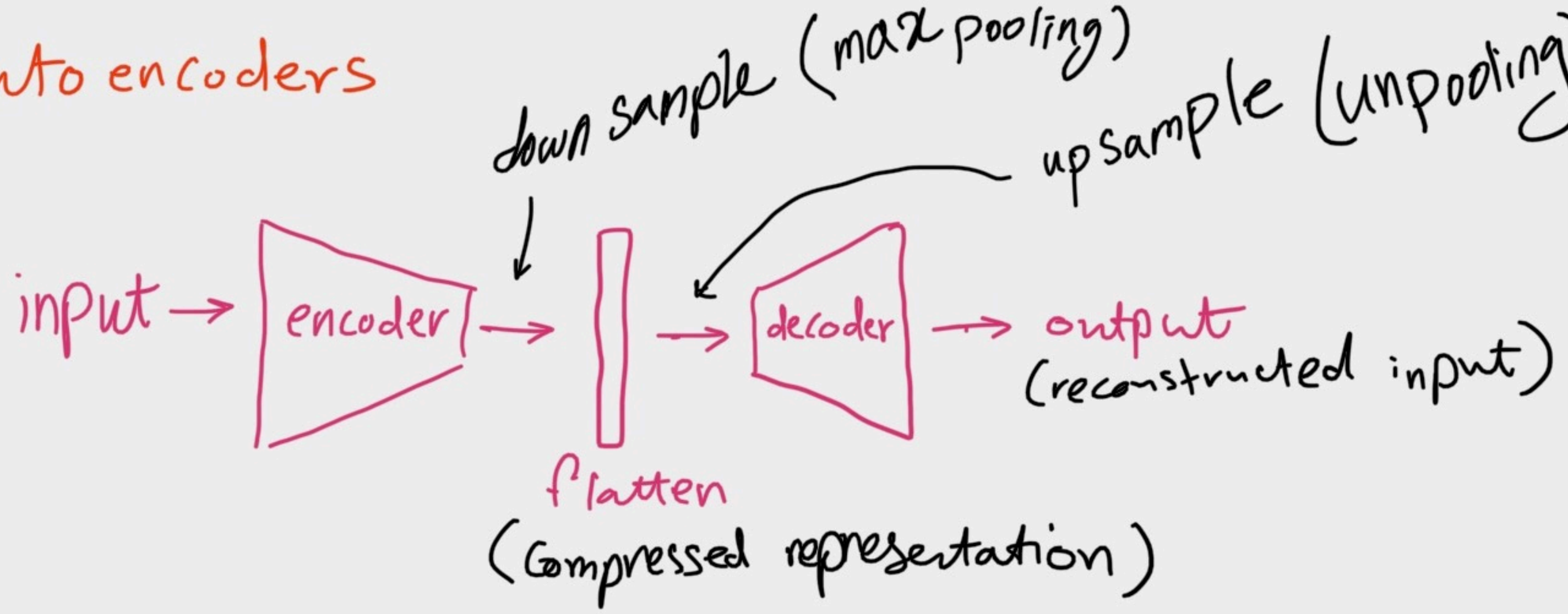
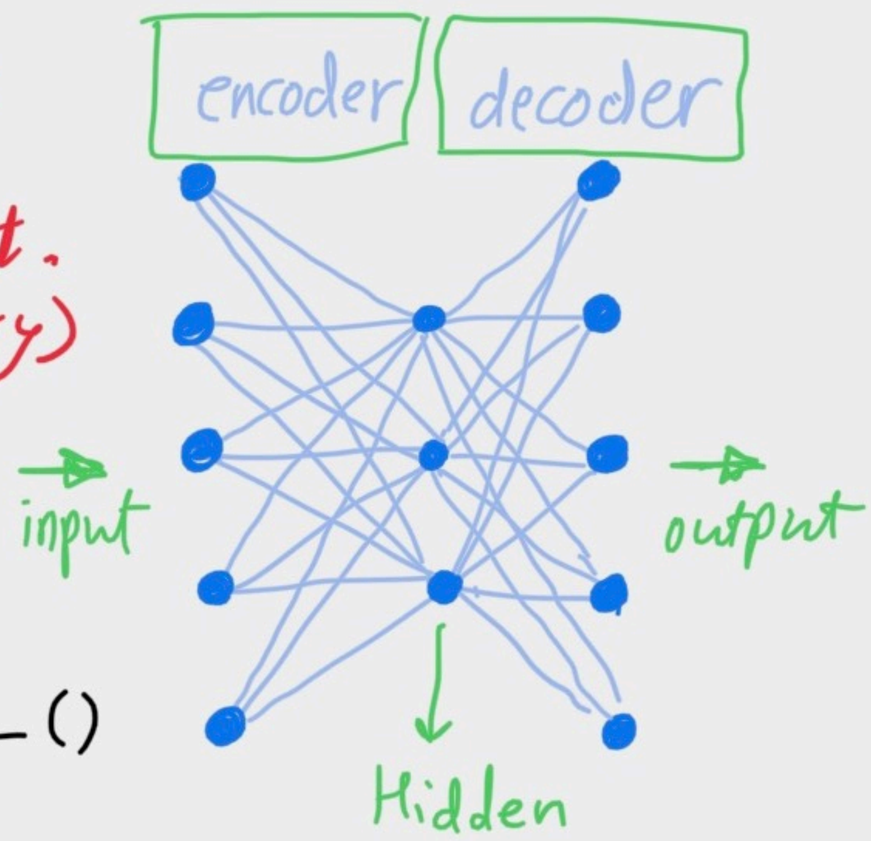
Auto encoders



