

파이썬 뽀수기

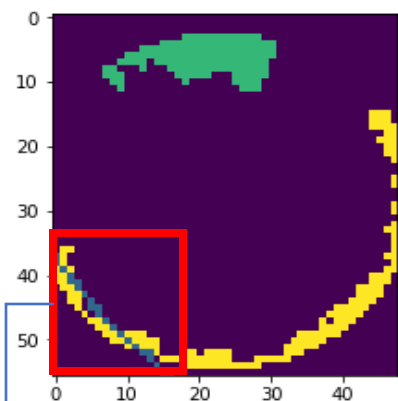
Fully Convolutional Network for Segmentation

2020.10.17

김현우

Image Segmentation

Segmentation – Pixel 단위의 Classification 방법



0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0
2	2	0	0	0	0	0	0
1	2	2	0	0	0	0	0
1	1	0	2	1	0	0	0
0	1	1	1	2	1	1	0
0	0	1	1	1	2	1	1

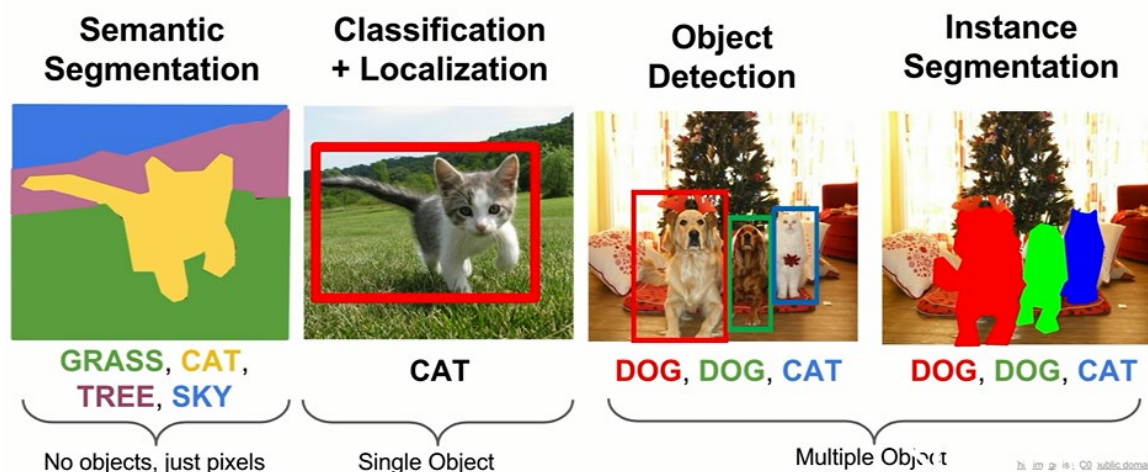
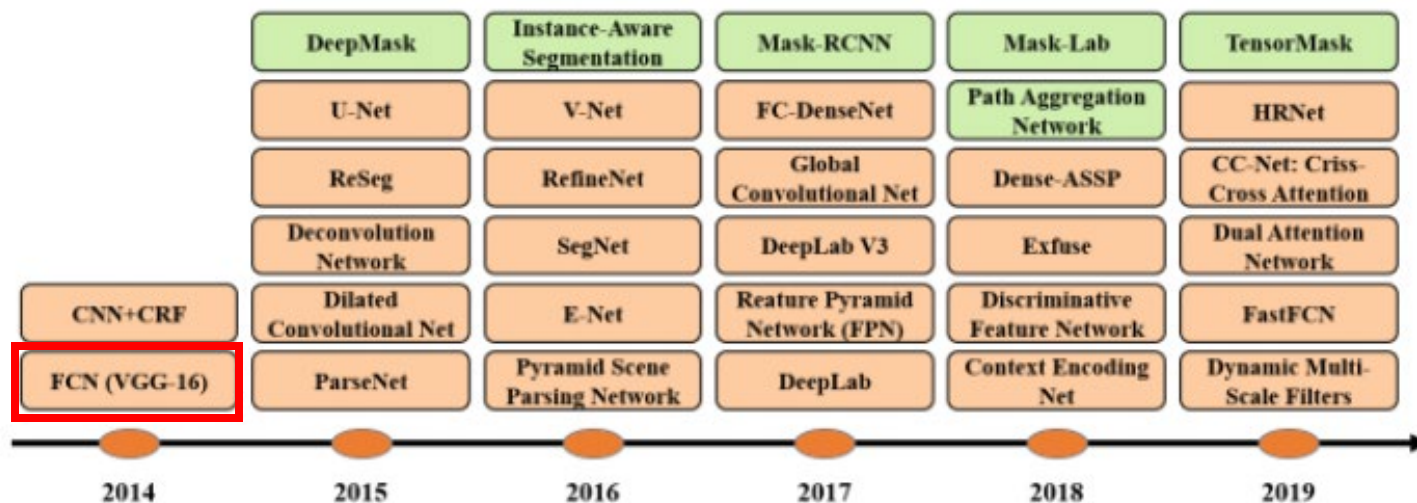


Image Segmentation : Pixel이 불량(Object)의 종류에 따라 Classification 가능

- Background : 0
- Ring : 1
- Scratch : 2
- Local Zone : 3
- Circle : 4

FCN (Fully Convolutional Networks for Semantic Segmentation)

