

Bsc Data Science for Responsible Business

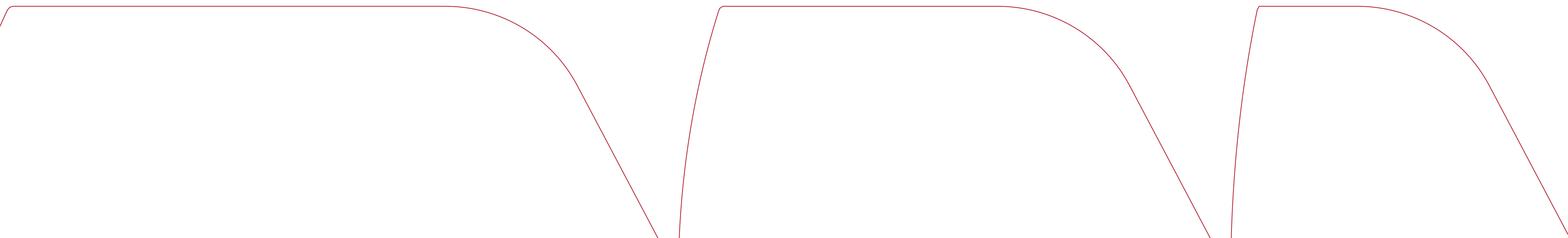
Deep Learning Course

Autoencoder, U-Net, VAE, GAN

Emmanuel Dellandrea - emmanuel.dellandrea@ec-lyon.fr

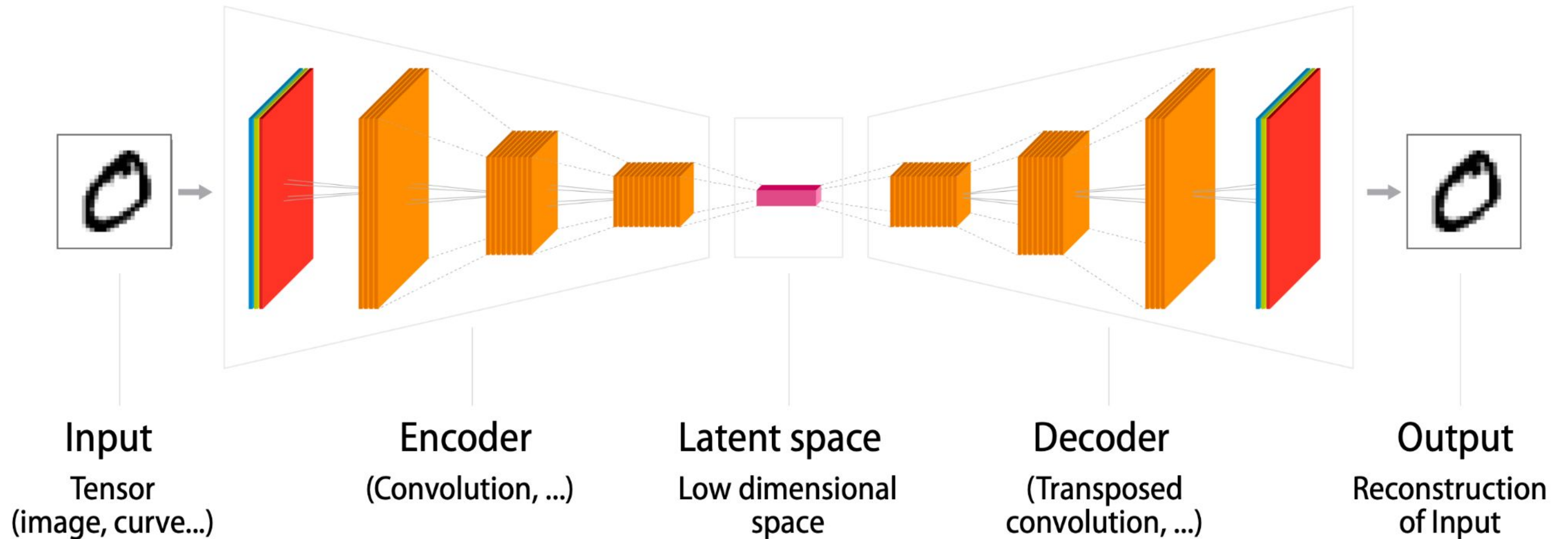


Autoencoder



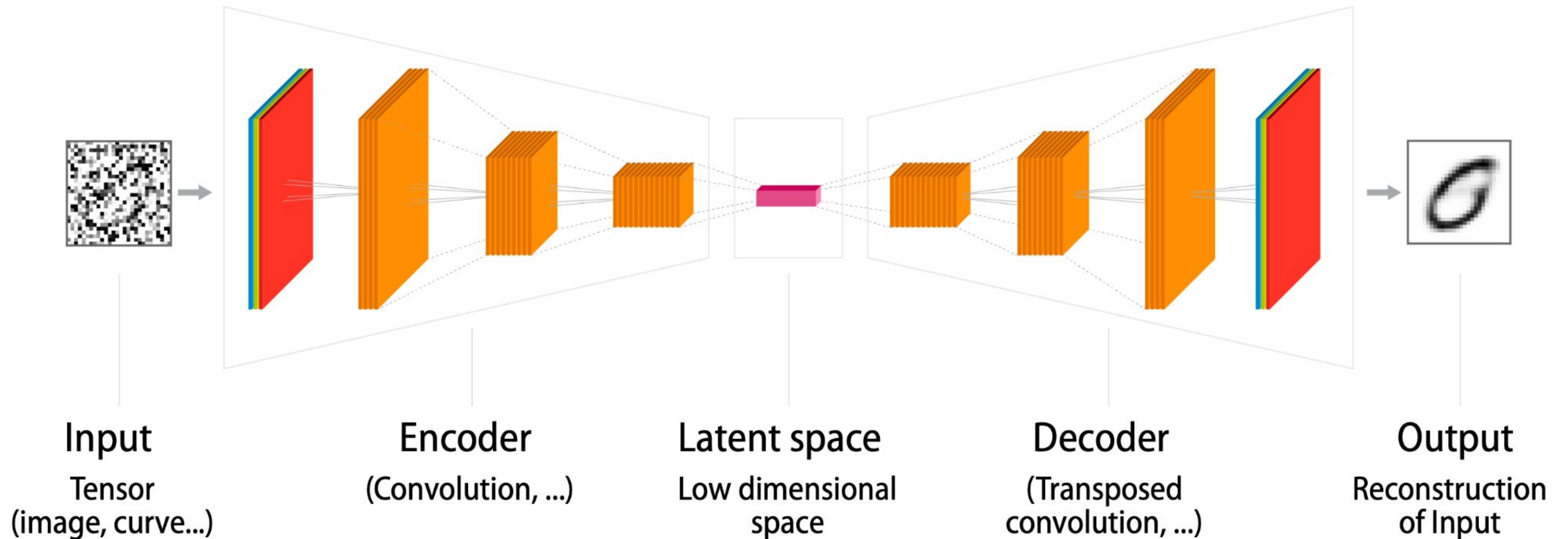
Autoencoders

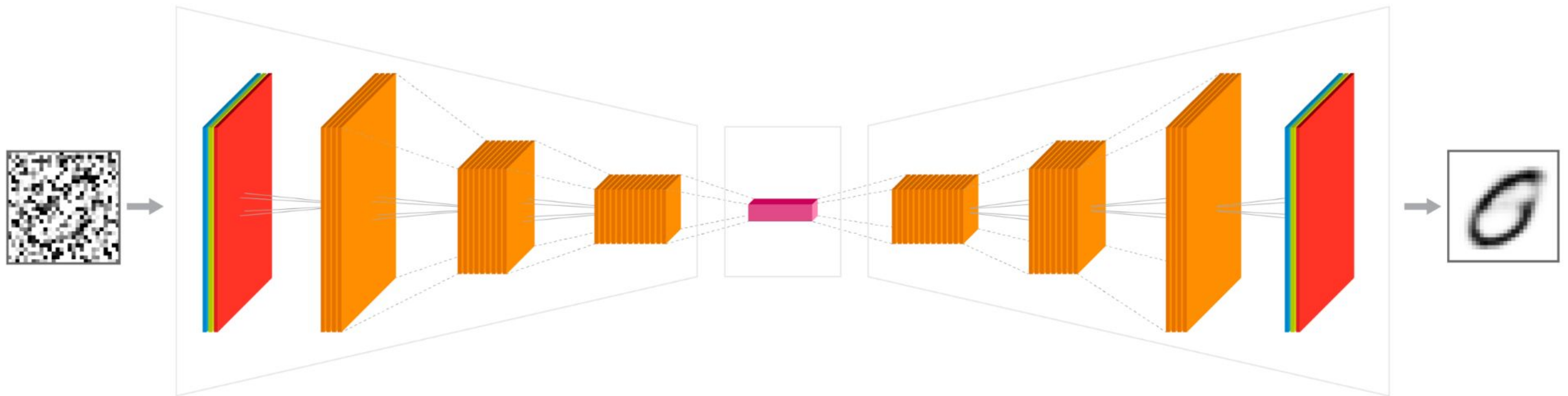
Autoencoders are trained to minimise reconstruction errors.



