COMPUTER VISION -
ATTENTION U - NET

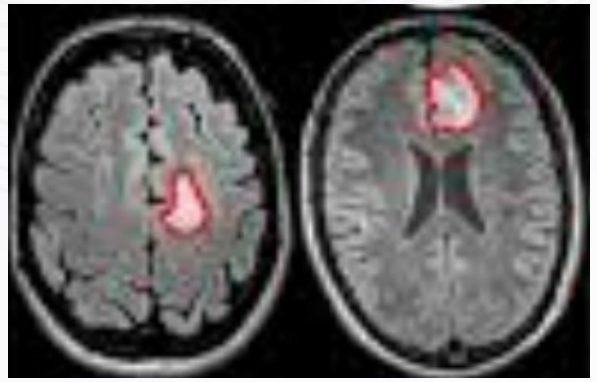
Prof., Dr.Eng Fitri Utaminingrum., ST., MT

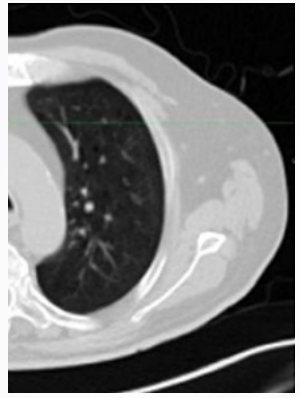
brone.ub.ac.id

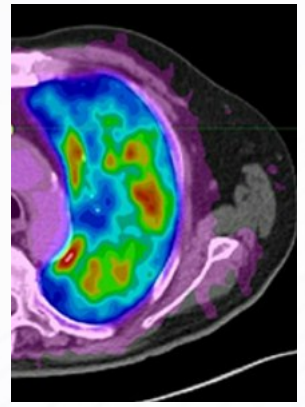
Introduction

Otomasi segmentasi pada citra medis merupakan topik yang penting, karena proses pelabelan manual dari sejumlah besar data citra medis merupakan tugas yang membosankan dan rawan kesalahan.

Sehingga diperlukan solusi yang akurat dan andal untuk meningkatkan efisiensi alur kerja klinis dan mampu mendukung pengambilan keputusan melalui ekstraksi pengukuran kuantitatif yang cepat dan otomatis



