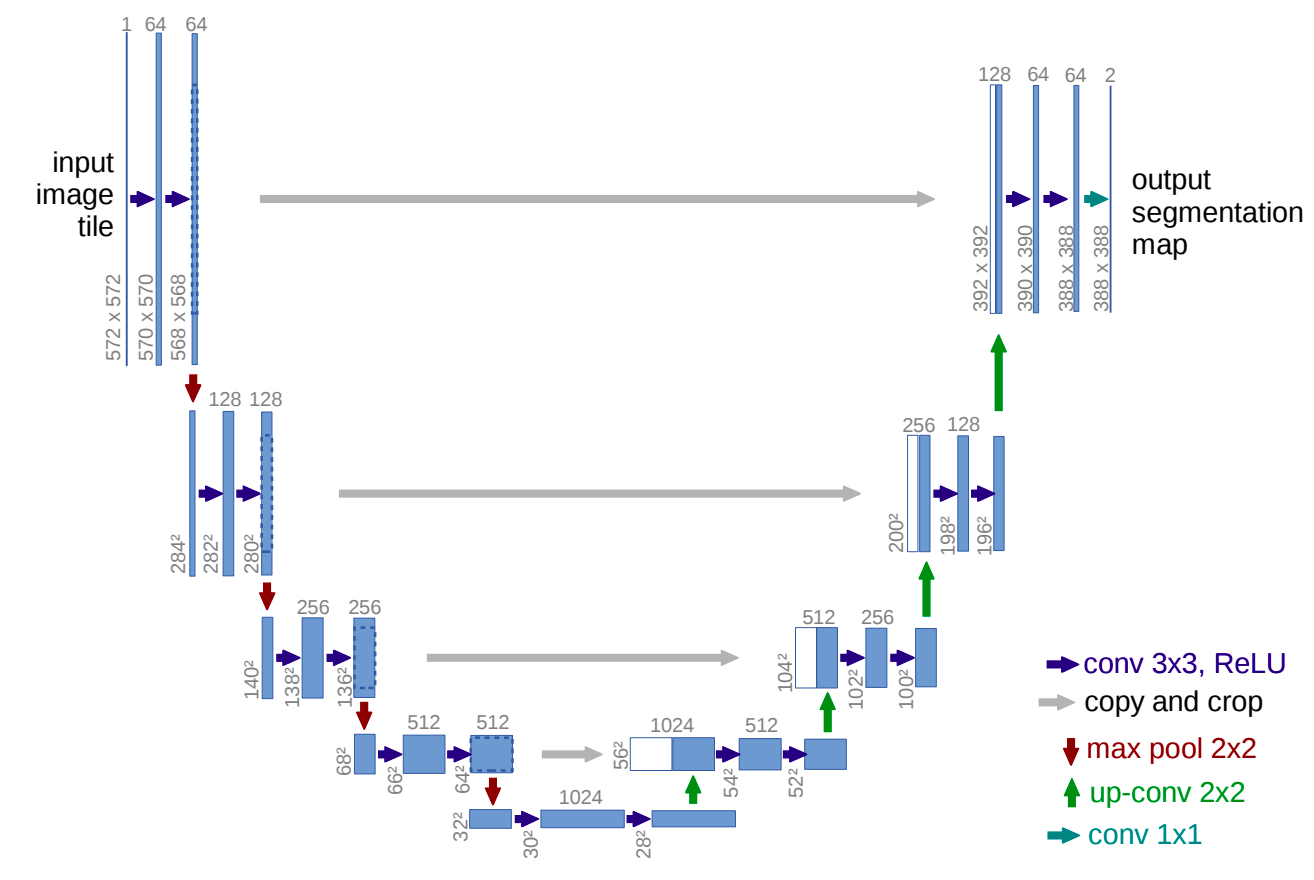
**U-Net Paper Implementation using PyTorch**

**Main Diagram**



**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.

| """ Created on Mon Jun 20 16:08:19 2021  @author: mr-siddy """  **import** torch  **import** torch.nn **as** nn   **def** **double\_conv**(in\_c, out\_c):  conv = nn.Sequential(  nn.Conv2d(in\_c, out\_c, kernel\_size=3),  nn.ReLU(inplace=**True**),  nn.Conv2d(out\_c, out\_c, kernel\_size=3),  nn.ReLU(inplace=**True**)  )  **return** conv  **def** **crop\_img**(tensor, target\_tensor):  target\_size = target\_tensor.size()[2]  tensor\_size = tensor.size()[2]  delta = tensor\_size - target\_size  delta = delta // 2  **return** tensor[:, :, delta:tensor\_size-delta, delta:tensor\_size-delta]    **class** **UNet**(nn.Module):   **def** **\_\_init\_\_**(self):  super(UNet, self).\_\_init\_\_()    self.max\_pool\_2x2 = nn.MaxPool2d(kernel\_size=2, stride=2)  self.down\_conv\_1 = double\_conv(1, 64)  self.down\_conv\_2 = double\_conv(64, 128)  self.down\_conv\_3 = double\_conv(128, 256)  self.down\_conv\_4 = double\_conv(256, 512)  self.down\_conv\_5 = double\_conv(512, 1024)    self.up\_trans\_1 = nn.ConvTranspose2d(  in\_channels=1024,  out\_channels=512,  kernel\_size=2,  stride=2)    self.up\_conv\_1 = double\_conv(1024, 512)    self.up\_trans\_2 = nn.ConvTranspose2d(  in\_channels=512,  out\_channels=256,  kernel\_size=2,  stride=2)    self.up\_conv\_2 = double\_conv(512, 256)    self.up\_trans\_3 = nn.ConvTranspose2d(  in\_channels=256,  out\_channels=128,  kernel\_size=2,  stride=2)    self.up\_conv\_3 = double\_conv(256, 128)    self.up\_trans\_4 = nn.ConvTranspose2d(  in\_channels=128,  out\_channels=64,  kernel\_size=2,  stride=2)    self.up\_conv\_4 = double\_conv(128, 64)    self.out = nn.Conv2d(  in\_channels=64,  out\_channels=2,  kernel\_size=1)      **def** **forward**(self, image):  *# bs, channel = c, height = h, width = w*    *# encoder*  x1 = self.down\_conv\_1(image) *# copy and crop*  x2 = self.max\_pool\_2x2(x1)  x3 = self.down\_conv\_2(x2) *# copy and crop*  x4 = self.max\_pool\_2x2(x3)  x5 = self.down\_conv\_3(x4) *# copy and crop*  x6 = self.max\_pool\_2x2(x5)  x7 = self.down\_conv\_4(x6) *# copy and crop*  x8 = self.max\_pool\_2x2(x7)  x9 = self.down\_conv\_5(x8)    *# decoder*  x = self.up\_trans\_1(x9)   y = crop\_img(x7, x)  x = self.up\_conv\_1(torch.cat([x, y], 1))    x = self.up\_trans\_2(x)   y = crop\_img(x5, x)  x = self.up\_conv\_2(torch.cat([x, y], 1))    x = self.up\_trans\_3(x)   y = crop\_img(x3, x)  x = self.up\_conv\_3(torch.cat([x, y], 1))    x = self.up\_trans\_4(x)   y = crop\_img(x1, x)  x = self.up\_conv\_4(torch.cat([x, y], 1))    x = self.out(x)  print(x.size())    **return** x     **if** \_\_name\_\_ == '\_\_main\_\_':  image = torch.rand((1, 1, 572, 572))  model = UNet()  print(model(image)) |
| --- |