Autoencoders

Autoencoders are trained to retain essential data

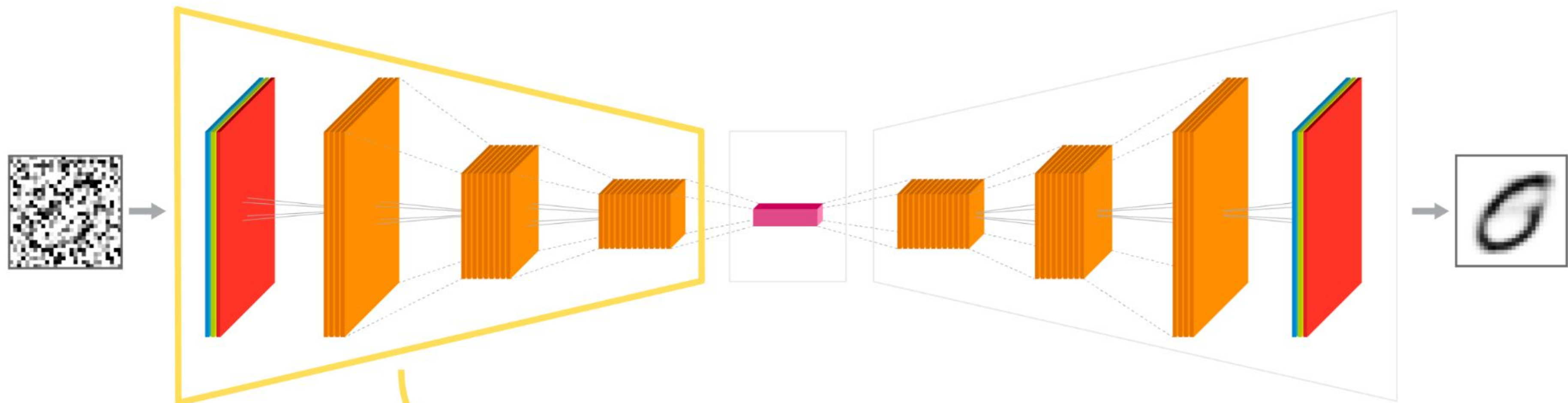




Great, but...
How to build an autoencoder?

Autoencoders

An autoencoders network have 2 parts :

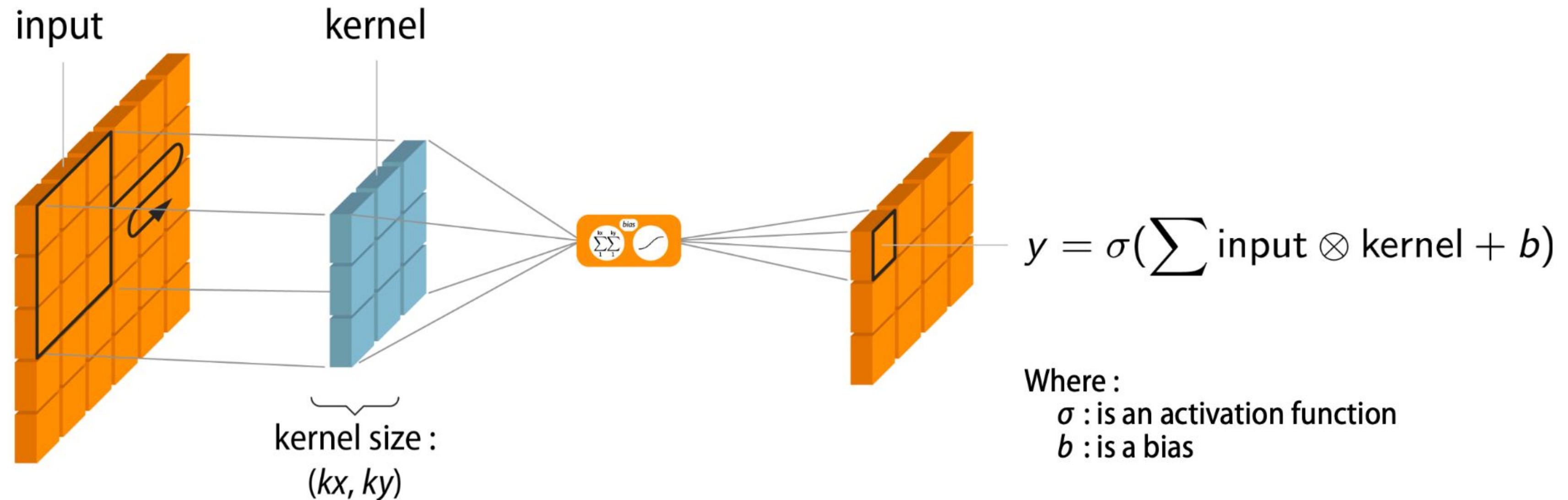


Encoder

Which can use, for example,
some convolutional layers

Convolutional layers

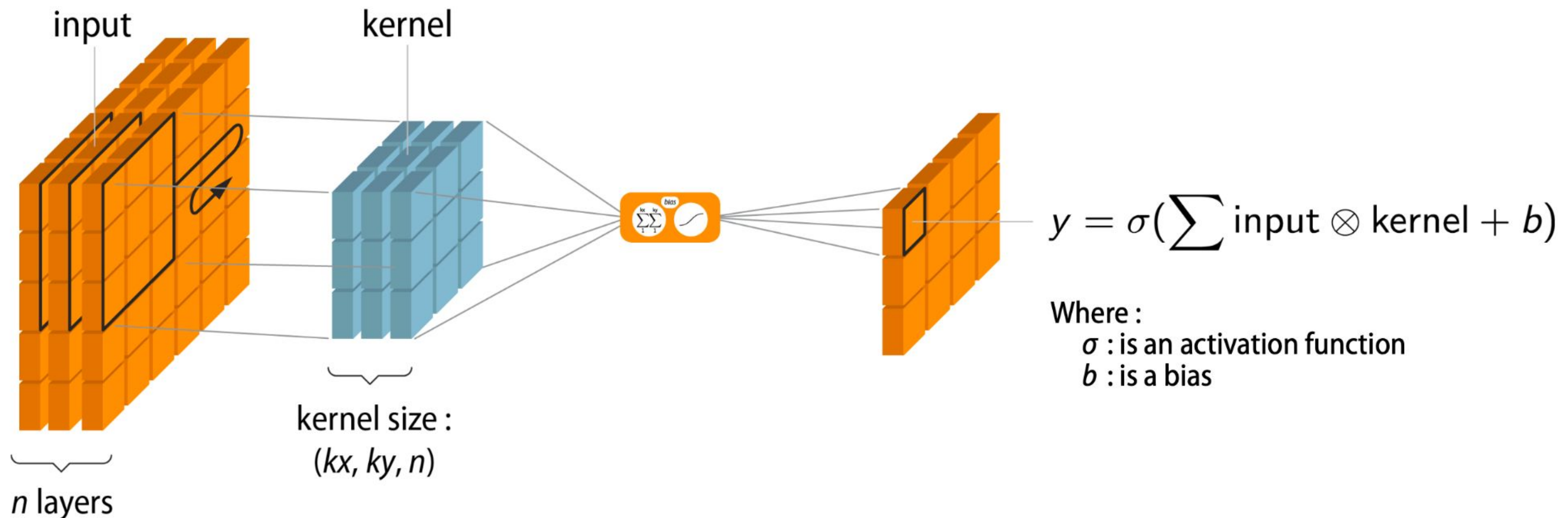
Objective is to **downsampling** an input :