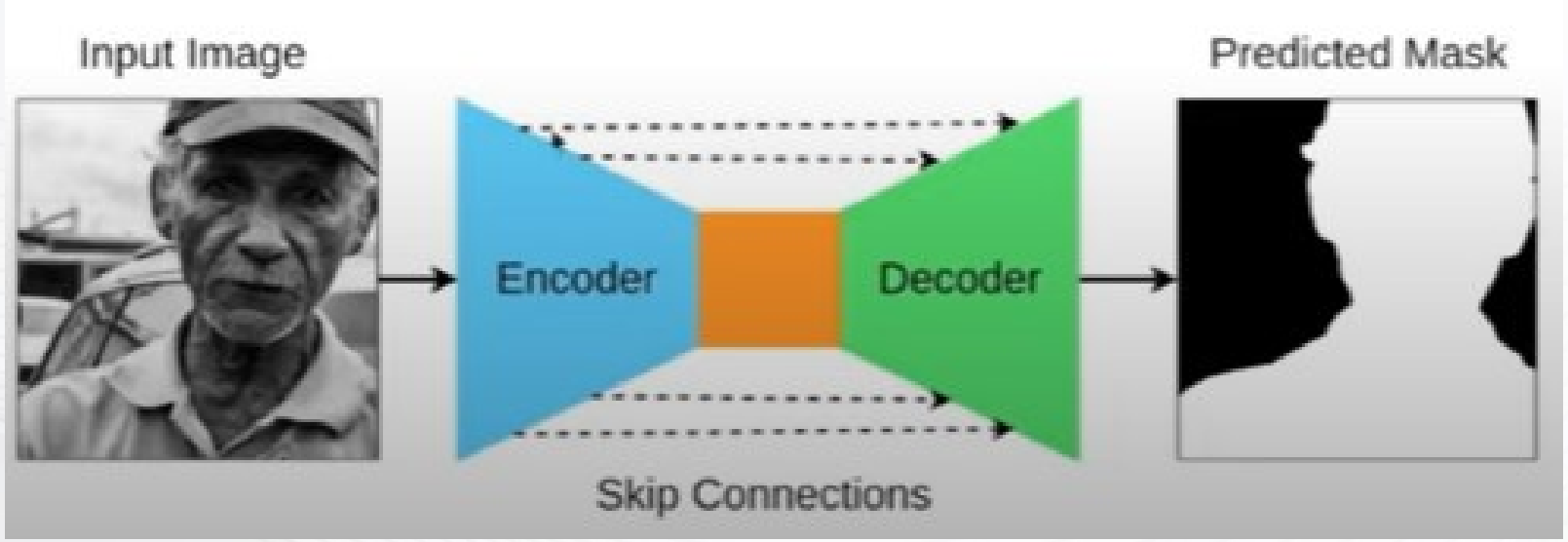




Introduction

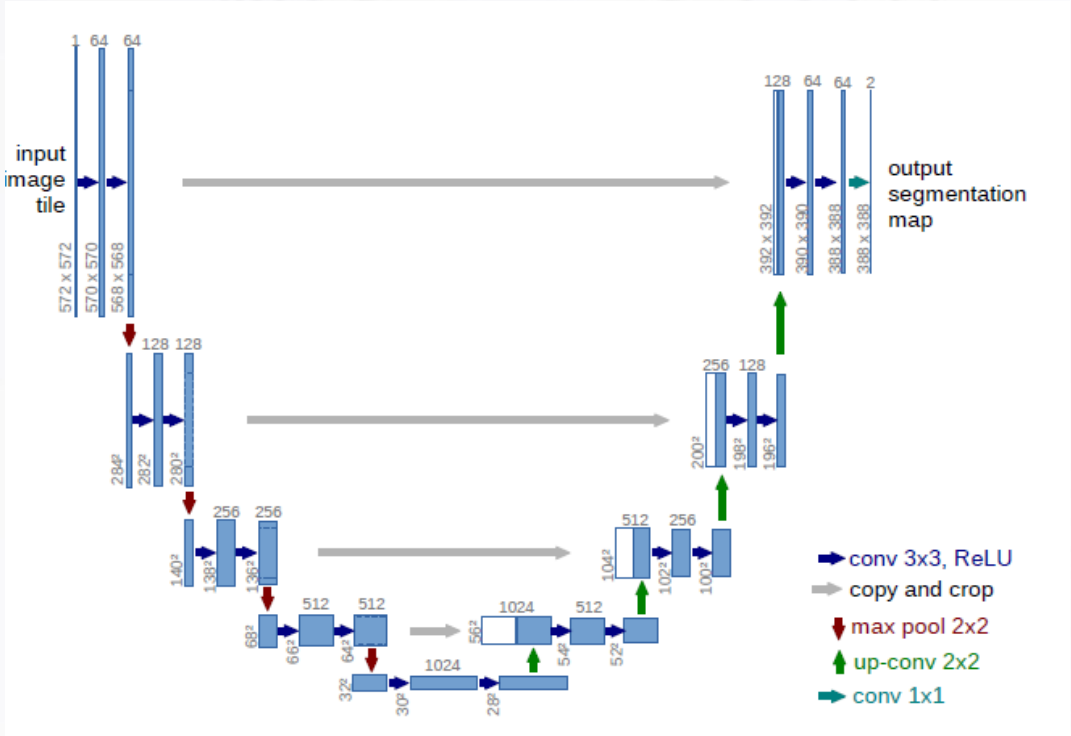
- Perkembangan Deep Learning di bidang Computer Vision membuat Convolutional Neural Networks (CNN) menjadi salah satu metode yang powerful untuk kasus segmentasi citra.
- Arsitektur umum yang digunakan:

