U-Net

U-Net 논문 리뷰

- **U-Net**: Convolutional Networks for Biomedical Image Segmentation

  : 의료 영상에서 객체(조직, 장기 등)을 분할(Segmentation)하는데 특화된 CNN 아키텍처

  이미지 그 자체의 특정 영역을 label로 표현하고자 하여 구현된 모델링

네트워크 구조

2

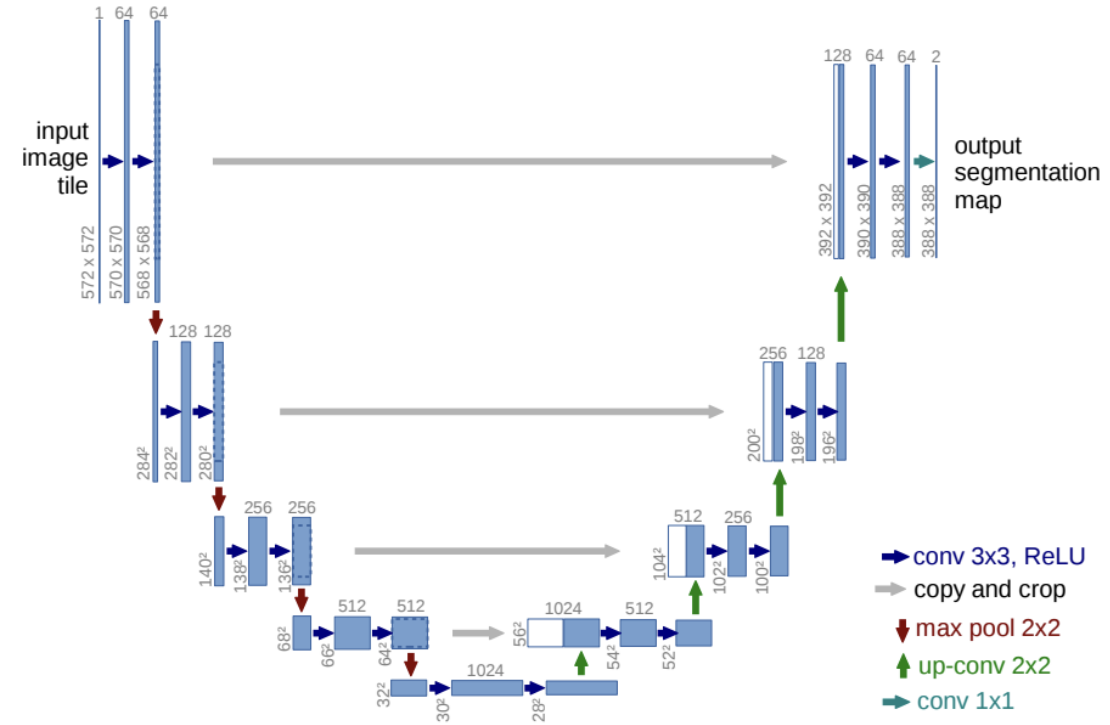
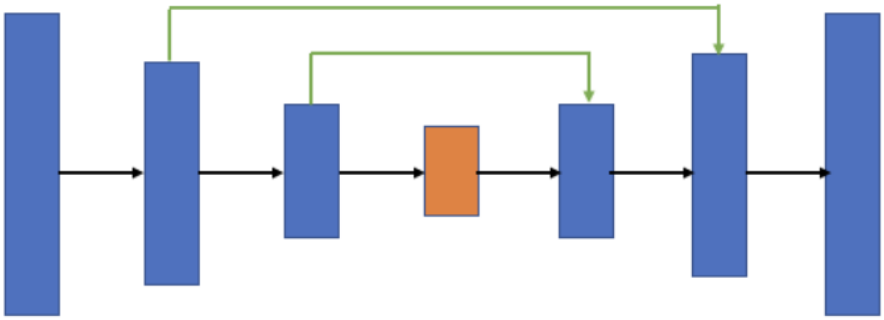


