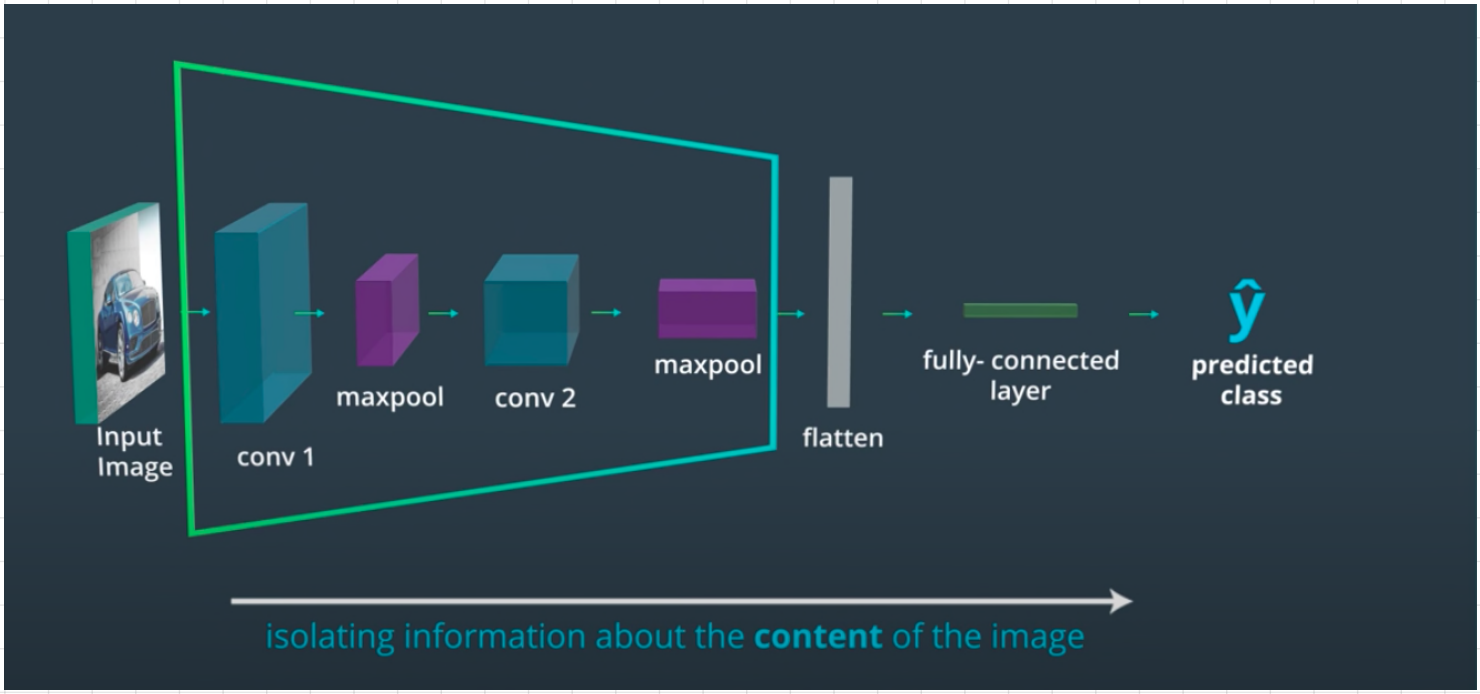
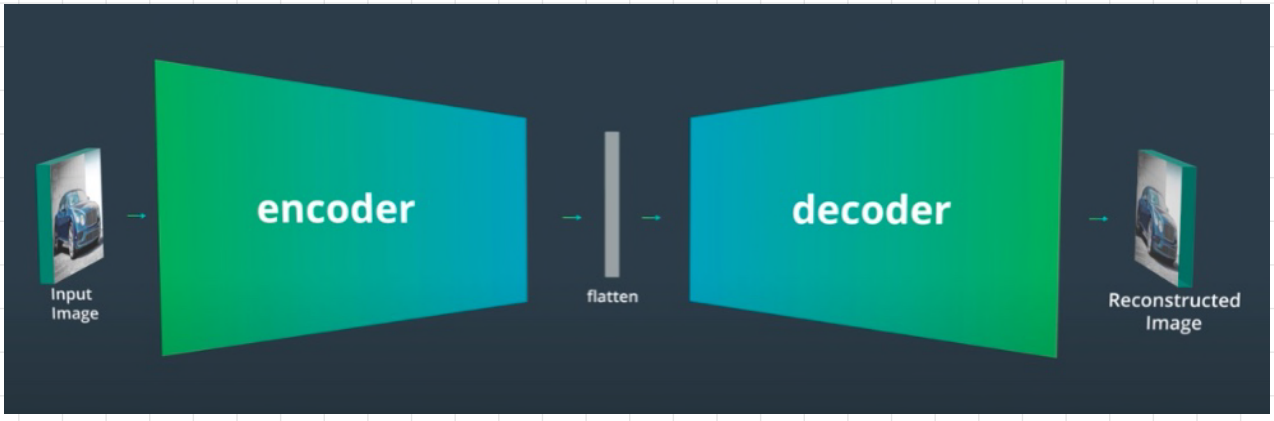
Autoencoder