1. Fully Convolutional Networks (FCN)
2. U-Net



U-Net

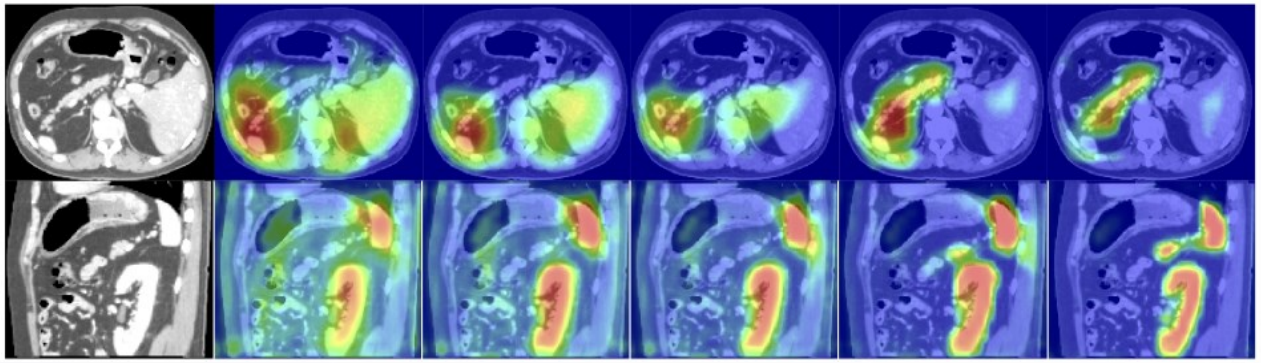
- Meskipun arsitektur FCN dan U-Net memiliki daya representasi yang baik
- Arsitektur tersebut mengandalkan multi-stage cascaded CNN ketika organ target menunjukkan *large inter-patient variations* dalam hal bentuk dan ukuran.



