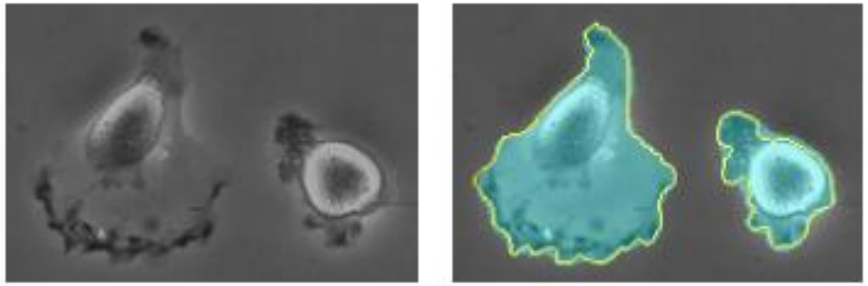
Segmentation by U-Net

First Lecture: U-Net

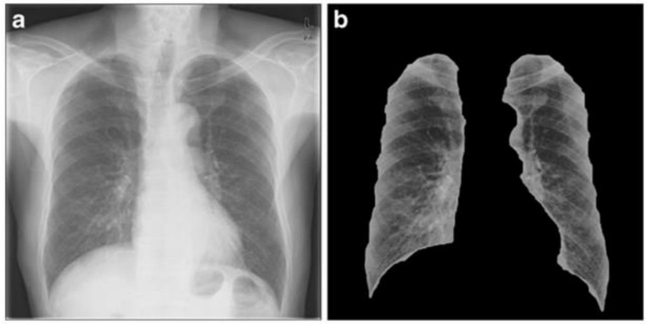
By Hongrun Zhang

Semantic Segmentation

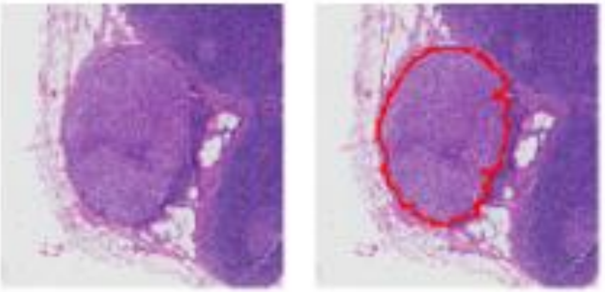
Only care about which category a pixel belongs to



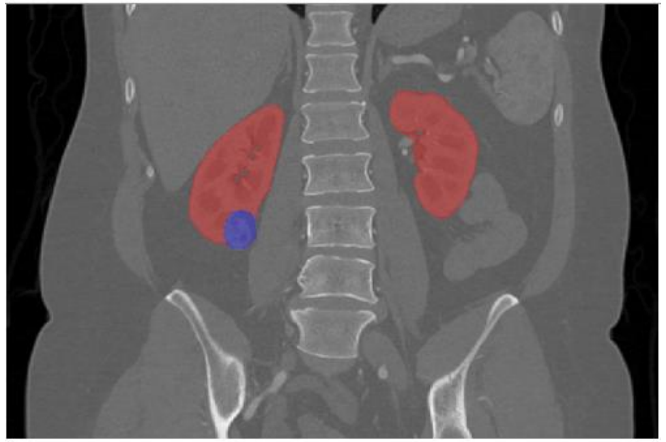
Cell



X-Ray



Microscopic images



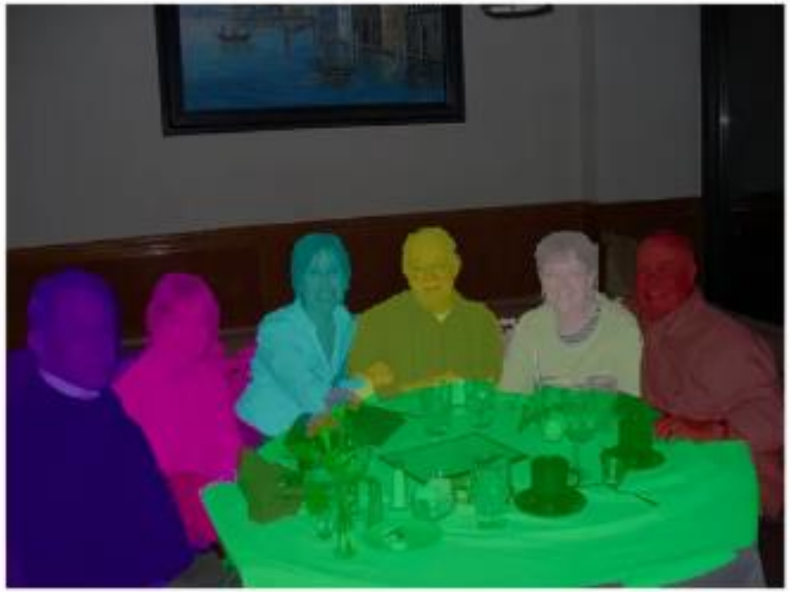
CT

Instance Segmentation

Two levels:

