Autoencoders

Unsupervised Learning

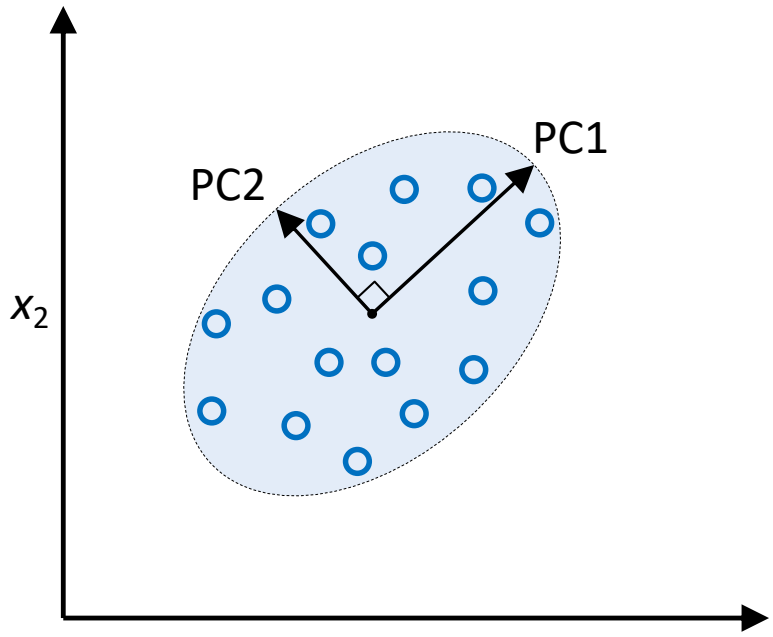
Working with datasets *without* considering the *target* variable

Some Applications and Goals:

- Finding hidden structures in data
- Data compression
- Clustering
- Retrieving similar objects
- Exploratory Data Analysis
- Generating new examples

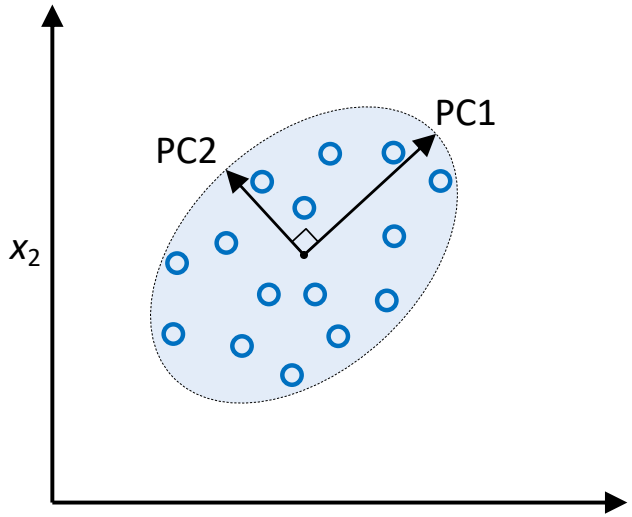
Principal Component Analysis (PCA)

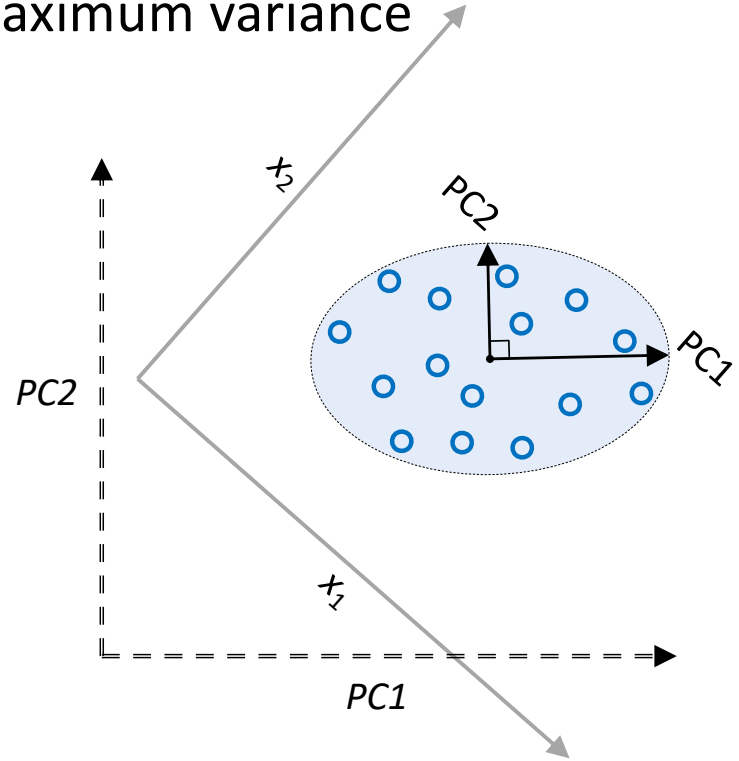
Find directions of maximum variance



Principal Component Analysis (PCA)

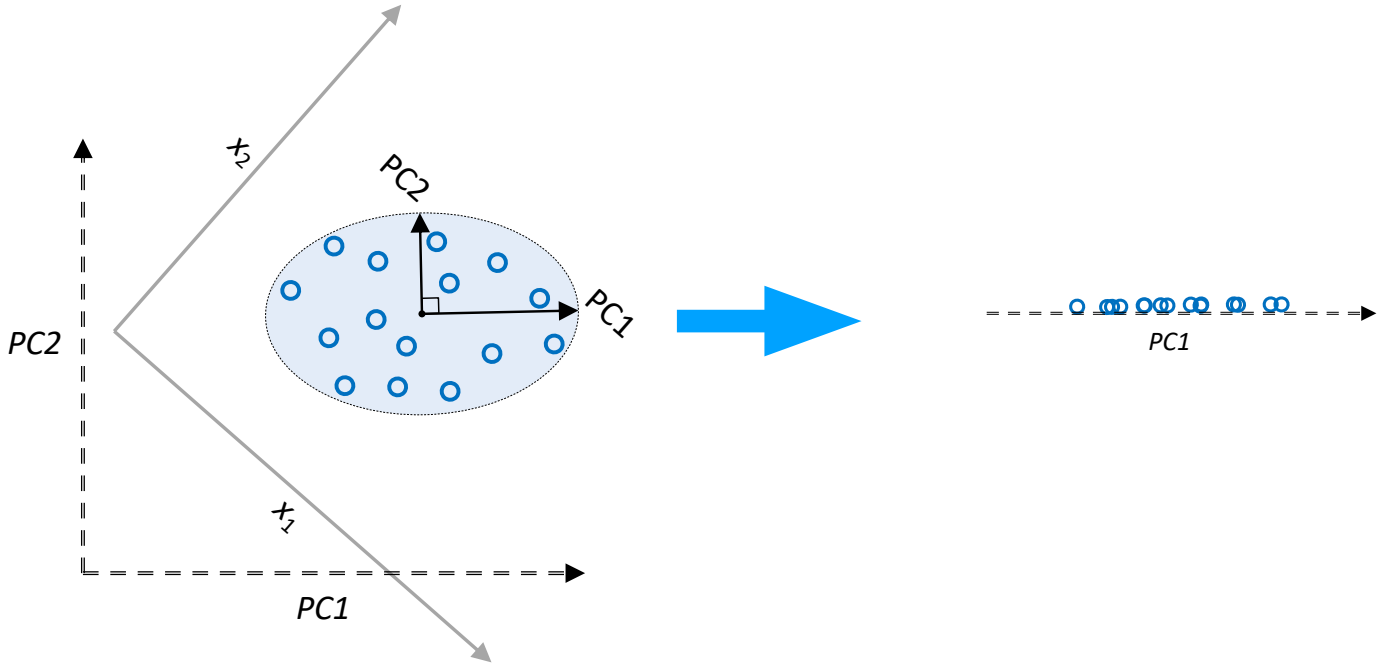
Transform features onto directions of maximum variance



