# 9. Autoencoder

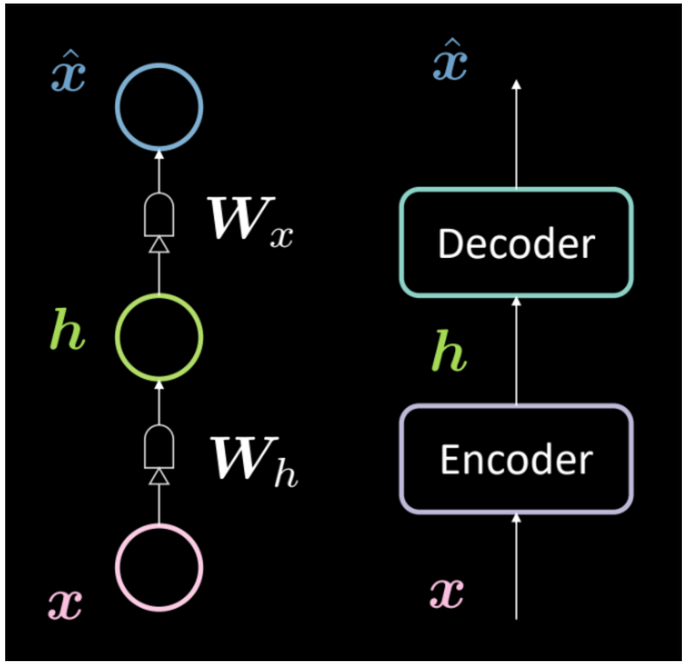
PyTorch master study, 김영은

# 9.1 소개 및 학습원리

오토인코더란 무엇인가?
사용 분야
구성 및 모델 아키텍처

# 오토인코더란 무엇인가?

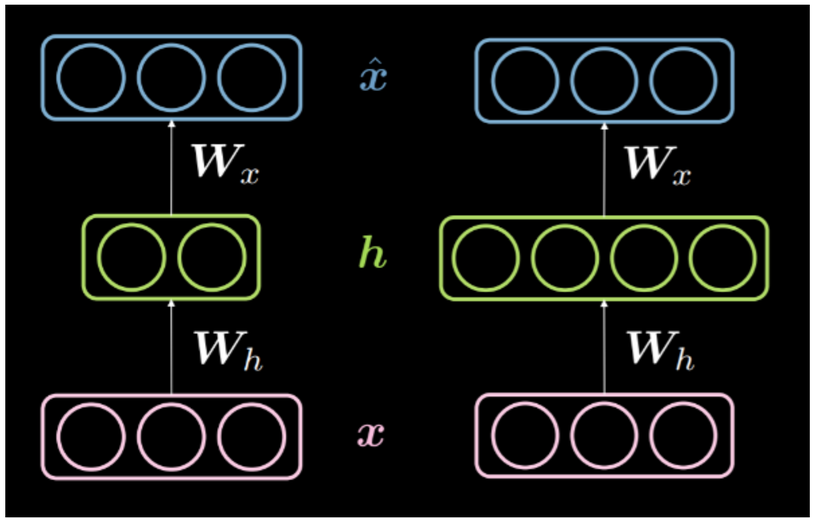
## 입력을 출력으로 복사하는 신경망




- 딥러닝 / 비지도학습 / 인공신경망

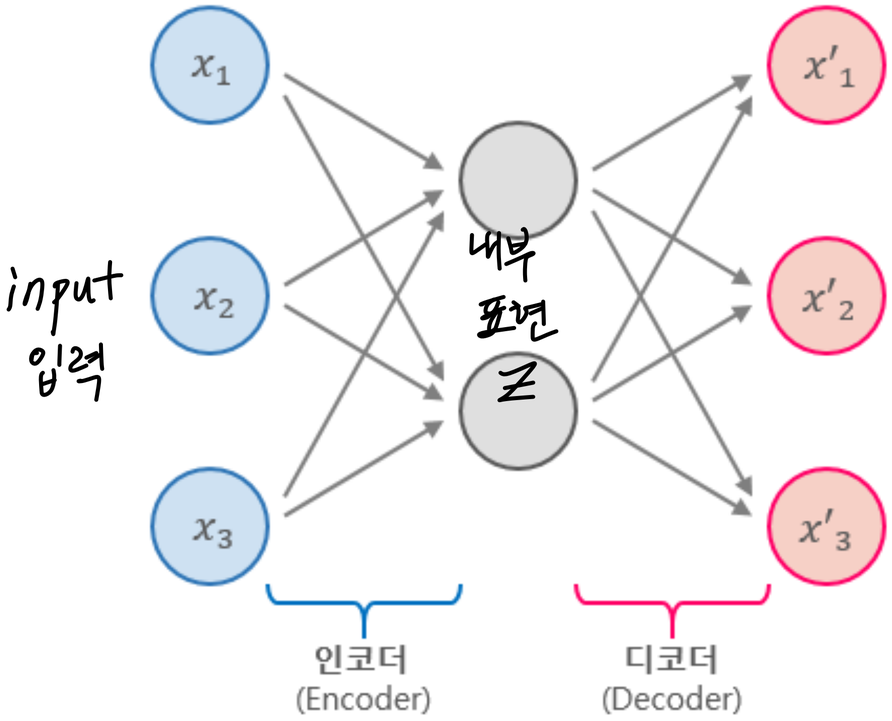
  비지도학습 : 지도학습과 달리 정답 라벨이 없는 데이터를 비슷한 특징끼리 군집화하여 새로운 데이터에 대해 결과를 예측하는 방법

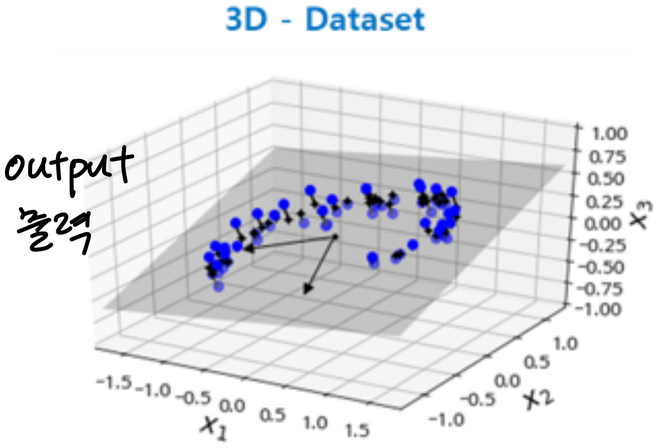
- Hidden layer의 뉴런 수를 Input layer보다 작게해서 데이터를 압축하여 차원 축소를 하기도하고, 입력 데이터에 noise를 추가한 후 원본 입력을 복원할 수 있도록 네트워크를 학습시키는 등의 다양한 오토인코더가 있다
- 주로 이미지 압축,이상감지, 노이즈 제거 분야에 응용
- 가장 기본적인 오토인코더 형태 " 압축 "