In PyTorch: option 1: MLP

* the output image and input image are kinda diffent. (blurry)

data handling as before




```
class AutoEncoder(nn.Module):
    def __init__(self, encoding_dim):
        super(AutoEncoder, self).__init__()
        # encoder:
        self.fc1 = nn.Linear(28*28, encoding_dim)
        # decoder
        self.fc2 = nn.Linear(encoding_dim, 28*28)

def forward(self, x):
    x = F.relu(self.fc1(x))
    x = F.sigmoid(self.fc2(x))
    return x
```

encoding_dim = 32

model = AutoEncoder(encoding_dim)

★ criterion = nn.MSELoss() → good for pixels comparison not probabilities

optimizer = optim.Adam(model.parameters(), lr = 0.001)

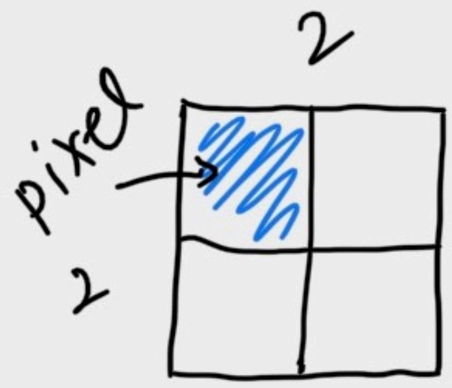
> train loop as before; but we are not interested in labels here. We just want to compare the input image with reconstructed image.

↳ at each step: loss = criterion(outputs, images)

this tells us how good of a reconstruction our model does

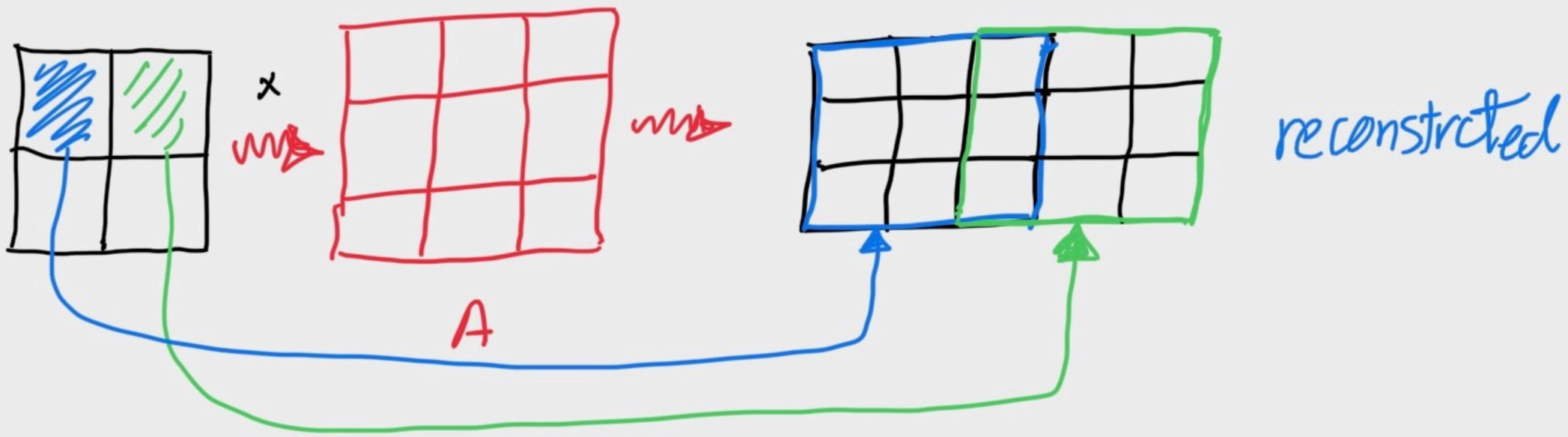
option 2: Conv. layers ★ a better solution but still has issue of "blocks" in the image.

