Auto encoders

```
input -> encoder -> [downsample (max pooling)] -> decoder -> output (reconstructed input)
```

```
flatten (compressed representation)
```

In PyTorch: option 1: MLP

```
data handling as before
```

```
class AutoEncoder (nn.Module):
    def __init__ (self, encoding_dim):
        super (AutoEncoder, self).__init__ ()
        # encoder:
        self.fc1 = nn.Linear (28x28, encoding_dim)
        # decoder
        self.fc2 = nn.Linear (encoding_dim, 28x28)

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 () \rightarrow 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.

\rightarrow at each step: \quad 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

pixel 2
\[ \begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array} \]
\rightarrow 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

input → 28x28x16 → Conv. layer 28x28x4 → Maxpool (7x7x4) → transposed Conv. layer 28x28x1 → 28x28x1

Encoder

Decoder

reconstructed output
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)
        self.conv2 = nn.Conv2d(16, 4, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)

        # 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 = Conv Autoencoder()
criterion = nn.MSELoss()

Most of the train loop is as before. Just note the following statements:

```
outputs = model(images)
loss = criterion(outputs, images)
```
i.e., we want outputs and images to 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 a Conv. layer + an upsampling 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 Auto-encoders

\[
\begin{align*}
\text{Encoder} & : \quad \text{self. Conv1} = \text{nn.Conv2d}(1, 32, 3, \text{padding}=1) \\
& \quad \text{self. Conv2} = \text{nn.Conv2d}(32, 16, 3, \text{padding}=1) \\
& \quad \text{self. Conv3} = \text{nn.Conv2d}(16, 8, 3, \text{padding}=1) \\
& \quad \text{self. Pool} = \text{nn.MaxPool2d}(2, 2) \\
\text{Decoder} & : \quad \text{self. t-Conv1} = \text{nn.ConvTranspose2d}(8, 8, 3, \text{stride}=2) \\
& \quad \text{self. t-Conv2} = \text{nn.ConvTranspose2d}(8, 16, 2, \text{stride}=2) \\
& \quad \text{self. t-Conv3} = \text{nn.ConvTranspose2d}(16, 32, 2, \text{stride}=2) \\
& \quad \text{self. Conv_out} = \text{nn.Conv2d}(32, 1, 3, \text{padding}=1)
\end{align*}
\]

In PyTorch

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

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