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

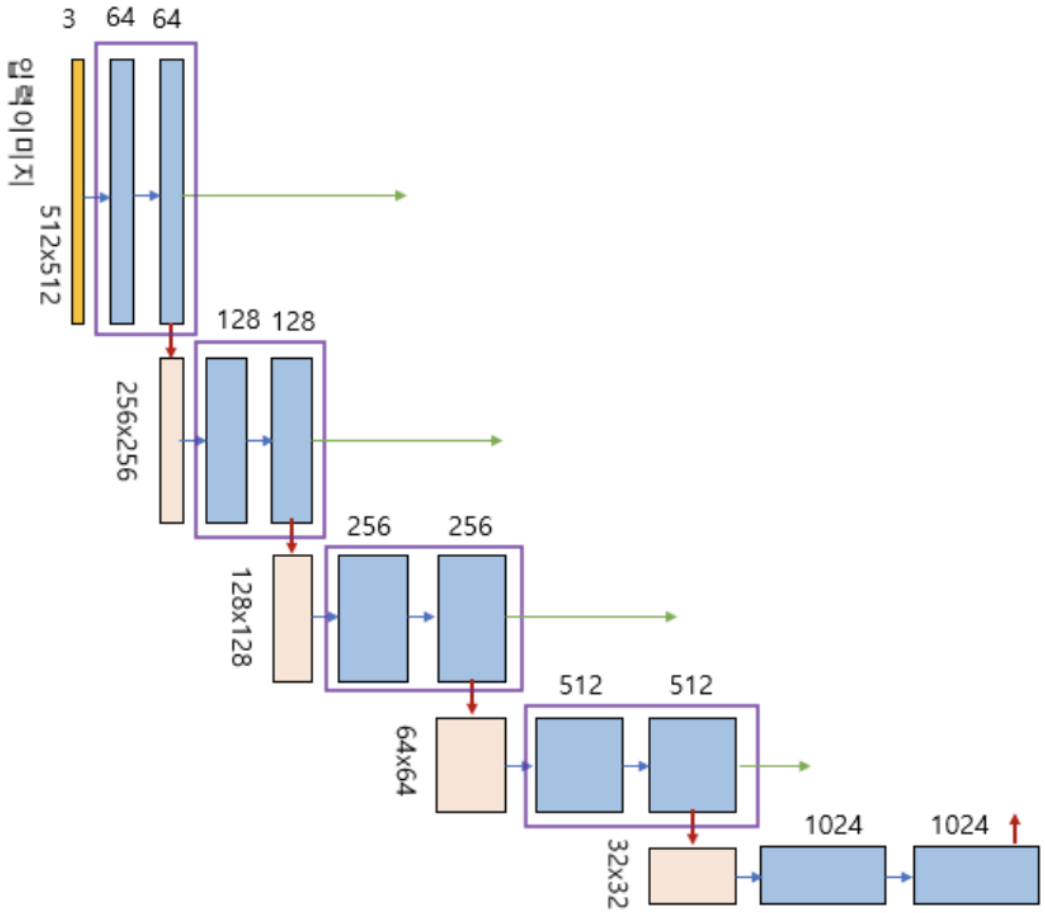
: 의료 영상 분할 작업을 위한 딥러닝 모델의 구조(incoder-decoder 구조 기반)

1. **Input**
    - 572 X 572, 단일 채널, 네트워크를 통과하며 점점 더 작은 크기로 다운샘플링되고, 이후에 복원
    - 입력 이미지의 특징을 포착할 수 있도록 채널 수를 늘리면서 차원을 축소해나감

1. **Encoder**(downsampling way, left)
    - 3X3으로 합성곱 연산, 활성화 함수 : ReLU, 출력 채널 수는 단계별로 증가
    - 빨간색 화살표 : pooling, 이미지 크기를 전반으로 줄임, 특징을 집약하여 공간적 크기 감소
2. **Bottleneck**(병목) : 가장 추상적인 특징을 학습, 채널 : 1024, 다운과 업샘플링 경로를 연결
3. **Decoder**(Upsampling way)
    - 인코더에서 추출한 특징을 기반으로 점진적으로 원래 크기로 복원
    - 채널 수를 줄이고 차원을 늘려서 고차원의 이미지를 복원
    - 인코더에서 생성된 특징 맵을 복사하여 디코더의 대응 단계에 결합(더 정밀한 seg 가능)
4. **Output** : 388 X 388, 클래스 수만큼의 채널, 픽셀별로 클래스가 할당된 세그멘테이션 맵을 생성
5. **Skip Connections**
    - 인코더의 낮은 레벨(고해상도) 특징을 디코더로 전달해 공간적 정보를 보존, 저차원 뿐만 아니라 고차원 정보도 이용하여 이미지의 특징 추출 동시에 정확한 위치 파악도 가능하게끔 함
    - 인코딩 단계의 각 레이어에서 얻은 특징을 디코딩 단계의 각 레이어에 합치는 방법을 사용