1. A pixel belongs to which category?
2. A pixel belongs to which instance?

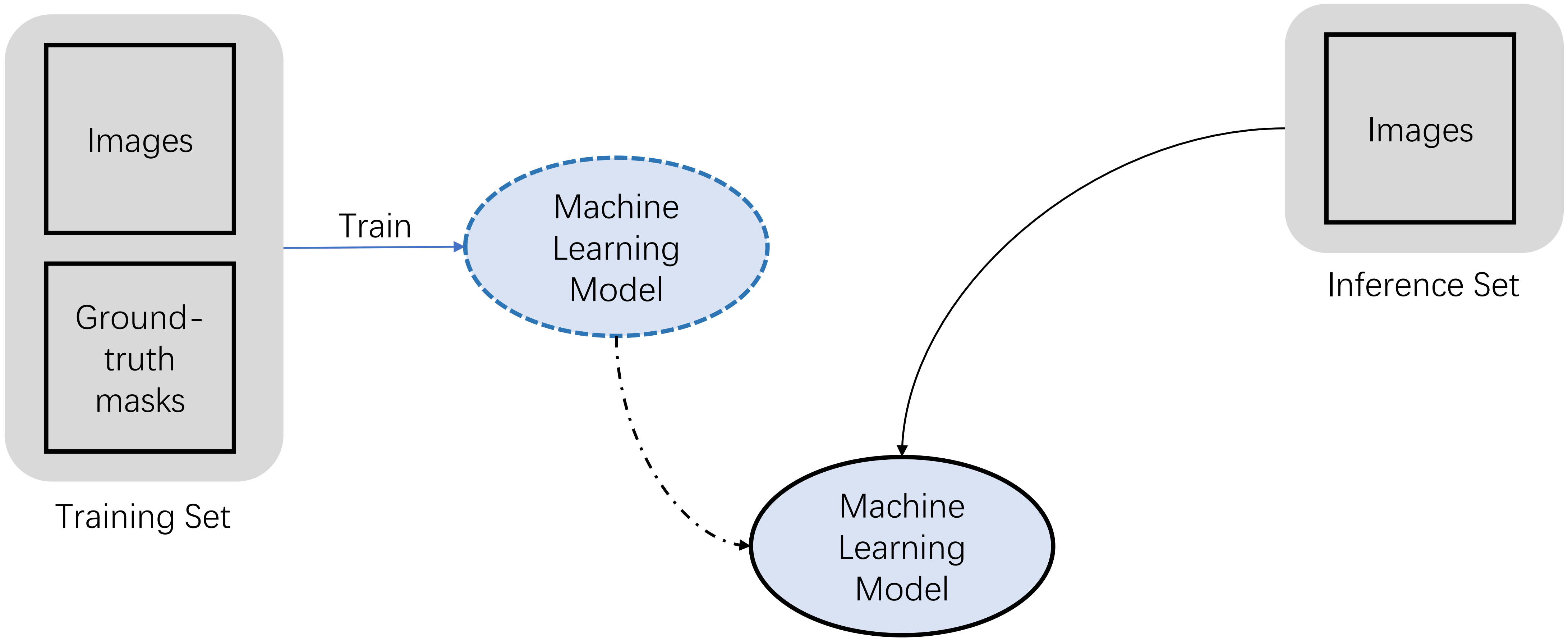
Semantic Segmentation



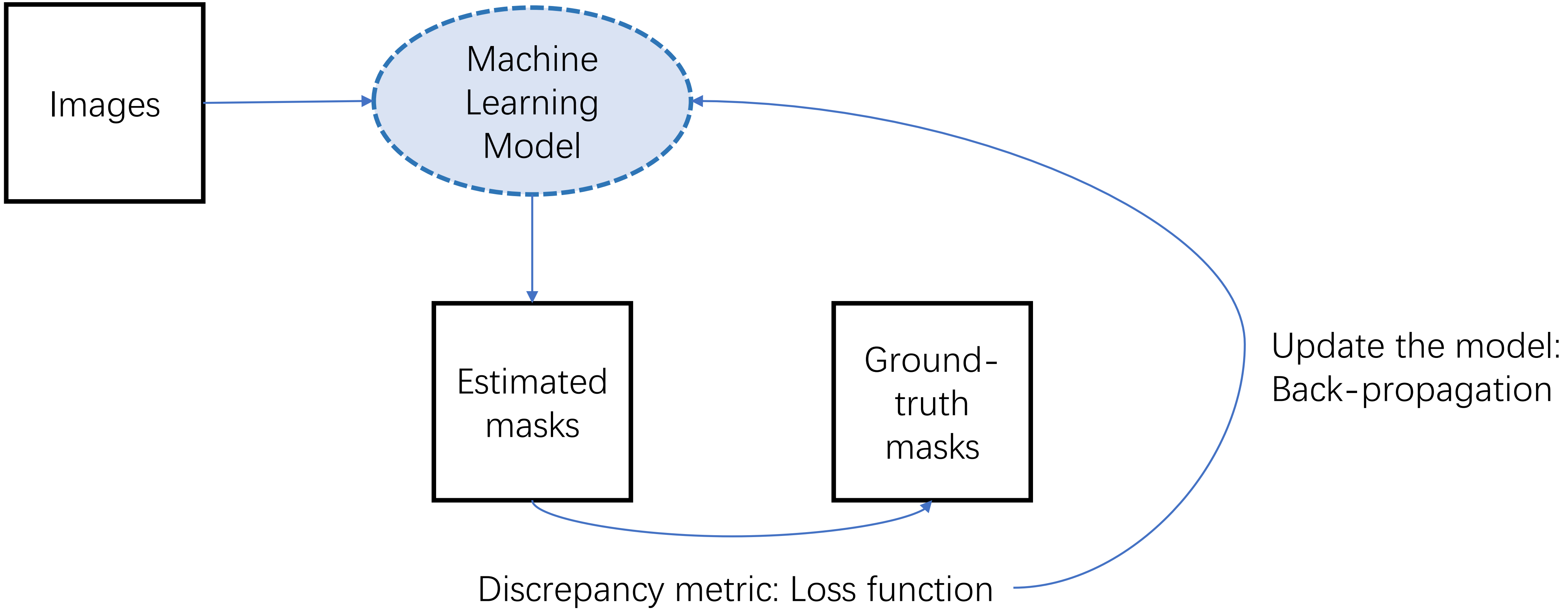
Instance Segmentation

https://blog.roboflow.com/difference-semantic-segmentation-instance-segmentation/