# 오토인코더의 구성

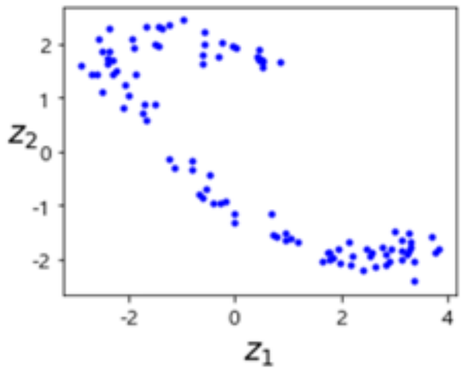
- Encoder : 인지 네트워크(recognition network)라고도 하며, 입력을 내부 표현으로 변환
- Decoder : 생성 네트워크(generator network)라고도 하며, 내부 표현을 출력으로 변환
- 잠재변수(잠재표현/잠재벡터/코드) z




"압축"
입력의 저차원 표현

Uncomplete
Autoencoder



# 오토인코더 모델 아키텍처 및 손실 정의

## 모델 Autoencoder()

```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder,self).__init__()
        self.encoder = nn.Linear(28*28,20)
        self.decoder = nn.Linear(20,28*28)

    def forward(self,x):
        x = x.view(batch_size,-1)
        encoded = self.encoder(x)
        out = self.decoder(encoded).view(batch_size,1,28,28)
        return out
```

