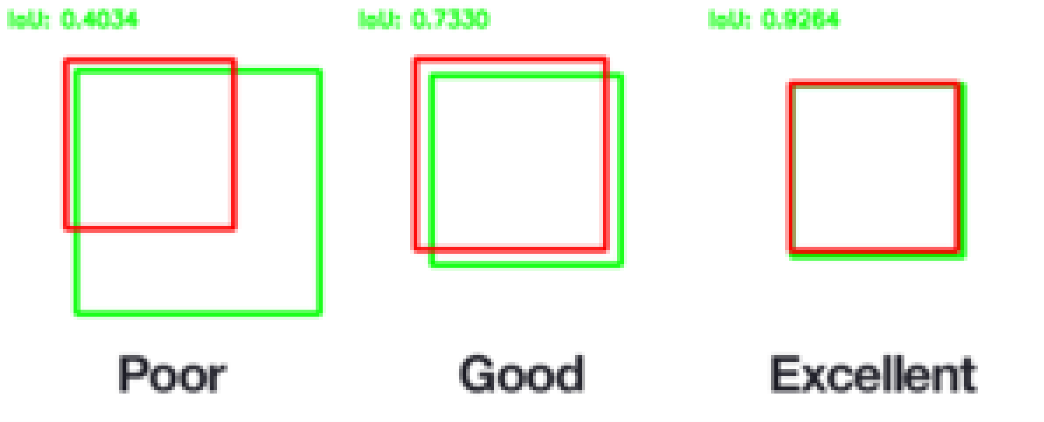
- IOU (Intersection over union)

DCE = 2A / (A+B)

IOU = A / B







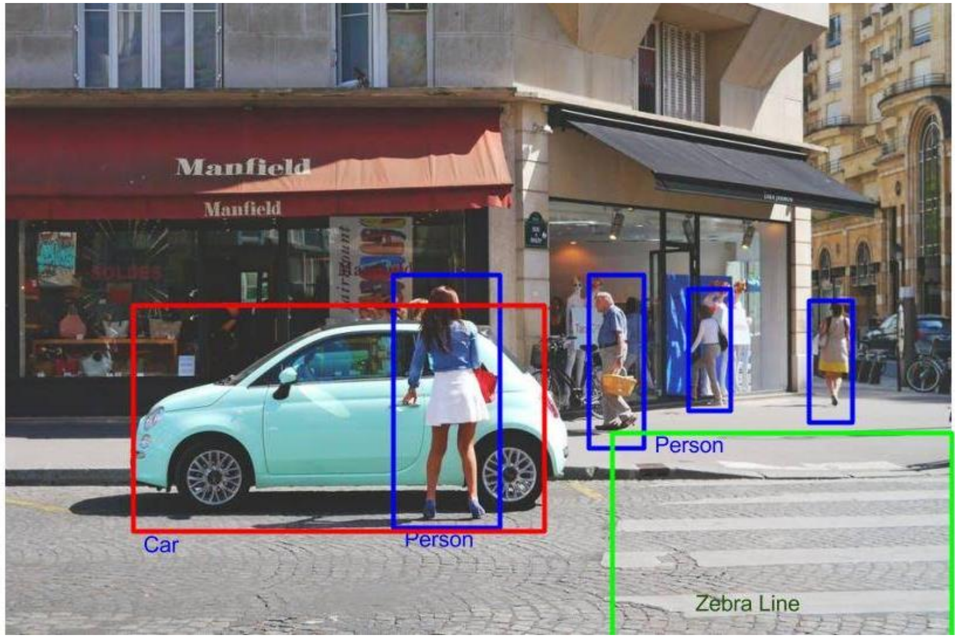
0.7 기준으로 보통 판단함

Evaluation Matrix

모든 IOU의 평균 값

- mIOU (Mean intersection over union)