Machine learning-based semantic segmentation



Machine learning-based semantic segmentation: Training process



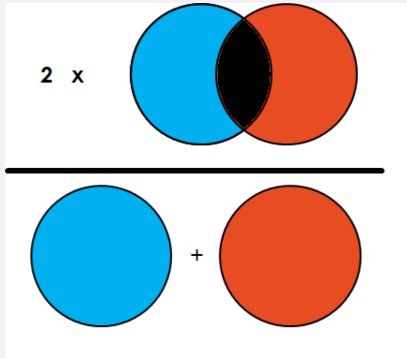
Loss functions

Cross-entropy

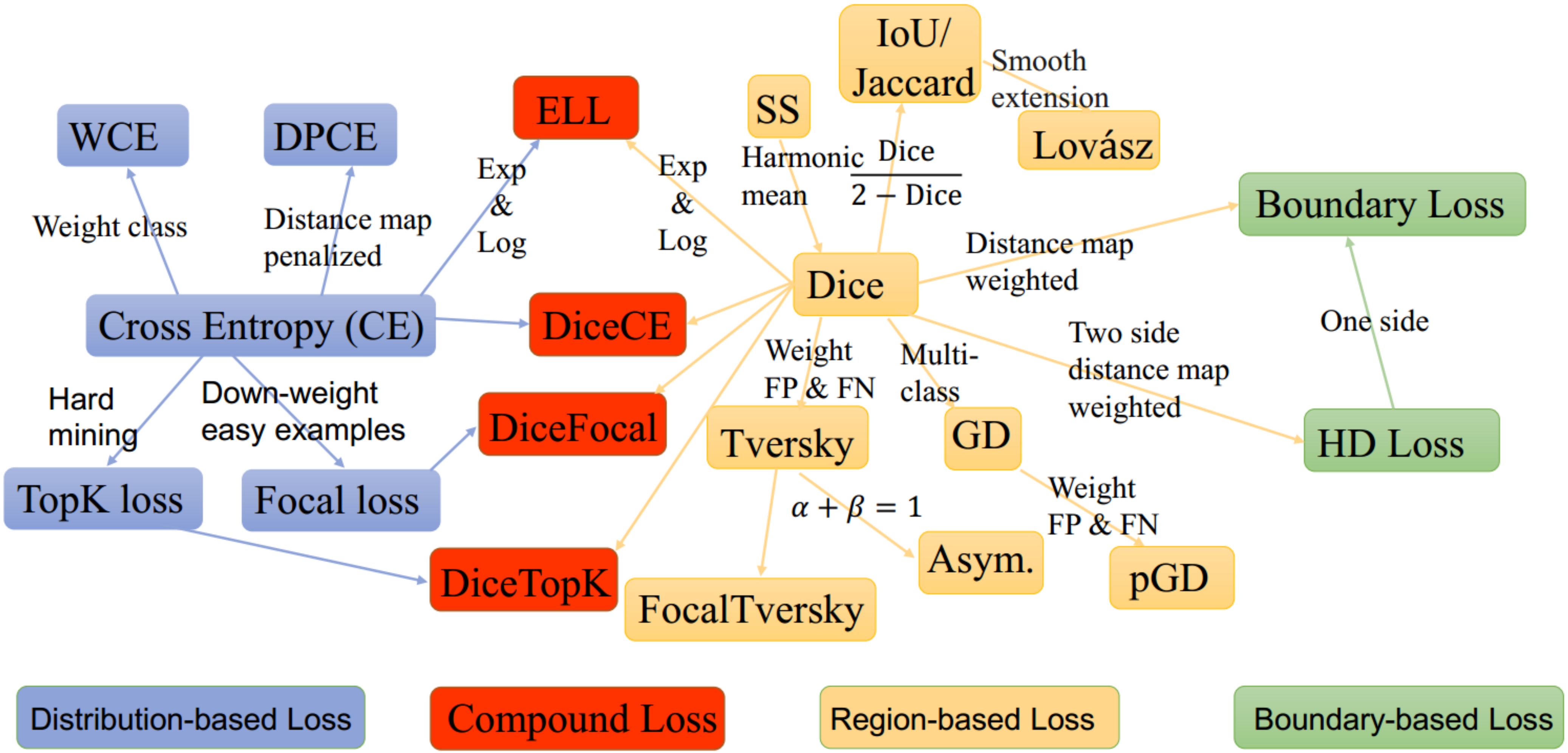
$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)log(1-\hat{y})$$

Dice coefficient

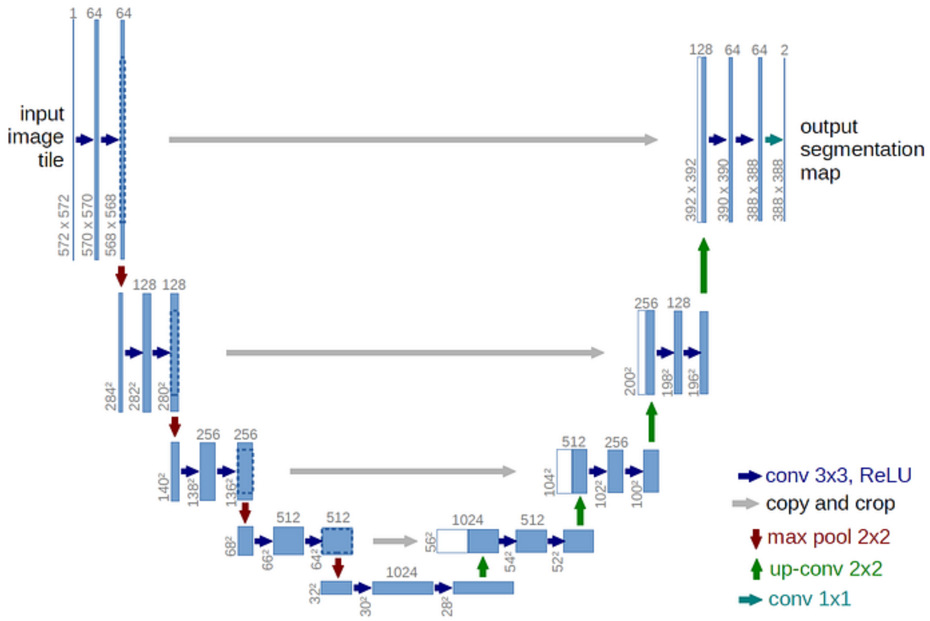
$$D = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2}$$