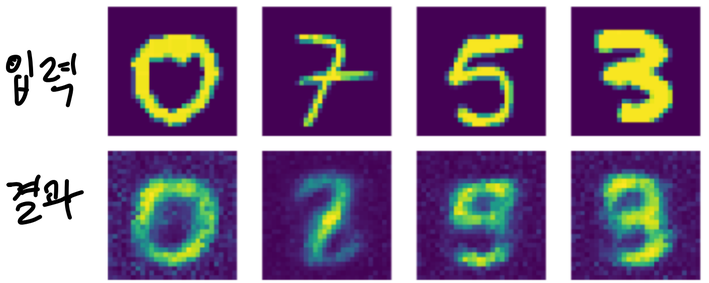
압축할 vector size

MNIST dataset size

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)

model = Autoencoder().to(device)
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

- self.encoder : 압축
- self.decoder : 되돌리기
- forward() 함수 : view함수를 통해 텐서 형태를 바꿔주고, encoder로 압축하고 decoder로 원래 이미지의 형태로 돌려놓는다.
- 손실함수와 최적화함수를 정의하는데, 여기서 손실은 입력값과 출력값의 차이의 절대값의 합을 말한다. $L(x,x') = \sum_{i=1}^{n} |x_i - x_i'|$
- 데이터 로더에서 입력값을 가져와서 모델에 넣어주고, 입력값과 모델에서 나온 출력값간의 차이로 loss를 구하여 모델의 가중치를 업데이트한다.



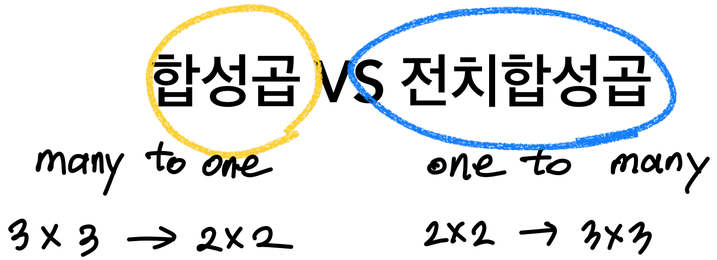
# 9.2 합성곱 오토인코더

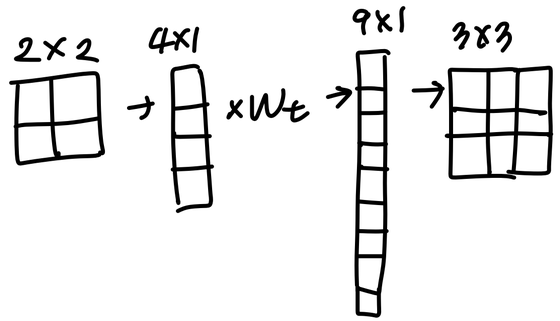
합성곱 오토인코더란 ?

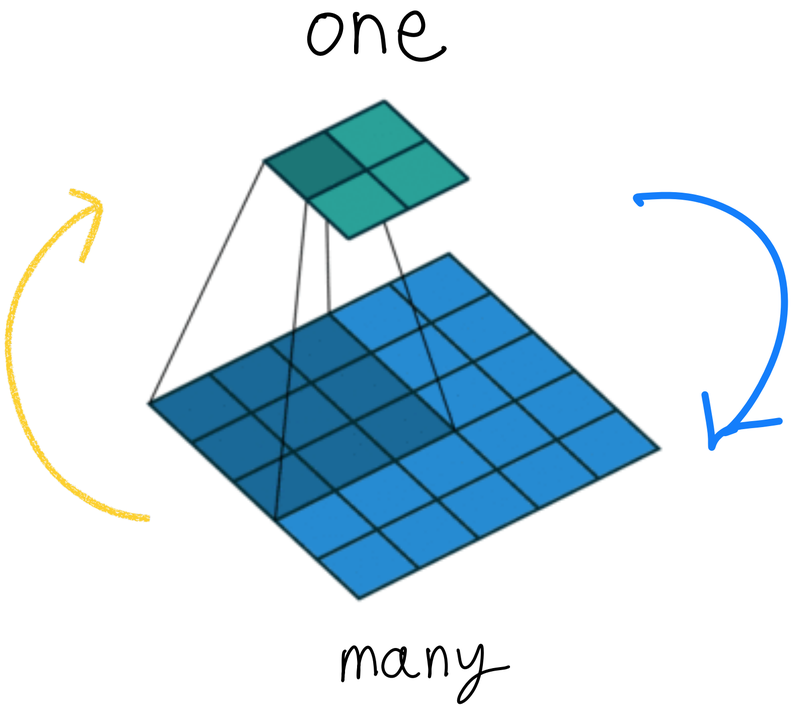
합성곱 VS 전치합성곱

# 합성곱 오토인코더

- 손실 들이기 위해, 그냥 신경망이 아닌 합성곱 신경망을 오토인코더에 적용해본다.
- Encoder : 합성곱 풀링 층으로 구성된 일반적인 CNN
- Decoder : 전치합성곱(역합성곱) - 인코더와 대칭되게 만들어야하기 때문에, 이미지의 스케일을 늘리고, 깊이를 원본 차원으로 돌리는 방법