Number of parameters for a convolutional layer : $kx \cdot ky + 1$

Convolutional layers

Objective is to **downsampling** an input :

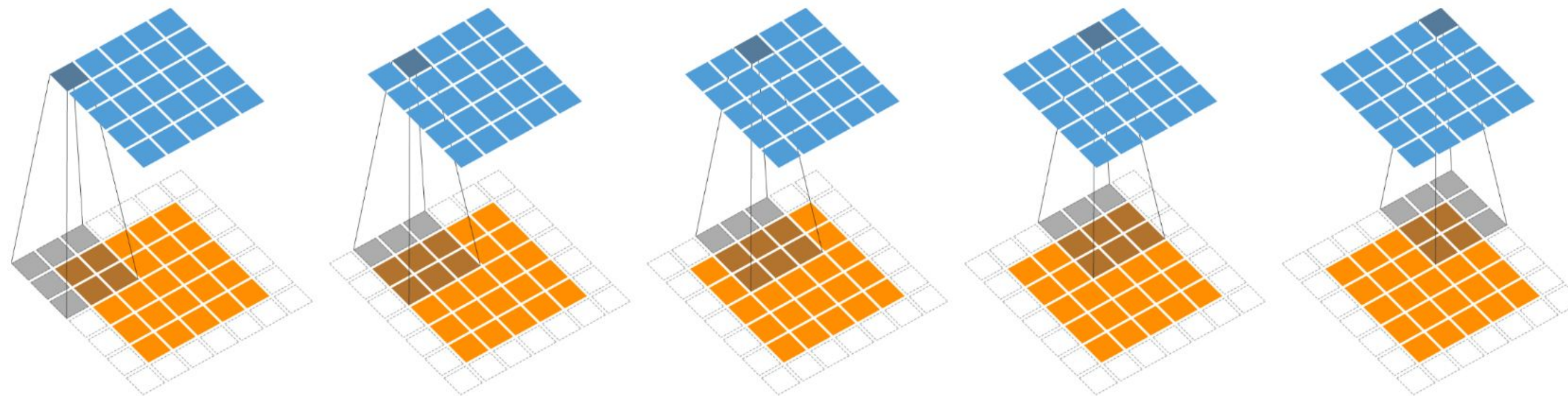
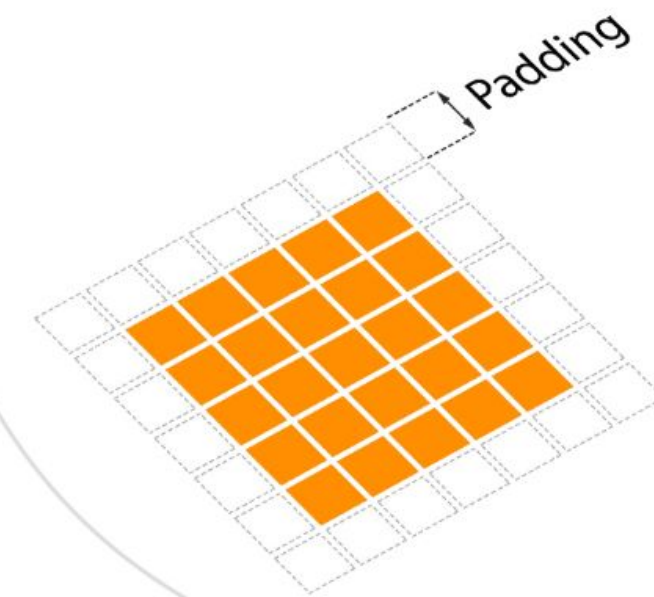


If we want to generate m convolutional layers, we will need m convolutional neurons

Convolutional parameters

Parameters of a convolutional layer : padding

Reminder !

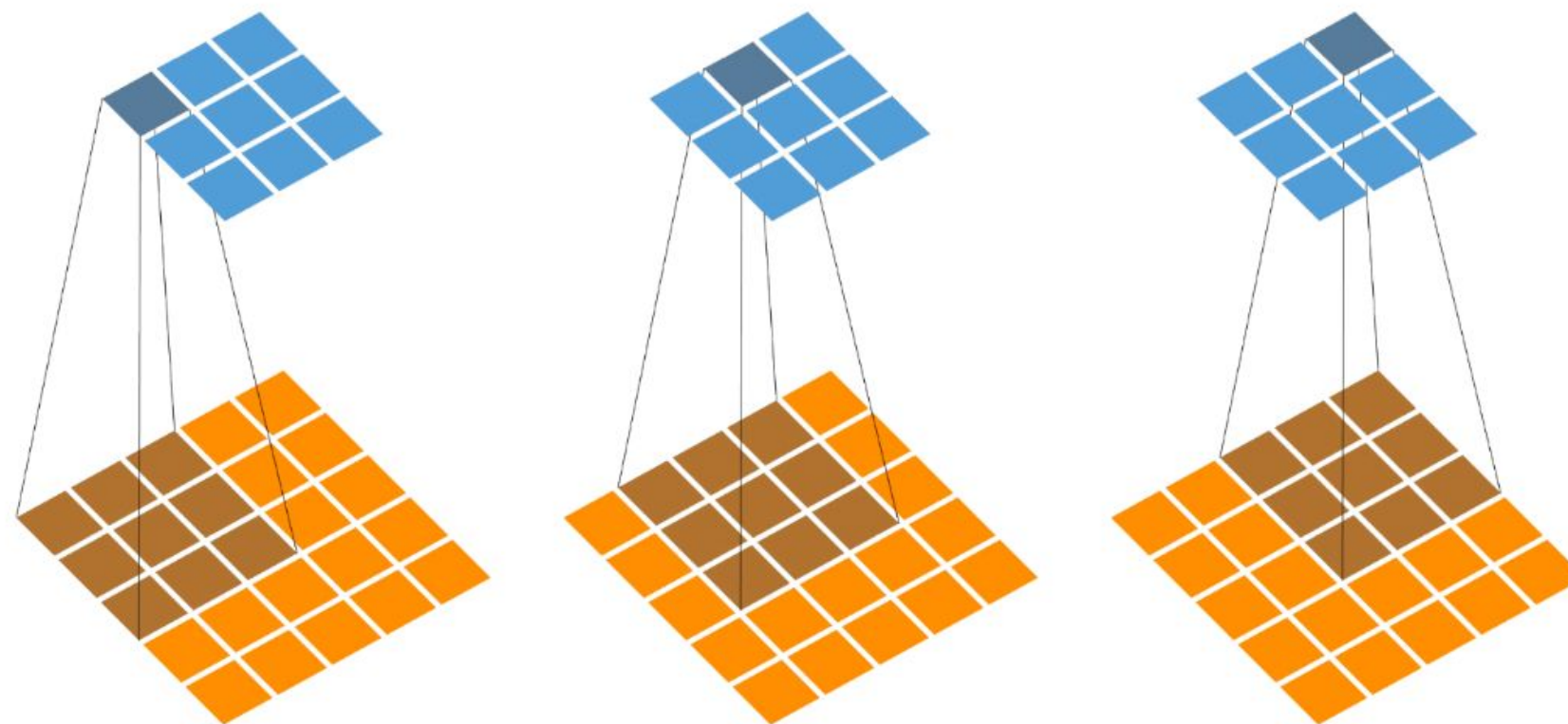
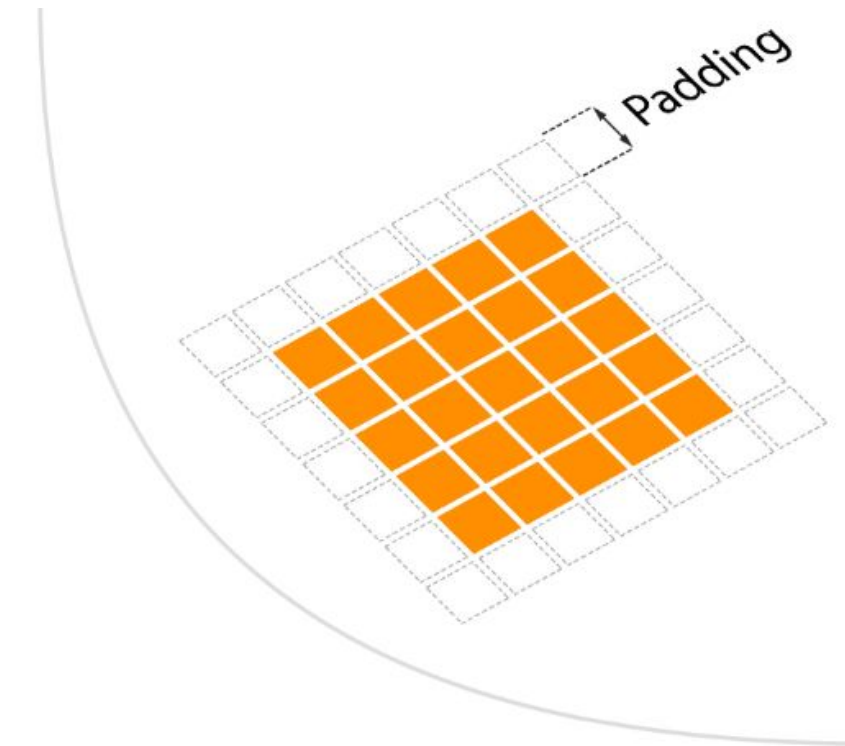