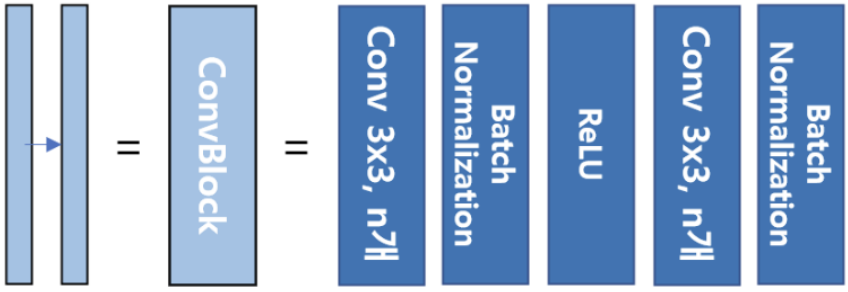
<Encoder>



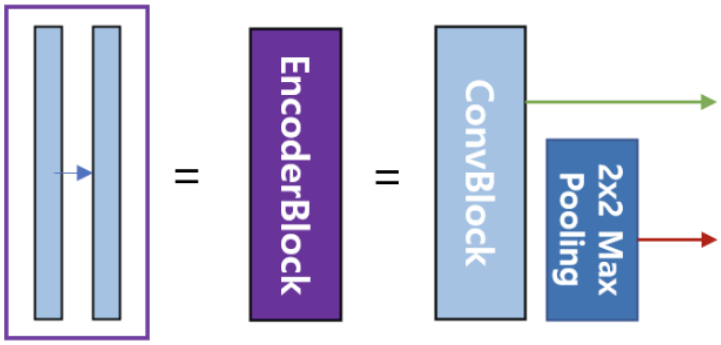
- 세로 방향 숫자 : 맵의 차원

- 가로 방향 숫자 : 채널 수
- 파란색 박스 : 3 X 3 convolution (입력 데이터의 특징을 추출한 레이어) + batch normalization + ReLU 활성화 함수가 차례로 배치됨

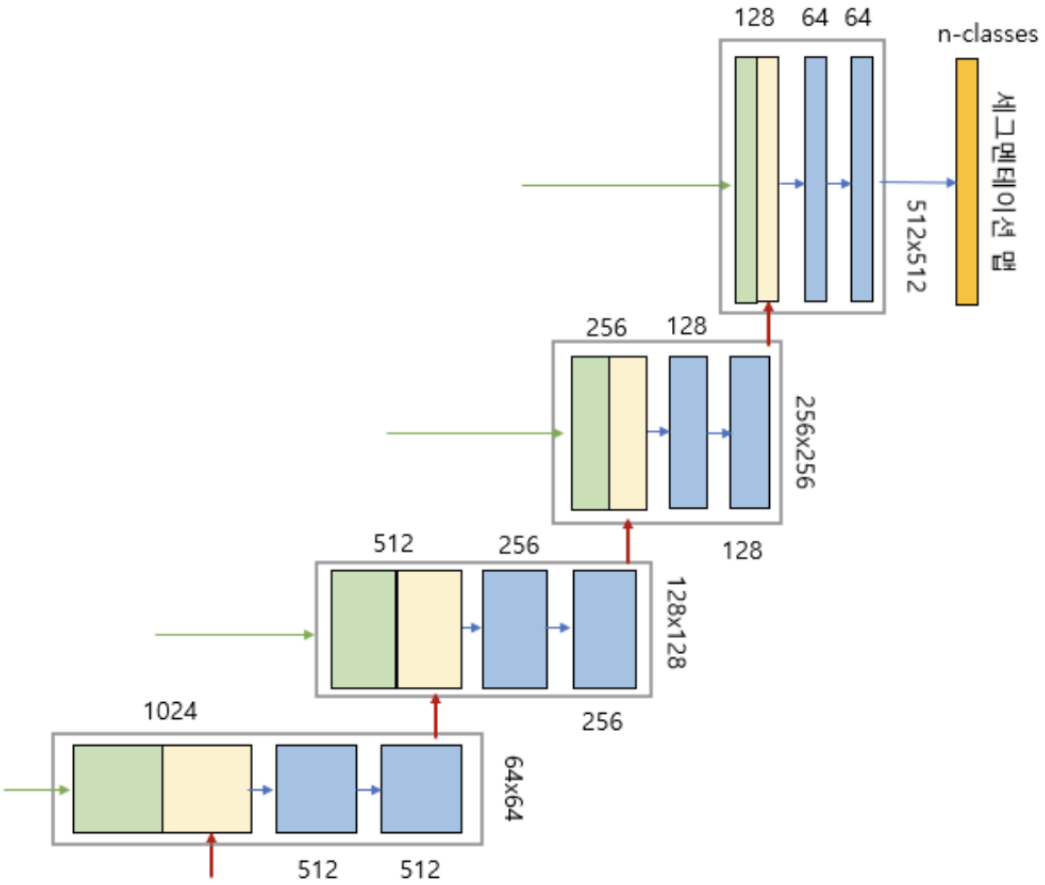
  이 파란색 박스 두 개를 묶어 한 개의 레이어 블록으로 구현하여 사용하면 편리 (=ConvBlock)