Loss functions

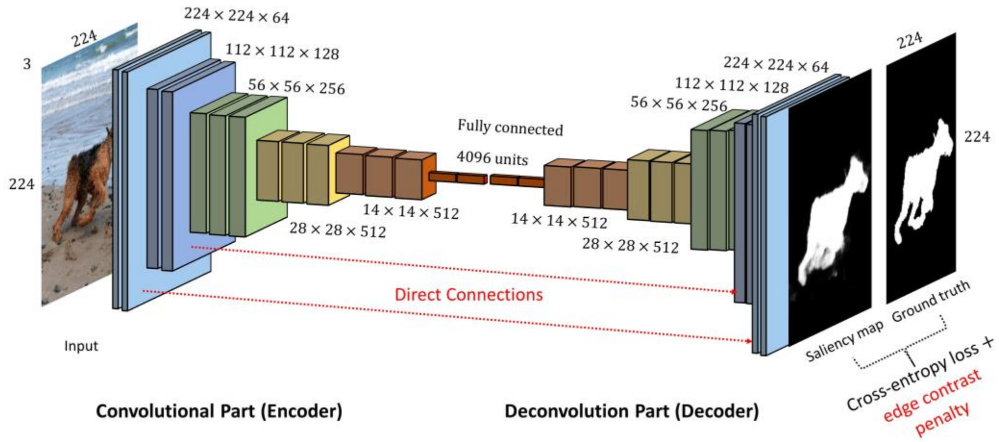


U-Net



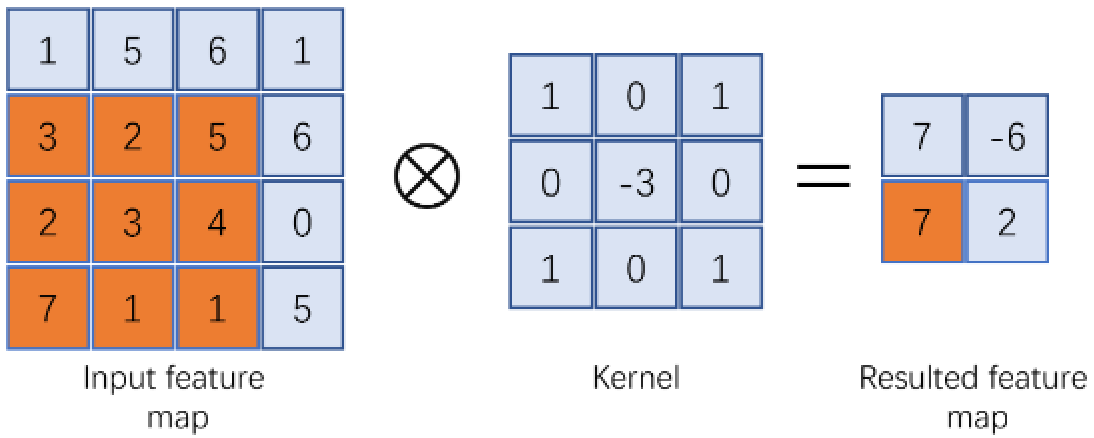
Basic Components

Convolution Auto-encoder



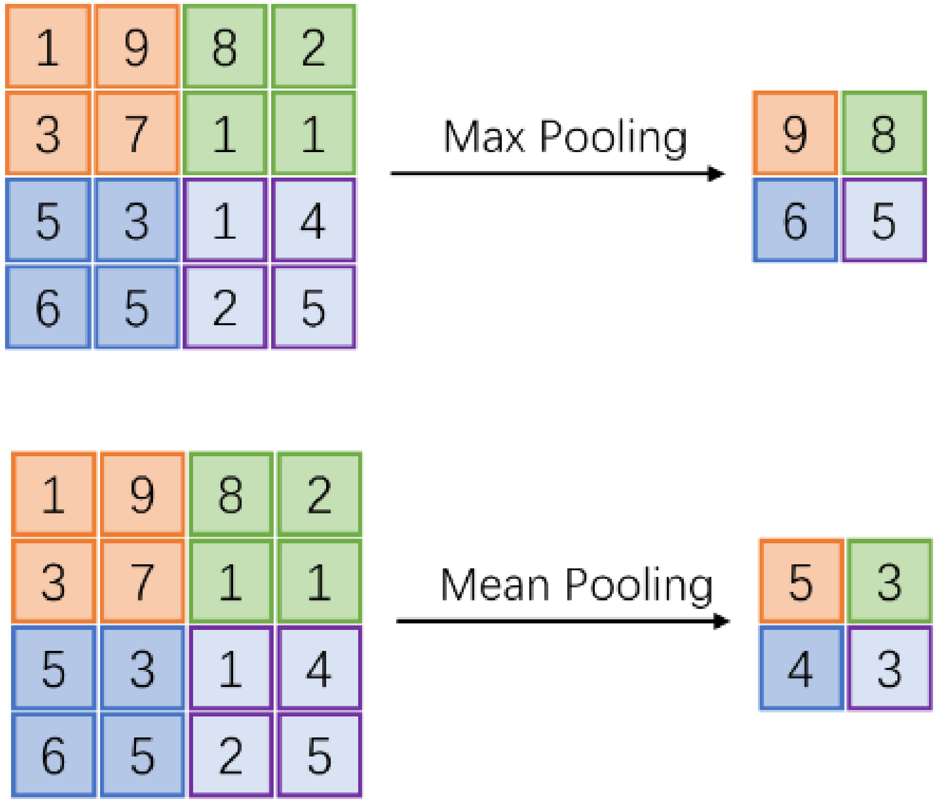
Basic convolution Module: 2D convolution

$$y[m,n] = x[m,n] * h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j]$$



$3 \times 1 + 2 \times 0 + 5 \times 1 + 2 \times 0 + 3 \times 3 + 4 \times 0 + 7 \times 1 + 1 \times 0 + 1 \times 1 = 7$