✓ FCN 논문을 선택한 이유



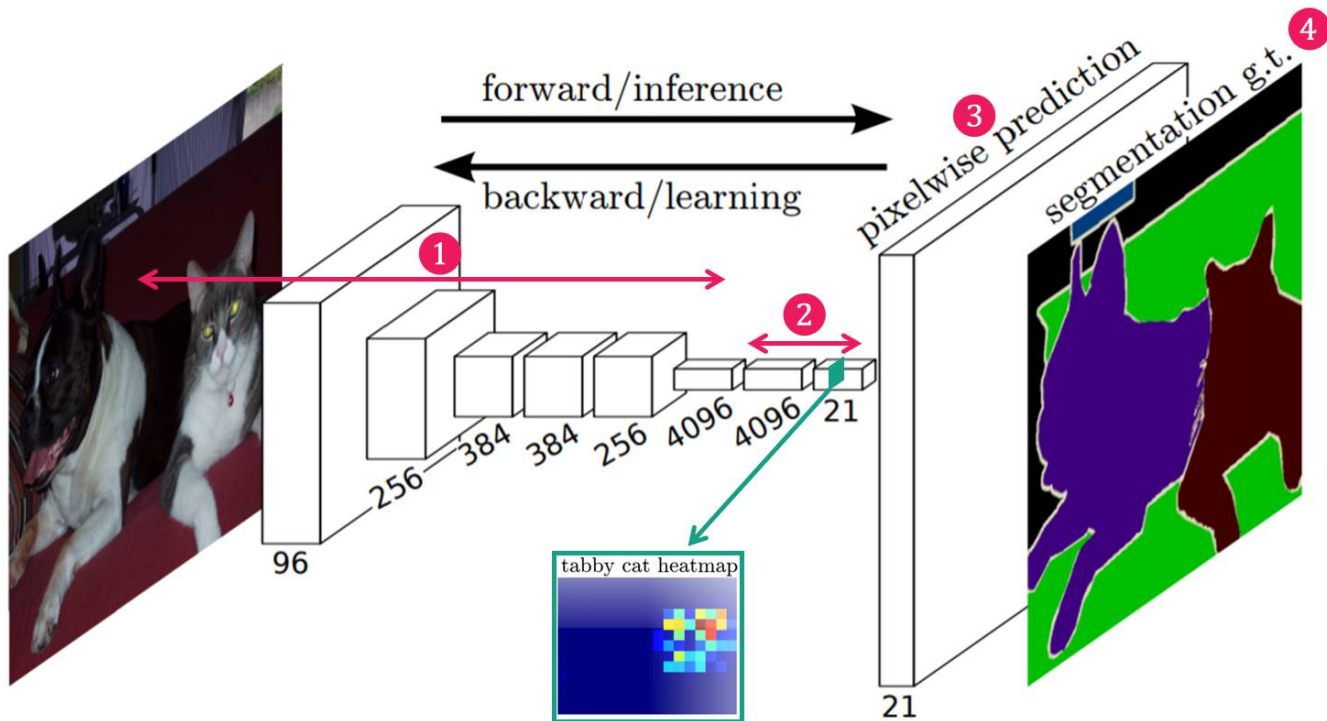
1. 2014년도에 나왔지만, Segmentation 계열에서 딥러닝을 쓴 최초의 논문
2. 이후의 논문을 이해하기 위한, 배경지식이 되는 논문

2

FCN (Fully Convolutional Networks for Semantic Segmentation)

✓ Abstract

1. AlexNet을 시작으로 하는 CNN 계열 모델의 발전을 Segmentation에 접목 (Resnet, VGG 등)
2. Fully Convolutional + Skip Architecture 두가지 방법을 도입



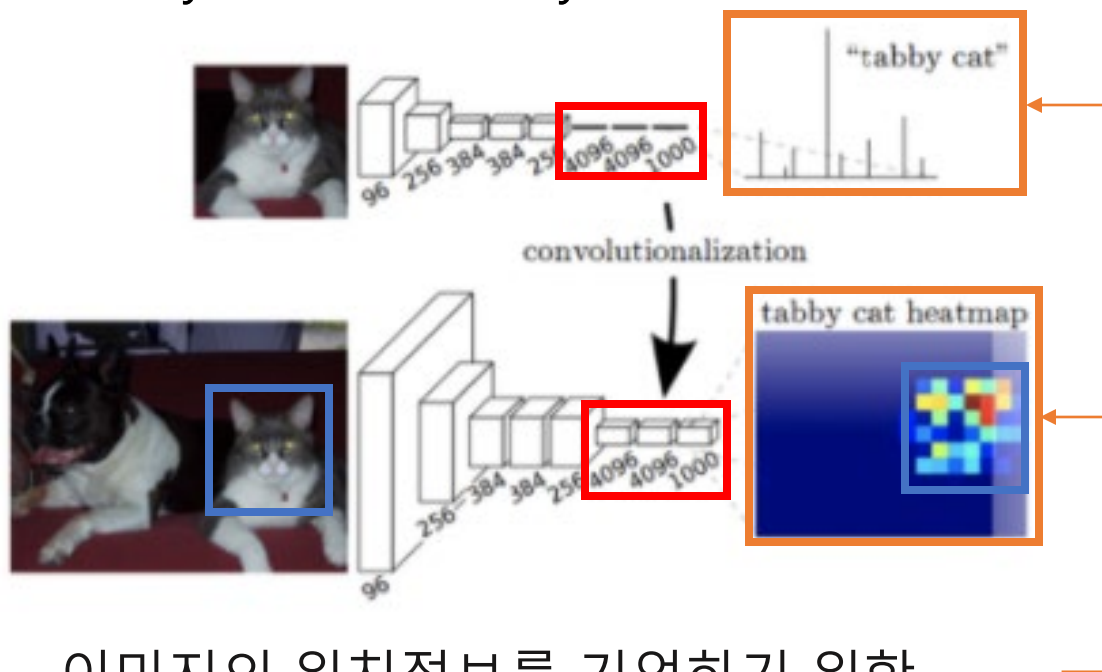
FCN (Fully Convolutional Networks for Semantic Segmentation)



Fully Convolutional

(자세한 내용은 Appendix 참고)

정의 : Fully Connected Layer를 1x1 Convolution으로 변경



Fully Connected vs Fully Convolution

- Fully Connected Layer는 단순히 개 vs 고양이의 정보만 제공
- Fully Convolutional Layer는 위치 정보를 같이 제공