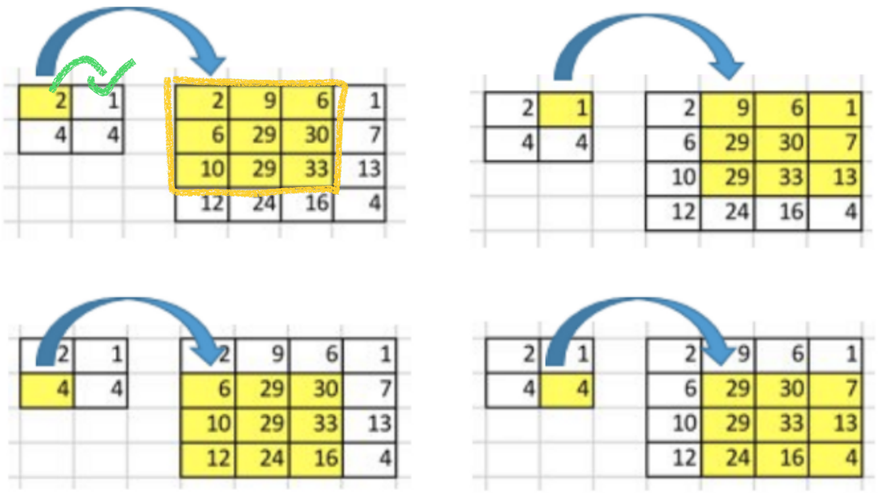




- 합성곱 연산 : 입력값에 필터의 가중치를 각각 곱한 결과의 합을 통해 계산
- 전치합성곱 연산 : 하나의 입력값을 받아 여기에 서로 다른 가중치를 곱해 필터의 크기만큼 입력값을 퍼뜨리는 역할

# 전치합성곱 예시

- Kernel 3, stride 1 input 2×2




| | | | |
|---|---|---|---|
| 1 | 2 | 2 | 1 |
| 2 | 4 | 4 | 2 |
| 2 | 4 | 4 | 2 |
| 1 | 2 | 2 | 1 |

output 4×4

- + Padding 1

| | | | |
|---|---|---|---|
| 1 | 2 | 2 | 1 |
| 2 | 4 | 4 | 2 |
| 2 | 4 | 4 | 2 |
| 1 | 2 | 2 | 1 |

→

| | |
|---|---|
| 4 | 4 |
| 4 | 4 |

2×2

- + Output padding 1

| | | |
|---|---|---|
| 4 | 4 | 2 |
| 4 | 4 | 2 |
| 2 | 2 | 1 |

3×3

- No padding + Output padding 1

4×4 → 5×5

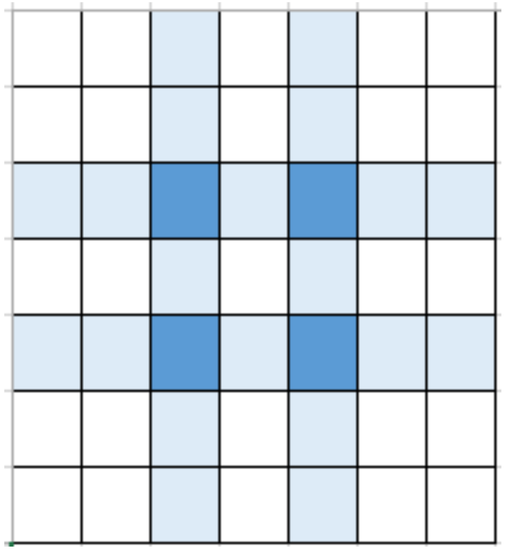
| | | | | |
|---|---|---|---|---|
| 1 | 2 | 2 | 1 | 0 |
| 2 | 4 | 4 | 2 | 0 |
| 2 | 4 | 4 | 2 | 0 |
| 1 | 2 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