Basic convolution Module: 2D Pooling

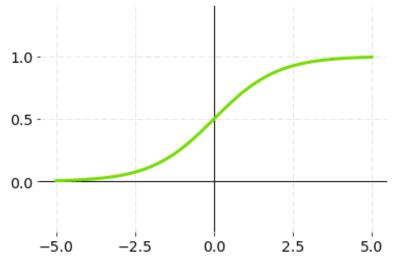


Basic convolution Module: Activation function

To introduce non-linearity for activation values

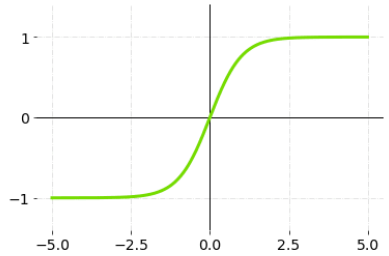
1. Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$



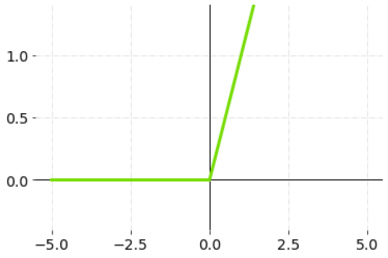
2. TanH

$$f(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$



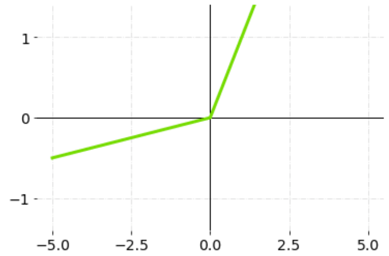
3. ReLU

$$f(x) = max\ (0, x)$$



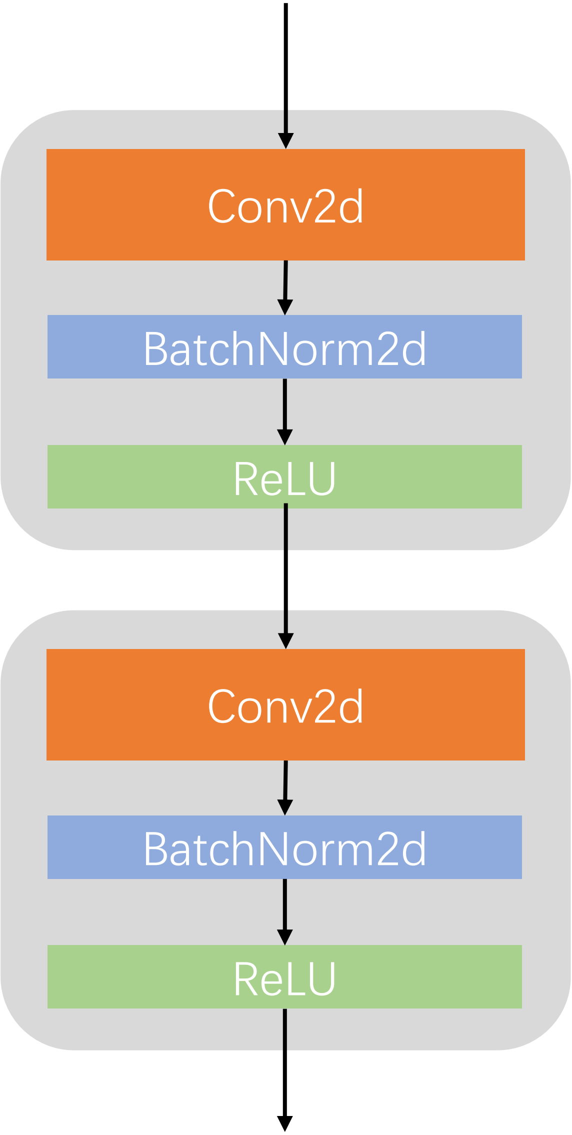
4. Leaky ReLU

$$f(x) = max\ (0.1x, x)$$



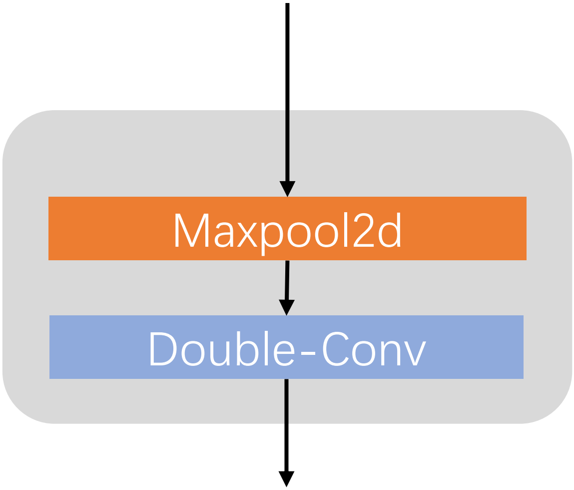
Double-conv block

```python
class DoubleConv(nn.Module):
    """(convolution => [BN] => ReLU) * 2"""

    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, bias=False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

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



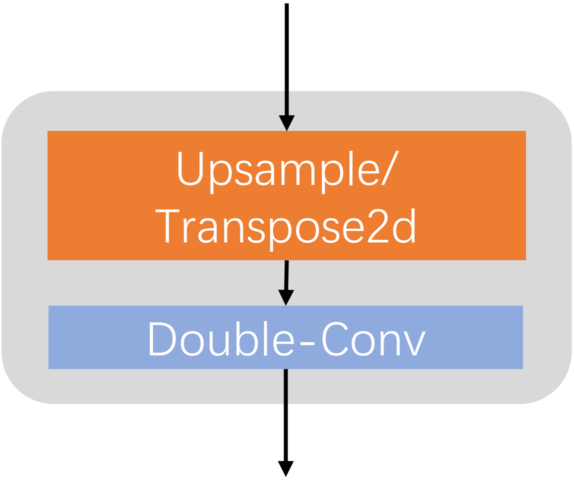
Encoder: down-sampling module

```
class Down(nn.Module):
    """Downscaling with maxpool then double conv"""