- ConvBlock 박스에서 나오는 출력 2개
  1. 연두색 화살표 : 디코더로 복사하기 위한 연결선
  2. 빨간색 화살표 : 2X2 max pooling 으로 다운샘플링해 인코더의 다음 단계로 내보냄



<Decoder>



- 녹색 박스 : Skip connection을 통해 인코더에 있는 맵 복사
- 노란색 박스 : 맵의 차원을 두 배로 늘리면서 채널 수를 반으로 줄임

  → 이 두 박스를 합쳐 저차원 + 고차원 정보도 이용 가능
- 맨 상단 오른쪽 박스 : U-Net의 출력부분, 1X1 convolution으로 특징 맵을 처리, 입력 이미지의 각 픽셀을 분류하는 seg 맵을 생성하는 부분

  convloution filter 개수 = 분류할 카테고리 개수

  활성 함수 : 카테고리수 1개면 sigmoid, 여러 개면 softmax 사용

U-Net 모델의 전체 코드

```
# U-Net model
# coded by st.watermelon

import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Conv2D
from tensorflow.keras.layers import Activation, BatchNormalizati


""" Conv Block """
class ConvBlock(tf.keras.layers.Layer):
    def __init__(self, n_filters):
        super(ConvBlock, self).__init__()

        self.conv1 = Conv2D(n_filters, 3, padding='same')
        self.conv2 = Conv2D(n_filters, 3, padding='same')

        self.bn1 = BatchNormalization()
        self.bn2 = BatchNormalization()

        self.activation = Activation('relu')

    def call(self, inputs):
        x = self.conv1(inputs)
        x = self.bn1(x)
        x = self.activation(x)

        x = self.conv2(x)
        x = self.bn2(x)
        x = self.activation(x)

        return x


""" Encoder Block """
class EncoderBlock(tf.keras.layers.Layer):
```

```
    def __init__(self, n_filters):
        super(EncoderBlock, self).__init__()

        self.conv_blk = ConvBlock(n_filters)
        self.pool = MaxPooling2D((2,2))

    def call(self, inputs):
        x = self.conv_blk(inputs)
        p = self.pool(x)
        return x, p


""" Decoder Block """
class DecoderBlock(tf.keras.layers.Layer):
    def __init__(self, n_filters):
        super(DecoderBlock, self).__init__()

        self.up = Conv2DTranspose(n_filters, (2,2), strides=2, p
        self.conv_blk = ConvBlock(n_filters)

    def call(self, inputs, skip):
        x = self.up(inputs)
        x = Concatenate()([x, skip])
        x = self.conv_blk(x)

        return x


""" U-Net Model """
class UNET(tf.keras.Model):
    def __init__(self, n_classes):
        super(UNET, self).__init__()

        # Encoder
        self.e1 = EncoderBlock(64)
        self.e2 = EncoderBlock(128)
```

```
        self.e3 = EncoderBlock(256)
        self.e4 = EncoderBlock(512)

        # Bridge
        self.b = ConvBlock(1024)

        # Decoder
        self.d1 = DecoderBlock(512)
        self.d2 = DecoderBlock(256)
        self.d3 = DecoderBlock(128)
        self.d4 = DecoderBlock(64)

        # Outputs
        if n_classes == 1:
            activation = 'sigmoid'
        else:
            activation = 'softmax'

        self.outputs = Conv2D(n_classes, 1, padding='same', acti

    def call(self, inputs):
        s1, p1 = self.e1(inputs)
        s2, p2 = self.e2(p1)
        s3, p3 = self.e3(p2)
        s4, p4 = self.e4(p3)

        b = self.b(p4)

        d1 = self.d1(b, s4)
        d2 = self.d2(d1, s3)
        d3 = self.d3(d2, s2)
        d4 = self.d4(d3, s1)

        outputs = self.outputs(d4)
```

```
        return outputs
```