- **Kernel 3 stride 2 padding 1 output padding 1 = 입력값 2배 만들기**

(예) 3X3 → 7x7 → 5x5 → 6x6

- 체크보드 형태의 artifact -> interpolation + convolution

보간법 : 함수는 미지이나, 특정 변수값들이 알려져있을 경우, 임의의 x에 대한 함수 값을 추정하는 방법
Nearest-neighbor interpolation / bilinear interpolation



DeepDream only applying the neural network to a fixed position.
*Severe artifacts.*

DeepDream applying the network to a different position each step.
*Reduced artifacts.*

https://www.notion.so/BOAZ-157dee0a5a454bdd8339bea3e91f8d32#331d658879884ae09405fd48c9360e2d

# 9.3 시맨틱 시그맨테이션

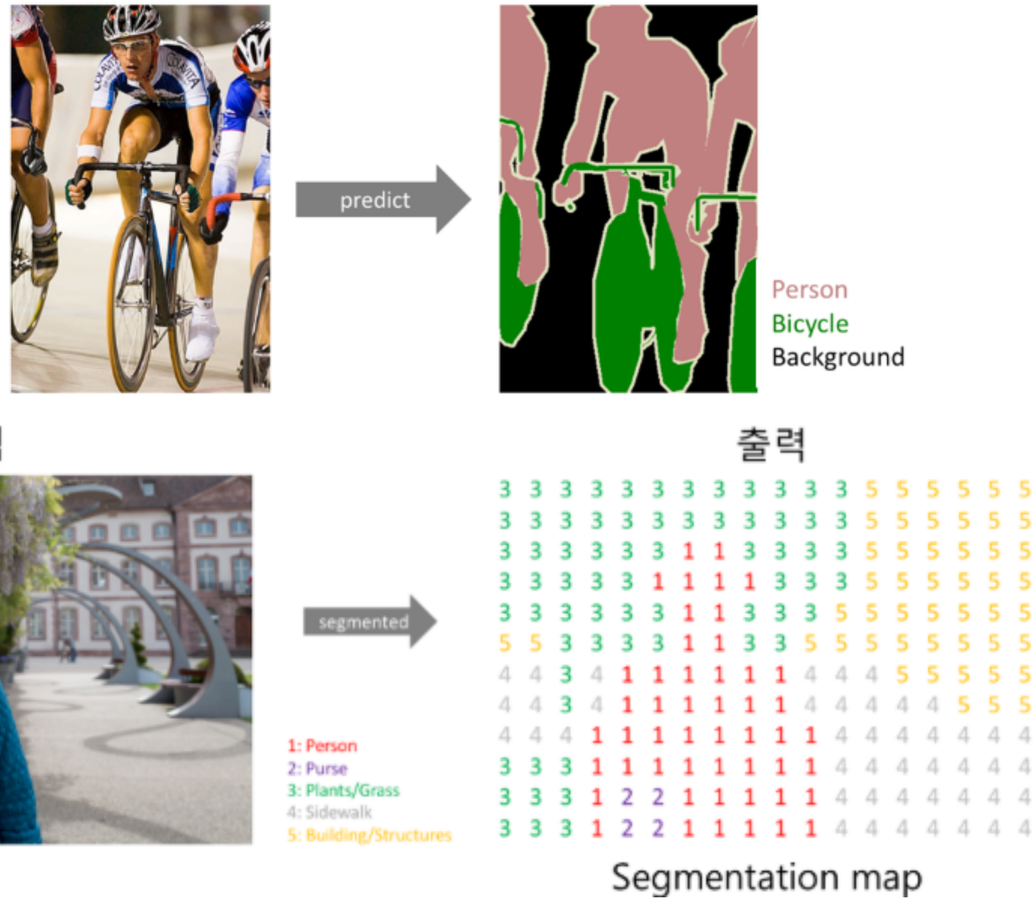
시맨틱 시그맨테이션이란 ?