With padding \Rightarrow size is preserved

Convolutional parameters

Reminder !

Parameters of a convolutional layer : padding

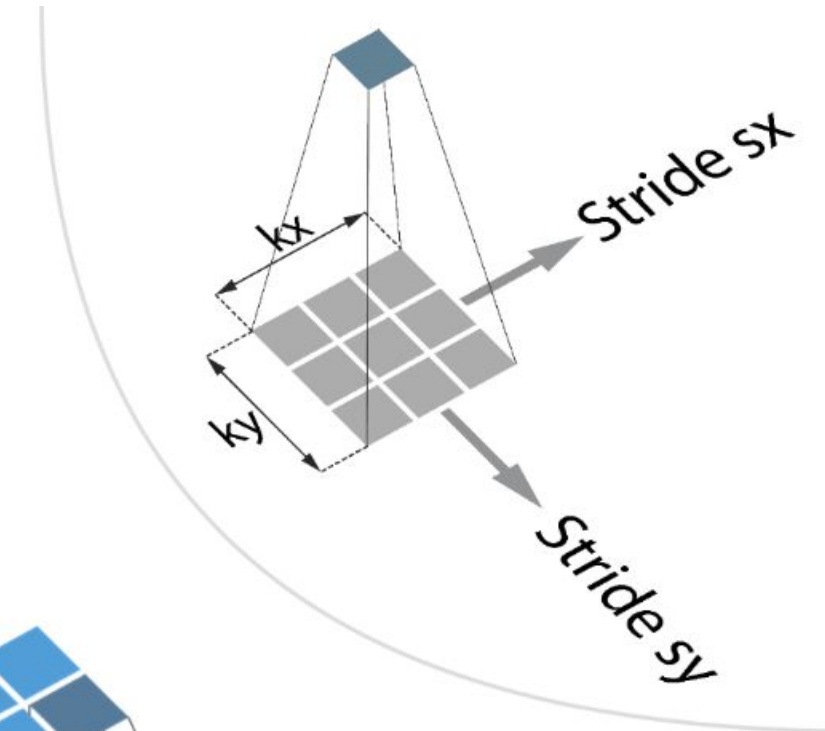
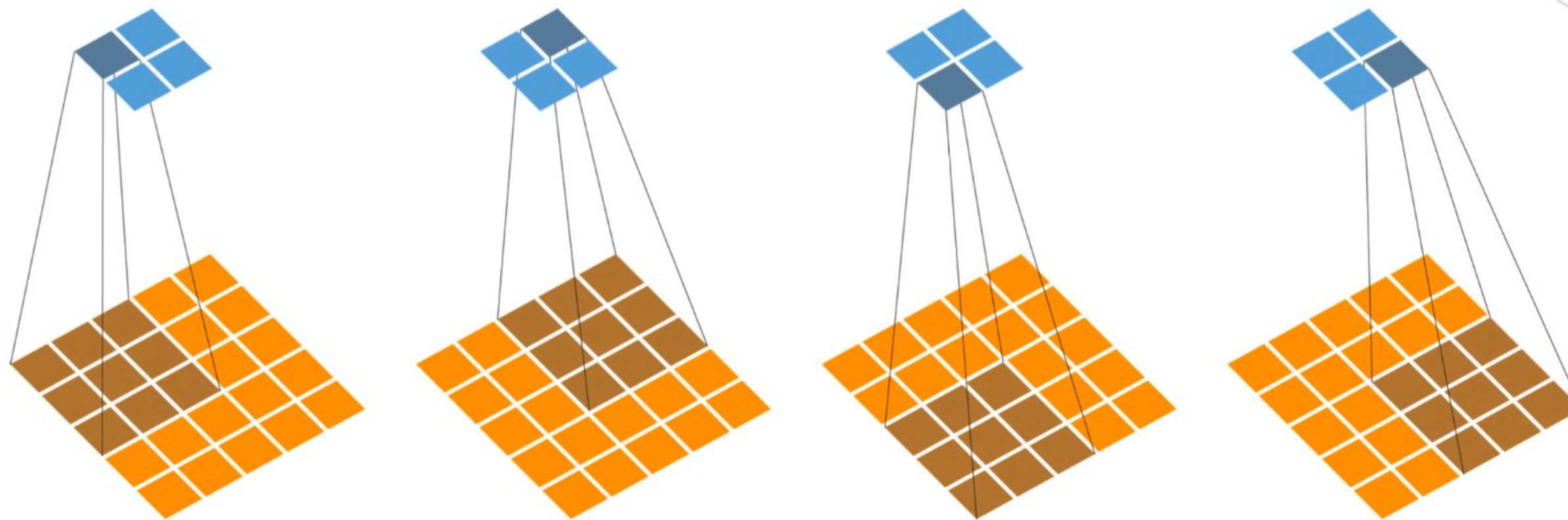


Without padding \Rightarrow Size is not preserved (no padding)

