

```

def make_transform(mode):
    if mode == 'train':
        train_transform = Compose([
            transforms.Resize(height = 512, width = 512),
            OneOf([transforms.MotionBlur(),
                    transforms.OpticalDistortion(),
                    transforms.GaussNoise(p = 0.5),
                    transforms.RandomContrast()]),
            transforms.ElasticTransform(),
            OneOf([transforms.HorizontalFlip(),
                    transforms.RandomRotate90(),
                    transforms.VerticalFlip()]), # oneof에 p 부여 가능,
            transforms.Normalize((0.5), (0.5))
        ])
        return train_transform
    else:
        test_transform = Compose([
            transforms.Resize(height = 512, width = 512),
            transforms.Normalize((0.5), (0.5))
        ])
        return test_transform

```

```

class DoubleConv(nn.Module):
    """반복되는 conv - BN - ReLU 구조 모듈화"""
    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)

```

<https://m.blog.naver.com/winddori2002/222111458214>

conv2d와 Relu함수 적용 사이에 배치정규화를 하는 코드, 3page 더 보강;

u-net 모델을 구성할 때의 적절한 높이?

깊이를 더욱 증가시키면 네트워크는 더 복잡한 패턴과 구조를 학습할 수 있지만, 과적합 문제가 발생할 수 있습니다. 따라서 적절한 깊이는 데이터셋의 크기, 문제의 복잡성 및 훈련 데이터의 다양성에 따라 결정되어야 합니다.

보통 이미지 세그멘테이션 작업에서는 4-5개의 다운샘플링 단계를 사용하는 것이 일반적입니다. 이 수는 경험적으로 확인된 결과이며, 일반적으로 좋은 결과를 제공합니다.

최적화 포인트

U-Net의 성능을 끌어올리기 위해서 최적화할 수 있는 포인트가 무엇인지 살펴보자.

U-Net을 비롯한 대부분의 세그멘테이션 모델이 인코더-디코더 아키텍처를 따른다. 인코더-디코더 아키텍처의 문제점은 얼마나 깊어야(다운샘플링을 얼마나 많이 해야) 최적의 성능을 낼 수 있는지 알기 힘들다는 것이다. 심지어 어떤 문제에 U-Net을 적용하느냐에 따라 최적의 깊이는 달라질 수 있다. 예를 들어 아래 그림 2에 다양한 깊이의 U-Net 모델이 제시되어 있고, 표1에 EM, Cell, Brain Tumor 데이터셋을 대상으로 각 모델의 성능을 평가했다. Cell과 Brain Tumor 세그멘테이션 문제에서는 상대적으로 얇은 모델인 U-Net(L3)의 성능이 가장 좋았고, EM 데이터셋에 대해서는 모델이 깊을수록 성능이 좋아졌다.

Architecture	DS	Params	EM	Cell	Brain Tumor
U-Net L ¹	✗	0.1M	86.83±0.43	88.58±1.68	86.90±2.25
U-Net L ²	✗	0.5M	87.59±0.34	89.39±1.64	88.71±1.45
U-Net L ³	✗	1.9M	88.16±0.29	90.14±1.57	89.62±1.41
U-Net (L ⁴)	✗	7.8M	88.30±0.24	88.73±1.64	89.21±1.55

표1 깊이에 따른 U-Net 성능

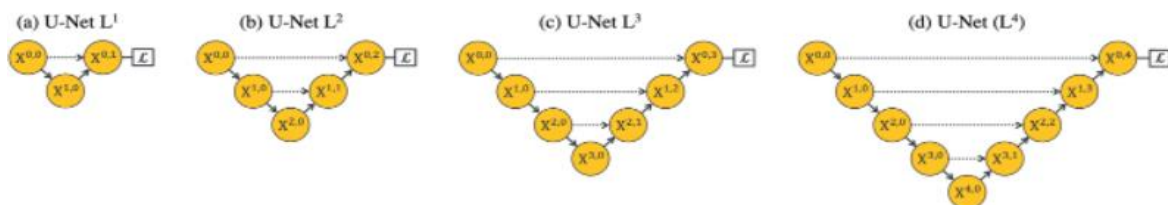


그림 2 다운샘플링 수준에 따른 U-Net 구조. L은 Level의 약자로 L1부터 L4까지 표현. 출처 : [3]

양상불 with deep supervision

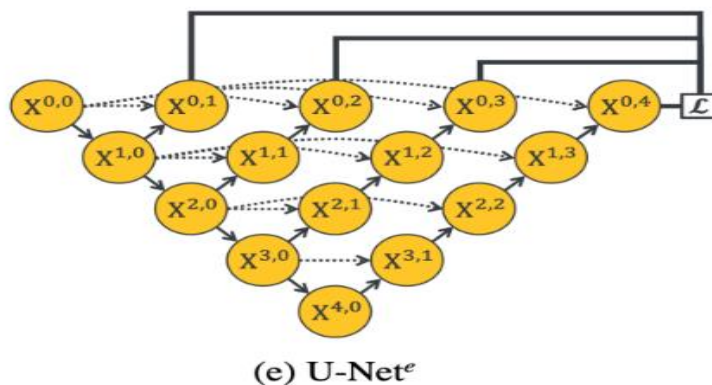


그림 3 그림 2의 U-Net(L1)부터 U-Net(L4)를 양상불한 모델. 출처 : [3]

<https://devbasket.tistory.com/26> : u-net 모델의 적절한 깊이와 해결책?

--유넷 구성에서 배치 정규화를 사용하는 코드 ?

```
def CB(in_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=True):
    layers = []
    layers += [nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                        kernel_size=kernel_size, stride=stride, padding=padding,
                        bias=bias)]
    layers += [nn.BatchNorm2d(num_features=out_channels)]
    layers += [nn.ReLU()]

    conv_block = nn.Sequential(*layers)
    return conv_block
```

+ 데이터 증강 참조 <https://magiccode.tistory.com/68> : random flip 등 페이지 내 코드 참조


-- 현재 sample code의 conv2d 함수 > 무작위의 가중치로 필터를 초기화 하고 최적화 함수등을 통해 맞추어 나감,

무작위로 초기화를 하지 않으려면? >적절한 가중치로 초기화 하는 것도 중요한 성능지표

`nn.Conv2d`를 사용하여 생성된 컨볼루션 필터의 초기화 방법을 제어하려면 다음과 같이 할 수 있습니다:

1. 무작위 초기화를 사용하지 않고 직접 가중치를 지정하려면, `nn.Conv2d` 객체를 생성한 후에 해당 레이어의 가중치에 원하는 값을 할당할 수 있습니다. 예를 들어, `conv_layer`라는 `nn.Conv2d` 객체를 생성한 후에 `conv_layer.weight`에 원하는 가중치 값을 할당할 수 있습니다.

python

 Copy code

```
import torch.nn as nn

conv_layer = nn.Conv2d(in_channels, out_channels, kernel_size)
conv_layer.weight = nn.Parameter(custom_weights)
```

`custom_weights`는 원하는 가중치 텐서입니다. 이 텐서의 크기와 데이터 타입은 해당 레이어의 크기와 데이터 타입과 일치해야 합니다.

+ u-net ++ 구현 참조 코드 <https://with-ahn-ssu.tistory.com/40>