U-net model

# 시맨틱 시그멘테이션

## 오토인코어의 활용

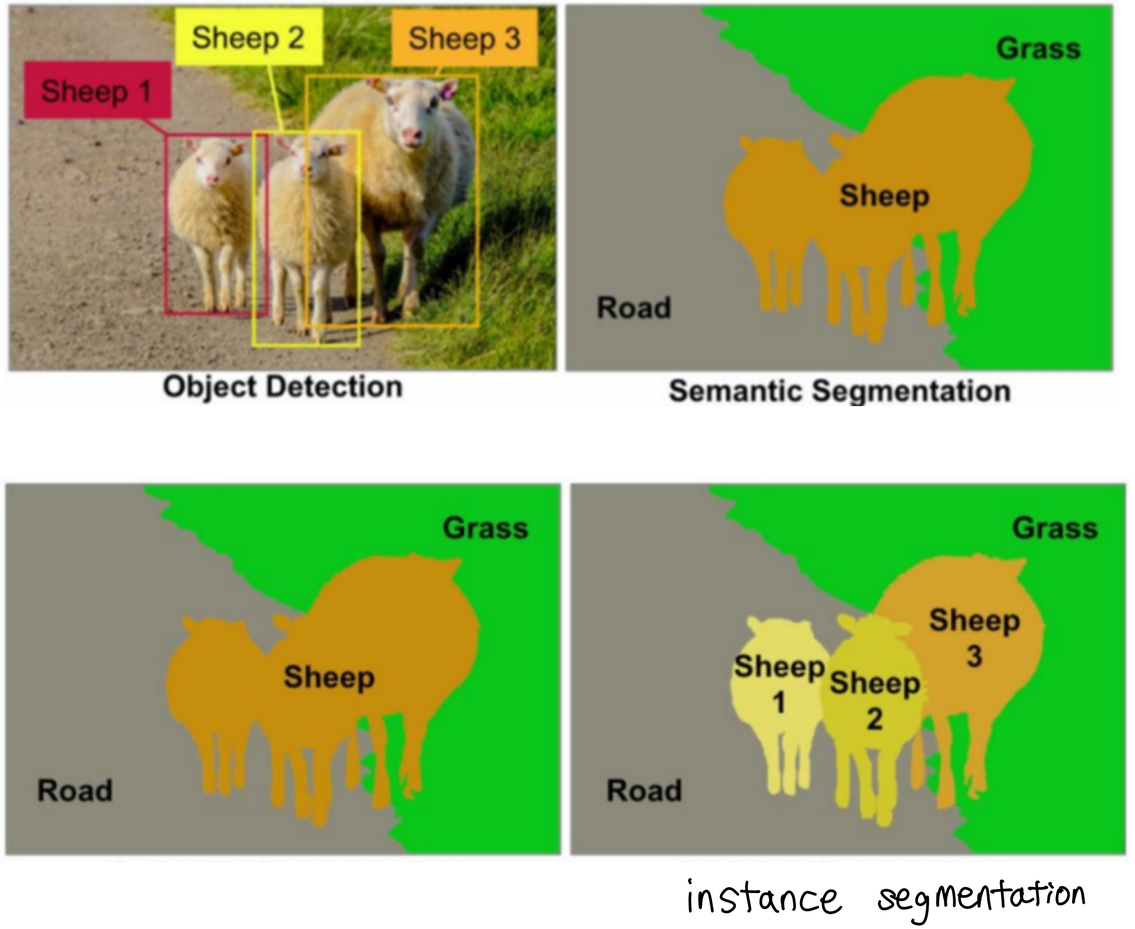
- 말 그대로 의미에 맞게 분할한다는 뜻
- 이미지 각 픽셀이 어느 클래스에 속하는지를 예측하는 것!
- 시맨틱 시그멘테이션 알고리즘의 입력값은 컬러 혹은 흑백 이미지이고, 출력값은 각 픽셀의 예측된 클래스를 나타내는 시그멘테이션 map이다.



