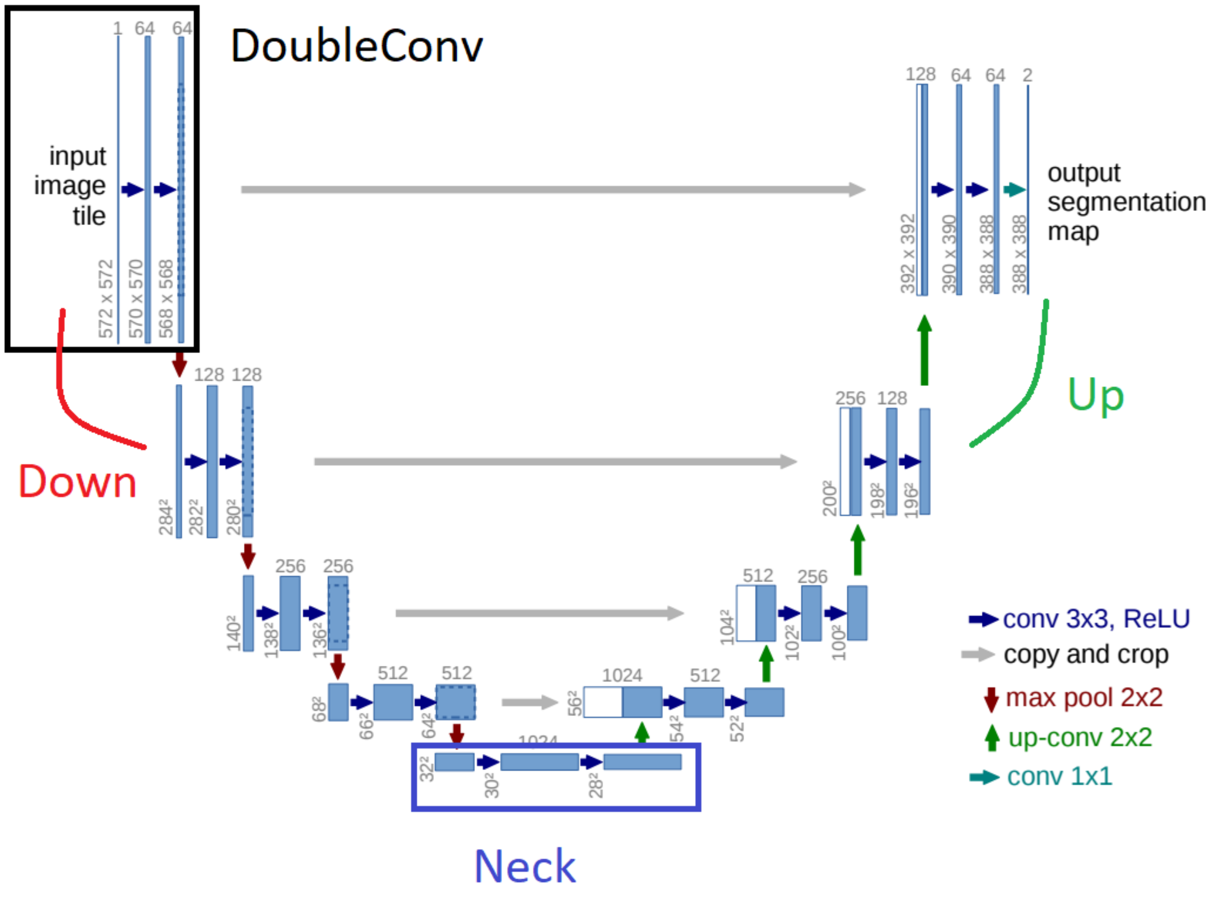