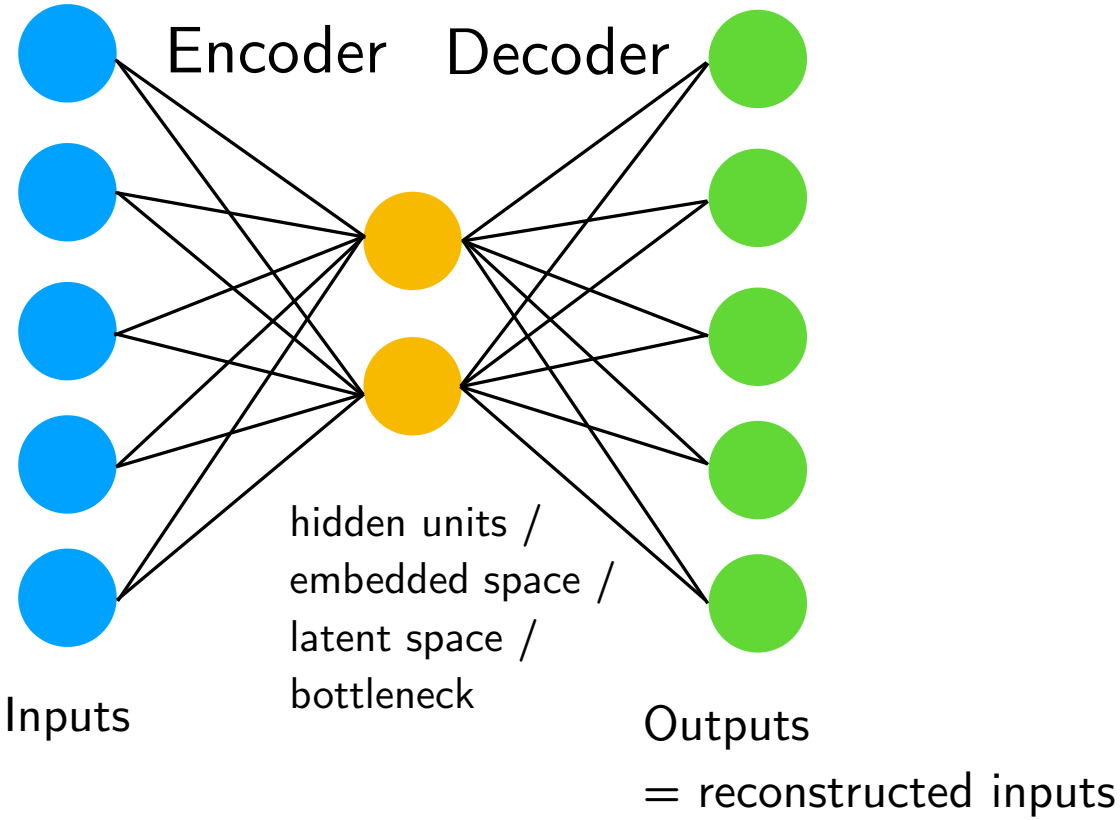


Principal Component Analysis (PCA)

Usually consider a subset of vectors of most variance (dimensionality reduction)

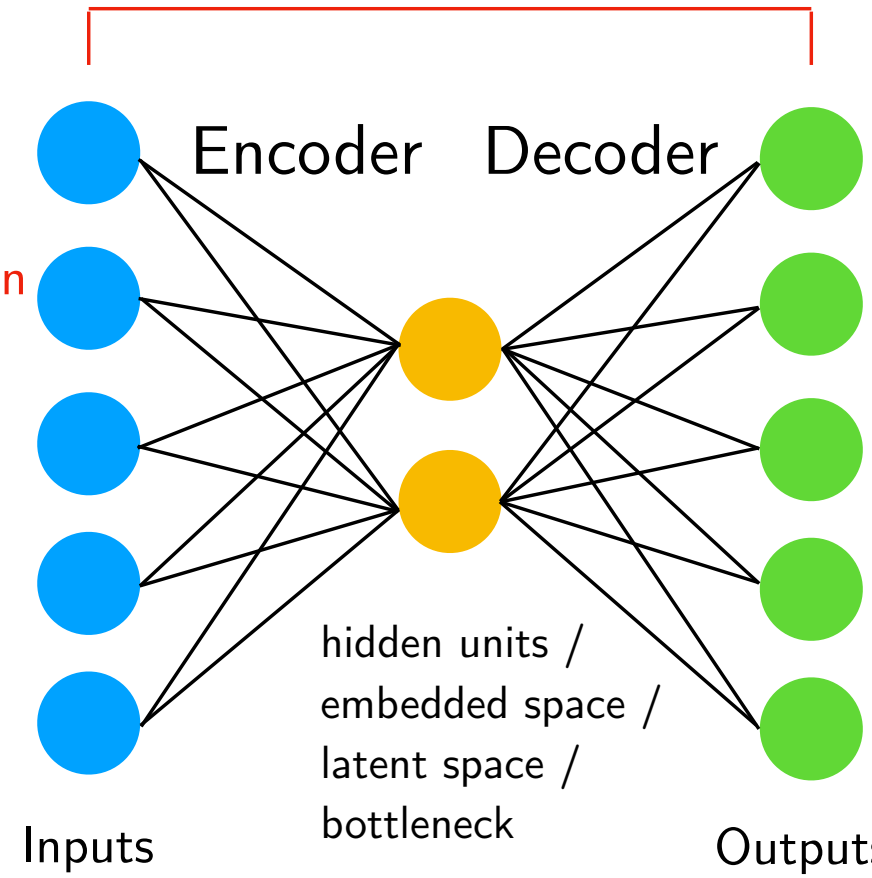


A Basic Fully-Connected (Multilayer-Perceptron) Autoencoder



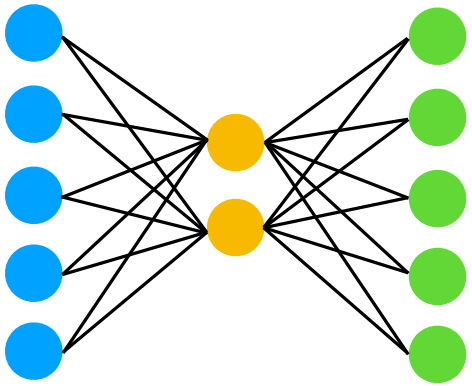
Autoencoders

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_2^2 = \sum_i (x_i - x_i')^2$$



If we don't use non-linear activation functions and minimize the MSE, this is very similar to PCA

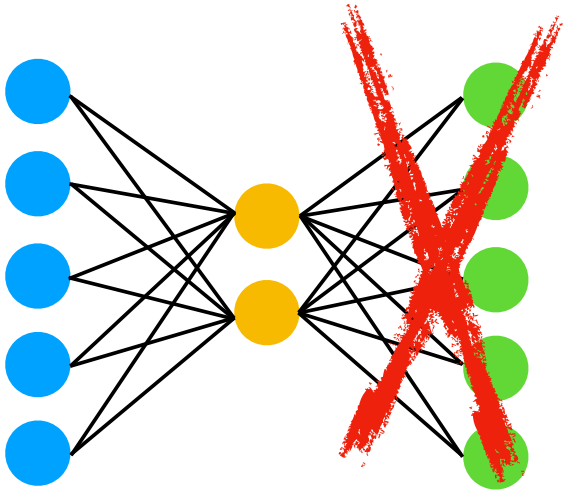
However, the latent dimensions will not necessarily be orthogonal and will have ~ same variance



Question:

If we can achieve the same with PCA, which is essentially a kind of matrix factorization that is more efficient than Backprop + SGD, why bother with autoencoders?

After training, disregard this part

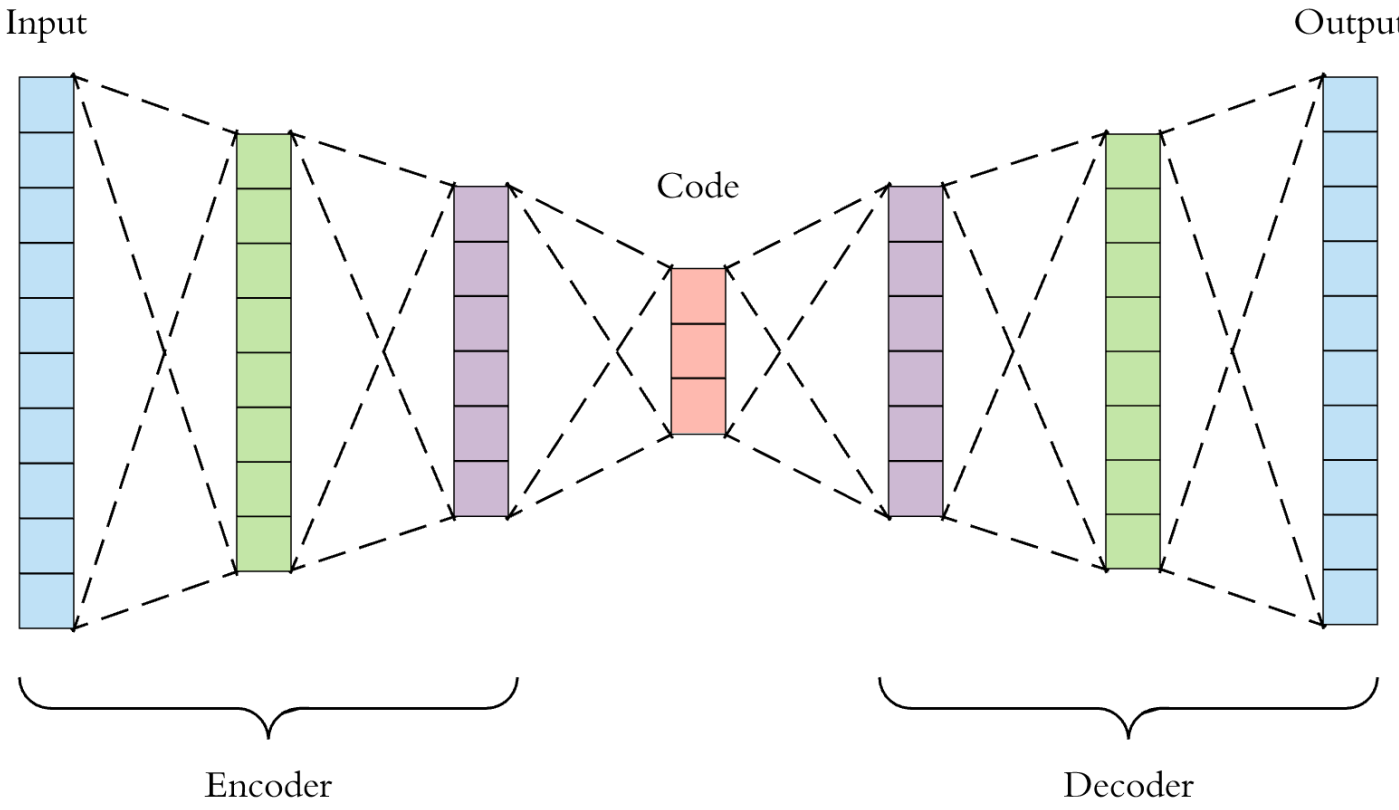


Use embedding as input to classic machine learning methods (SVM, KNN, Random Forest, ...