장점

- Skip Connection 과정을 통해 인코더의 특징 맵을 디코더로 직접 전달하여 더 정교한 출력을 만듦
- U-Net은 데이터 증강을 통해 작은 데이터셋에서도 효과적으로 학습 가능
    - 이를 위해 회전, 이동, 크기 조정 등의 변환 과정을 통해 데이터를 인위적으로 늘림
- U-Net은 의료 영상의 특성에 적합한 모델로 설계됨, 2D와 3D 데이터 모두에서 사용 가능

데이터

- frame01, frame12 : 두 종류의 이미지 사용 가능 → 이걸 사용해 학습 및 평가
- GT(Ground Truth) : seg label이 포함된 데이터, 0,1,2,3으로 구성된 클래스

문제 정의

: 딥러닝 모델을 사용해 의료 이미지를 입력으로 받아 GT에 기반한 segmentation 결과를 예측하는 모델을 만듦

- 출력물 : 0,1,2,3 클래스로 이루어진 Segmentation map

U-Net 모델 사용

2D에서의 U-Net : 한 번에 한 장씩 2D 이미지를 처리 (H, W)

3D에서의 U-Net : 여러 슬라이스를 한 번에 입력하여 예측 (D, H, W)

평가 기준

1. Loss : 예측된 seg와 GT 간의 오차를 측정. 일반적으로 Cross-Entropy Loss 또는 Dice Loss 사용
2. Dice Score : GT와 예측된 seg 간의 일치도를 측정하는 지표. 주로 의료 영상 seg에서 사용됨

⇒ 예측한 픽셀 클래스와 실제 정답(GT) 간의 차이를 줄이는 데 집중해야 함

성능

- U-Net은 이미지 분할 문제에서 뛰어난 성능을 보임. 특히, 신경 구조 분할 및 세포 추적에서 높은 IOU 점수와 낮은 오류율을 기록
- 데이터 증강 기법 덕분에 적은 양의 데이터로도 우수한 성능 발휘

프로젝트

ACDC라는 학회에서 공개한 데이터셋, 심장 데이터들 test,train 데이터 있음, nifti파일로 있음. 이걸 이용해서 segmentaition 모델 만들어보자.

frame01, 12 둘 중에 하나쓰면 됨.

GT : seg 해야 하는 모델, 0,1,2,3으로 seg이 있음.

이걸 딥러닝을 써서 이 세 가지를 seg하는 모델을 만들어라. medical에서 가장 유명한 게 유넷이라는 모델(2d(2d 예측하는 모델 -> 10장을 각각 쪼개서), 3d → 전체를 한번에 유넷) → input 사이즈 디멘션이 다름 (두 가지 이미지에 대해 학습시켜보자),

GT와 이미지를 줘서, 학습을 해서 예측된 output를 예측하는 모델을 만들면 됨, 성능은 loss나 dice score seg 대포 스킬을 써보자

플로우

- **입력 데이터 준비**:
  - `frame01.nii` : 모델의 입력으로 사용.
  - `frame01-gt.nii` : 학습 중 손실 계산 및 평가를 위한 라벨.

        - **0**: 배경
        - **1**: 좌심실
        - **2**: 우심실
        - **3**: 심근
    - **3D MRI 이미지**를 2D 슬라이스로 나눔.
    - GT(Ground Truth)도 동일한 방식으로 2D로 슬라이스하여 라벨링.
- **학습 과정**:
    - 원본 이미지(`frame01.nii`)를 모델에 입력.
    - U-Net 모델에 각 슬라이스(이미지)와 GT를 입력하여 학습.
    - 모델은 세그멘테이션 맵(예: 0, 1, 2, 3 클래스)을 출력.
    - 출력 세그멘테이션 맵과 GT(`frame01-gt.nii`)를 비교하여 손실(Loss)을 계산.
- **결과 도출**:
    - 학습된 모델을 사용해 새로운 이미지(`frame02.nii` 등)에 대해 예측.
    - 예측 결과를 GT 데이터와 비교해 **Dice Score**, **IoU** 등의 지표를 계산.

1. input
    - patient001_frame01_gt.nii.gz : 실제 MRI 원본 의료 이미지
    - patient001_frame01.nii.gz : GT 데이터로, 각 픽셀이 클래스(0,1,2,3)을 나타내는 라벨 맵

      원본 이미지에 대한 정답으로, 모델 학습 시 손실을 계산할 때 사용

입력 : 각각의 슬라이싱된 GT, 일반 이미지

과정 : U-Net 모델에 각각의 이미지들을 입력해 학습