Segmentation 에서 가장 일반적으로 사용하는 성능지표

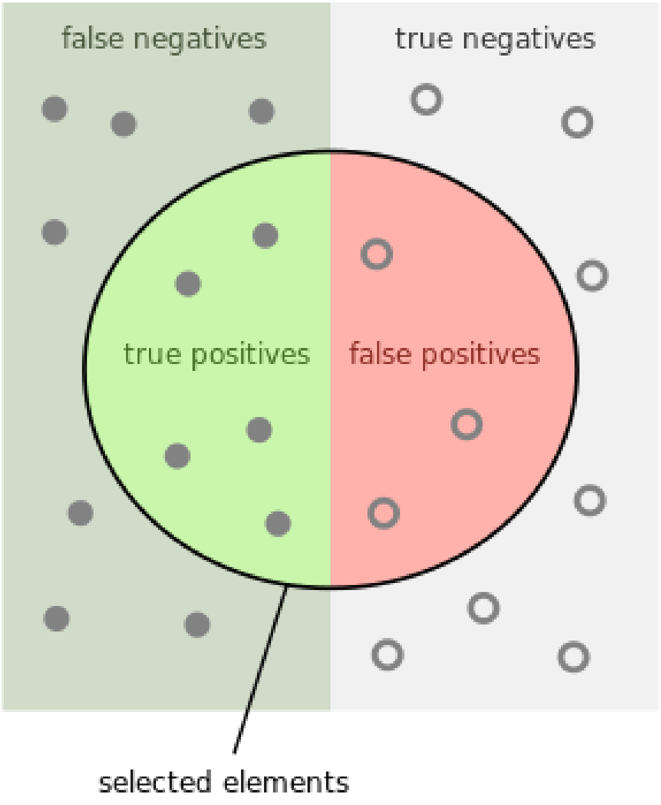


Evaluation Matrix

- Dice Coefficient

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|}$$

$$DSC = \frac{2TP}{2TP + FP + FN}.$$



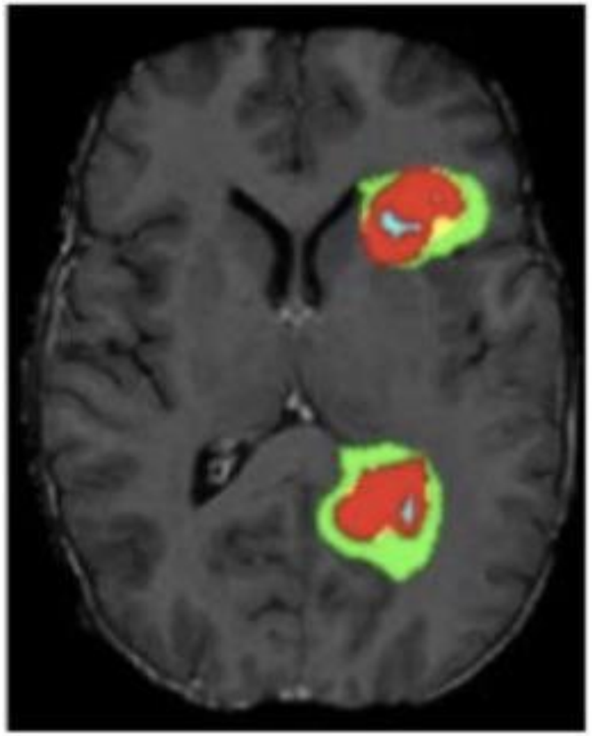
답지에 해당하는 부분
큰 물체 + 자주 등장 : mIOU
작은 물체 + 적게 등장 : DCE

데이터에 따른 유동적인 평가 지표 사용이 필요하다.!!