Or, similar to transfer learning, train autoencoder on large image dataset, then fine tune encoder part on your own, smaller datset and/or provide your own output (classification) layer

Latent space can also be used for visualization (EDA, clustering), but there are better methods for that

Autoencoder



- Encoder: compresses the input data into latent space
- Code: the compressed input
- Decoder: reconstructing input from the latent code

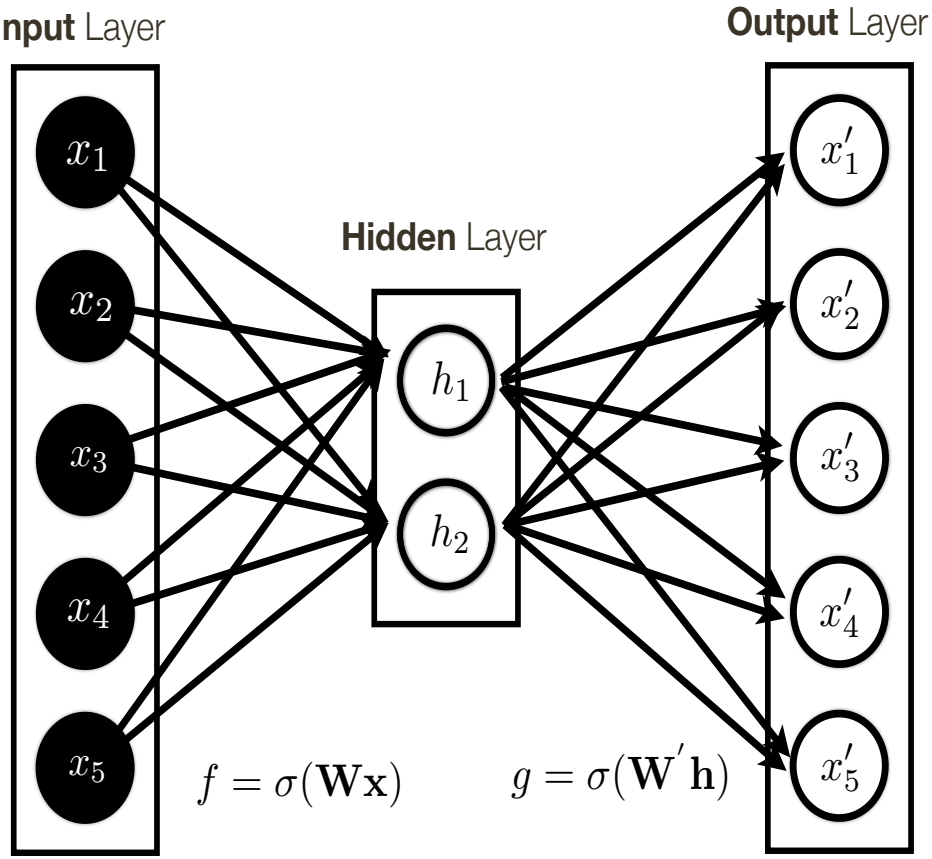
Autoencoders with tied-weights

- Using untied weights it would look like:

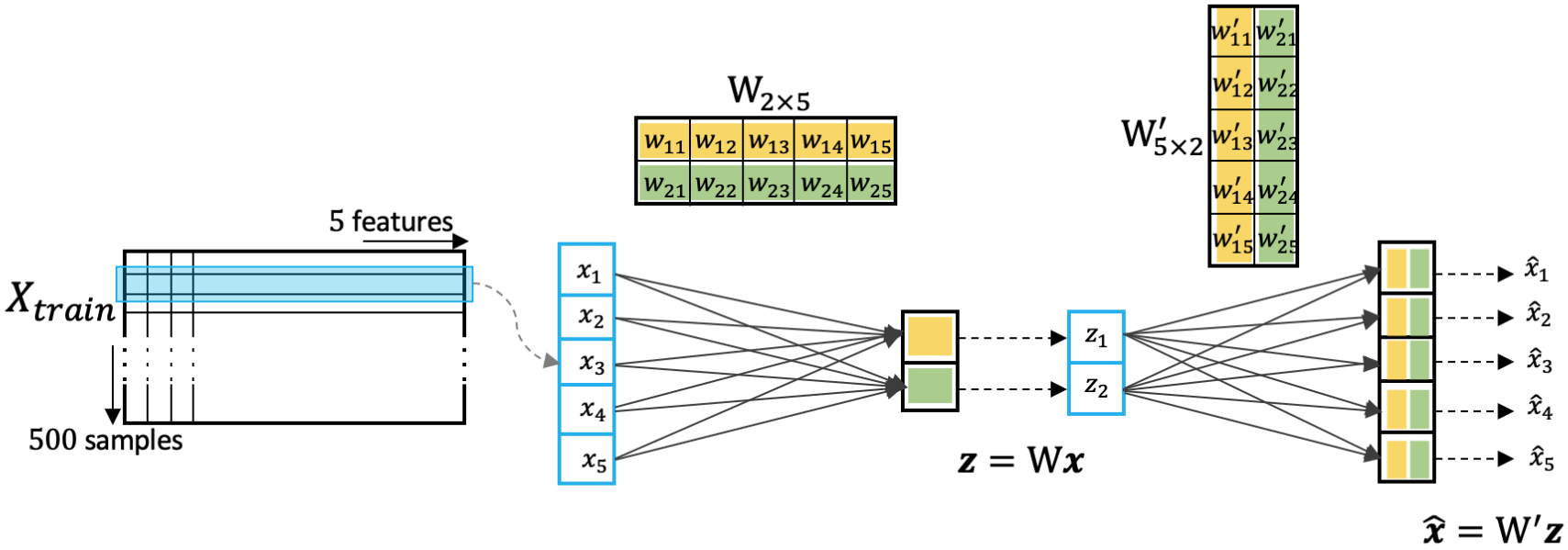
  $f(x)=\sigma_2(\mathbf{b_2}+\mathbf{W_2}*\sigma_1(\mathbf{b_1}+\mathbf{W_1}*x))$

- Using tied weights it would look like:

  $f(x)=\sigma_2(\mathbf{b_2}+\mathbf{W_1}^{T}*\sigma_1(\mathbf{b_1}+\mathbf{W_1}*x))$



Autoencoders with constraints



Required Constraints.

1. Tied weights, $W' = W^T$.
2. Orthogonal weights, $W^T W = I$.
3. Uncorrelated Encodings, $cor(z_i, z_j) = 0, \text{if } i \neq j$.
4. Weights are Unit Norm, $\sum_{j=1}^{p} w_{ij}^2 = 1, i = 1, \ldots, k$.