    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)
```

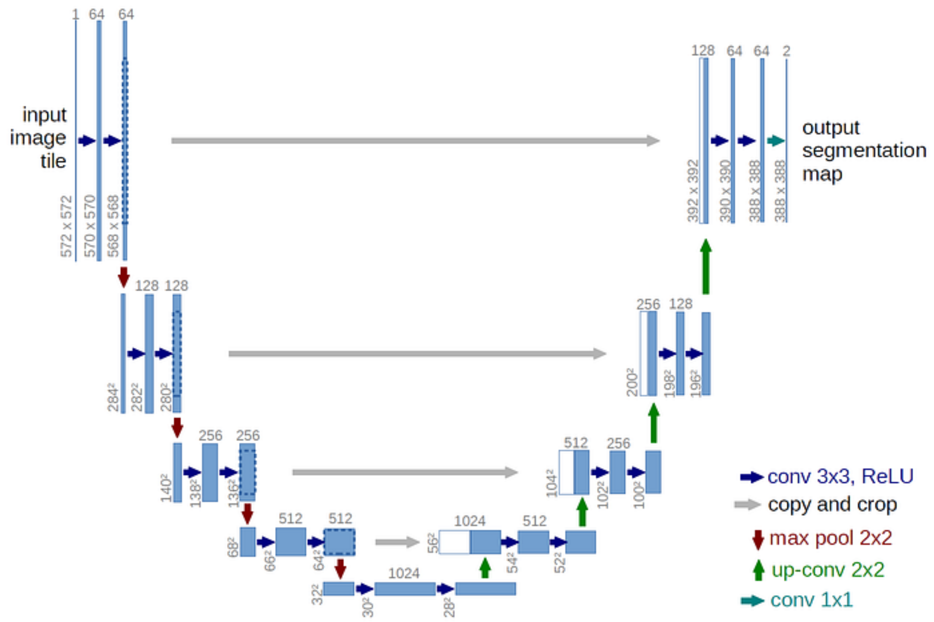


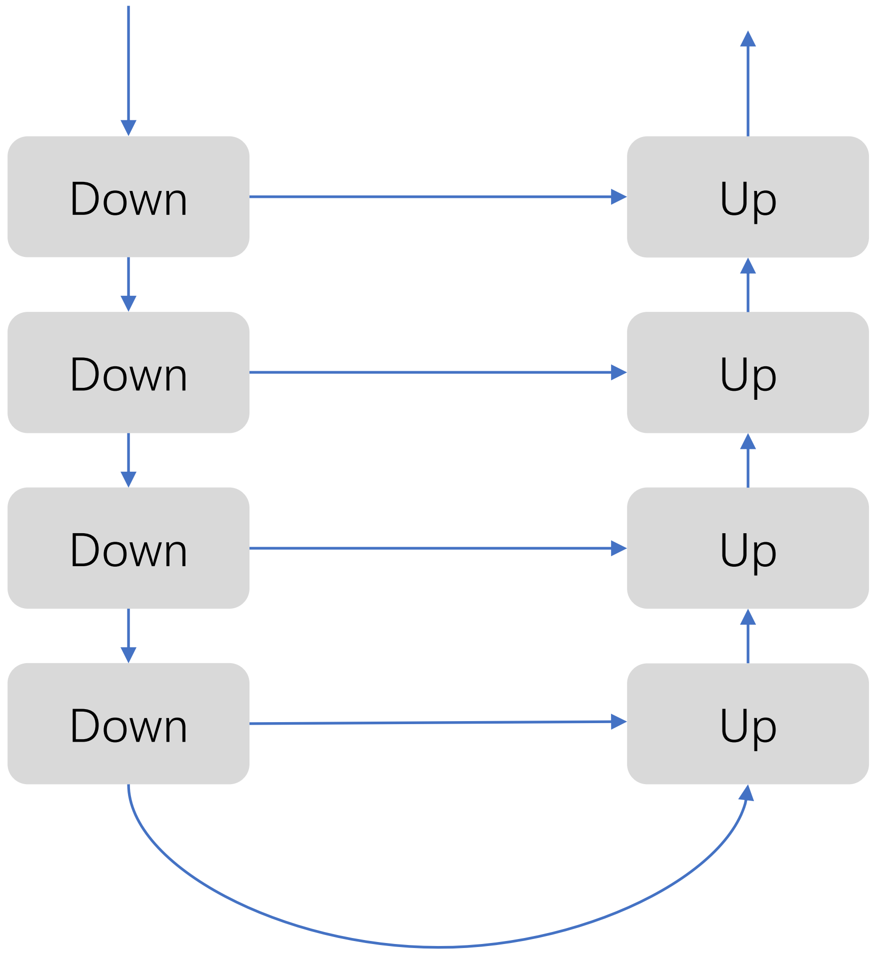
Decoder: Up-sampling module

```python
class Up(nn.Module):
    """Upscaling then double conv"""

    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()

        # if bilinear, use the normal convolutions to reduce the number of channels
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
            self.conv = DoubleConv(in_channels, out_channels, in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels, in_channels // 2, kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        # input is CHW
        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]