1. 이미지의 위치정보를 기억하기 위함
2. **임의의 입력 크기**에 대해서도 일관성있는 결과를 생성하려는 의도

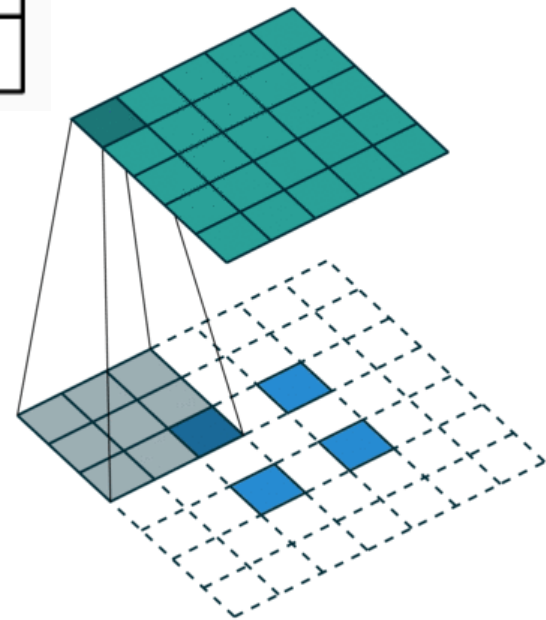
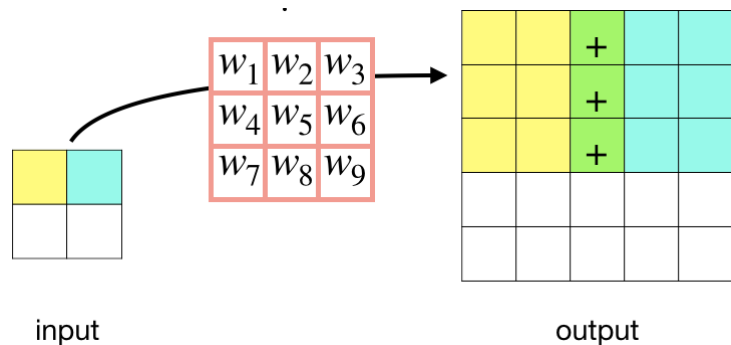
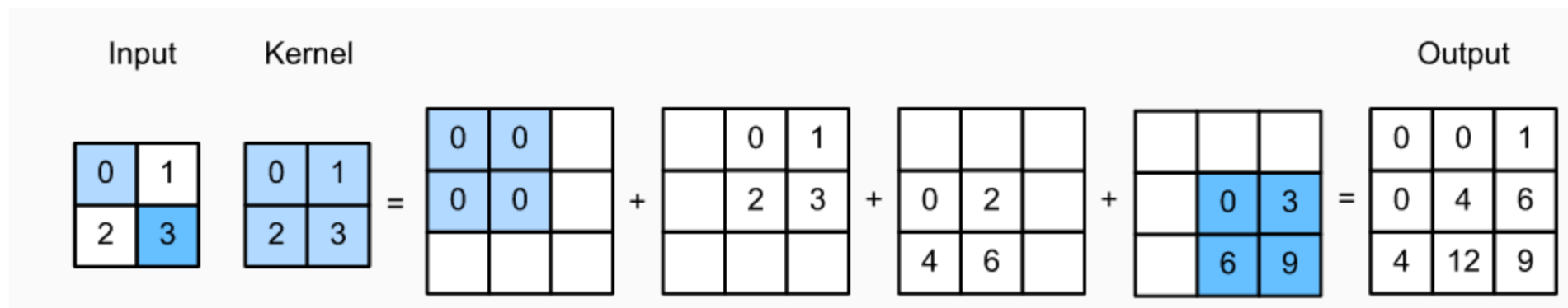
(Fully connected layer는 입력의 크기가 동일해야 하는데, Convolution 는 상관없음)

2

FCN (Fully Convolutional Networks for Semantic Segmentation)

✓ UpSampling (Deconvolution, Transposed Convolution) (자세한 내용은 Appendix 참고)

정의 : Max Pooling에 의해서 감소한 이미지의 크기를 원본 이미지의 크기로 복원

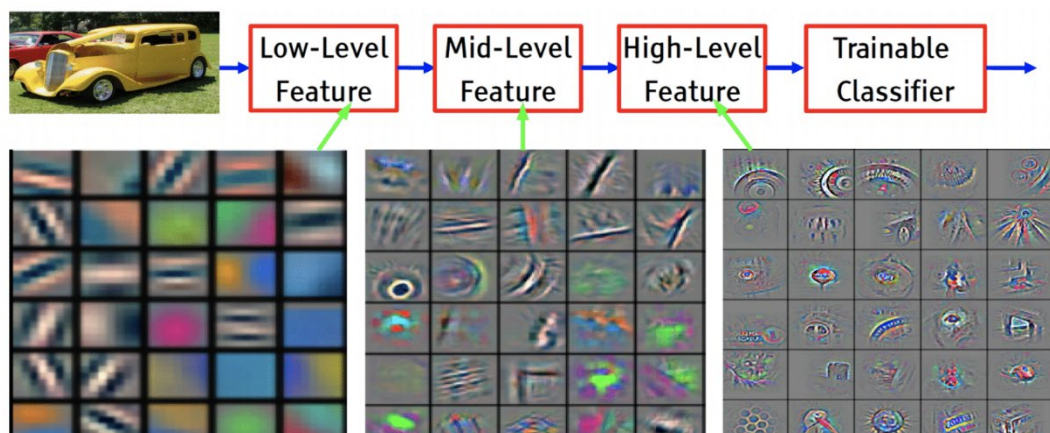


2

FCN (Fully Convolutional Networks for Semantic Segmentation)

✓ Skip Architecture

아이디어: 얇은 층과 깊은 층의 다른 특징을 결합하려는 시도

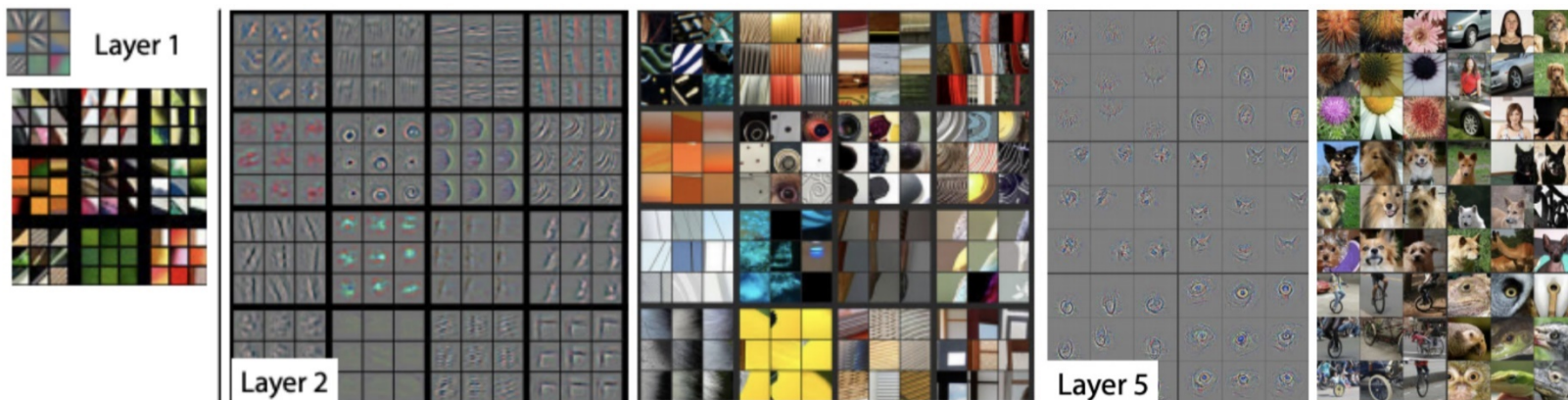


Low Layer

- 직선 및 곡선, 색상 등의 낮은 수준의 특징 (local feature)