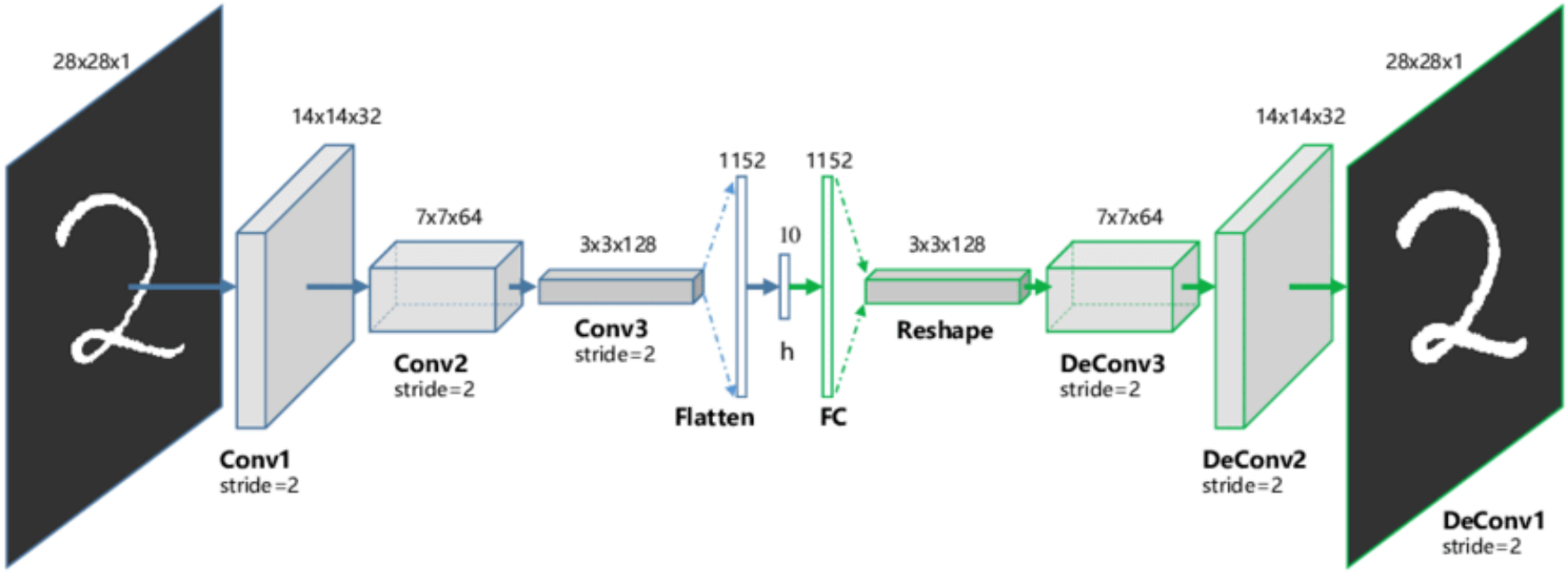
Input Layer.
Size 5 x 1.

Encoding Layer.
Input 5 x 1.
Output 2 x 1.
Weights $W_{2\times5}$

Encodings, output of the Encoding Layer. Size 2 x 1.

Decoding Layer.
Input 2 x 1.
Output 5 x 1.
Weights $W'_{5\times2}$

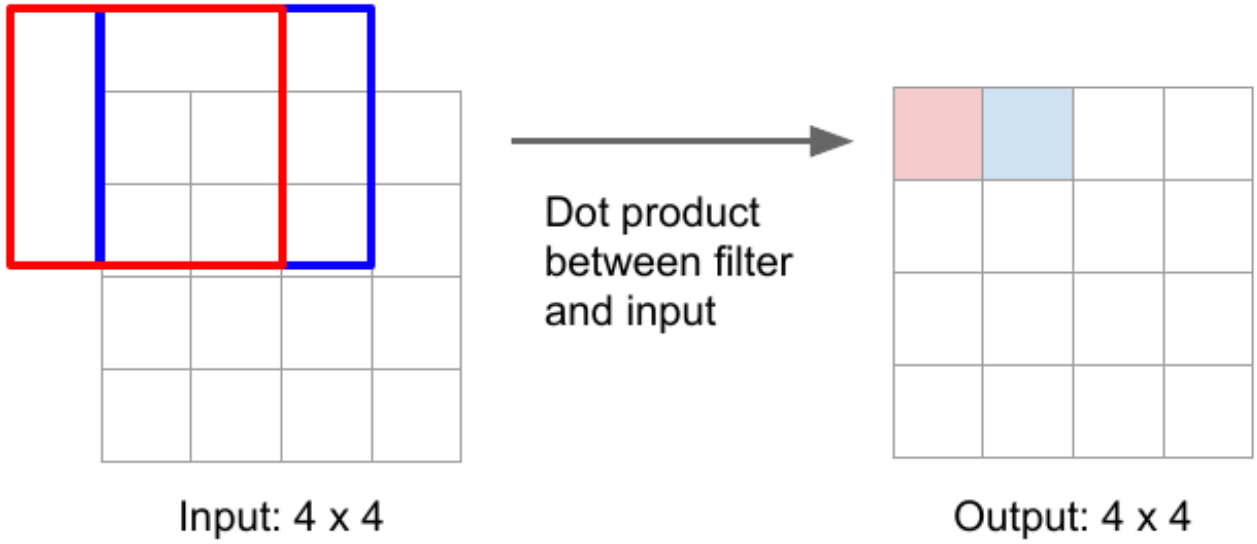
Convolutional autoencoders



Convolutional autoencoders

Downsampling

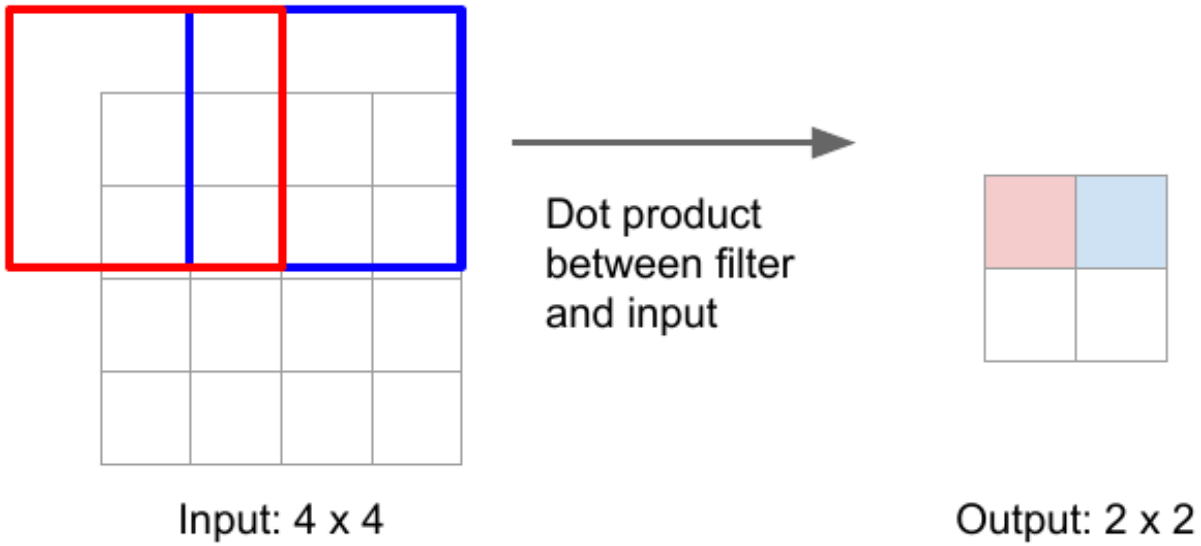
The normal convolution *(without stride)* operation gives the same size output image as input image e.g. 3x3 kernel (filter) convolution on 4x4 input image with stride 1 and padding 1 gives the same-size output.



Convolutional autoencoders

Downsampling

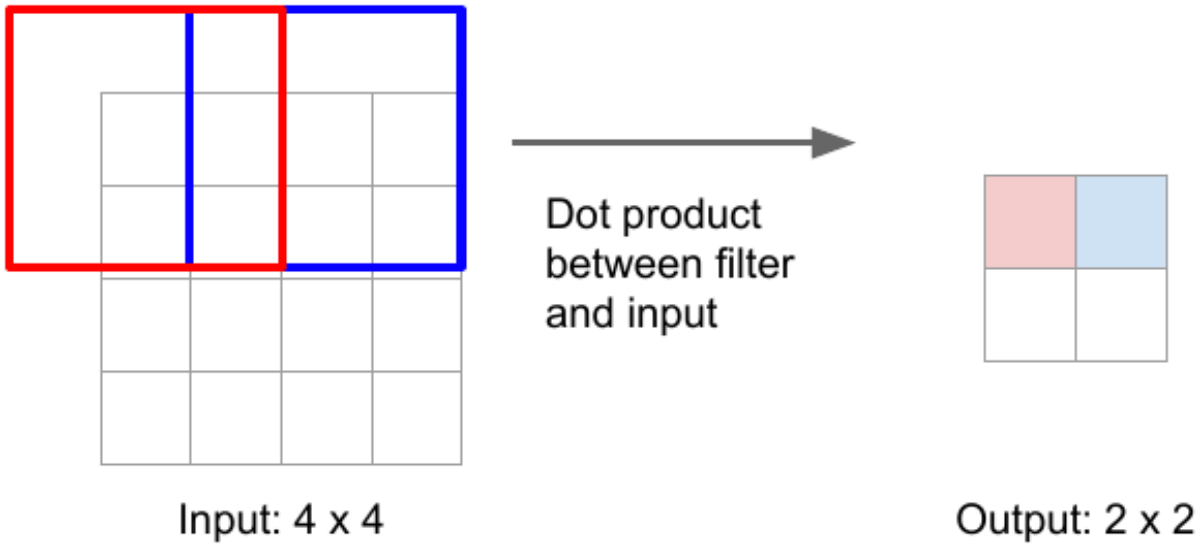
But strided convolution results in downsampling i.e. reduction in size of input image e.g. 3x3 convolution with stride 2 and padding 1 convert image of size 4x4 to 2x2.