U-Net: Convolutional Networks for Biomedical Image Segmentation
(Olaf Ronneberger, Philipp Fischer, Thomas Brox)

Attention Gates (AG)

- Sehingga untuk mengatasi permasalahan tersebut, diusulkan cara yang lebih efektif yaitu menerapkan **Attention Gates (AG)** pada sisi skip connection U-Net.
- AG dapat secara otomatis belajar untuk **fokus pada struktur target** tanpa pengawasan tambahan.
- Pada tahap pengujian, AG mampu menghasilkan ***soft region proposal* secara implisit** saat itu juga dan menyorot fitur menonjol yang berguna untuk tugas tertentu.



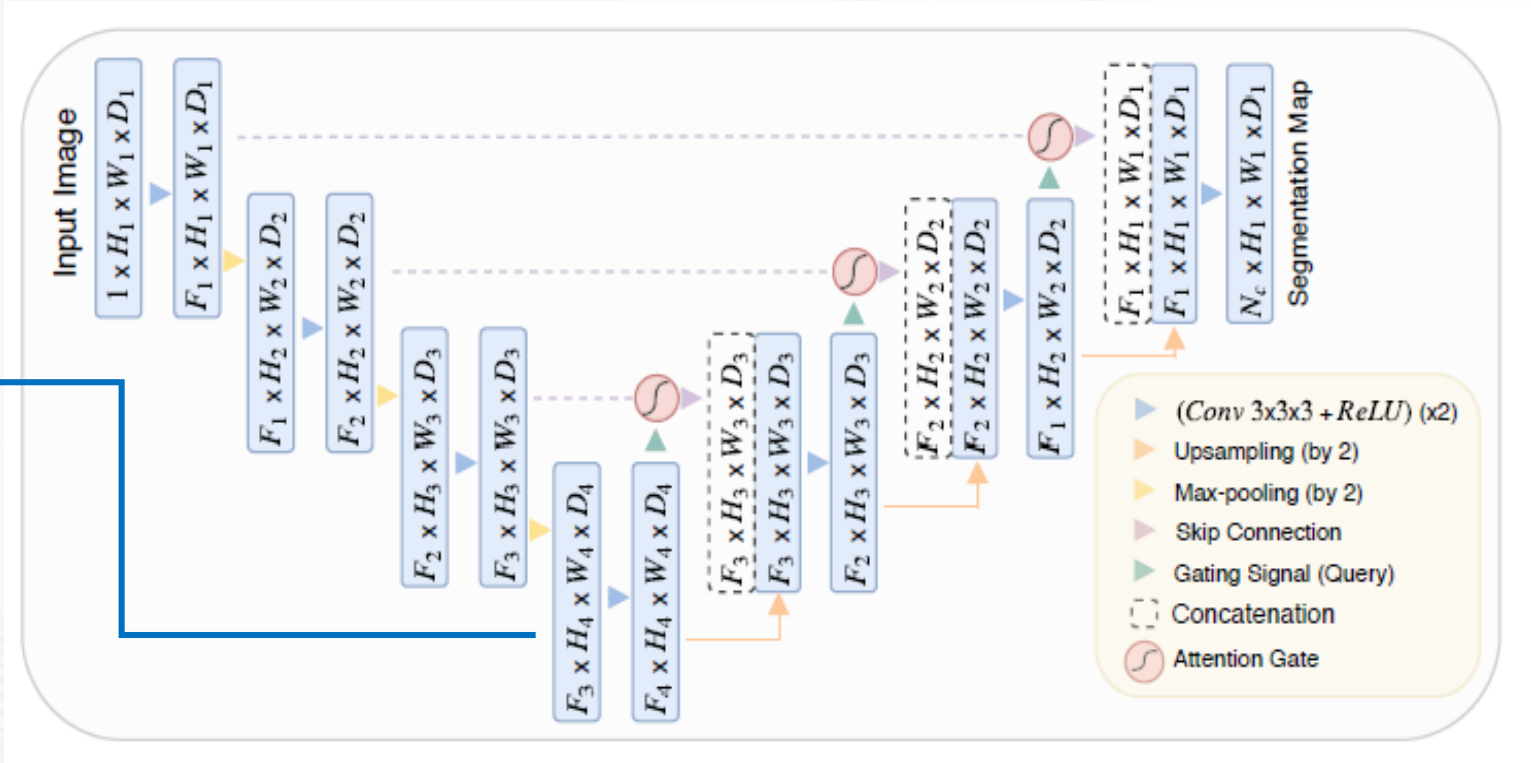
- Gambar menunjukan Attention Coefficient pada setiap epoch (3, 6, 10, 60, 150)
- Model secara bertahap belajar untuk fokus pada pankreas, ginjal, dan limpa