High Layer

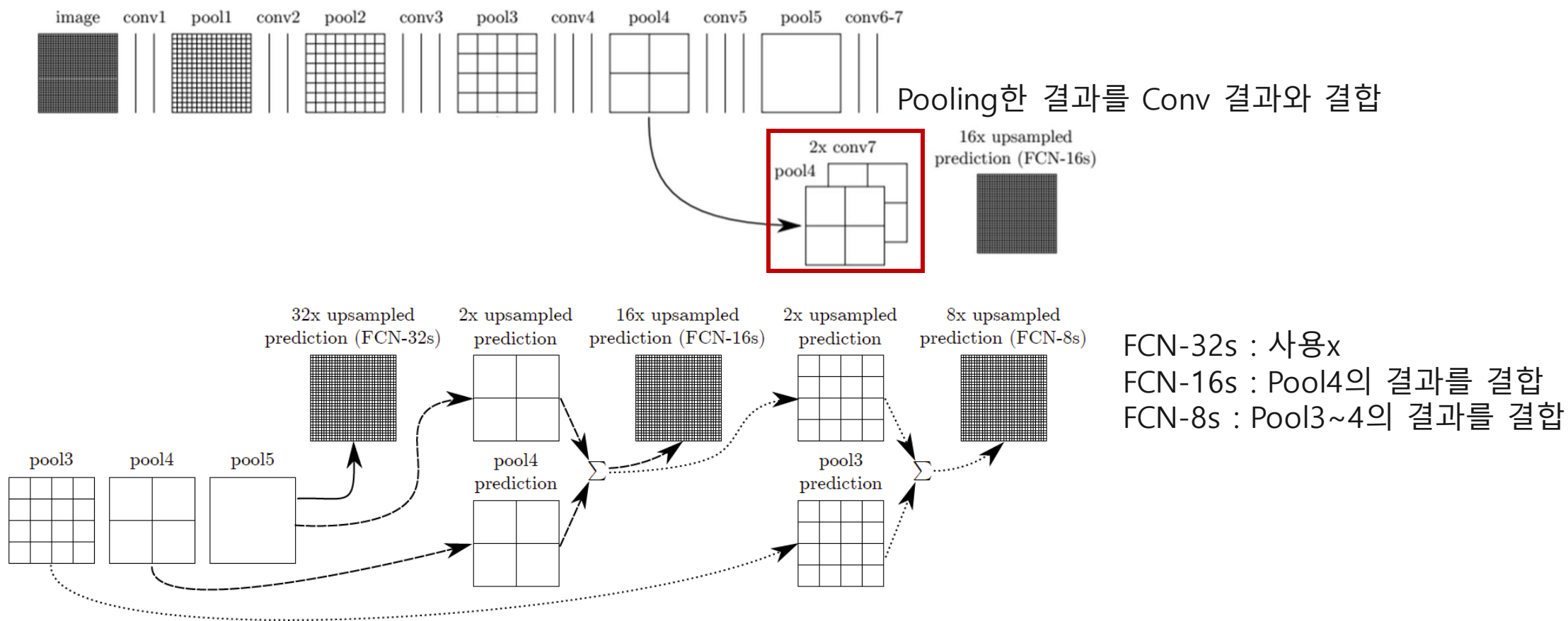
- 복잡하고 포괄적인 개체 정보가 활성화 (global feature)



FCN (Fully Convolutional Networks for Semantic Segmentation)

✓ Skip Architecture

아이디어: Resnet의 아이디어를 이용해서, 낮은 층의 결과를 결합



3

평가함수 설정 – Pixel Accuracy, mIoU

✓ 내용

1. Pixel Accuracy

입력



segmented

- 1: Person
- 2: Purse
- 3: Plants/Grass
- 4: Sidewalk
- 5: Building/Structures

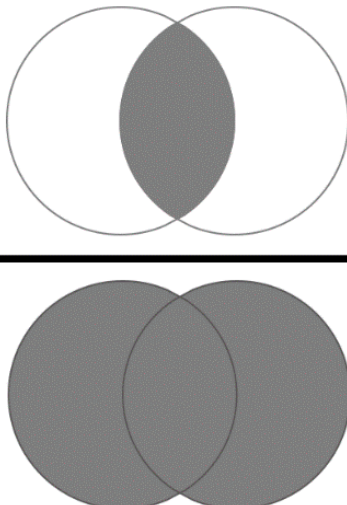
출력

3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	5	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4	4

Segmentation map

✓ 내용

2. mIoU (5개의 Class에 대해서 계산하고 평균)

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$




FCN32

```
# conv1
self.conv1_1 = nn.Conv2d(3, 64, 3, padding=100)
self.relu1_1 = nn.ReLU(inplace=True)
self.conv1_2 = nn.Conv2d(64, 64, 3, padding=1)
self.relu1_2 = nn.ReLU(inplace=True)
self.pool1 = nn.MaxPool2d(2, stride=2, ceil_mode=True) # 1/2

# conv2
self.conv2_1 = nn.Conv2d(64, 128, 3, padding=1)
self.relu2_1 = nn.ReLU(inplace=True)
self.conv2_2 = nn.Conv2d(128, 128, 3, padding=1)
self.relu2_2 = nn.ReLU(inplace=True)
self.pool2 = nn.MaxPool2d(2, stride=2, ceil_mode=True) # 1/4

# conv3
self.conv3_1 = nn.Conv2d(128, 256, 3, padding=1)
self.relu3_1 = nn.ReLU(inplace=True)
self.conv3_2 = nn.Conv2d(256, 256, 3, padding=1)
self.relu3_2 = nn.ReLU(inplace=True)
self.conv3_3 = nn.Conv2d(256, 256, 3, padding=1)
self.relu3_3 = nn.ReLU(inplace=True)
self.pool3 = nn.MaxPool2d(2, stride=2, ceil_mode=True) # 1/8

# conv4
self.conv4_1 = nn.Conv2d(256, 512, 3, padding=1)
self.relu4_1 = nn.ReLU(inplace=True)
self.conv4_2 = nn.Conv2d(512, 512, 3, padding=1)
self.relu4_2 = nn.ReLU(inplace=True)
self.conv4_3 = nn.Conv2d(512, 512, 3, padding=1)
self.relu4_3 = nn.ReLU(inplace=True)
self.pool4 = nn.MaxPool2d(2, stride=2, ceil_mode=True) # 1/16

# conv5
self.conv5_1 = nn.Conv2d(512, 512, 3, padding=1)
self.relu5_1 = nn.ReLU(inplace=True)
self.conv5_2 = nn.Conv2d(512, 512, 3, padding=1)
self.relu5_2 = nn.ReLU(inplace=True)
self.conv5_3 = nn.Conv2d(512, 512, 3, padding=1)
self.relu5_3 = nn.ReLU(inplace=True)
self.pool5 = nn.MaxPool2d(2, stride=2, ceil_mode=True) # 1/32

# fc6
self.fc6 = nn.Conv2d(512, 4096, 7)
self.relu6 = nn.ReLU(inplace=True)
self.drop6 = nn.Dropout2d()

# fc7
self.fc7 = nn.Conv2d(4096, 4096, 1)
self.relu7 = nn.ReLU(inplace=True)
self.drop7 = nn.Dropout2d()

self.score_fr = nn.Conv2d(4096, n_class, 1)
self.upscore = nn.ConvTranspose2d(n_class, n_class, 64, stride=32,
                                  bias=False)
```