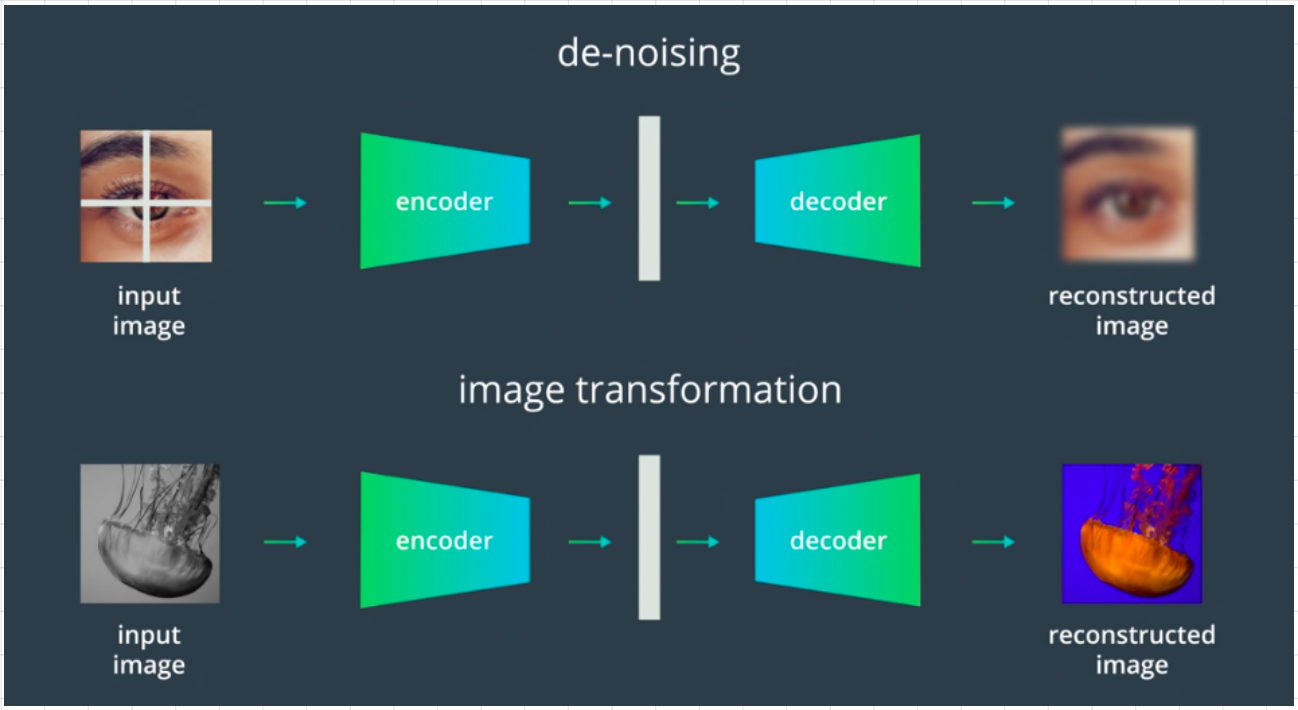
CNN Review

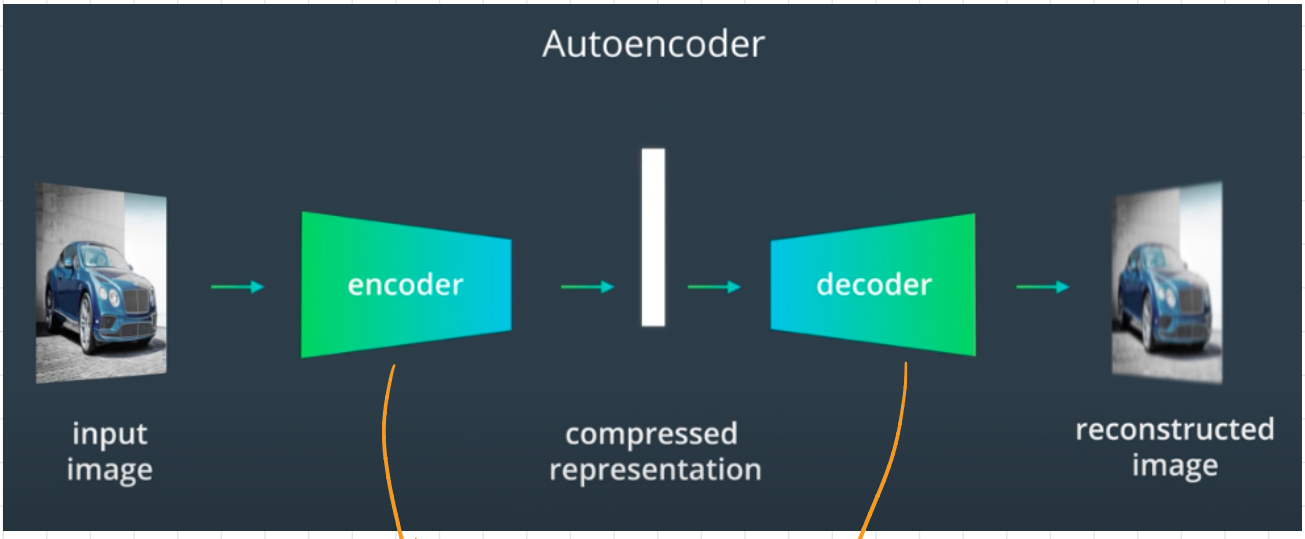


- Disgard spatial info
- Isolate high level info about content
- Aka data compression: image compress feature vector



- Autoencoder compresses input, ↓ dimensionality
  - Read faster
- Compresses but retain content!