        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])
        # if you have padding issues, see
        # https://github.com/HaiyongJiang/U-Net-Pytorch-Unstructured-Buggy/commit/0e854509c2cea854e247a9c615f175f76fbb2e3a
        # https://github.com/xiaopeng-liao/Pytorch-UNet/commit/8ebac70e633bac59fc22bb5195e513d5832fb3bd
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)
```

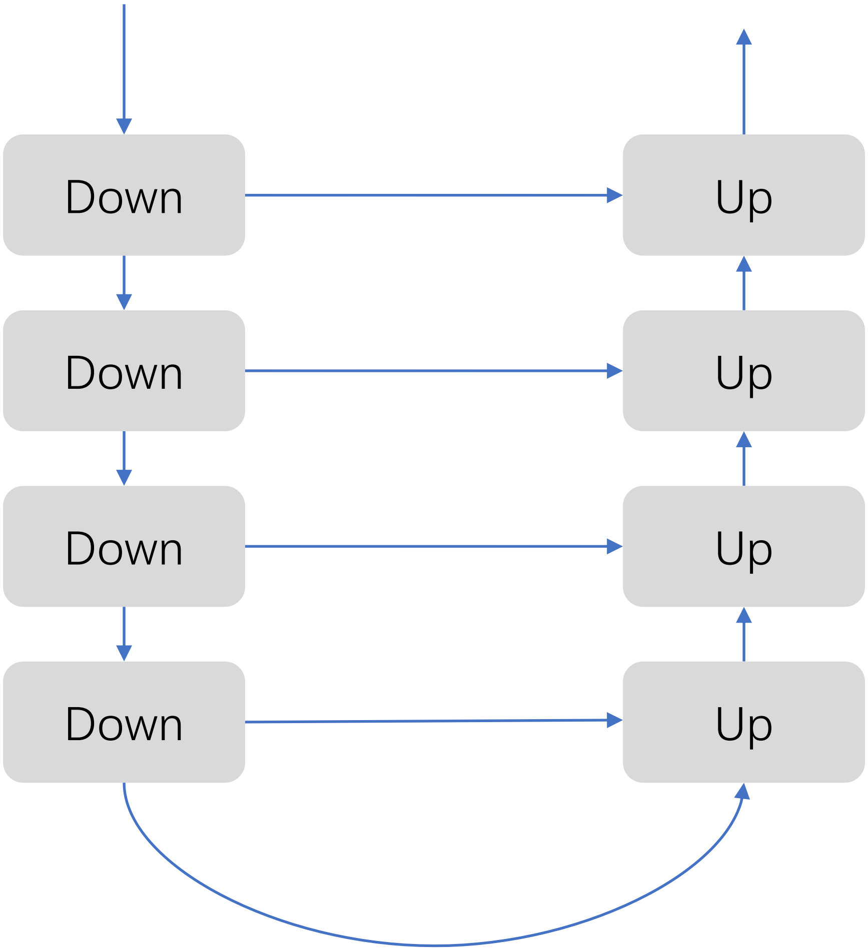


Whole U-Net structure





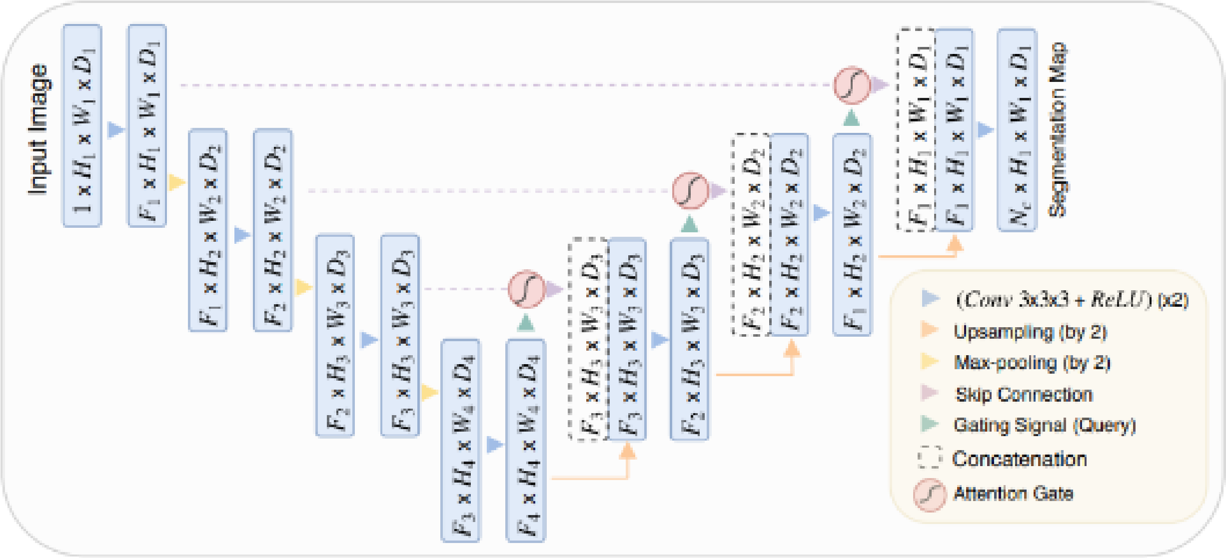
U-Net Variant



U-Net Variant

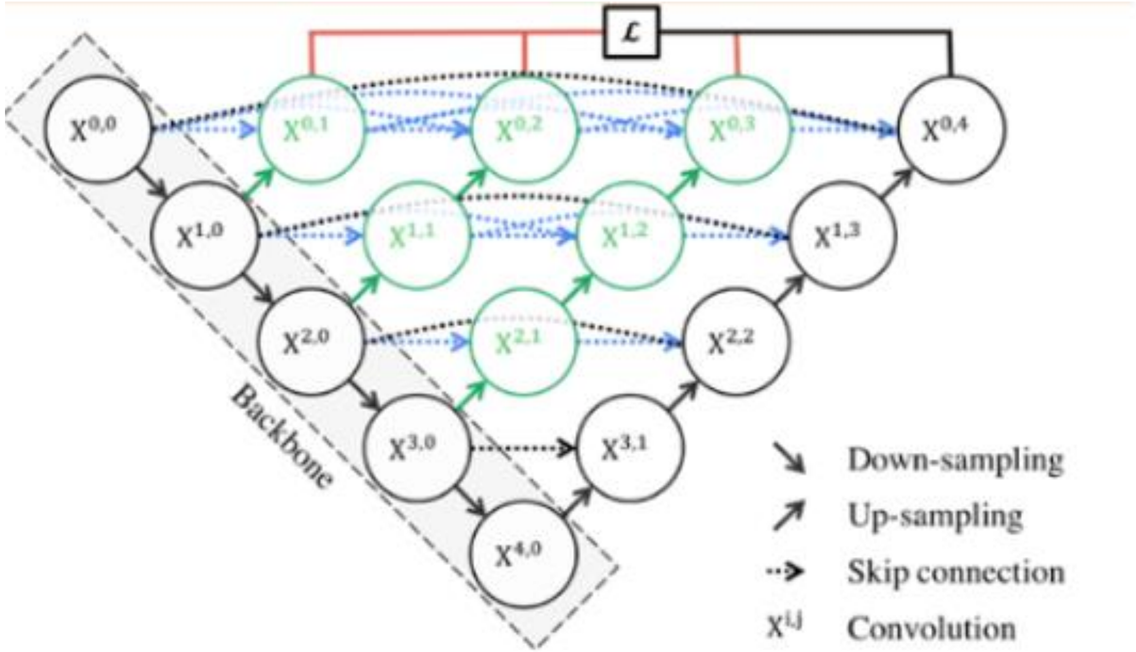
- Res-UNet
- Dense-UNet
- Attention-UNet
- Unet++
- Transformer-UNet

Attention U-Net



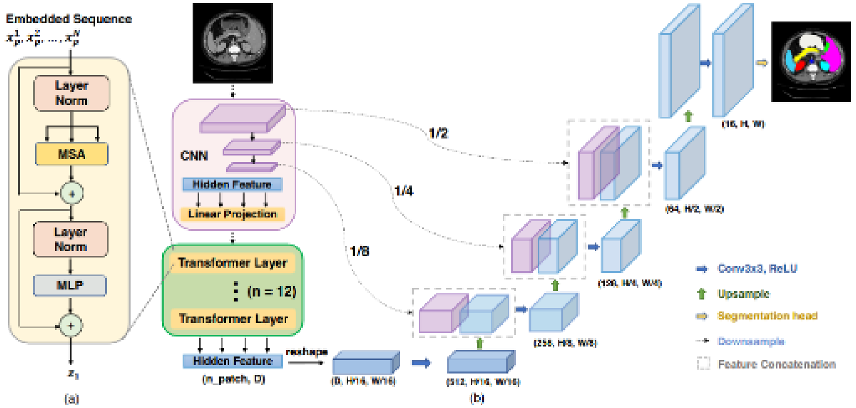
Ozan Oktay etal. ‘attention u-net: Learning where to look for the pancreas attention’

U-Net++

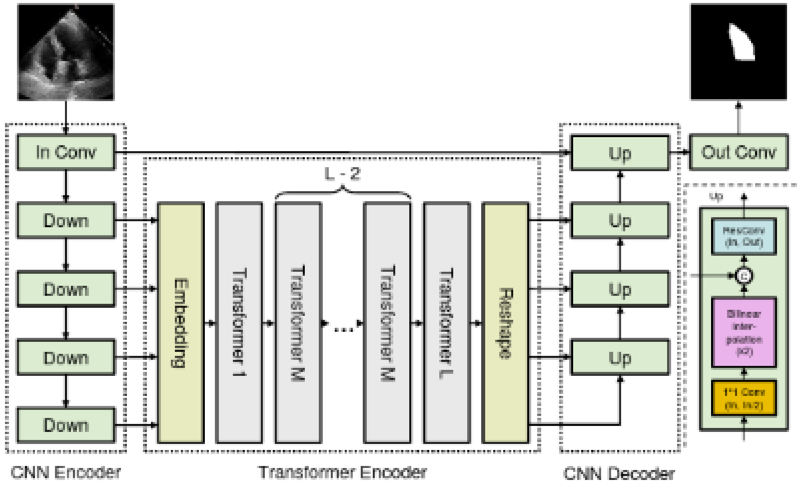