Arsitektur Attention U-Net

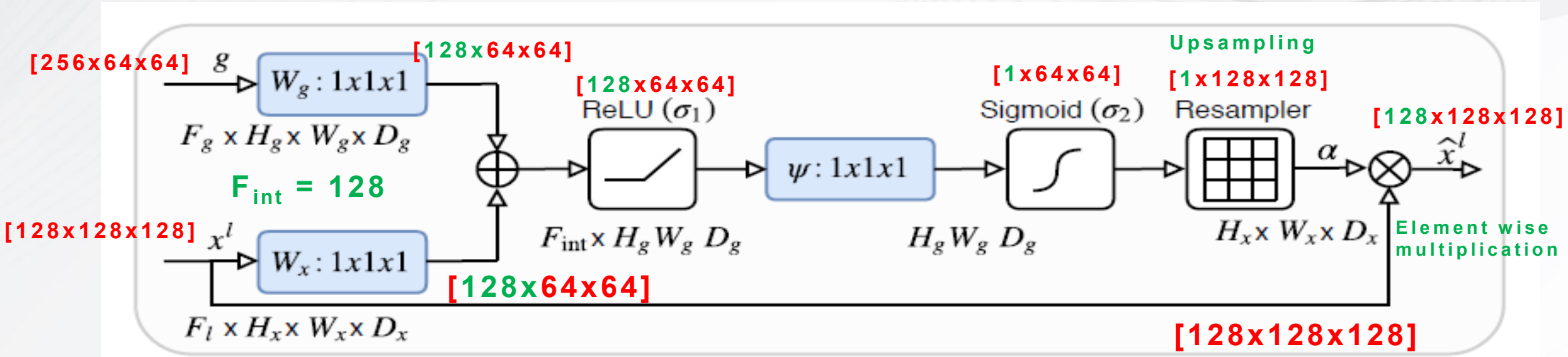
- Input image pada setiap block encoder akan melalui proses downsample dengan faktor 2

(Misalnya *H4* = $\frac{H1}{8}$)

- Attention Gates melakukan filter fitur yang dipropagasikan melalui skip connection.



[Channel x Height x Width]



- X^l = Input dari skip connection (low-level feature)
- g = Input dari high-level feature

$$q_{att}^l = \psi^T \left(\sigma_1 \left(W_x^T x_i^l + W_g^T g_i + b_g \right) \right) + b_\psi$$
$$\alpha_i^l = \sigma_2 \left(q_{att}^l (x_i^l, g_i; \Theta_{att}) \right),$$

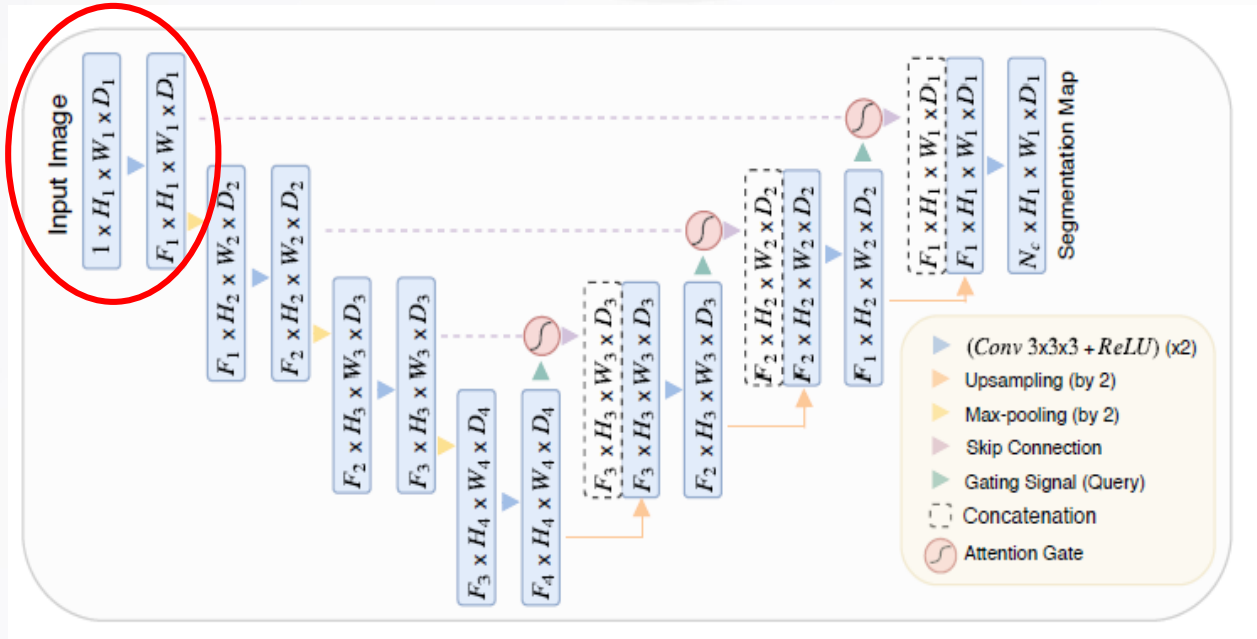
$$W_x \in \mathbb{R}^{F_l \times F_{int}}, W_g \in \mathbb{R}^{F_g \times F_{int}}$$
$$\psi \in \mathbb{R}^{F_{int} \times 1}$$