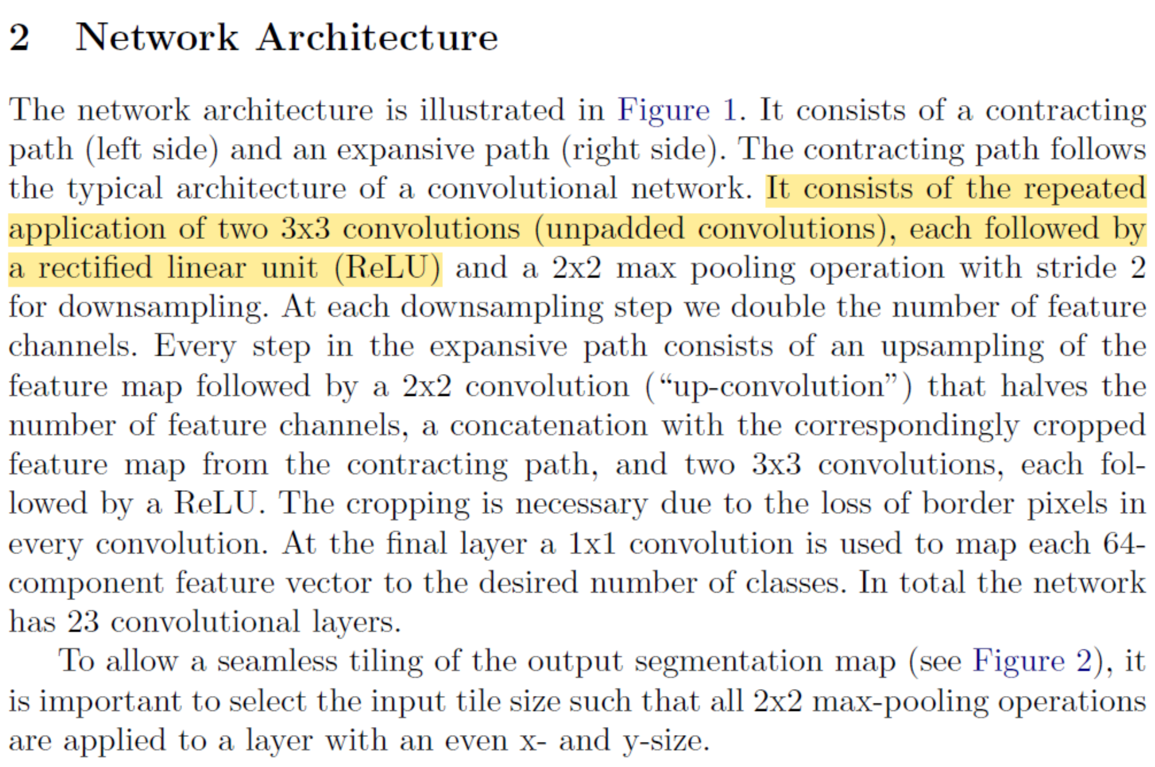
U-NET PyTorch