# 시맨틱 시그멘테이션

## 오토인코어의 활용

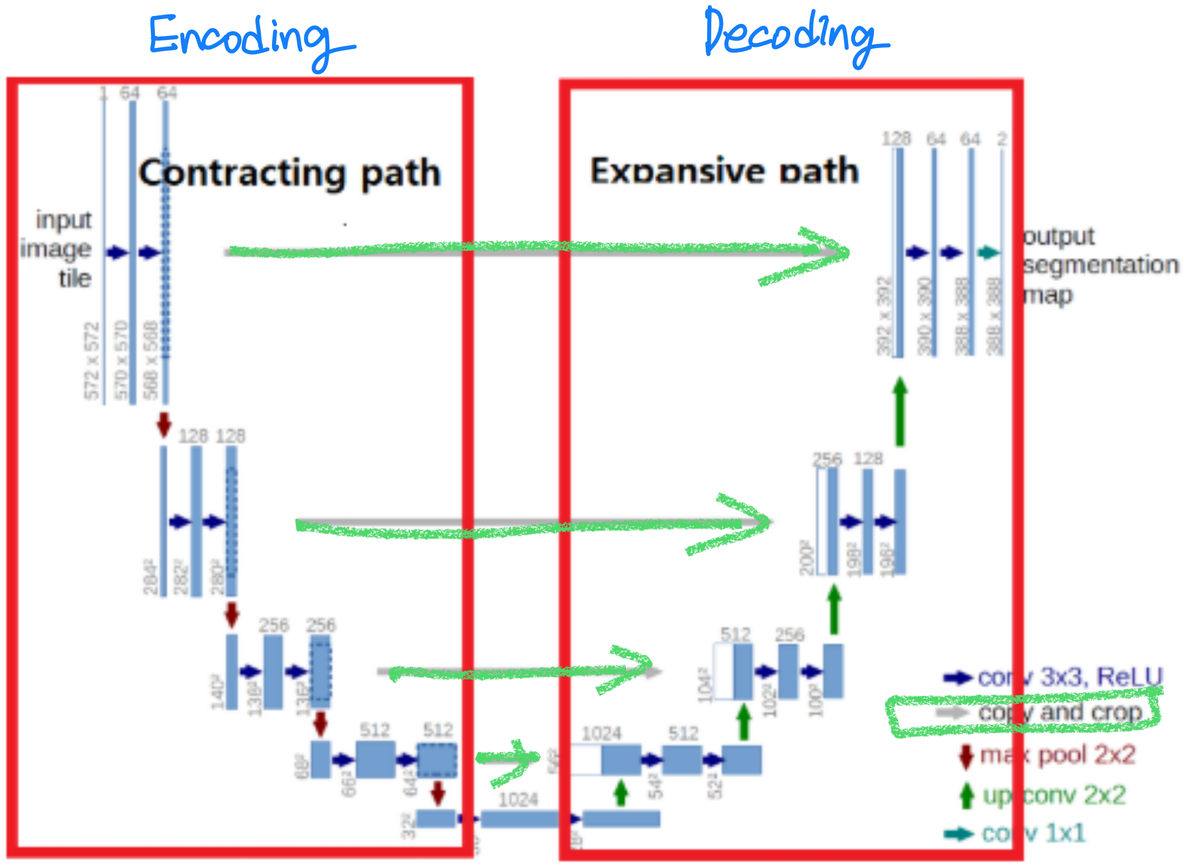
- 말 그대로 의미에 맞게 분할한다는 뜻
- 이미지 각 픽셀이 어느 클래스에 속하는지를 예측하는 것!
- 시맨틱 시그멘테이션 알고리즘의 입력값은 컬러 혹은 흑백 이미지이고, 출력값은 각 픽셀의 예측된 클래스를 나타내는 시그멘테이션 map이다.



# U-net model

## 시맨틱 시그멘테이션 알고리즘 중 하나

- 생체 데이터 이미지에서 시그멘테이션 작업을 수행하기 위해 만들어진 모델
- End to End
- Encoding -> Decoding
- 인코딩에서 이미지의 context를 포착하고, 디코딩에서 피쳐맵을 업샘플링하고 인코딩에서 포착한 피쳐맵의 context와 결합하여 더욱 정확한 localization을 하는 역할을 합니다.



# U-net model

## 시맨틱 시그멘테이션 알고리즘 중 하나

- Fully Convolution Network(FCN)를 기반으로 구축
- 적은 데이터를 가지고도 더욱 정확한 분류를 하기위하여 FCN 구조를 수정하였다.

  1. 업샘플링 과정에서 피쳐채널의 수가 많다는 것

  2. 각 Convoultion의 valid part만 사용한다는 것

  valid part : full context가 들어있는 segmentation map

  이는 Overlap-tile 기법을 사용하여 원활한 segmentation 이 가능하도록 한다.