Attention U-Net: Learning Where to Look for the Pancreas (Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, Ben Glocker, Daniel Rueckert)

Implementasi U-Net Encoder Block

```python
import torch
import torch.nn as nn

class EncoderBlock(nn.Module):
    """Some Information about EncoderBlock"""
    def __init__(self, input_channel, output_channel, kernel_size):
        super(EncoderBlock, self).__init__()
        alpha = 0.2
        self.conv1 = nn.Conv2d(in_channels=input_channel, out_channels=output_channel,
                               kernel_size=kernel_size, stride=1, padding="same", bias=False)
        self.conv2 = nn.Conv2d(in_channels=output_channel, out_channels=output_channel,
                               kernel_size=kernel_size, stride=1, padding="same", bias=False)
        self.bn1 = nn.BatchNorm2d(num_features=output_channel)
        self.bn2 = nn.BatchNorm2d(num_features=output_channel)

        self.act = nn.LeakyReLU(negative_slope=alpha, inplace=True)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.act(x)

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

        return x, self.pool(x)
```

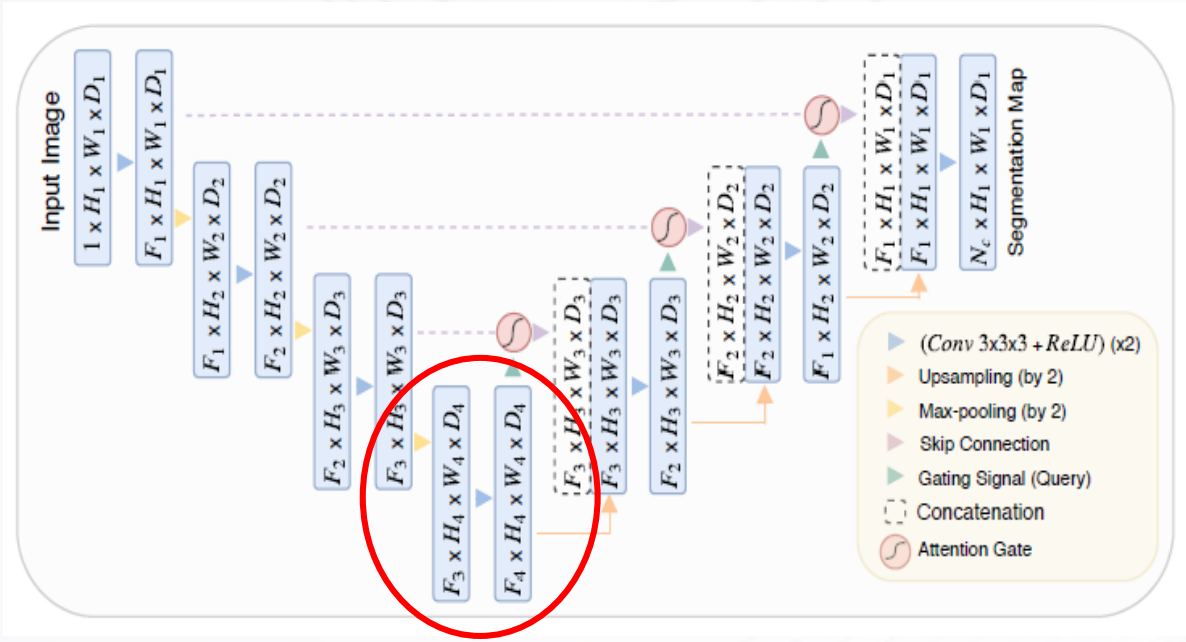
U-Net Encoder Block



Implementasi U-Net Bottleneck Block

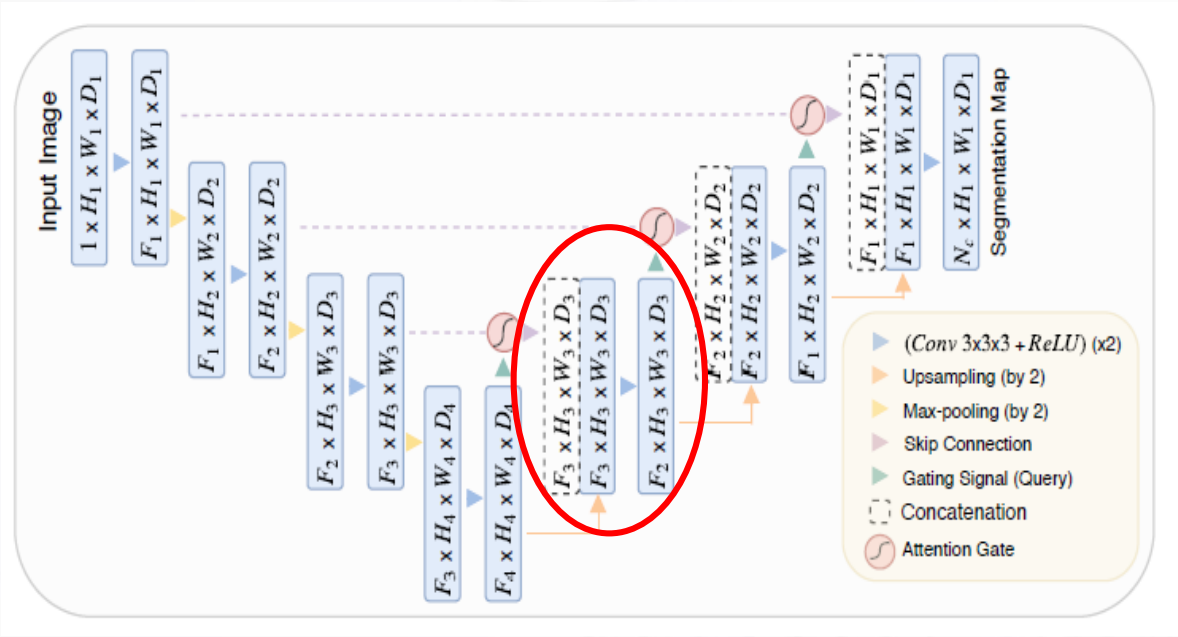
```python
class BottleneckBlock(nn.Module):
    """Some Information about BottleneckBlock"""

    def __init__(self, input_channel, output_channel, kernel_size):
        super(BottleneckBlock, self).__init__()
        alpha = 0.2
        self.conv1 = nn.Conv2d(in_channels=input_channel, out_channels=output_channel,
            kernel_size=kernel_size, stride=1, padding="same", bias=False)
        self.conv2 = nn.Conv2d(in_channels=output_channel, out_channels=output_channel,
            kernel_size=kernel_size, stride=1, padding="same", bias=False)

        self.bn1 = nn.BatchNorm2d(num_features=output_channel)
        self.bn2 = nn.BatchNorm2d(num_features=output_channel)

        self.act = nn.LeakyReLU(negative_slope=alpha, inplace=True)

    def forward(self, x):
        x = self.act(self.bn1(self.conv1(x)))
        x = self.act(self.bn2(self.conv2(x)))

        return x
```