Key point: transpose Conv. layers

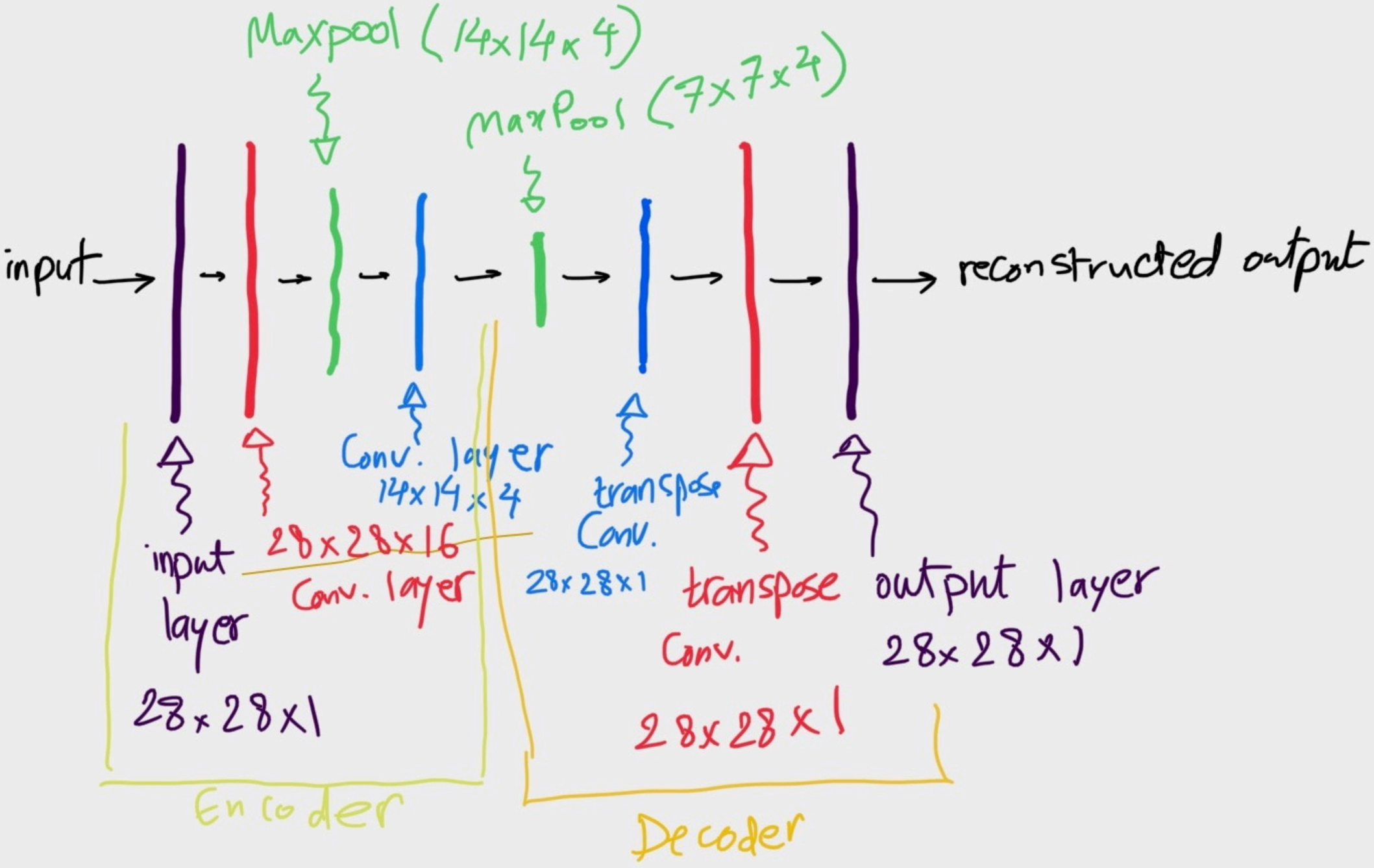


→ we take each pixel, and use a kernel for upsampling. e.g., we use a 3x3 kernel for a 2x2 image. (let's call this A)

For each pixel in the image, we multiply the value of it with each element of the kernel.