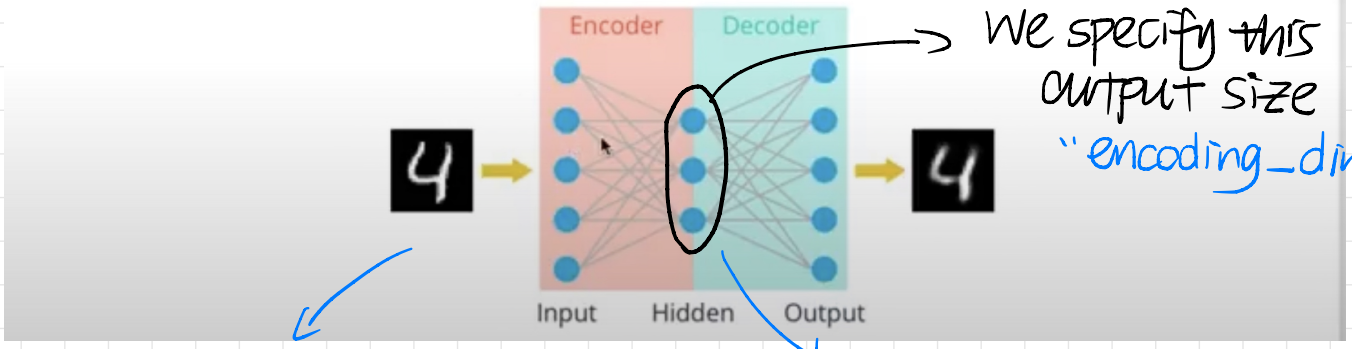
Built w/ Neural Networks

Minimize the difference bt. input & output!

Implementation (with MLP)

We'll train an autoencoder with these images by flattening them into 784 length vectors. The images from this dataset are already normalized such that the values are between 0 and 1. Let's start by building a simple autoencoder. The encoder and decoder should be made of **one linear layer**. The units that connect the encoder and decoder will be the *compressed representation*.

Since the images are normalized between 0 and 1, we need to use a **sigmoid activation on the output layer** to get values that match this input value range.



We specify this output size "encoding_dim"

flatten input image into column vector

1 linear layer

fed into decoder

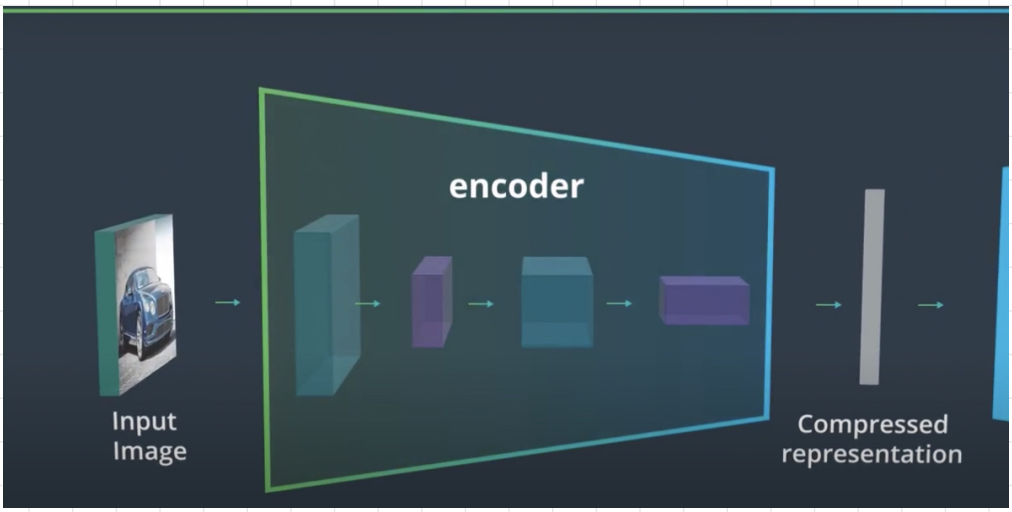
increase dim. output vector length of 784. Reshape.

Training

- Loss: use MSE instead of CrossEntropy
  - Comparing pixel values bt. input & output so it resembles a regression task, not a classification.
- Not interested in labels, just want to compare input & compressed images.

Implementation (with CNN)