Convolutional autoencoders

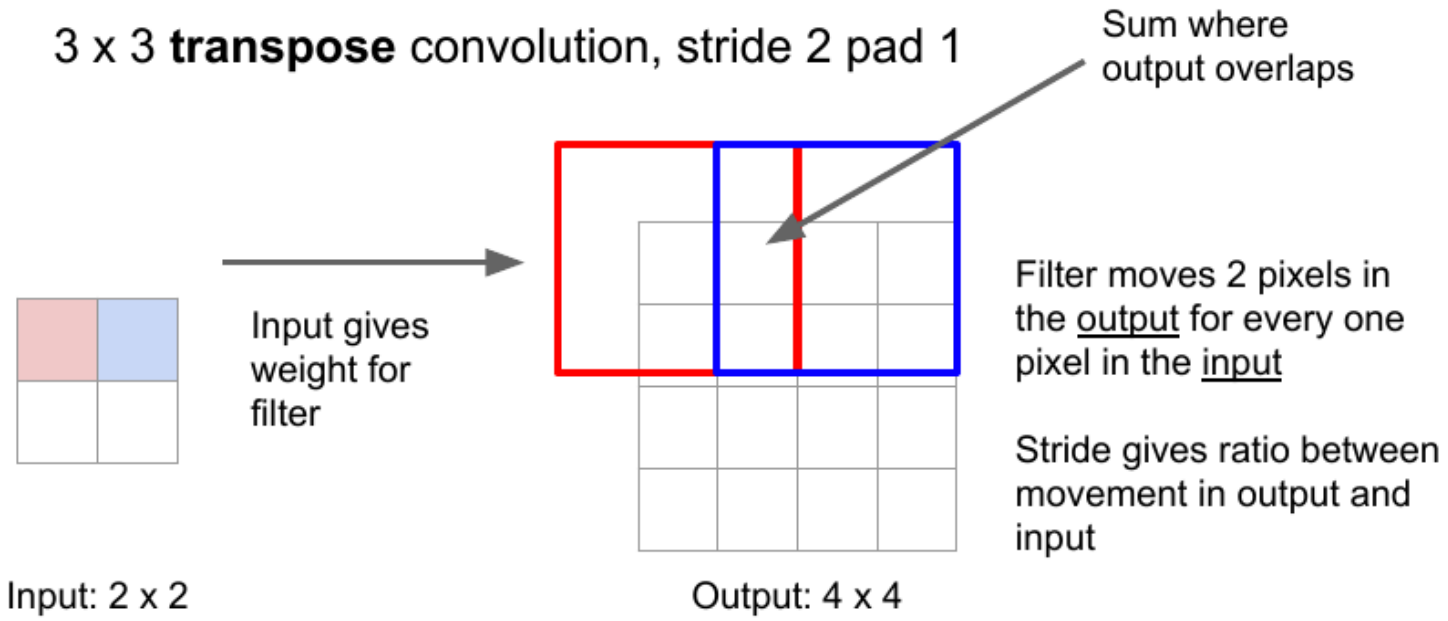
Upsampling

One of the ways to upsample the compressed image is by **Unpooling** *(the reverse of pooling)* using Nearest Neighbor or by max unpooling.



Convolutional autoencoders

Transpose convolution



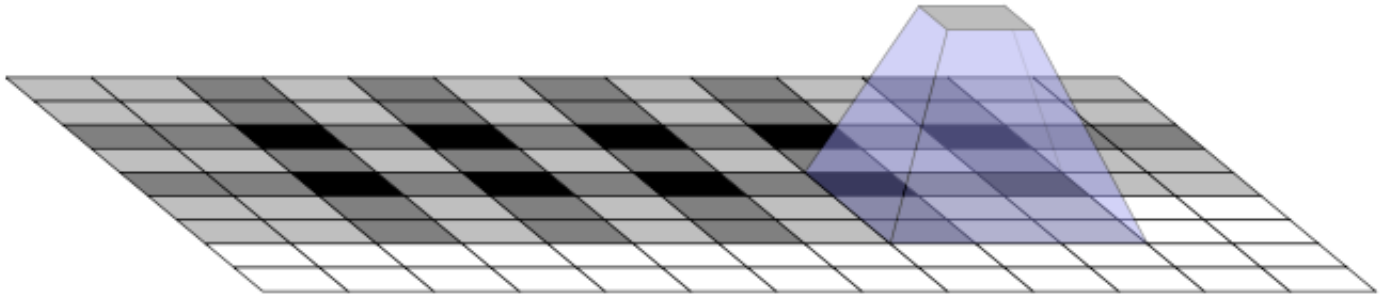
Convolutional autoencoders

Transpose convolution

The transpose convolution is reverse of the convolution operation. Here, the kernel is placed over the input image pixels. The pixel values are multiplied successively by the kernel weights to produce the upsampled image. In case of overlapping, the values are summed. The kernel weights in upsampling are learned the same way as in convolutional operation that’s why it’s also called learnable upsampling.

Convolutional autoencoders

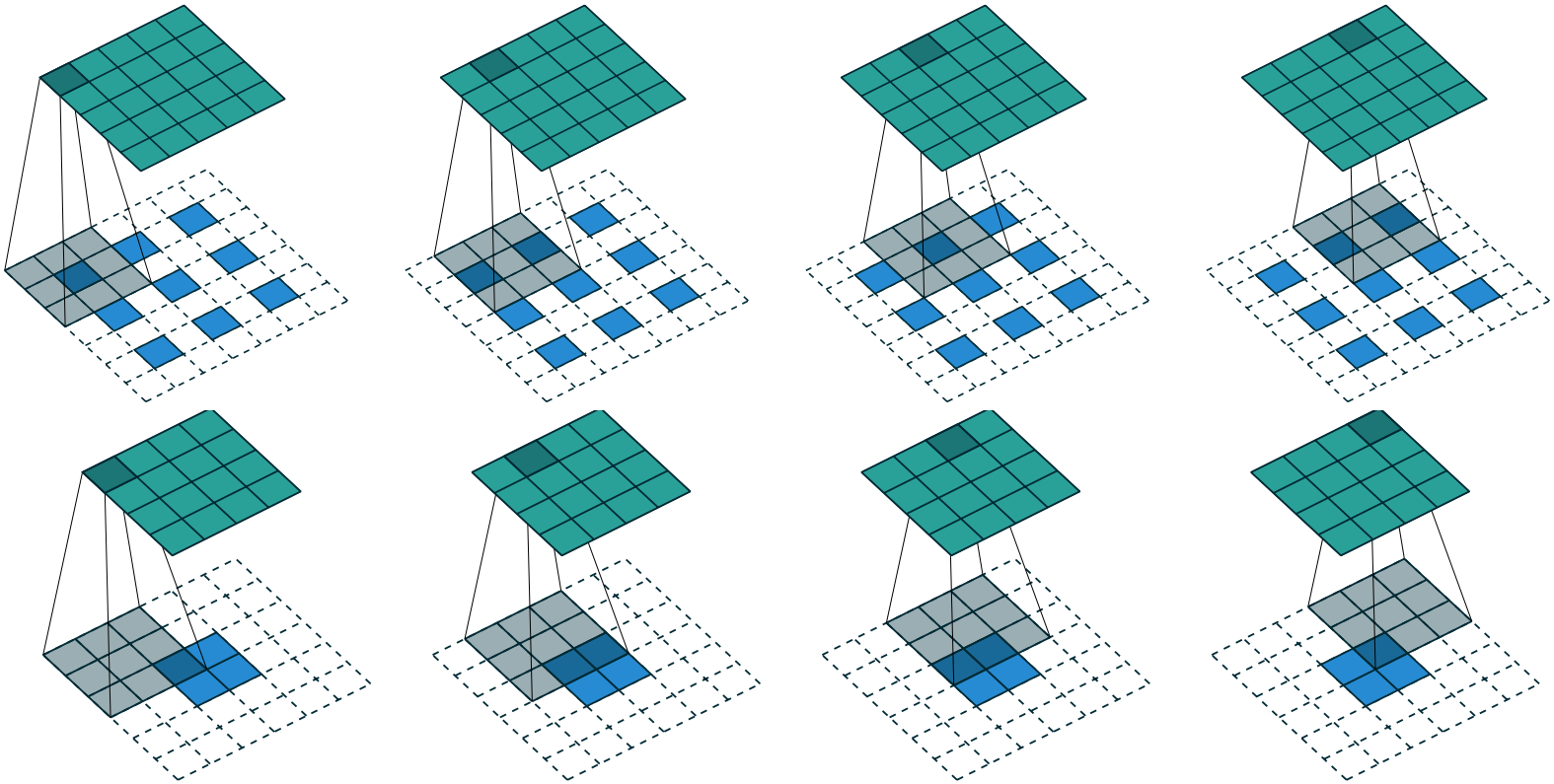
Dangers of transposed convolution



In short, it is recommended to replace transposed conv.
by upsampling (interpolation) followed by regular convolution