Convolutional Autoencoders



In pytorch, transpose Conv Layers are nn.ConvTranspose2d.

```
class ConvAutoencoder(nn.Module):
    def __init__(self):
        super(ConvAutoencoder, self).__init__()

        #encoder layers
        self.conv1 = nn.Conv2d(1,16, 3, padding=1)   # depth: 1→16 ; number of feature maps
        self.conv2 = nn.Conv2d(16,4,3, padding=1)    # depth: 16→4
        self.pool = nn.MaxPool2d(2,2)                # size, stride

        #decoder layers
        self.t_conv1 = nn.ConvTranspose2d(4, 16, 2, stride=2)
        self.t_conv2 = nn.ConvTranspose2d(16,1, 2, stride=2)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)

        x = F.relu(self.t_conv1(x))
        x = F.Sigmoid(self.t_conv2(x))
        return x
```

```
model = ConvAutoencoder()
Criterion = nn.MSELoss()
```

most of the train loop is as before.
just note following statements:

```
outputs = model(images)
loss = Criterion(outputs, images)
```

i.e., we want outputs and images be ideally identical.

option 3: upsampling

smoother and more similar images

We change option 2's decoder by adding a few layers. (upsampling)

* idea: replace a transpose Conv. layer with with a conv. layer + an upsample layer.

The change with option 2's code is just in the "forward" portion of the module.

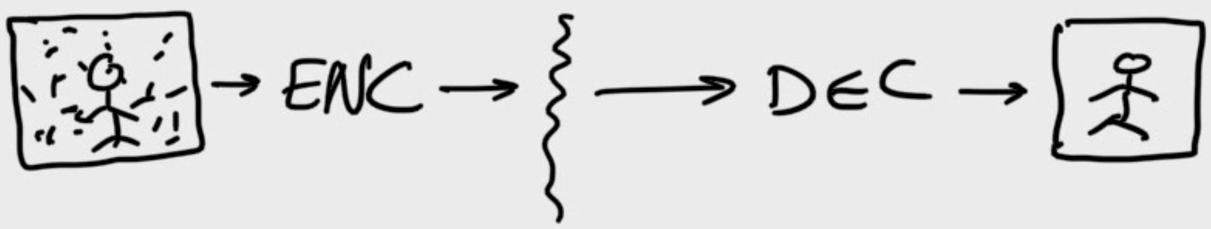
#encoder

⋮

#decoder

```
x = F.upsample(x, scale_factor=2, mode='nearest')
x = F.relu(self.conv4(x))
x = F.upsample(x, scale_factor=2, mode='nearest')
x = F.sigmoid(self.conv5(x))
```

Denoising Autoencoders



In PyTorch

```
class ConvDenoiser(nn.Module):
    def __init__(self):
        super(ConvDenoiser, self).__init__()
```

Encoder

```
self.conv1 = nn.Conv2d(1, 32, 3, padding=1)
self.conv2 = nn.Conv2d(32, 16, 3, padding=1)
self.conv3 = nn.Conv2d(16, 8, 3, padding=1)
self.pool = nn.MaxPool2d(2,2)
```

Decoder

```
self.t_conv1 = nn.ConvTranspose2d(8, 8, 3, stride=2)
self.t_conv2 = nn.ConvTranspose2d(8, 16, 2, stride=2)
self.t_conv3 = nn.ConvTranspose2d(16, 32, 2, stride=2)
self.conv_out = nn.Conv2d(32, 1, 3, padding=1)
```

```
def forward(self, x):
    x = F.relu(self.conv1(x))
    x = self.pool(x)
    x = self.relu(self.conv2(x))
    x = self.pool(x)
    x = F.relu(self.conv3(x))
    x = self.pool(x)
    x = F.relu(self.t_conv1(x))
    x = F.relu(self.t_conv2(x))
    x = F.relu(self.t_conv3(x))
    x = F.sigmoid(self.conv_out(x))
    return x
```