Convolutional parameters

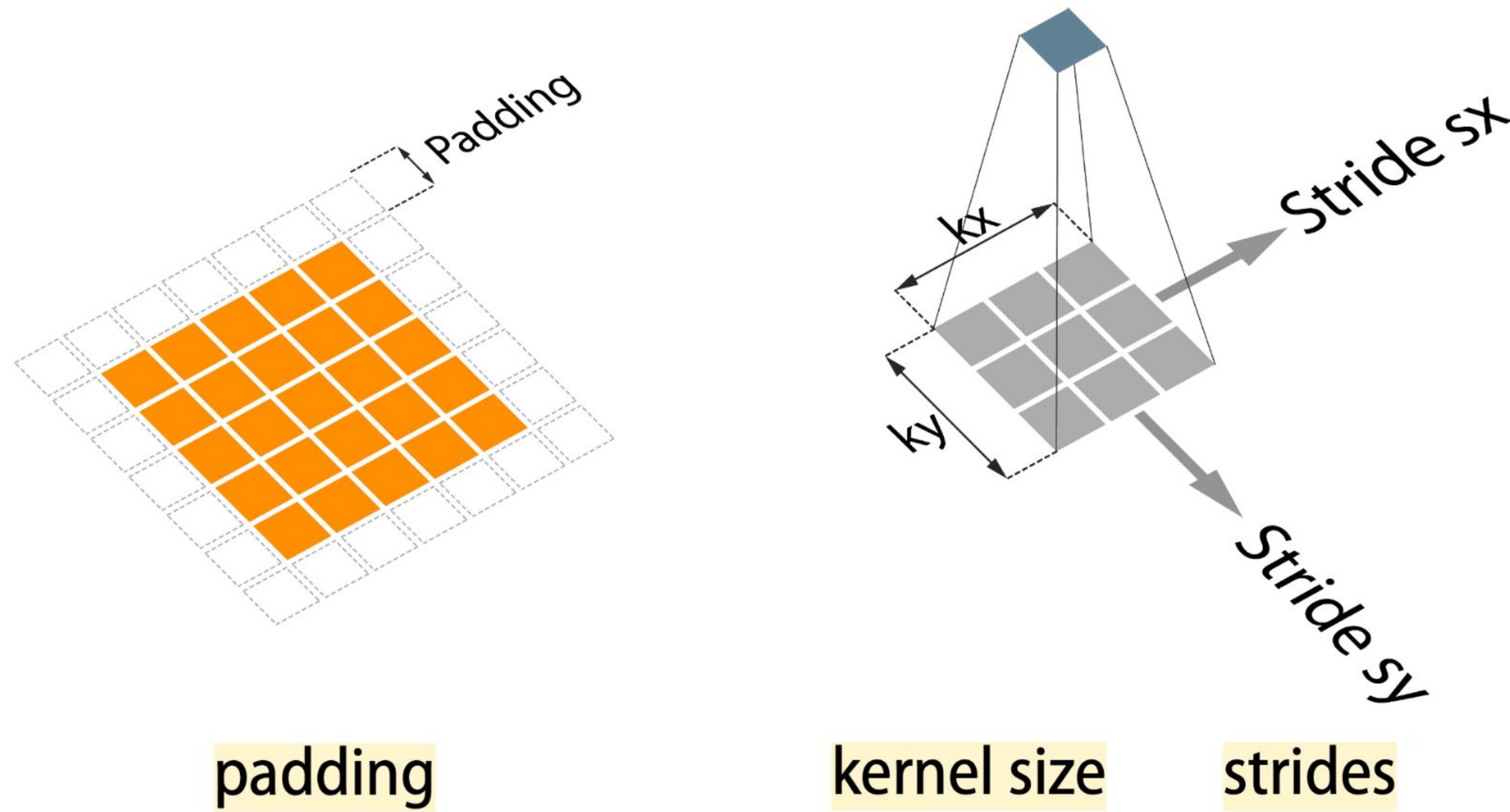
Reminder !

Parameters of a convolutional layer : Strides



$\text{strides} = (sx, sy) \Rightarrow$ A $\text{strides}=(2,2)$ will reduce by 2 the output size

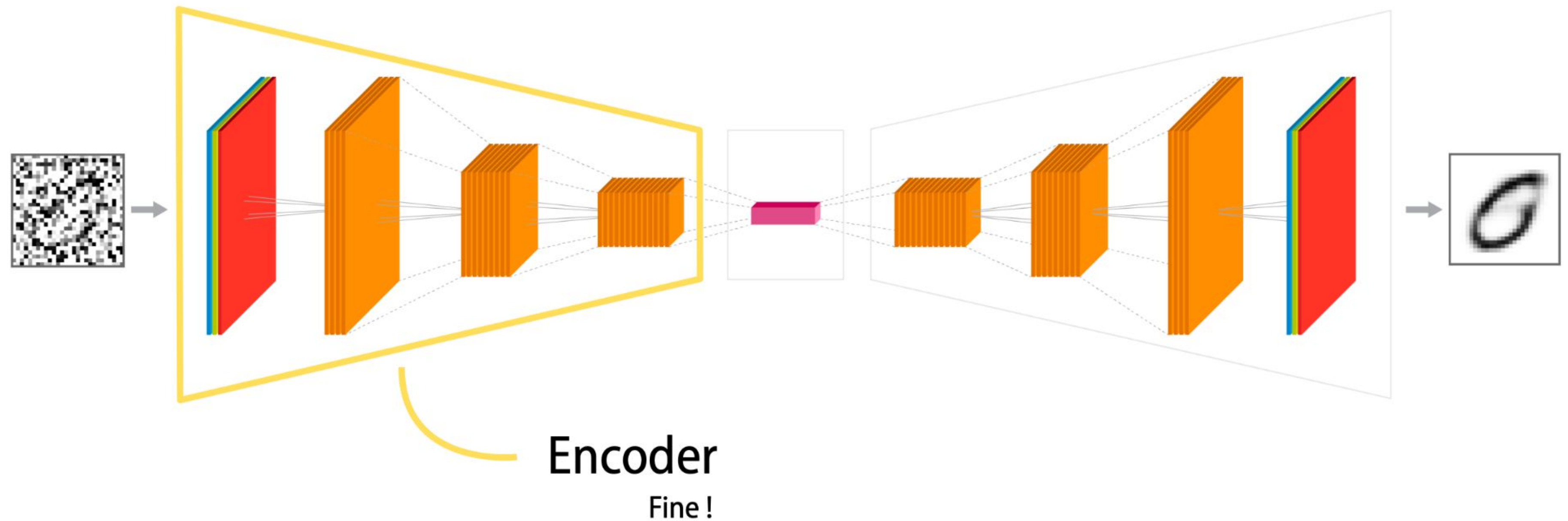
Convolutional parameters



$$\text{Output_dimension} = (\text{input_dimension} - \text{kernel_size} + 2 \times \text{padding}) / \text{stride} + 1$$

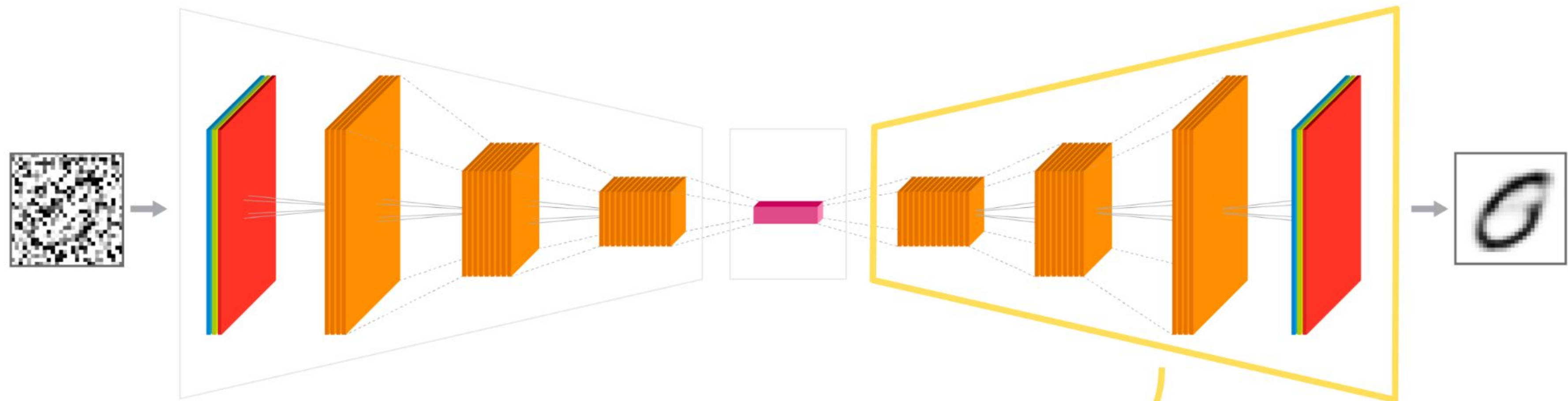
Autoencoder

An autoencoders network have 2 parts :



Autoencoder

An autoencoders network have 2 parts :