FCN16

```
self.score_fr = nn.Conv2d(4096, n_class, 1)
self.score_pool4 = nn.Conv2d(512, n_class, 1)

self.upscore2 = nn.ConvTranspose2d(
    n_class, n_class, 4, stride=2, bias=False)
self.upscore16 = nn.ConvTranspose2d(
    n_class, n_class, 32, stride=16, bias=False)

h = self.score_fr(h)
h = self.upscore2(h)
upscore2 = h # 1/16

h = self.score_pool4(pool4)
h = h[:, :, 5:5 + upscore2.size()[2], 5:5 + upscore2.size()[3]]
score_pool4c = h # 1/16

h = upscore2 + score_pool4c

h = self.upscore16(h)
h = h[:, :, 27:27 + x.size()[2], 27:27 + x.size()[3]].contiguous()
```

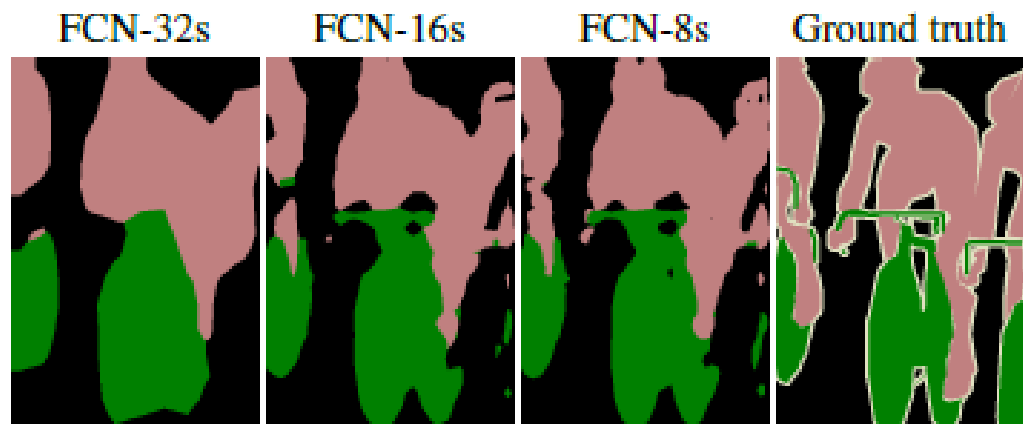
✓ Results

Skip Connection에 의한 결과

	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s-fixed	83.0	59.7	45.4	72.0
FCN-32s	89.1	73.3	59.4	81.4
FCN-16s	90.0	75.7	62.4	83.0
FCN-8s	90.3	75.9	62.7	83.2

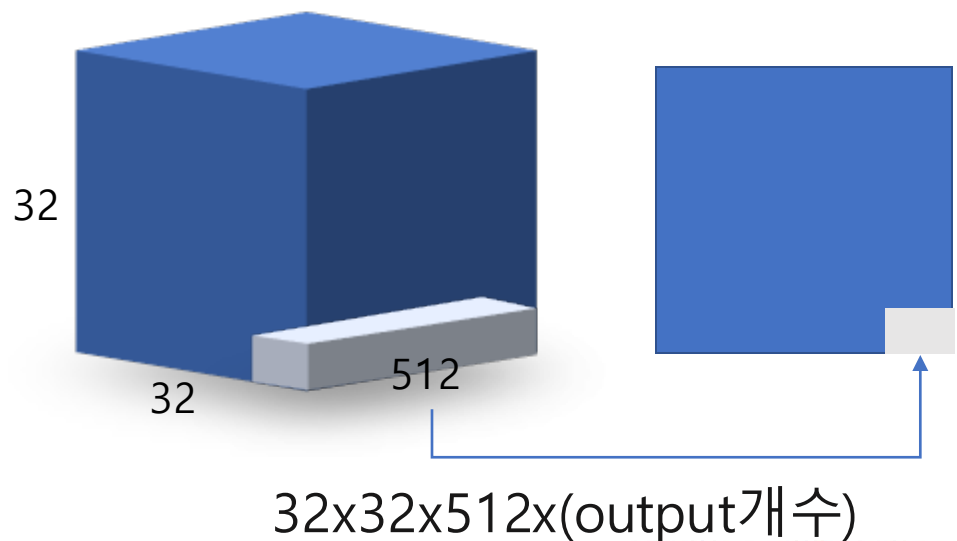
Network 구조에 의한 결과

	FCN- AlexNet	FCN- VGG16	FCN- GoogLeNet ⁴
mean IU	39.8	56.0	42.5
forward time	50 ms	210 ms	59 ms
conv. layers	8	16	22
parameters	57M	134M	6M
rf size	355	404	907
max stride	32	32	32