- **Double Convolution**

```
class DoubleConv(nn.Module):
    def __init__(self, in_, out_):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_, out_, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_, out_, kernel_size=3, padding=1),
            nn.ReLU(inplace=True)
        )
    def forward(self, x):
        return self.conv(x)
```



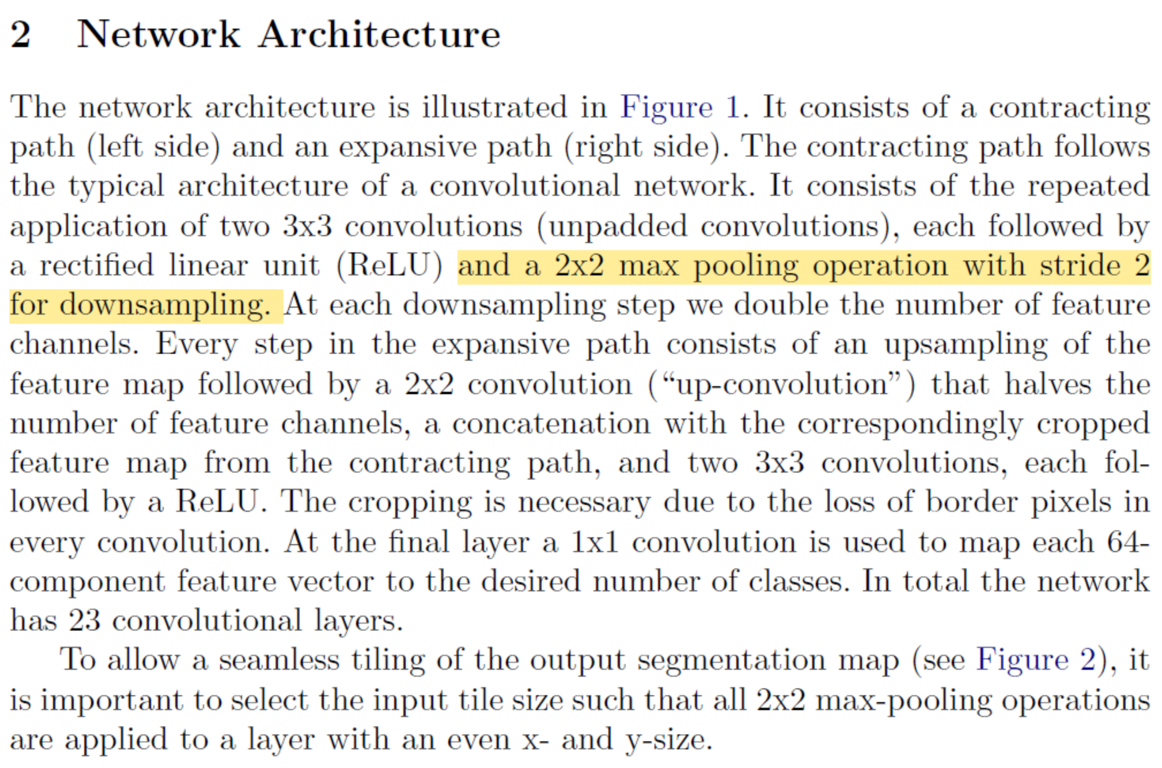
2 Network Architecture

The network architecture is illustrated in Figure 1. It consists of a contracting path (left side) and an expansive path (right side). The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers.

To allow a seamless tiling of the output segmentation map (see Figure 2), it is important to select the input tile size such that all 2x2 max-pooling operations are applied to a layer with an even x- and y-size.

- Down Sample

```
class Down(nn.Module):
    def __init__(self, in_, out_):
        super().__init__()
        self.conv = DoubleConv(in_, out_)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
    def forward(self, x):
        down = self.conv(x)
        pool = self.pool(down)
        return down, pool
```