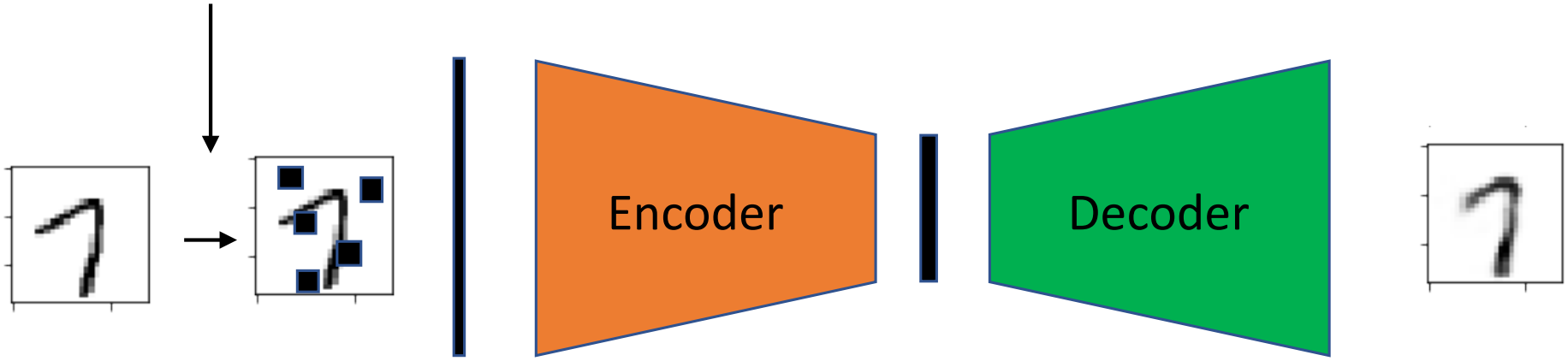
Convolutional autoencoders

Upsampling by padding



Denoising by autoencoders

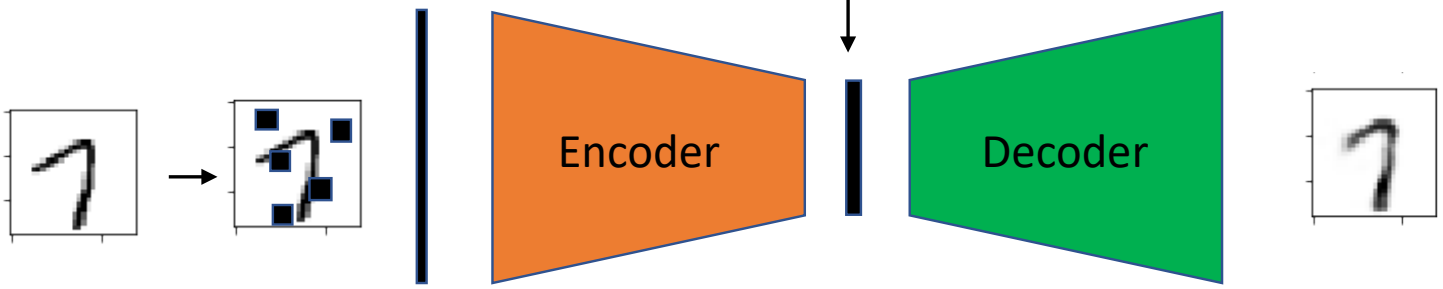
Add dropout after the input, or add noise to the input to learn to denoise images



Sparse autoencoders

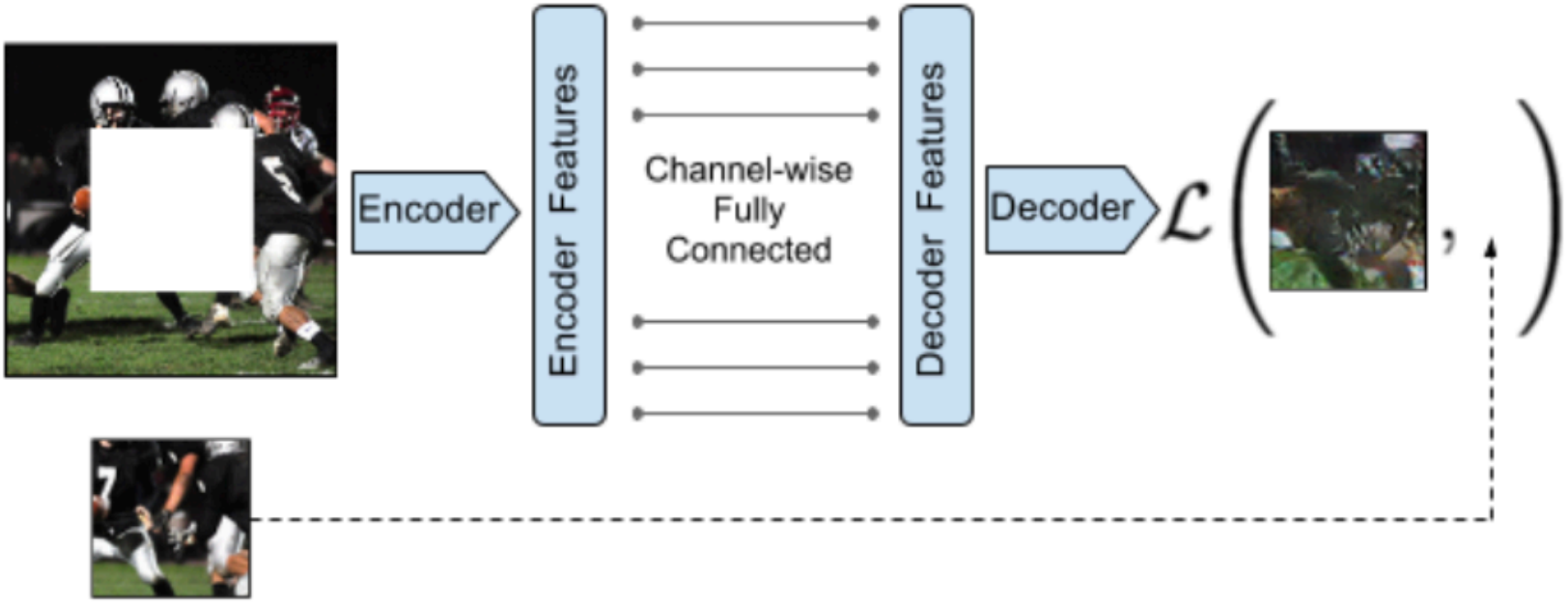
Add L1 penalty to the loss to learn sparse feature representations

$$\sum_i |Enc_i(\mathbf{x})|$$

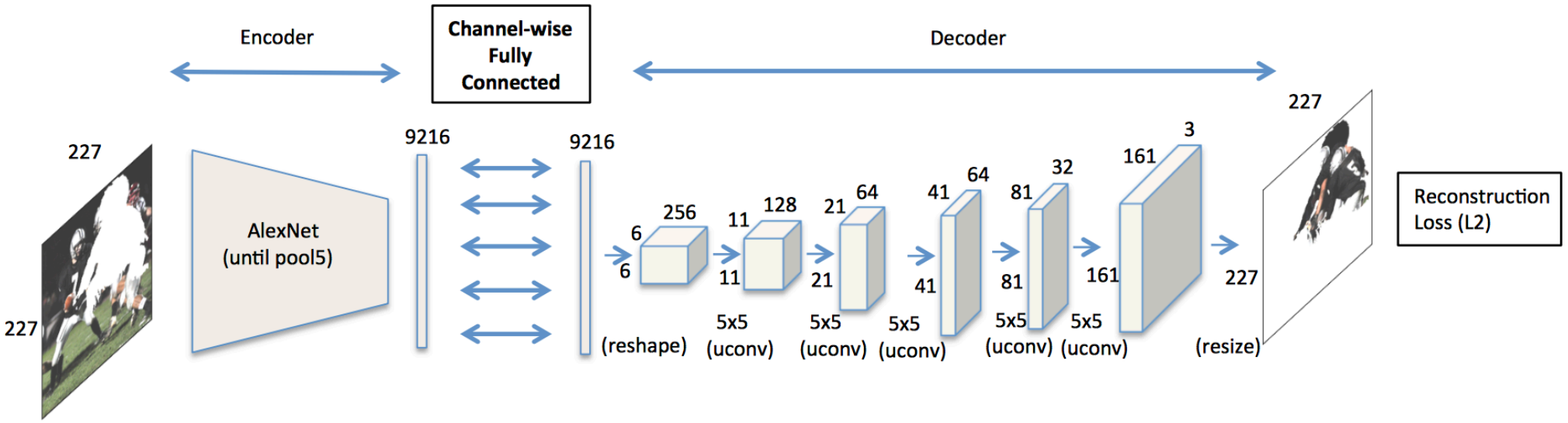


$$\mathcal{L} = ||\mathbf{x} - Dec(Enc(\mathbf{x}))||_2^2 + \sum_i |Enc_i(\mathbf{x})|$$