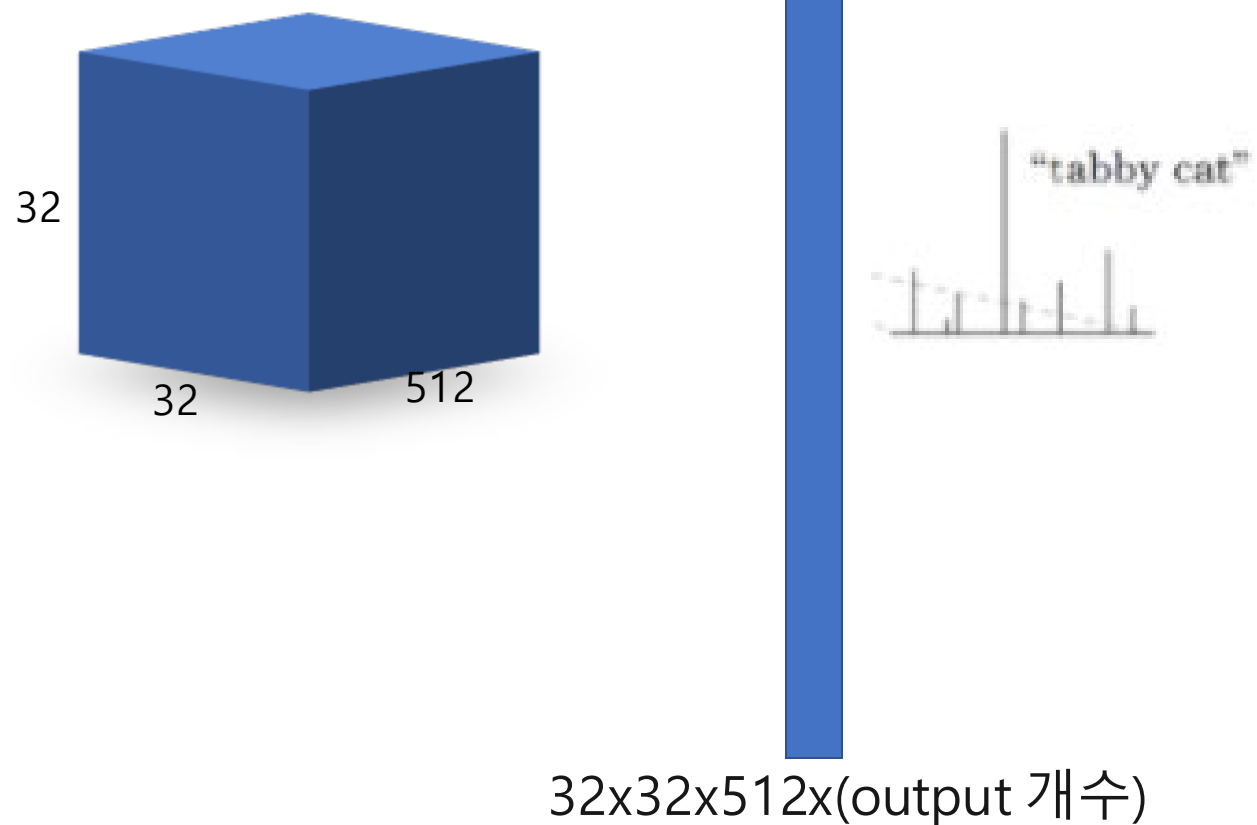
✓ 1x1 Convolution

각 픽셀의 위치정보를 해치지 않은채로 특징 추출



✓ Fully Connected

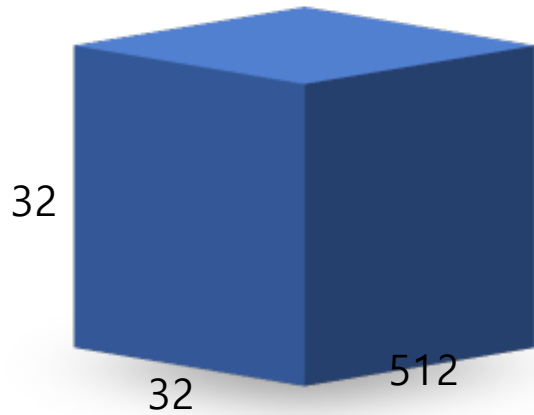
각 픽셀의 위치정보를 해침



✓ 1x1 Convolution

1x1 Convolution을 사용할 경우, 임의의 입력값에 대해서도 상관 없는 이유

-> Convolution은 kernel의 파라미터에 의해 영향을 받고, 이미지 혹은 레이어의 크기에 대해서는 상관 없음



`nn.Conv2d(input_channel, output_channel, 1, 1)`

-> Height, width와 상관이 없음



입력

`nn.Linear(input_channel * height * width, output_size)`

-> Height, width와 상관이 있음



Transposed Convolutions

의미 : Convolution 된 결과에 Deconvolution을 해도 그대로 나오지 않음. Convolution을 입력값에 대해 미분한 값에 Y를 곱한 값이 출력됨 (의미상으로 Convolution의 Transpose를 계산한 것임)

No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
No padding, strides, transposed	Padding, strides, transposed	Padding, strides, transposed (odd)	

$$\text{TransposedConvolution}(Y, W) = Y \cdot \frac{\partial \text{Conv}(X, W)}{\partial X}$$

크기는 같지만, 연산의 결과는 다를 수 있기에 Deconvolution이라는 표현은 수학적으로는 정확하지 않음



Convolutions

kernel (3x3)

$W_{0,0}$	$W_{0,1}$	$W_{0,2}$
$W_{1,0}$	$W_{1,1}$	$W_{1,2}$
$W_{2,0}$	$W_{2,1}$	$W_{2,2}$



input (4x4)

$X_{0,0}$	$X_{0,1}$	$X_{0,2}$	$X_{0,3}$
$X_{1,0}$	$X_{1,1}$	$X_{1,2}$	$X_{1,3}$
$X_{2,0}$	$X_{2,1}$	$X_{2,2}$	$X_{2,3}$
$X_{3,0}$	$X_{3,1}$	$X_{3,2}$	$X_{3,3}$



output (2x2)

Y_0	Y_1
Y_2	Y_3

$W_{0,0}$	$W_{0,1}$	$W_{0,2}$	0	$W_{1,0}$	$W_{1,1}$	$W_{1,2}$	0	$W_{2,0}$	$W_{2,1}$	$W_{2,2}$	0	0	0	0	0
0	$W_{0,0}$	$W_{0,1}$	$W_{0,2}$	0	$W_{1,0}$	$W_{1,1}$	$W_{1,2}$	0	$W_{2,0}$	$W_{2,1}$	$W_{2,2}$	0	0	0	0
0	0	0	0	$W_{0,0}$	$W_{0,1}$	$W_{0,2}$	0	$W_{1,0}$	$W_{1,1}$	$W_{1,2}$	0	$W_{2,0}$	$W_{2,1}$	$W_{2,2}$	0
0	0	0	0	0	$W_{0,0}$	$W_{0,1}$	$W_{0,2}$	0	$W_{1,0}$	$W_{1,1}$	$W_{1,2}$	0	$W_{2,0}$	$W_{2,1}$	$W_{2,2}$



$X_{0,0}$
$X_{0,1}$
$X_{0,2}$
$X_{0,3}$
$X_{1,0}$
$X_{1,1}$
$X_{1,2}$
$X_{1,3}$
$X_{2,0}$
$X_{2,1}$
$X_{2,2}$
$X_{2,3}$
$X_{3,0}$
$X_{3,1}$
$X_{3,2}$
$X_{3,3}$



Y_0
Y_1
Y_2
Y_3

✓ Transposed Convolutions

W0, 0	0	0	0
W0, 1	W0, 0	0	0
W0, 2	W0, 1	0	0
0	W0, 2	0	0
W1, 0	0	W0, 0	0
W1, 1	W1, 0	W0, 1	W0, 0
W1, 2	W1, 1	W0, 2	W0, 1
0	W1, 2	0	W0, 2
W2, 0	0	W1, 0	0
W2, 1	W2, 0	W1, 1	W1, 0
W2, 2	W2, 1	W1, 2	W1, 1
0	W2, 2	0	W1, 2
0	0	W2, 0	0
0	0	W2, 1	W2, 0
0	0	W2, 2	W2, 1
0	0	0	W2, 2



Y0
Y1
Y2
Y3



X_new0, 0
X_new0, 1
X_new0, 2
X_new0, 3
X_new1, 0
X_new1, 1
X_new1, 2
X_new1, 3
X_new2, 0
X_new2, 1
X_new2, 2
X_new2, 3
X_new3, 0
X_new3, 1
X_new3, 2
X_new3, 3

input (4x4)