- Advantage: preserve content information

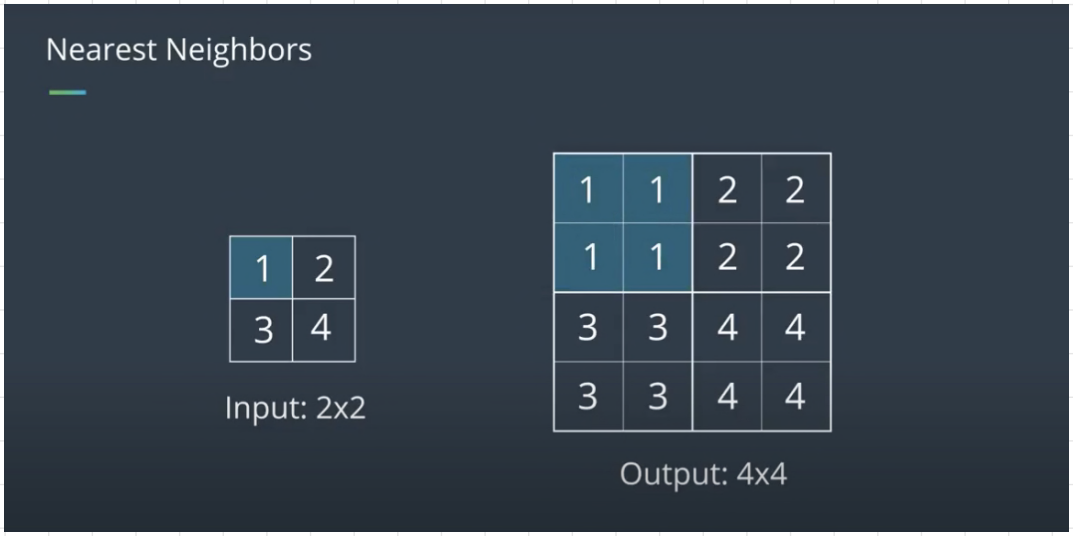


→ Spatial dims ↓

For decode, we want to increase spatial dim.

via "un-pooling" to up-sampling

Crude.

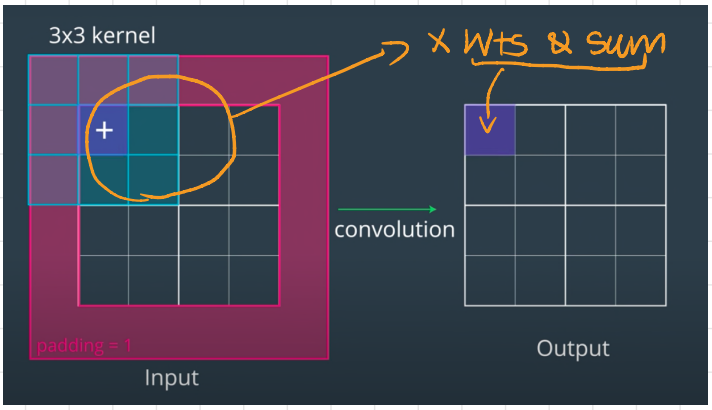


Instead, we can "learn" how to best up-sample.

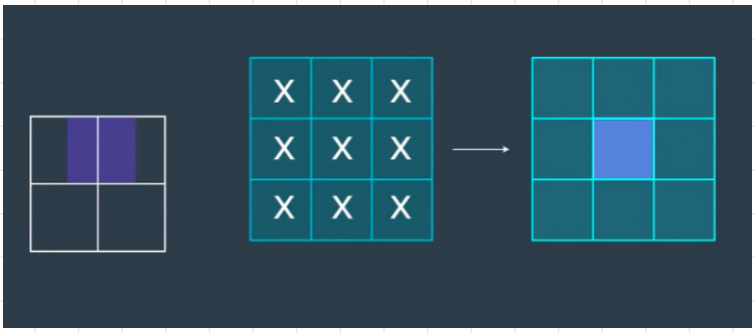
Via Transpose CNN!