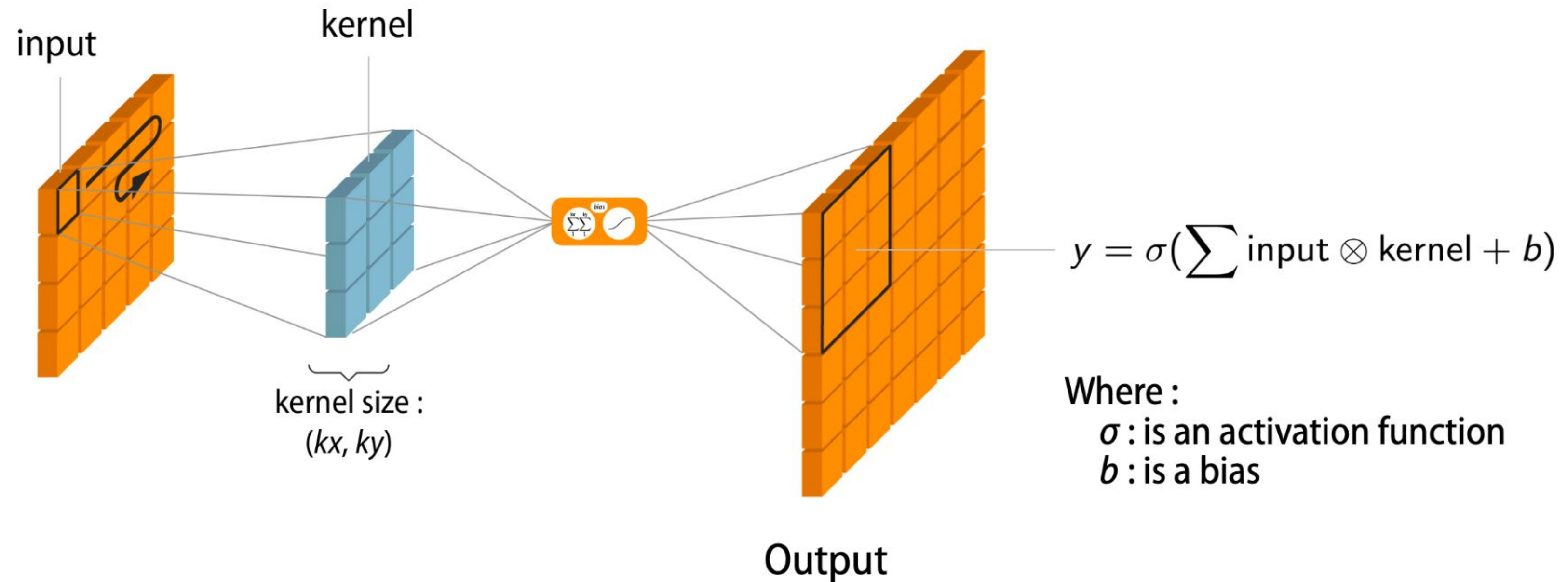


Decoder

Which can use, for example,
transposed convolutional layers

Transposed convolution

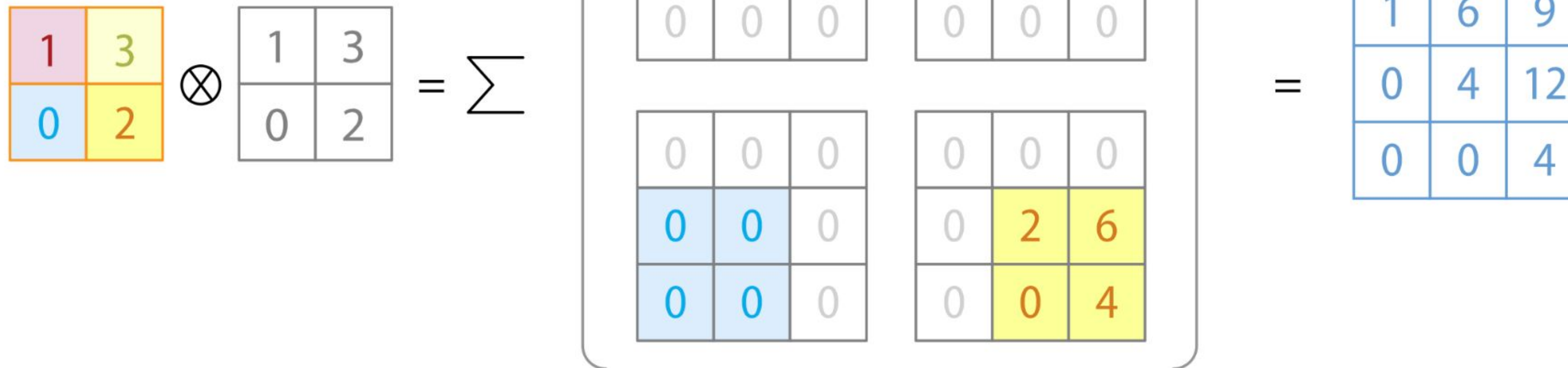
Objective is to **upsampling** an input :



Transposed convolution

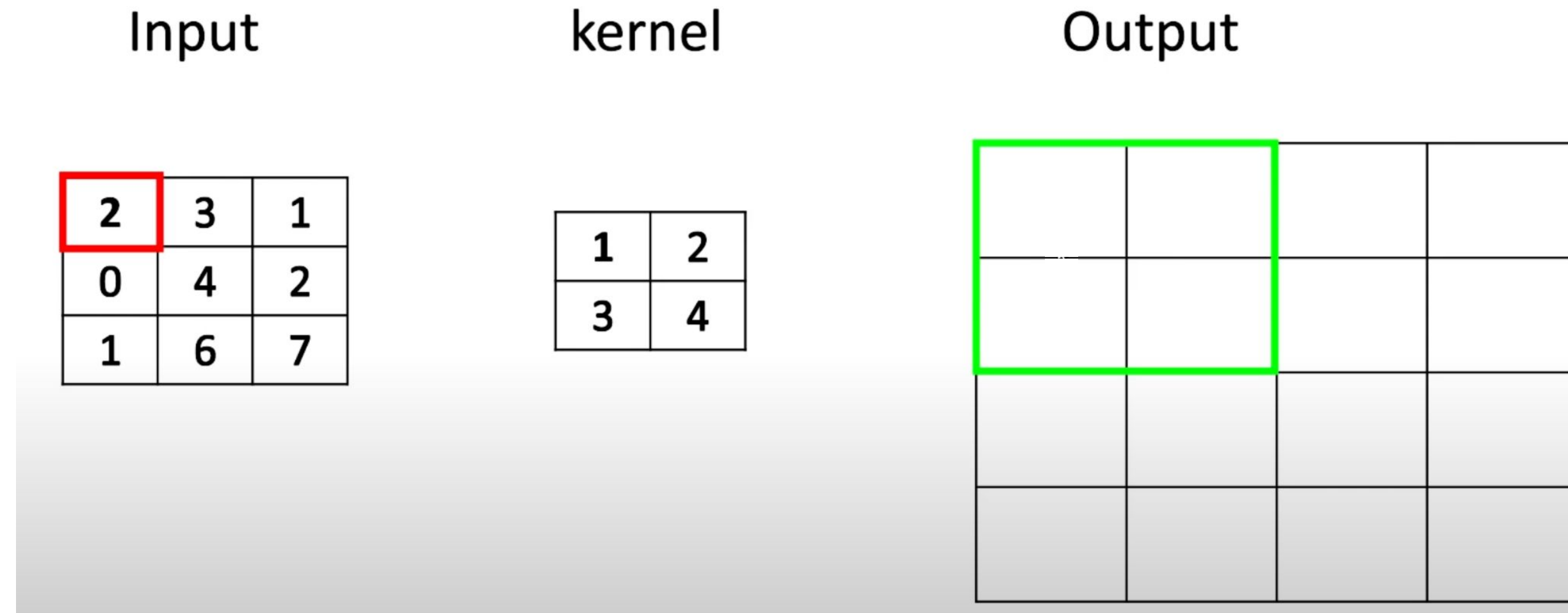
Basic example :

Input image : (2,2)
Kernel : (2,2)
Output image : (3,3)


$$\begin{bmatrix} 1 & 3 \\ 0 & 2 \end{bmatrix} \otimes \begin{bmatrix} 1 & 3 \\ 0 & 2 \end{bmatrix} = \sum \left(\begin{bmatrix} 1 & 3 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 3 & 9 \\ 0 & 0 & 6 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 6 \\ 0 & 0 & 4 \end{bmatrix} \right) = \begin{bmatrix} 1 & 6 & 9 \\ 0 & 4 & 12 \\ 0 & 0 & 4 \end{bmatrix}$$