X0, 0	X0, 1	X0, 2	X0, 3
X1, 0	X1, 1	X1, 2	X1, 3
X2, 0	X2, 1	X2, 2	X2, 3
X3, 0	X3, 1	X3, 2	X3, 3

$$C = \frac{\partial X^{(l+1)}}{\partial X^{(l)}} = \frac{\partial Vec(X^{(l+1)})}{\partial Vec(X^{(l)})}$$

$$= \begin{bmatrix} \frac{\partial x_{11}^{(l+1)}}{\partial x_{11}^{(l)}} & \frac{\partial x_{12}^{(l+1)}}{\partial x_{11}^{(l)}} & \frac{\partial x_{13}^{(l+1)}}{\partial x_{11}^{(l)}} & \frac{\partial x_{14}^{(l+1)}}{\partial x_{11}^{(l)}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial x_{11}^{(l+1)}}{\partial x_{44}^{(l)}} & \frac{\partial x_{12}^{(l+1)}}{\partial x_{44}^{(l)}} & \frac{\partial x_{13}^{(l+1)}}{\partial x_{44}^{(l)}} & \frac{\partial x_{14}^{(l+1)}}{\partial x_{44}^{(l)}} \end{bmatrix}$$

$$= \begin{bmatrix} w_{11} & 0 & 0 & 0 \\ w_{12} & w_{11} & 0 & 0 \\ w_{13} & w_{12} & 0 & 0 \\ 0 & w_{13} & 0 & 0 \\ w_{21} & 0 & w_{11} & 0 \\ w_{22} & w_{21} & w_{12} & w_{11} \\ w_{23} & w_{22} & w_{13} & w_{12} \\ 0 & w_{23} & 0 & w_{13} \\ w_{31} & 0 & w_{21} & 0 \\ w_{32} & w_{31} & w_{22} & w_{21} \\ w_{33} & w_{32} & w_{23} & w_{22} \\ 0 & w_{33} & 0 & w_{23} \\ 0 & 0 & w_{31} & 0 \\ 0 & 0 & w_{32} & w_{31} \\ 0 & 0 & w_{33} & w_{32} \\ 0 & 0 & 0 & w_{33} \end{bmatrix}$$

출력값을 입력값으로 미분한 값 (출력값 : Convolution한 결과)

✓ Transposed Convolutions

W _{0,0}	0	0	0
W _{0,1}	W _{0,0}	0	0
W _{0,2}	W _{0,1}	0	0
0	W _{0,2}	0	0
W _{1,0}	0	W _{0,0}	0
W _{1,1}	W _{1,0}	W _{0,1}	W _{0,0}
W _{1,2}	W _{1,1}	W _{0,2}	W _{0,1}
0	W _{1,2}	0	W _{0,2}
W _{2,0}	0	W _{1,0}	0
W _{2,1}	W _{2,0}	W _{1,1}	W _{1,0}
W _{2,2}	W _{2,1}	W _{1,2}	W _{1,1}
0	W _{2,2}	0	W _{1,2}
0	0	W _{2,0}	0
0	0	W _{2,1}	W _{2,0}
0	0	W _{2,2}	W _{2,1}
0	0	0	W _{2,2}



Y ₀
Y ₁
Y ₂
Y ₃



X _{new0,0}
X _{new0,1}
X _{new0,2}
X _{new0,3}
X _{new1,0}
X _{new1,1}
X _{new1,2}
X _{new1,3}
X _{new2,0}
X _{new2,1}
X _{new2,2}
X _{new2,3}
X _{new3,0}
X _{new3,1}
X _{new3,2}
X _{new3,3}

input (4x4)

X _{0,0}	X _{0,1}	X _{0,2}	X _{0,3}
X _{1,0}	X _{1,1}	X _{1,2}	X _{1,3}
X _{2,0}	X _{2,1}	X _{2,2}	X _{2,3}
X _{3,0}	X _{3,1}	X _{3,2}	X _{3,3}

둘은 크기만 같지, 서로 다른 값을
가지는 것을 기억해야 함 !!!
- Gradient 값에 Y를 곱한 값과 동일한

```
1 w = torch.randn((1, 1, 3, 3), requires_grad = True)
2 x = torch.randn((1, 1, 4, 4), requires_grad = True)
3 z = F.conv2d(x, w)
4
5 z.backward(z)
6 print(x.grad)
```

```
tensor([[[[ 3.9034, -4.6706, -0.6917,  0.5771],
           [ 2.4653, -6.3977,  2.2842,  0.8189],
           [ 1.3904, -3.8509,  4.2874, -1.0926],
           [ 1.1141, -2.1222,  1.2977, -1.2373]]]])
```

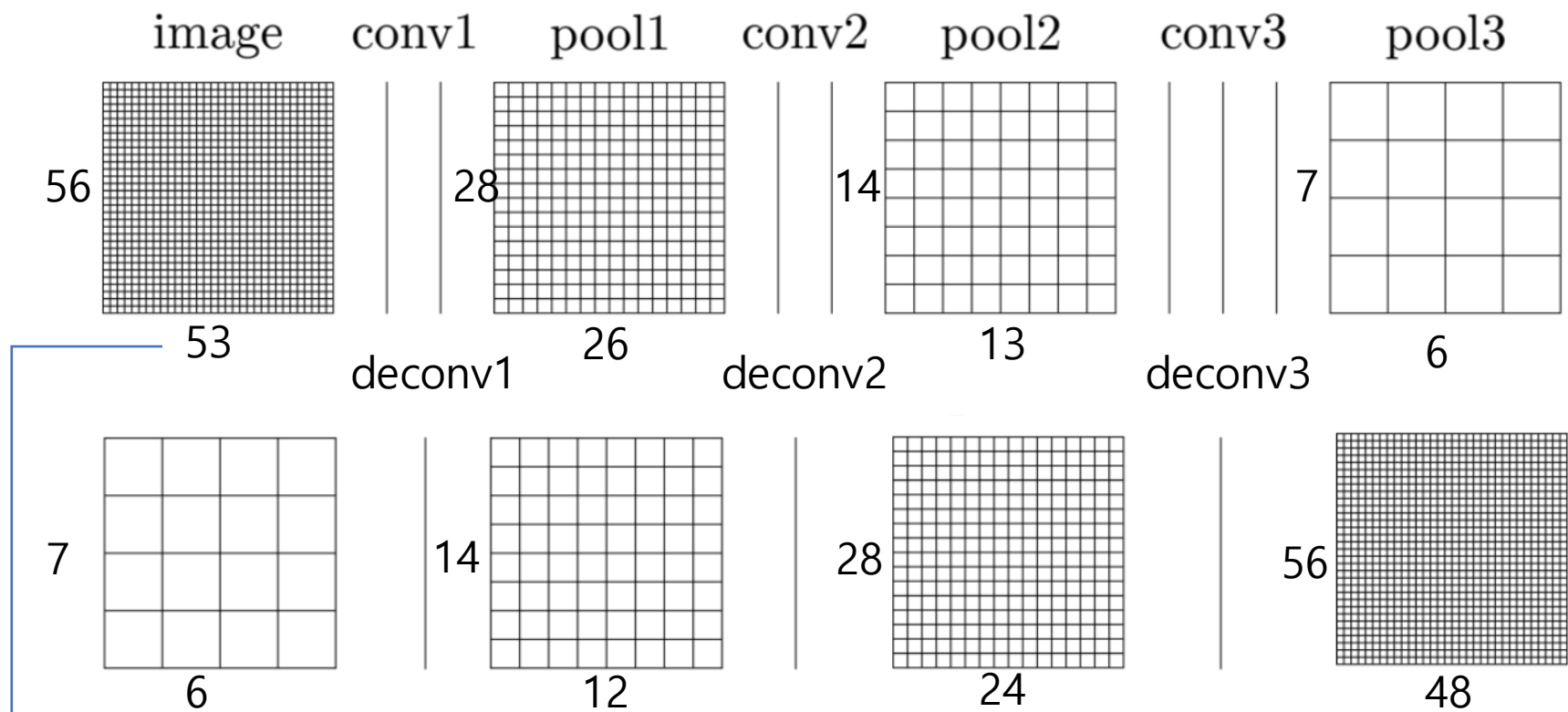
```
1 x_prime = F.conv_transpose2d(z, w)
2 print(x_prime)
```

```
tensor([[[[ 3.9034, -4.6706, -0.6917,  0.5771],
           [ 2.4653, -6.3977,  2.2842,  0.8189],
           [ 1.3904, -3.8509,  4.2874, -1.0926],
           [ 1.1141, -2.1222,  1.2977, -1.2373]]]])
grad_fn=<SlowConvTranspose2DBackward>
```