Zongwei Zhou. ‘Unet++: A nested u-net architecture for medical image segmentation’

Transformer-based U-Net



TransUNet

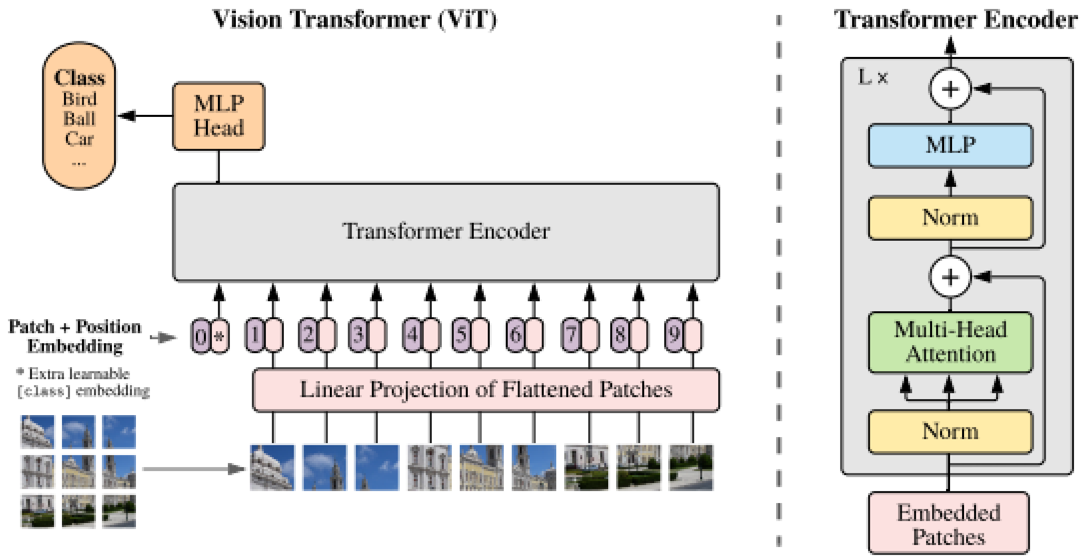


TransBridge

Jieneng Teng etal ‘Transformers make strong encoders for medical image segmentation’

Kaizhong Deng et al ‘Transbridge: A lightweight transformer for left ventricle segmentation in echocardiography’

Vision transformer (ViT)



Dosovitskiy A, Beyer L, Kolesnikov A, et al. An image is worth 16x16 words: Transformers for image recognition at scale[J]. arXiv preprint arXiv:2010.11929, 2020.

Task

- Run the code provided in github to come out with some segmentation results
- Try to replace the vanilla u-net with other architectures and redo the experiments. Compare the results.

https://github.com/LeeJunHyun/Image_Segmentation