Transposed convolution

From: <https://www.youtube.com/watch?v=Vu5xXCKG5q8&t=29s>



$$\text{Output} = (\text{input} - 1) * \text{stride} - 2 * \text{padding} + \text{kernel} + \text{output_padding}$$

- transposed convolution with a 3x3 input, 2x2 kernel, stride of 1, no padding, no output padding
- output of dimension $(3-1) \times 1 - 2 \times 0 + 2 + 0 = 4$

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4		
6	8		

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4		
6	8		

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4+3	6	
6	8+9	12	

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4+3	6	
6	8+9	12	

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4+3	6+1	2
6	8+9	12+3	4

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	7	7	2
6	21	25	8
1	20	41	22
3	22	45	28

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

$$\text{Output} = (\text{input} - 1) * \text{stride} - 2 * \text{padding} + \text{kernel} + \text{output_padding}$$

- transposed convolution with a 3x3 input, 2x2 kernel, **stride of 2**, no padding, no output padding
- output of dimension $(3-1) \times 2 - 2 \times 0 + 2 + 0 = 6$

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

Transposed convolution


Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output



2	4				
6	8				

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4	3	6		
6	8	9	12		

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4	3	6		
6	8	9	12		

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4	3	6	1	2
6	8	9	12	3	4

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4	3	6	1	2
6	8	9	12	3	4

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4	3	6	1	2
6	8	9	12	3	4
0	0				
0	0				

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4	3	6	1	2
6	8	9	12	3	4
0	0				
0	0				

Transposed convolution

Input

2	3	1
0	4	2
1	6	7

kernel

1	2
3	4

Output

2	4	3	6	1	2
6	8	9	12	3	4
0	0	4	8		
0	0	12	16		

Transposed convolution

Output (padding = 0)

Output (padding = 1)

Transposed convolution

Output (padding = 0)

2	4	3	6	1	2
6	8	9	12	3	4
0	0	4	8	2	4
0	0	12	16	6	8
1	2	6	12	7	14
3	4	18	24	21	28

Output (padding = 1)

2	4	3	6	1	2
6	8	9	12	3	4
0	0	4	8	2	4
0	0	12	16	6	8
1	2	6	12	7	14
3	4	18	24	21	28

Transposed convolution

Output (padding = 0)

2	4	3	6	1	2
6	8	9	12	3	4
0	0	4	8	2	4
0	0	12	16	6	8
1	2	6	12	7	14
3	4	18	24	21	28

Output (padding = 1)

8	9	12	3
0	4	8	2
0	12	16	6
2	6	12	7

$$\text{Output} = (\text{input} - 1) * \text{stride} - 2 * \text{padding} + \text{kernel} + \text{output_padding}$$

- transposed convolution with a 3x3 input, 2x2 kernel, **stride of 2**, padding of 1, no output padding
- output of dimension $(3-1) \times 2 - 2 \times 1 + 2 + 0 = 4$

Transposed convolution

Output (padding = 0)

Output (padding = 2)

Transposed convolution

Output (padding = 0)

2	4	3	6	1	2
6	8	9	12	3	4
0	0	4	8	2	4
0	0	12	16	6	8
1	2	6	12	7	14
3	4	18	24	21	28

Output (padding = 2)

2	4	3	6	1	2
6	8	9	12	3	4
0	0	4	8	2	4
0	0	12	16	6	8
1	2	6	12	7	14
3	4	18	24	21	28

Transposed convolution

Output (padding = 0)

2	4	3	6	1	2
6	8	9	12	3	4
0	0	4	8	2	4
0	0	12	16	6	8
1	2	6	12	7	14
3	4	18	24	21	28

Output (padding = 2)

4	8
12	16

$$\text{Output} = (\text{input} - 1) * \text{stride} - 2 * \text{padding} + \text{kernel} + \text{output_padding}$$

- transposed convolution with a 3x3 input, 2x2 kernel, **stride of 2**, padding of 2, no output padding
- output of dimension $(3-1) \times 2 - 2 \times 2 + 2 + 0 = 2$

Transposed convolution

Output padding = 0

2	4	3	6	1	2
6	8	9	12	3	4
0	0	4	8	2	4
0	0	12	16	6	8
1	2	6	12	7	14
3	4	18	24	21	28

Output padding = 1

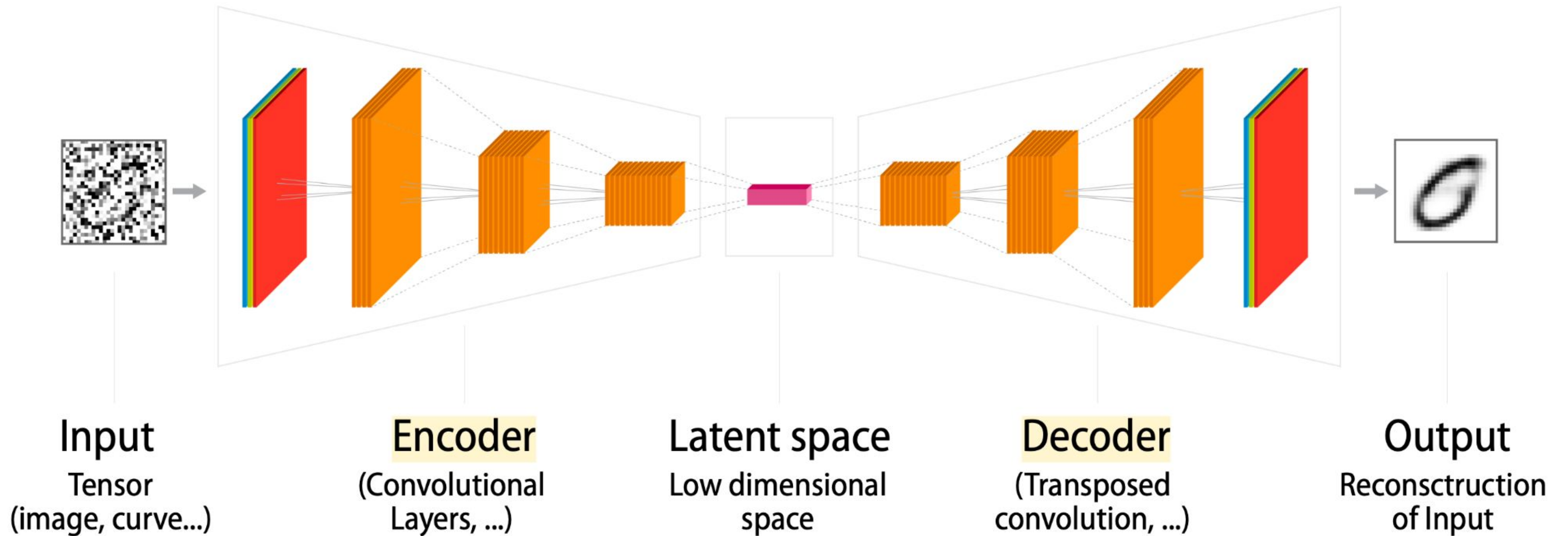
2	4	3	6	1	2	0
6	8	9	12	3	4	0
0	0	4	8	2	4	0
0	0	12	16	6	8	0
1	2	6	12	7	14	0
3	4	18	24	21	28	0
0	0	0	0	0	0	0

$$\text{Output} = (\text{input} - 1) * \text{stride} - 2 * \text{padding} + \text{kernel} + \text{output_padding}$$