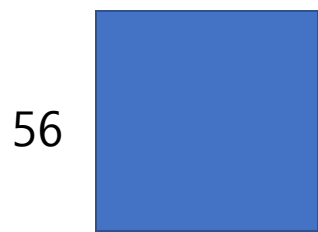
$$\text{TransposedConvolution}(Y, W) = Y \cdot \frac{\partial \text{Conv}(X, W)}{\partial X}$$

✓ Upsampling

문제점 : 원본 데이터는 56×53 이어서 Max pooling에 의해, 절반으로 줄어들지 않음



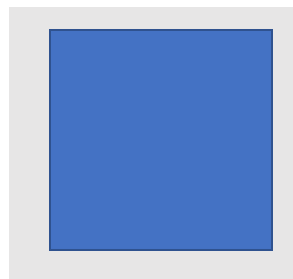
원본 이미지와 크기가 달라짐



56

53

64

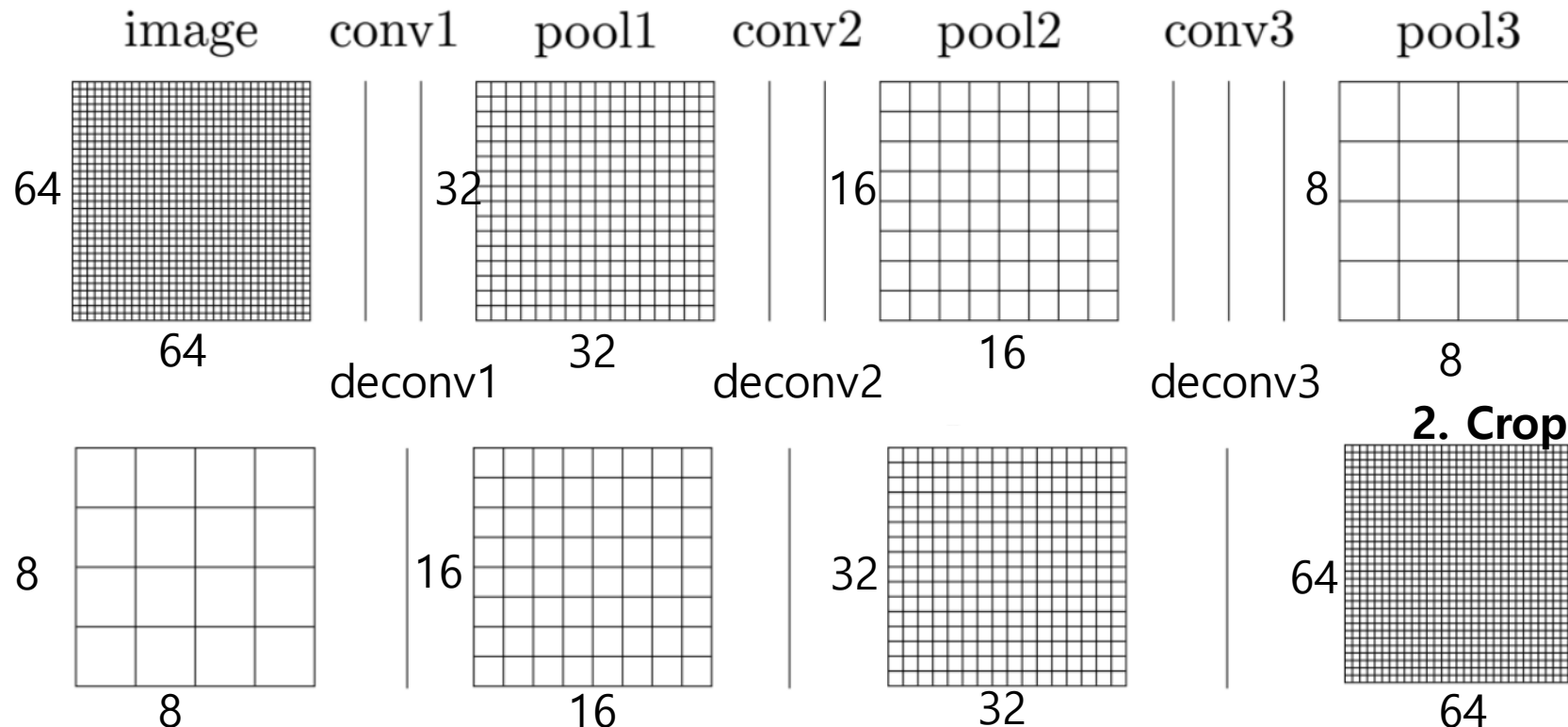


64

1. Padding을 통해서 64x64으로 수정

✓ Upsampling

해결 : Padding을 해서 임의로 2의 제곱배를 만들고, deconvolutional 이후 해당 부분을 Crop함



2. Cropping을 이용해서 56, 53으로 복원