Context encoders



Context encoders



Context Encoder

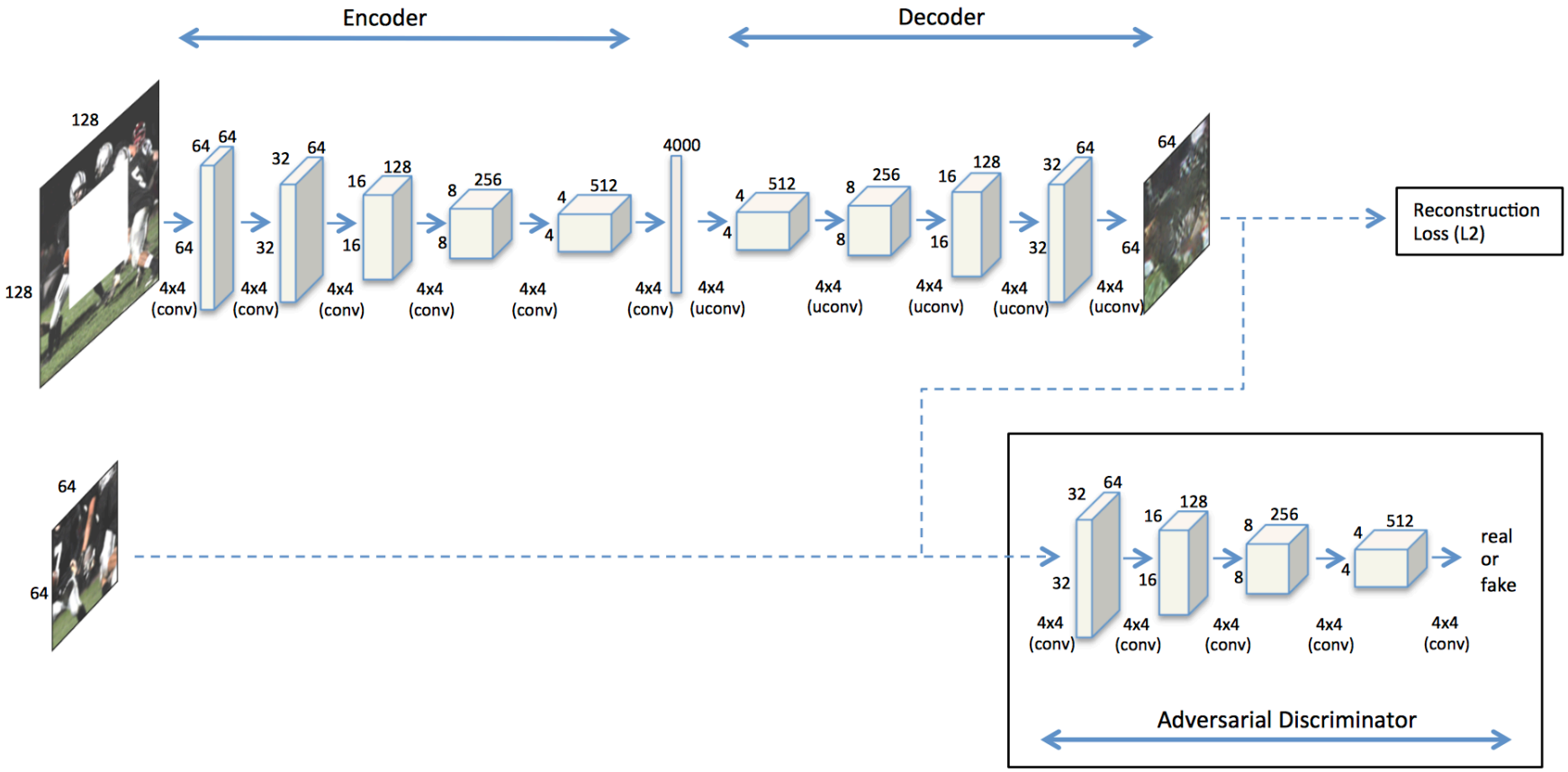


Image colorization

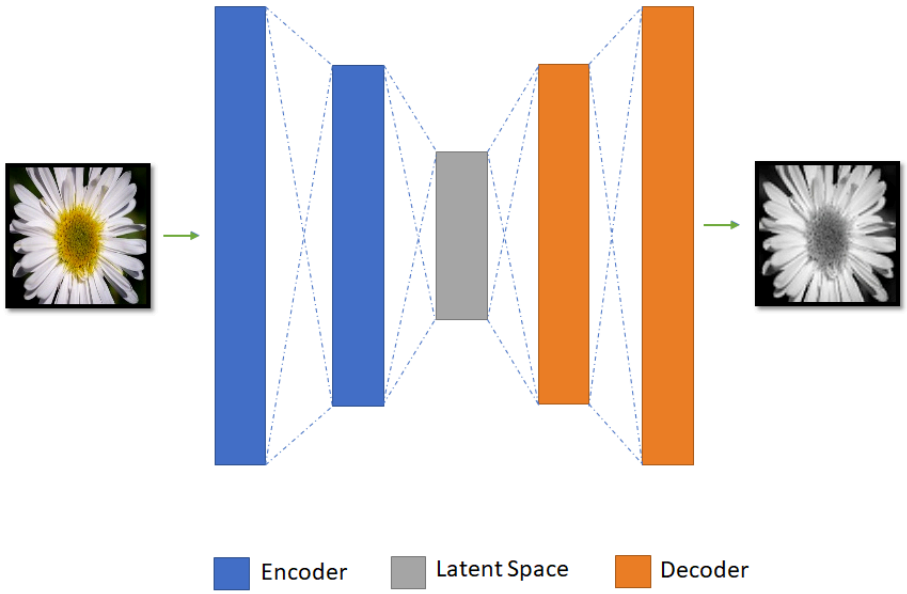
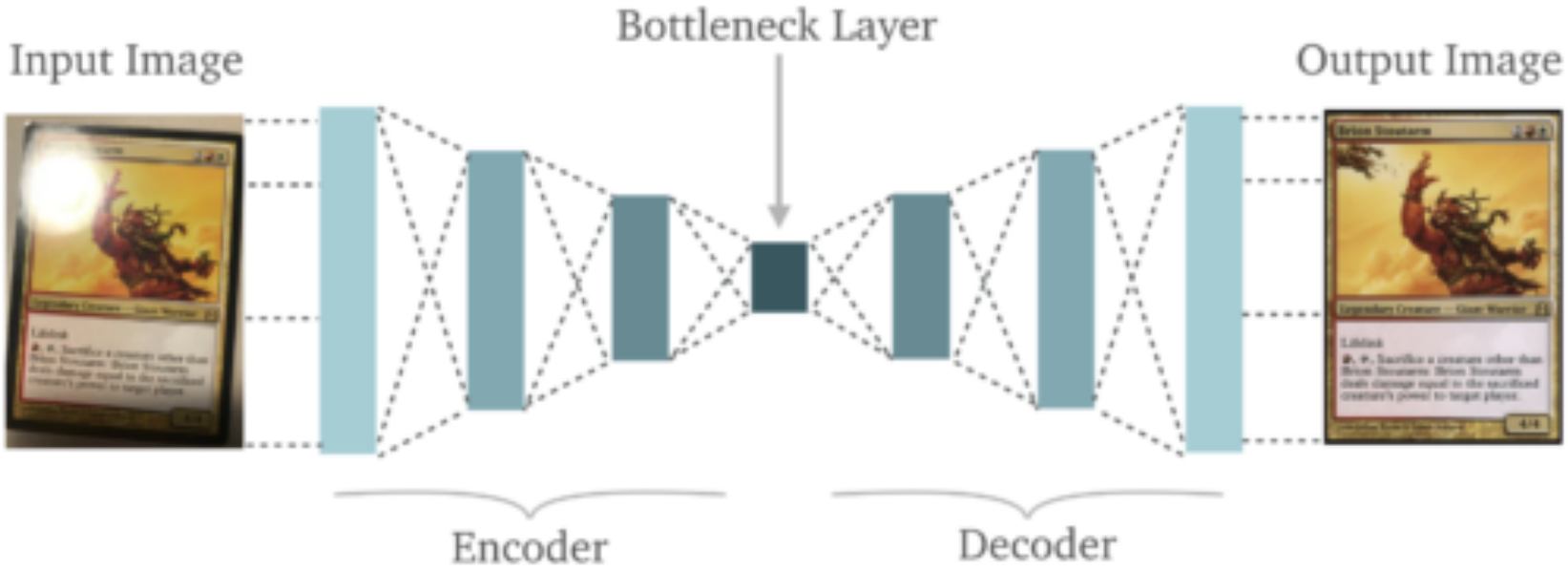
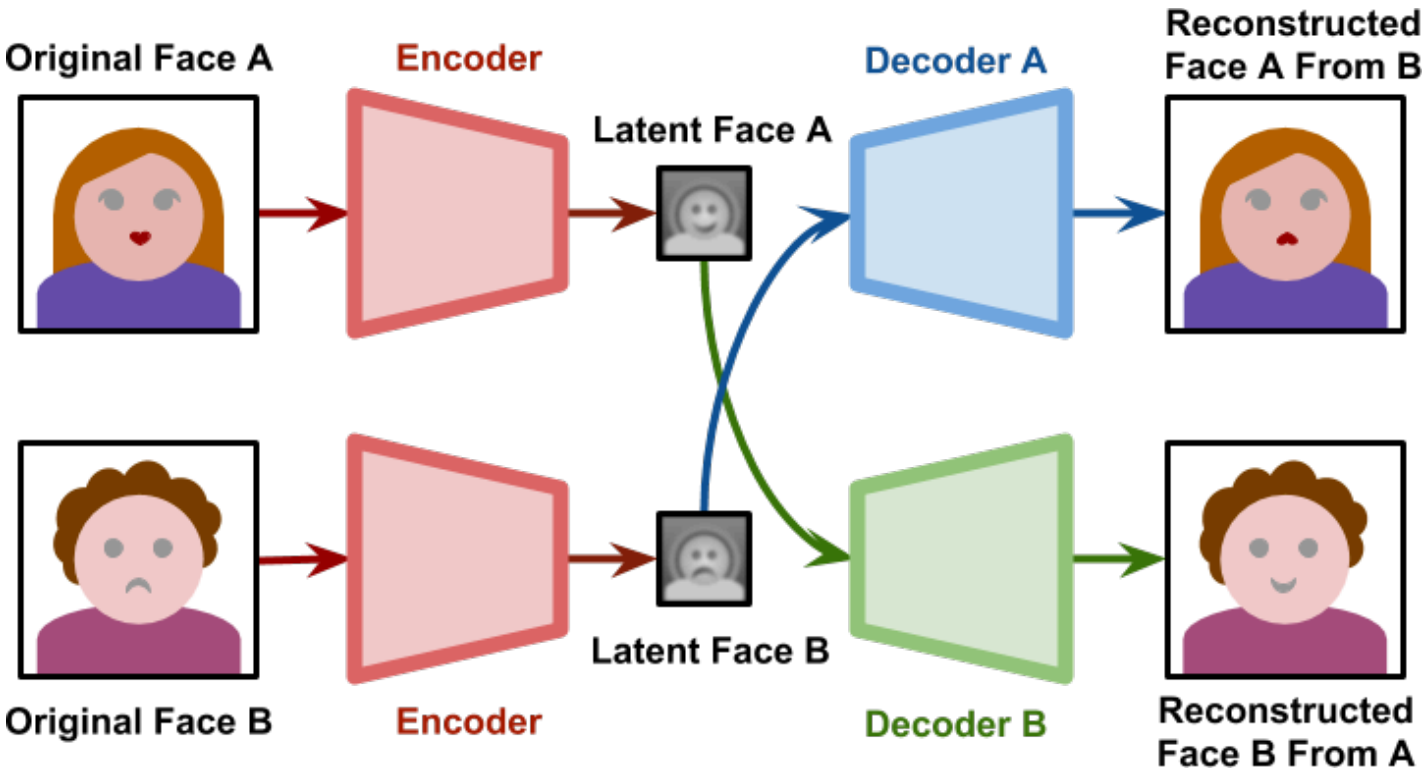