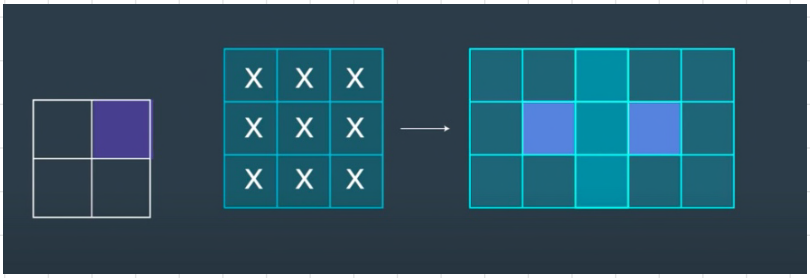
Transpose Convolution

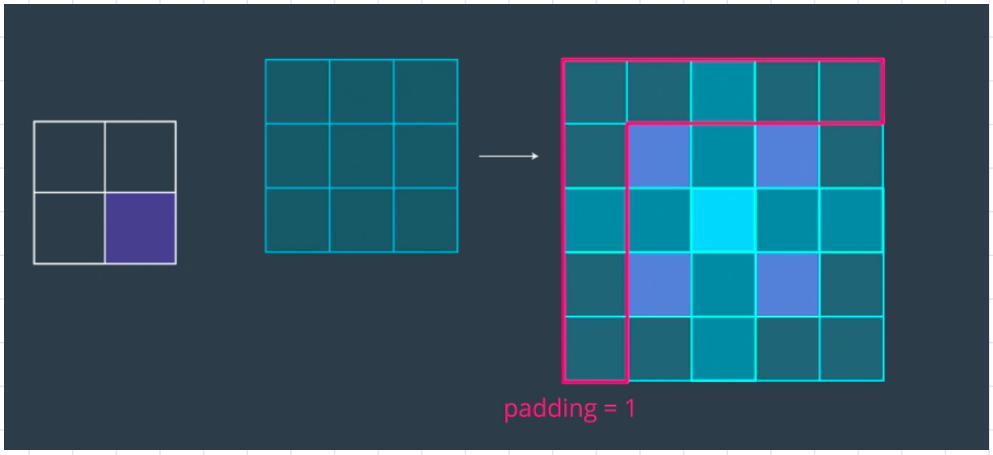
Conv. review

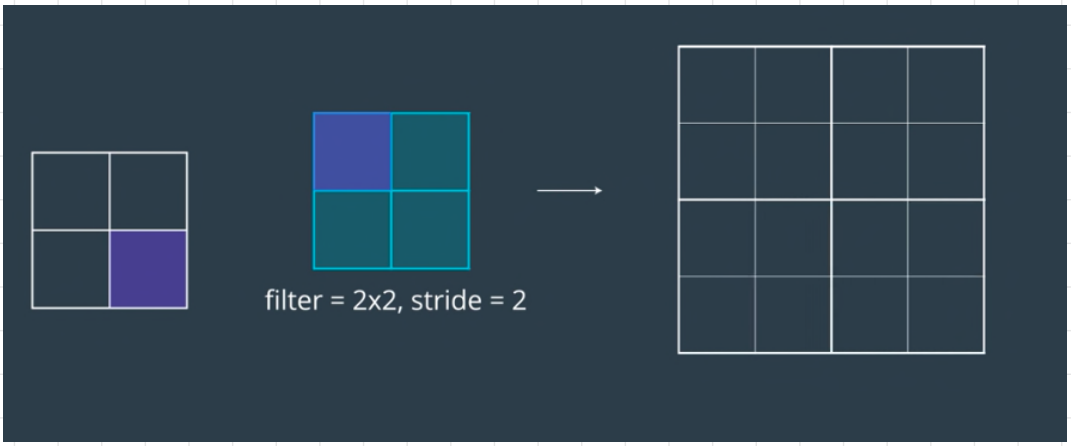


The wts here are also learned, like a regular CNN

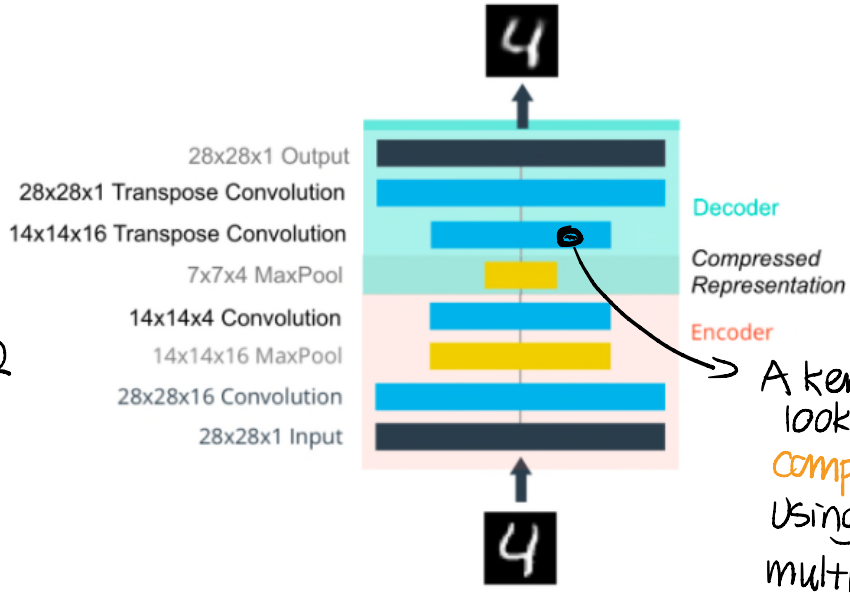








Implementation (Method #1)

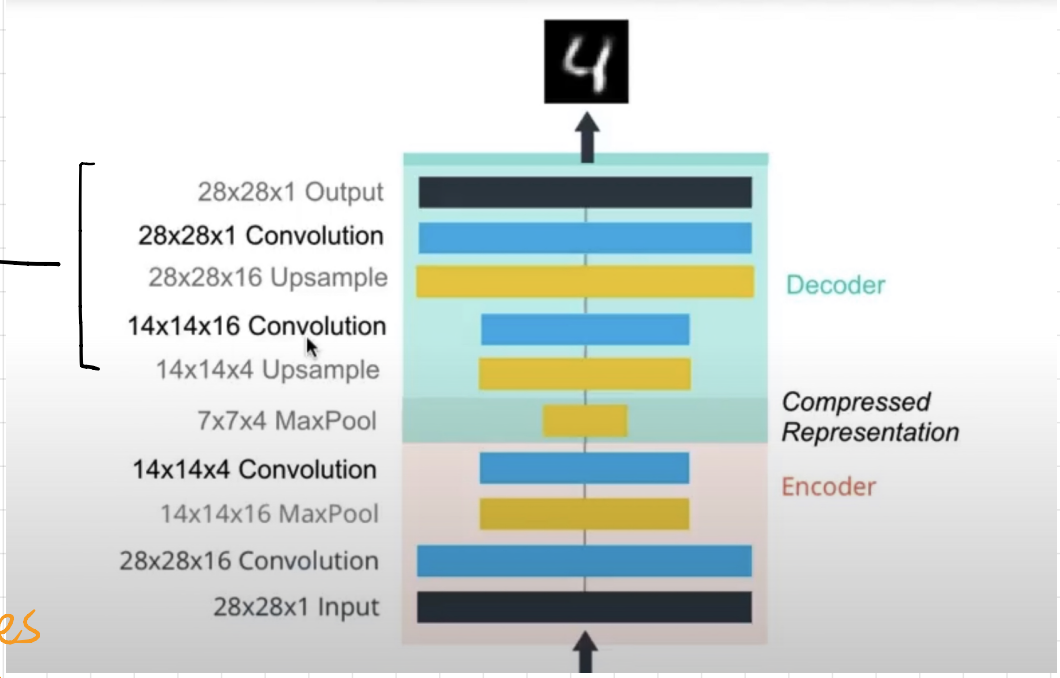


In this case, we use kernel = 2 stride = 2

A kernel in this layer looking at the compressed representation using a larger stride to multiply existing pixel values by some wts & ↑ spatial size of output!

Method #2

Replace conv. T



Convolution ↑ dimension

Create filtered images to help analyze input images

Nearest neighbor interpolation

$$\frac{(W - in - f + 2p)}{s} + 1$$

$i \equiv$ width

$k \equiv$ kernel size