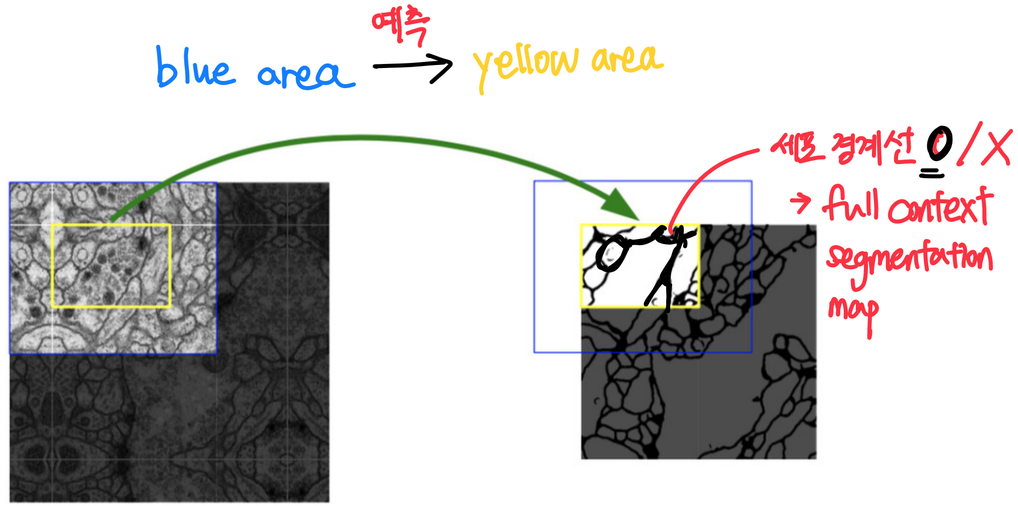



**Fig. 2.** Overlap-tile strategy for seamless segmentation of arbitrary large images (here segmentation of neuronal structures in EM stacks). Prediction of the segmentation in the yellow area, requires image data within the blue area as input. Missing input data is extrapolated by mirroring

# U-net model

## 시맨틱 시그멘테이션 알고리즘 중 하나

- 초반 부분의 레이더와 후반 부분의 레이더에 skip-connection을 추가함으로써 높은 공간 frequency 정보를 유지하고자 하는 방법
- skip-connection : 하나의 layer의 output을 몇개의 layer를 건너뛰고 다음 layer의 input을 추가하는 방법
- Convolution 연산을 진행하면서 인코딩 되는 과정의 이미지들을 기억해놨다가 디코딩 하는 과정에 같이  소스로 추가해주는 것
- Decoder 부분에 skip-connection layer를 추가하여, Encoder의 Convolution layer와 Decoder의 Upsampling layer를 연결해준다.

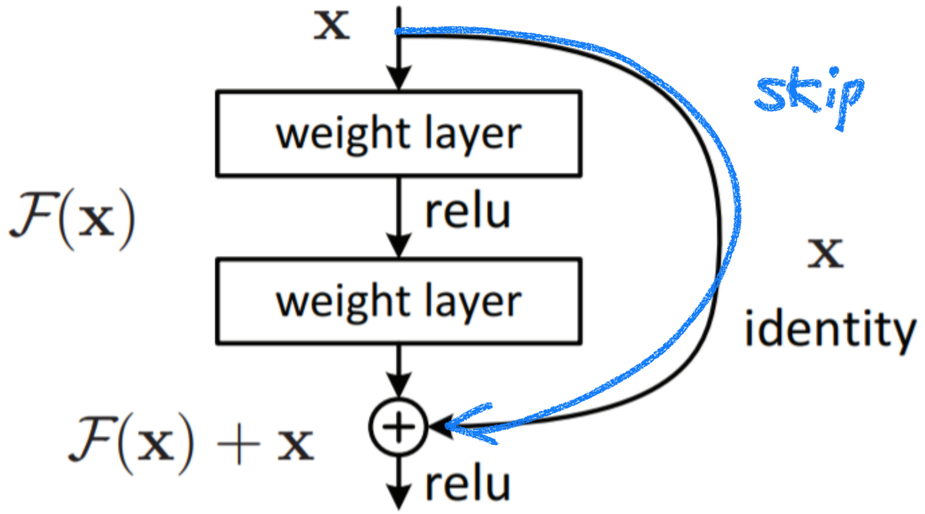



Figure 2. Residual learning: a building block.

# Thank you.

PyTorch master study, 김영은