U-Net Bottleneck Block



Implementasi U-Net Decoder Block

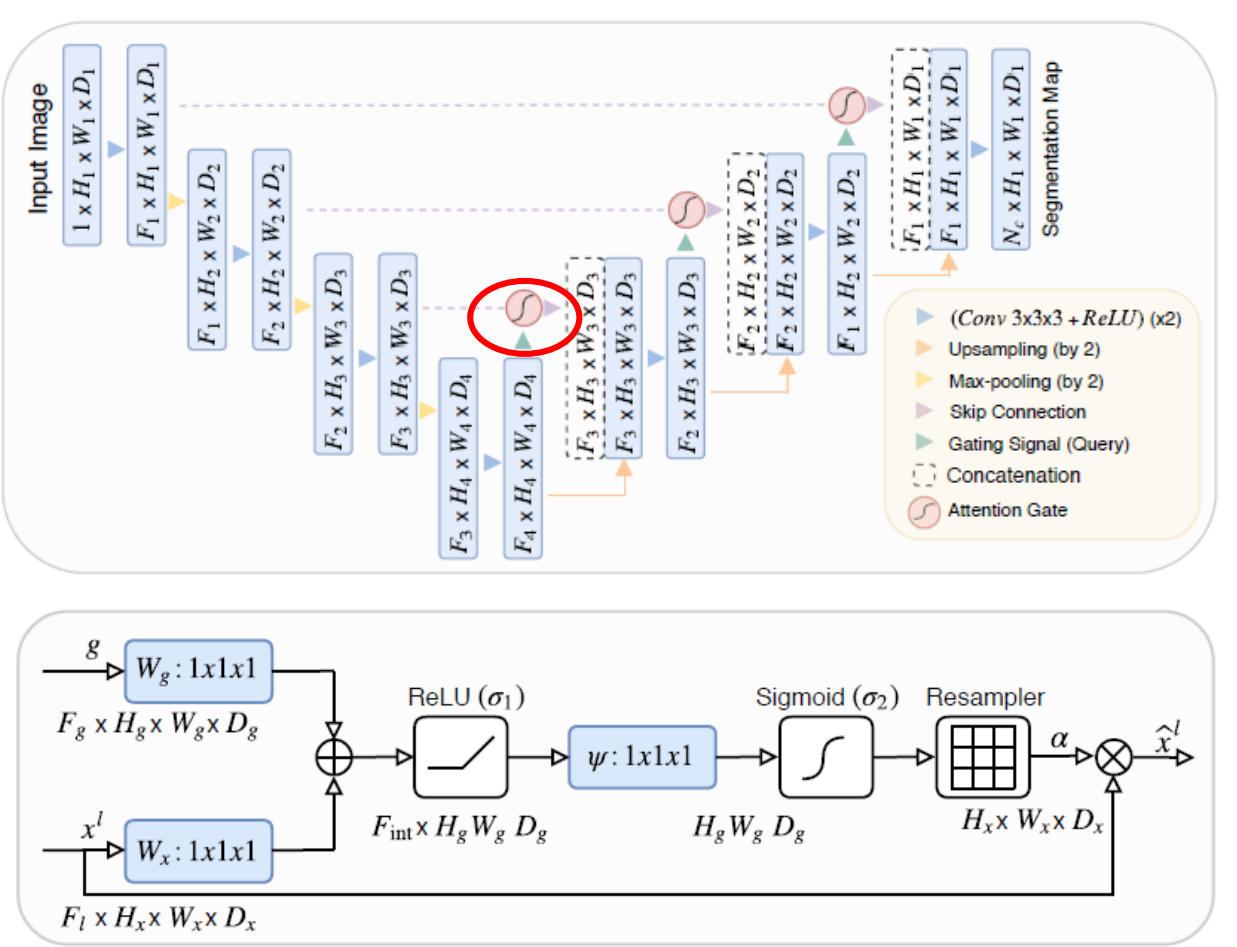
```python
class DecoderBLock(nn.Module):
    """Some Information about DecoderBLock"""

    def __init__(self, input_channel, output_channel, kernel_size):
        super(DecoderBLock, self).__init__()
        alpha = 0.2
        transconv_kernel = 2
        self.transconv = nn.ConvTranspose2d(in_channels=input_channel, out_channels=output_channel,
            kernel_size=transconv_kernel, stride=2, padding=(transconv_kernel - 1) // 2,
            dilation=1, bias=True,)

        self.conv1 = nn.Conv2d(in_channels=output_channel * 2, out_channels=output_channel,
            kernel_size=kernel_size, stride=1, padding="same", bias=True)
        self.conv2 = nn.Conv2d(in_channels=output_channel, out_channels=output_channel,
            kernel_size=kernel_size, stride=1, padding="same", bias=True)

        self.act = nn.LeakyReLU(negative_slope=alpha, inplace=True)

    def forward(self, x, skip):
        x = self.transconv(x)
        x = torch.cat([x, skip], dim=1)
        x = self.act(self.conv1(x))
        x = self.act(self.conv2(x))
        return x
```



U-Net Decoder Block

Implementasi Additive Attention Gate (AG)

```python
class AdditiveAttentionGate(nn.Module):
    """Some Information about AttentionGate"""

    def __init__(self, x_channel, g_channel, desired_channel):
        super(AdditiveAttentionGate, self).__init__()
        self.conv_x = nn.Sequential(
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(in_channels=x_channel, out_channels=desired_channel, kernel_size=1, stride=1, padding='same'),
        )
        self.conv_g = nn.Conv2d(in_channels=g_channel, out_channels=desired_channel, kernel_size=1, stride=1, padding="same")
        self.psi = nn.Conv2d(in_channels=desired_channel, out_channels=1, kernel_size=1, stride=1, padding="same")

        self.relu = nn.ReLU(inplace=True)
        self.sigmoid = nn.Sigmoid()

        self.upsample = nn.ConvTranspose2d(
            in_channels=1,
            out_channels=x_channel,
            kernel_size=2,
            stride=2,
            padding=(2 - 1) // 2,
            dilation=1,
            bias=True,
        )

    def forward(self, x, g):
        # misal x = (128, 64, 64) -> convert to (128, 32, 32) with strided conv
        #       g = (256, 32, 32) -> convert to (128, 32, 32) with 1x1 conv
        wx = self.conv_x(x) # (128, 32, 32)
        wg = self.conv_g(g) # (128, 32, 32)
        psi = self.psi(self.relu(wx + wg))
        psi = self.sigmoid(psi)
        # psi = self.upsample(psi, output_size=x.size()) digunakan kalau kernel upsample ganjil
        psi = self.upsample(psi)

        return psi * x
```



Additive Attention Gate (AG)