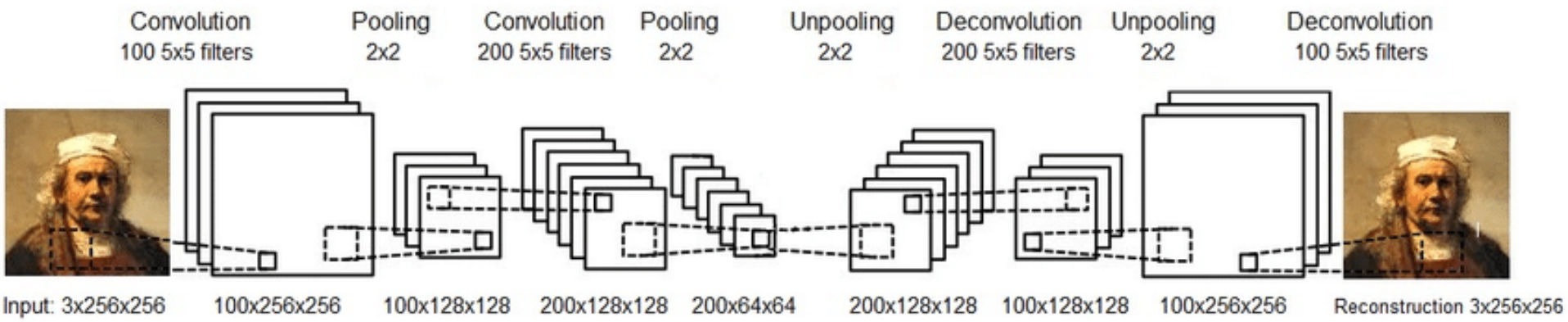
Image variation



Fake laugh


Original Face A
Encoder
Latent Face A
Decoder A
Reconstructed
Face A From B
Original Face B
Encoder
Latent Face B
Decoder B
Reconstructed
Face B From A

Fake painting



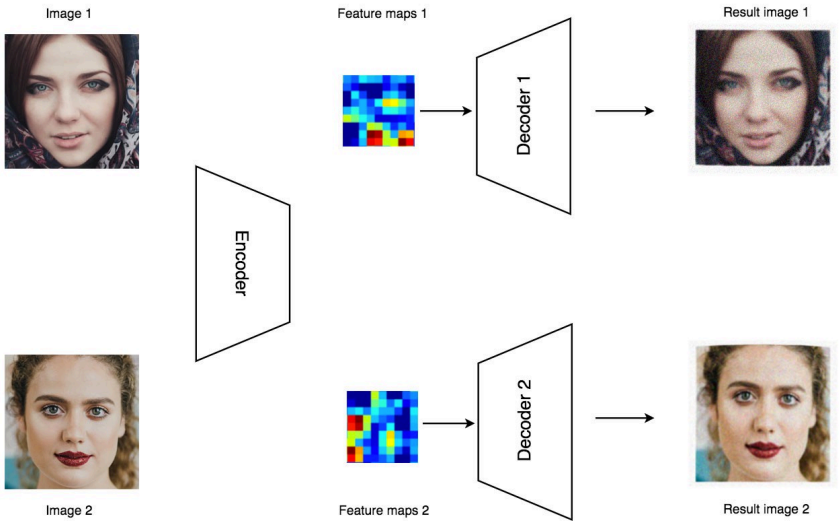
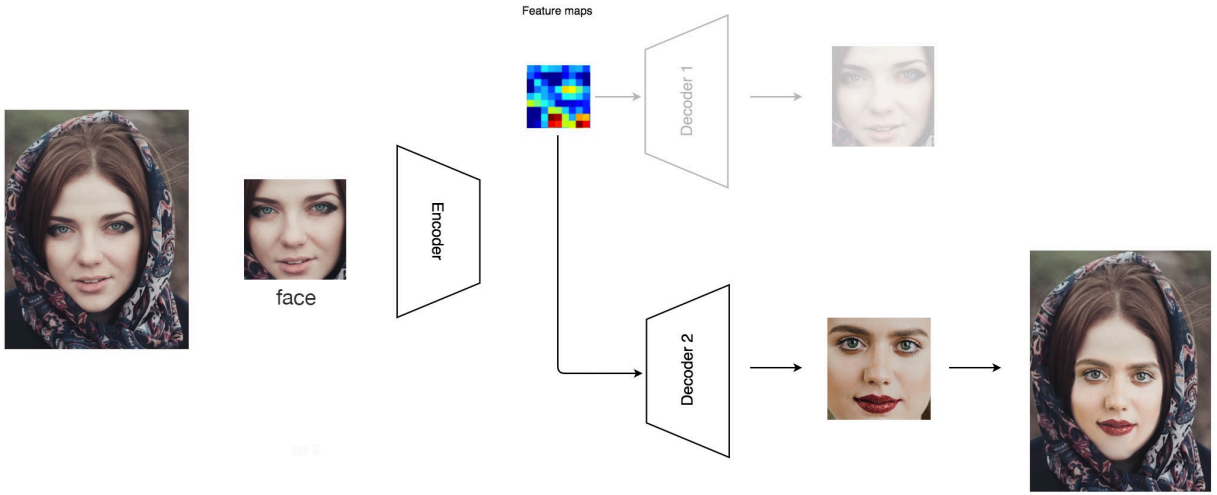


Image 1
Feature maps 1
Result image 1
Decoder 1
Encoder
Decoder 2
Image 2
Feature maps 2
Result image 2

1

Fake face

2


Feature maps
Decoder 1
Encoder
face
Decoder 2

FMRI decoding