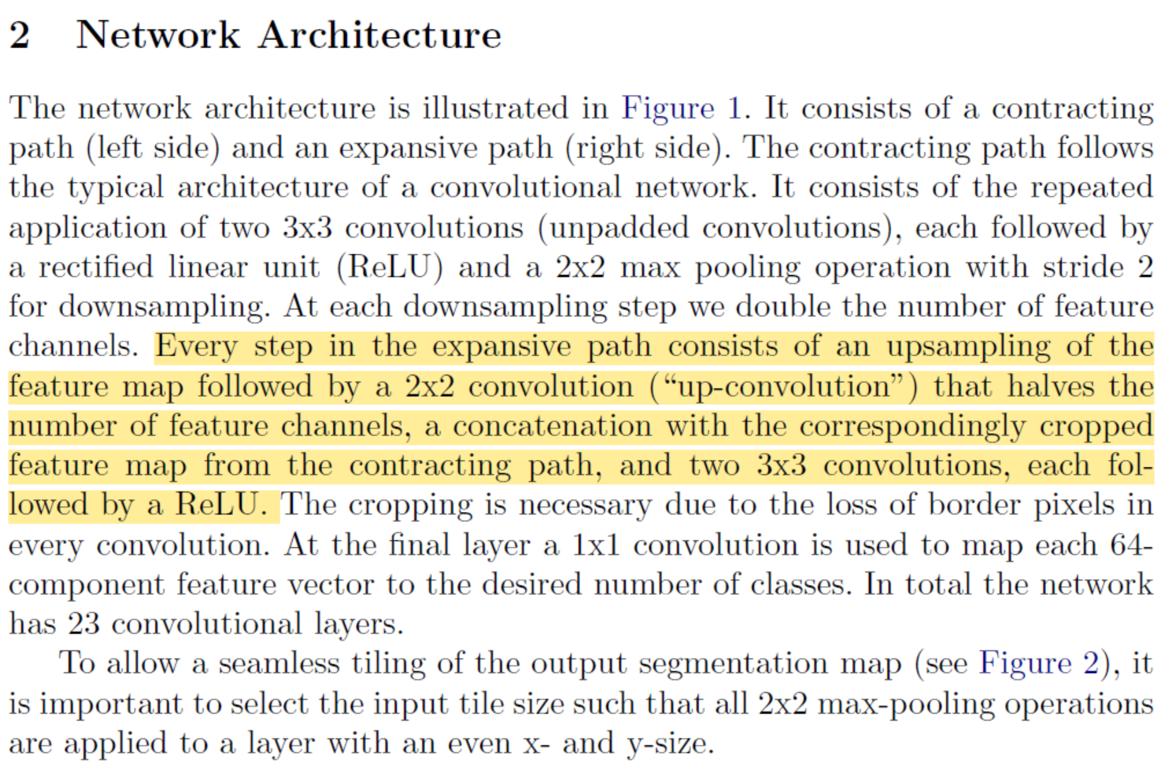
2 Network Architecture

The network architecture is illustrated in Figure 1. It consists of a contracting path (left side) and an expansive path (right side). The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers.

To allow a seamless tiling of the output segmentation map (see Figure 2), it is important to select the input tile size such that all 2x2 max-pooling operations are applied to a layer with an even x- and y-size.

- Up Sample

```
class Up(nn.Module):
    def __init__(self, in_, out_):
        super().__init__()
        self.up = nn.ConvTranspose2d(in_, in_//2, kernel_siz
        self.conv = DoubleConv(in_, out_)
    def forward(self, x1, x2):
        up = self.up(x1)
        out = torch.cat([up, x2], dim=1)
        out = self.conv(out)
        return out
```



2 Network Architecture

The network architecture is illustrated in Figure 1. It consists of a contracting path (left side) and an expansive path (right side). The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers.

To allow a seamless tiling of the output segmentation map (see Figure 2), it is important to select the input tile size such that all 2x2 max-pooling operations are applied to a layer with an even x- and y-size.

- U-Net

```
class UNet(nn.Module):
  def __init__(self, in_, n_class):
    super().__init__()
    self.step_down = 64
    self.n_class = n_class

    self.down_1 = Down(in_, self.step_down)
    self.down_2 = Down(in_, 2*self.step_down)
    self.down_3 = Down(in_, 4*self.step_down)
    self.down_4 = Down(in_, 8*self.step_down)

    self.neck = DoubleConv(8*self.step_down, 16*self.step_do

    self.up_1 = Up(16*self.step_down, 8*self.step_down)
    self.up_2 = Up(8*self.step_down, 4*self.step_down)
    self.up_3 = Up(4*self.step_down, 2*self.step_down)
    self.up_4 = Up(2*self.step_down, self.step_down)
    self.out = nn.Conv2d(in_channels=self.step_down,
                                             out_channels=n_c
                                             kernel_size=1)

  def forward(self, x):
    down1, pool1 = self.down_1(x)
    down2, pool2 = self.down_2(pool1)
    down3, pool3 = self.down_3(pool2)
    down4, pool4 = self.down_4(pool3)

    neck = self.neck(pool4)

    up1 = self.up_1(neck, down4)
    up2 = self.up_2(up1, down3)
    up3 = self.up_3(up2, down2)
    up4 = self.up_4(up3, down1)

    out = self.out(up4)
    return out
```

2 Network Architecture

The network architecture is illustrated in Figure 1. It consists of a contracting path (left side) and an expansive path (right side). The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step we double the number of feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a 1x1 convolution is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers.

To allow a seamless tiling of the output segmentation map (see Figure 2), it is important to select the input tile size such that all 2x2 max-pooling operations are applied to a layer with an even x- and y-size.