Evaluation Matrix

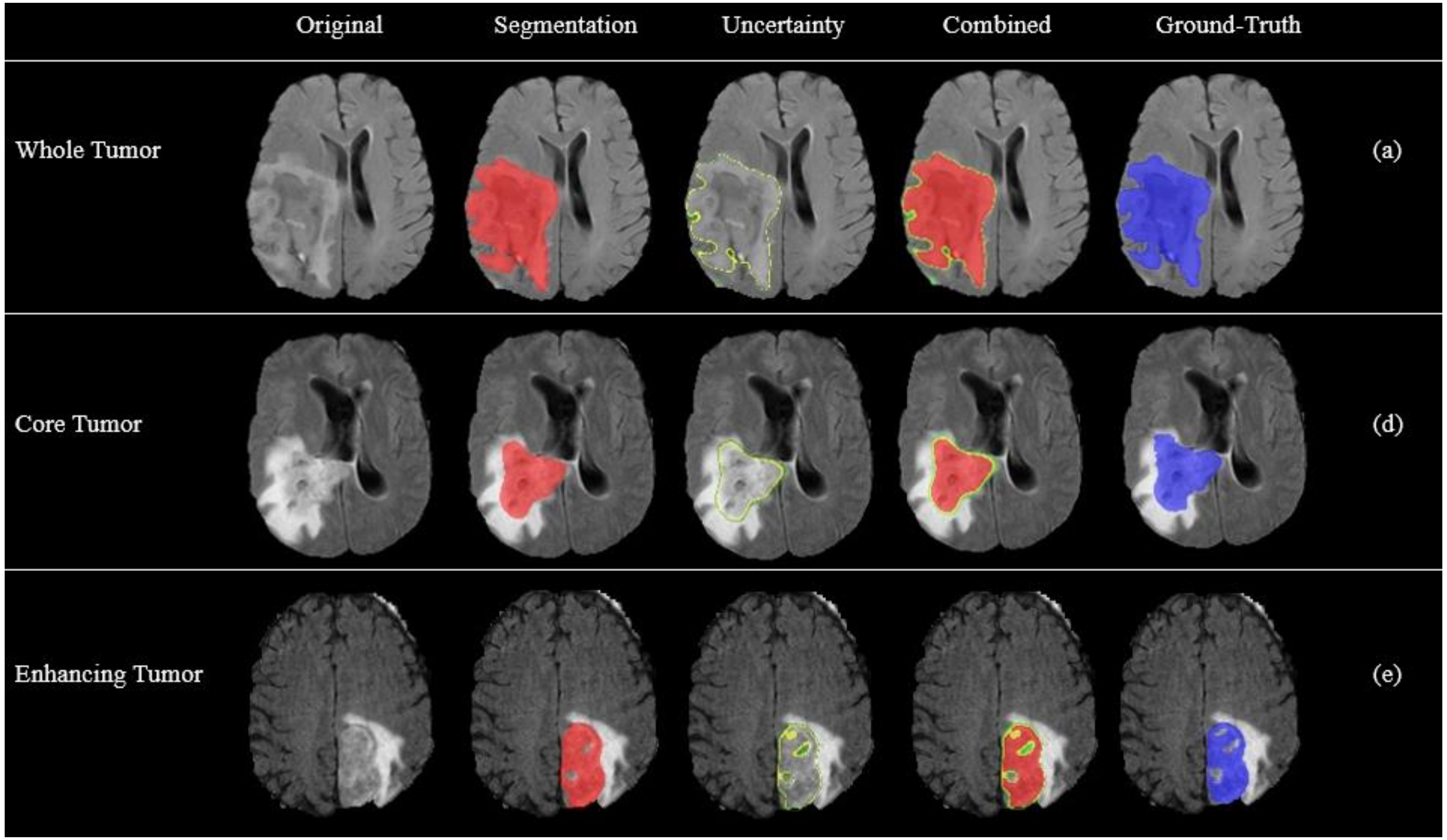
- Dice Coefficient

$$DSC = \frac{2TP}{2TP + FP + FN}.$$

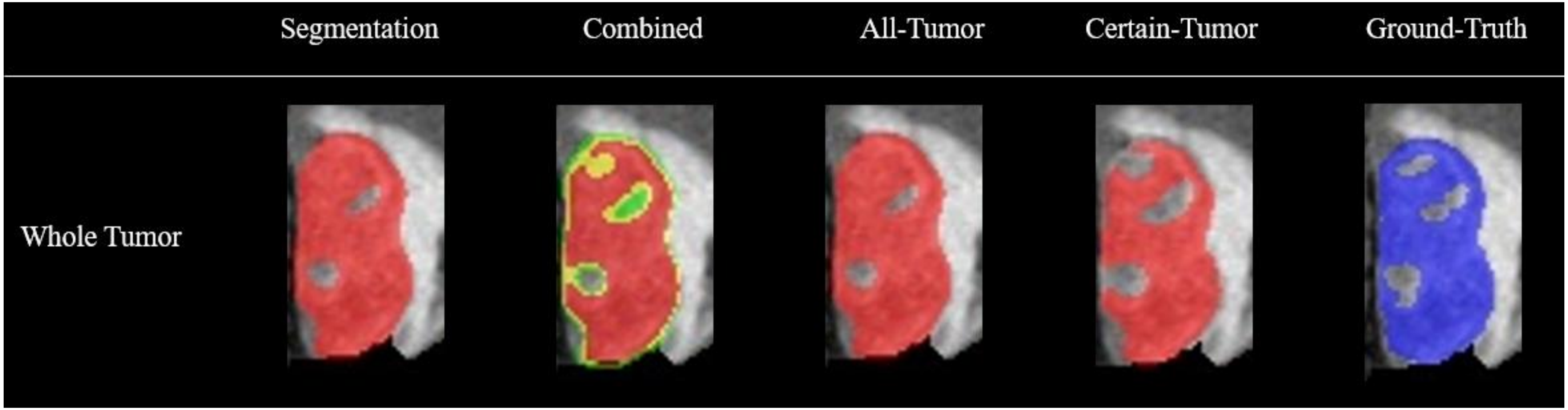


Very Effective to train Imbalanced dataset

Uncertainty Quantification



Uncertainty Quantification



Let’s try it !

(code)