Implementasi Attention U-Net

```python
class AttentionUNet(nn.Module):
    """Some Information about UNetResNet"""

    def __init__(self, device):
        super(AttentionUNet, self).__init__()
        f = [32, 64, 128, 256, 512]
        kernel = 3
        self.encoder = nn.ModuleList([
                            EncoderBlock(input_channel=3, output_channel=f[0], kernel_size=kernel),
                            EncoderBlock(input_channel=f[0],output_channel=f[1], kernel_size=kernel),
                            EncoderBlock(input_channel=f[1], output_channel=f[2], kernel_size=kernel),
                            EncoderBlock(input_channel=f[2], output_channel=f[3], kernel_size=kernel),
                        ]).to(device)

        self.bottleneck = BottleneckBlock(input_channel=f[3], output_channel=f[4], kernel_size=3).to(device)

        self.decoder = nn.ModuleList([
                            DecoderBLock(input_channel=f[4], output_channel=f[3], kernel_size=kernel),
                            DecoderBLock(input_channel=f[3], output_channel=f[2], kernel_size=kernel),
                            DecoderBLock(input_channel=f[2], output_channel=f[1], kernel_size=kernel),
                            DecoderBLock(input_channel=f[1], output_channel=f[0], kernel_size=kernel),
                        ]).to(device)

        self.attention = nn.ModuleList([
                            AdditiveAttentionGate( x_channel=f[3], g_channel=f[4], desired_channel=f[3]),
                            AdditiveAttentionGate(x_channel=f[2], g_channel=f[3], desired_channel=f[2]),
                            AdditiveAttentionGate(x_channel=f[1], g_channel=f[2], desired_channel=f[1]),
                            AdditiveAttentionGate(x_channel=f[0], g_channel=f[1], desired_channel=f[0]),
                        ]).to(device)

        self.conv = nn.Conv2d(in_channels=f[0], out_channels=1, kernel_size=1, stride=1, padding="same").to(device)
        self.sigmoid = nn.Sigmoid()
```

Attention U-Net

```python
def forward(self, x):
    s1, p1 = self.encoder[0](x)  # Channel = 16
    s2, p2 = self.encoder[1](p1)  # Channel = 32
    s3, p3 = self.encoder[2](p2)  # Channel = 64
    s4, p4 = self.encoder[3](p3)  # Channel = 128

    botneck = self.bottleneck(p4)  # Channel = 256

    skip_attention1 = self.attention[0](s4, botneck)
    u1 = self.decoder[0](botneck, skip_attention1)

    skip_attention2 = self.attention[1](s3, u1)
    u2 = self.decoder[1](u1, skip_attention2)

    skip_attention3 = self.attention[2](s2, u2)
    u3 = self.decoder[2](u2, skip_attention3)

    skip_attention4 = self.attention[3](s1, u3)
    u4 = self.decoder[3](u3, skip_attention4)

    op = self.conv(u4)

    return self.sigmoid(op)
```

```python
if __name__ == '__main__':
    device = torch.device('cuda')
    model = AttentionUNet(device)
    img = torch.randn((5, 3, 256, 256)).to(device)

    print(model(img).shape)
```

Implementasi Pre-trained model / Transfer Learning

```python
class RenseNetEncoderBlock(nn.Module):
    """Some Information about EncoderBlock"""

    def __init__(self, backbone, freeze=False):
        super(RenseNetEncoderBlock, self).__init__()
        if backbone.lower() == 'resnet18':
            self.resnet = resnet18(weights=ResNet18_Weights.IMAGENET1K_V1)

        elif backbone.lower() == 'resnet34':
            self.resnet = resnet34(weights=ResNet34_Weights.IMAGENET1K_V1)

        elif backbone.lower() == 'resnet50':
            self.resnet = resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)

        elif backbone.lower() == 'resnet101':
            self.resnet = resnet101(weights=ResNet101_Weights.IMAGENET1K_V1)

        elif backbone.lower() == 'resnet152':
            self.resnet = resnet152(weights=ResNet152_Weights.IMAGENET1K_V1)

        if freeze:
            for v in self.resnet.parameters():
                v.requires_grad = False

    def forward(self, x):
        features = [x]
        modules = list(self.resnet.children())
        encoder = torch.nn.Sequential(*(list(modules)[:-2]))
        # i = 1
        for layer in encoder:
            features.append(layer(features[-1]))

        return features
```

- Contoh implementasi Residual Network (ResNet) pada arsitektur Attention U-Net
- Pre-Trained Model diimplementasikan pada **sisi Encoder** dari arsitektur Attention U-Net

Implementasi Pre-trained model / Transfer Learning

```python
def forward(self, x):
    enc = self.encoder(x)
    block1, block2, block3, block4, block5 = enc[3], enc[5], enc[6], enc[7], enc[8]

    ag1 = self.attention[0](block4, block5)
    u1 = self.decoder[0](ag1, block5)

    ag2 = self.attention[1](block3, u1)
    u2 = self.decoder[1](ag2, u1)

    ag3 = self.attention[2](block2, u2)
    u3 = self.decoder[2](ag3, u2)

    ag4 = self.attention[3](block1, u3)
    u4 = self.decoder[3](ag4, u3)

    op = self.head(u4)

    return op
```

- Dikarenakan Fitur yang diperoleh pada setiap block ResNet disimpan pada sebuah list, maka fitur dapat diperoleh dengan menggunakan index 3, 5, 6, 7, dan 8.
- Fitur pada setiap block encoder akan dipropagasikan pada AG dan decoder (sama seperti implementasi sebelumnya)