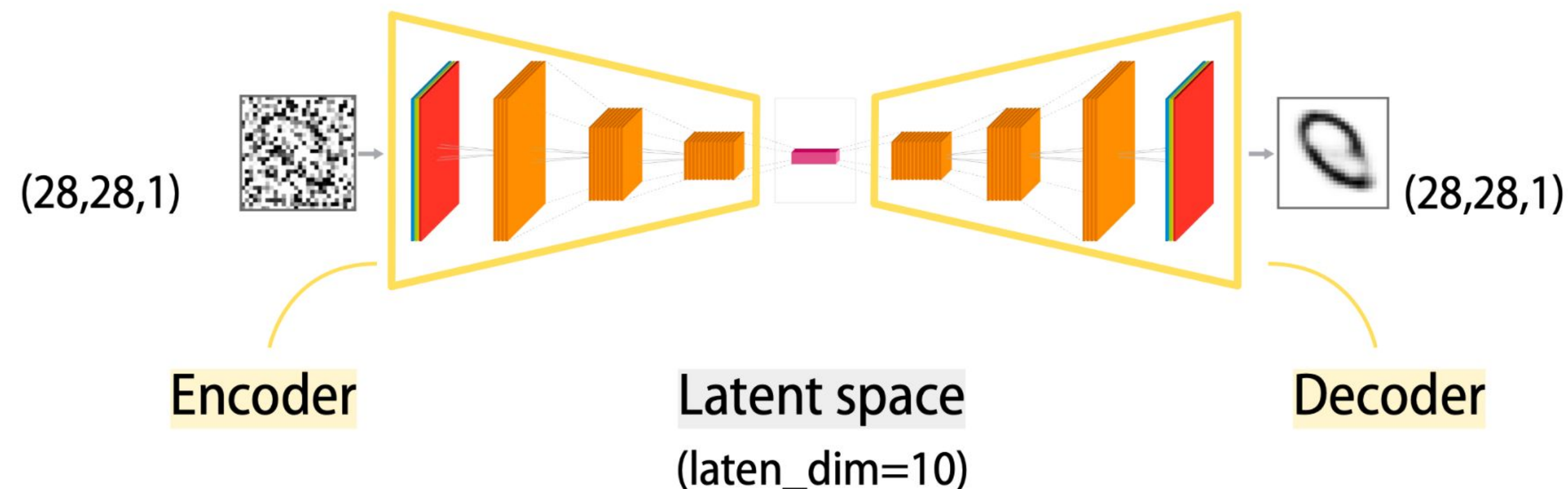
→ transposed convolution with a 3x3 input, 2x2 kernel, **stride of 1**, no padding, output padding of 1

→ output of dimension $(3-1) \times 2 - 2 \times 0 + 2 + 1 = 7$

Autoencoder



Autoencoder

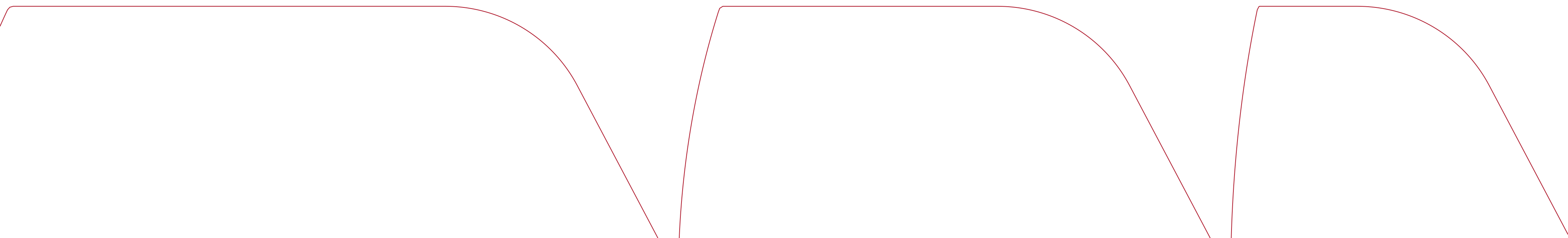


Layer (type)	Output Shape
input_1 (InputLayer)	[(None, 28, 28, 1)]
conv2d (Conv2D)	(None, 14, 14, 32)
conv2d_1 (Conv2D)	(None, 7, 7, 64)
flatten (Flatten)	(None, 3136)
dense (Dense)	(None, 16)
dense_1 (Dense)	(None, 10)

Layer (type)	Output Shape
input_2 (InputLayer)	[(None, 10)]
dense_2 (Dense)	(None, 3136)
reshape (Reshape)	(None, 7, 7, 64)
conv2d_t1 (Conv2DT)	(None, 14, 14, 64)
conv2d_t2 (Conv2DT)	(None, 28, 28, 32)
conv2d_t3 (Conv2DT)	(None, 28, 28, 1)



U-Net networks



U-Net for semantic image segmentation

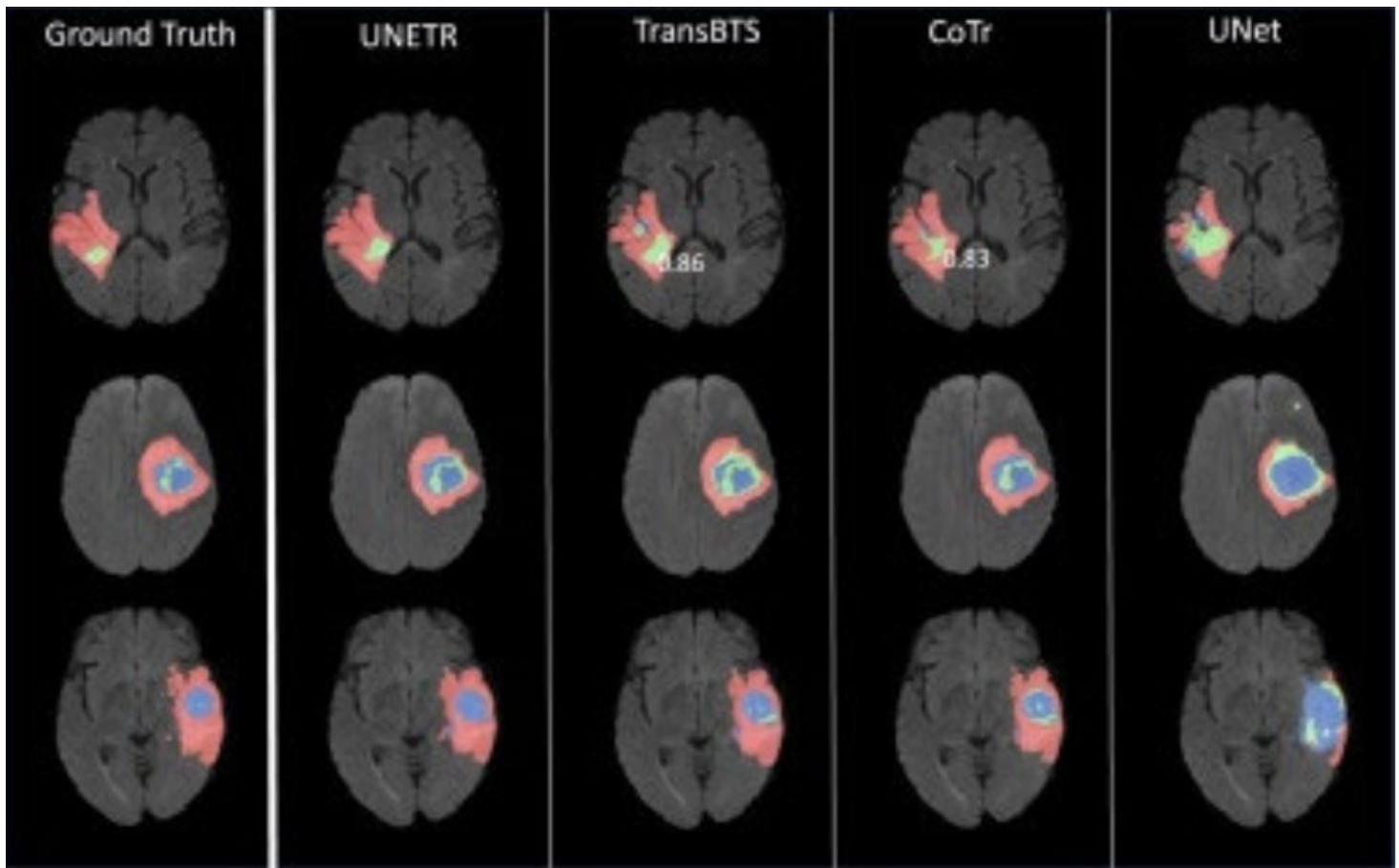
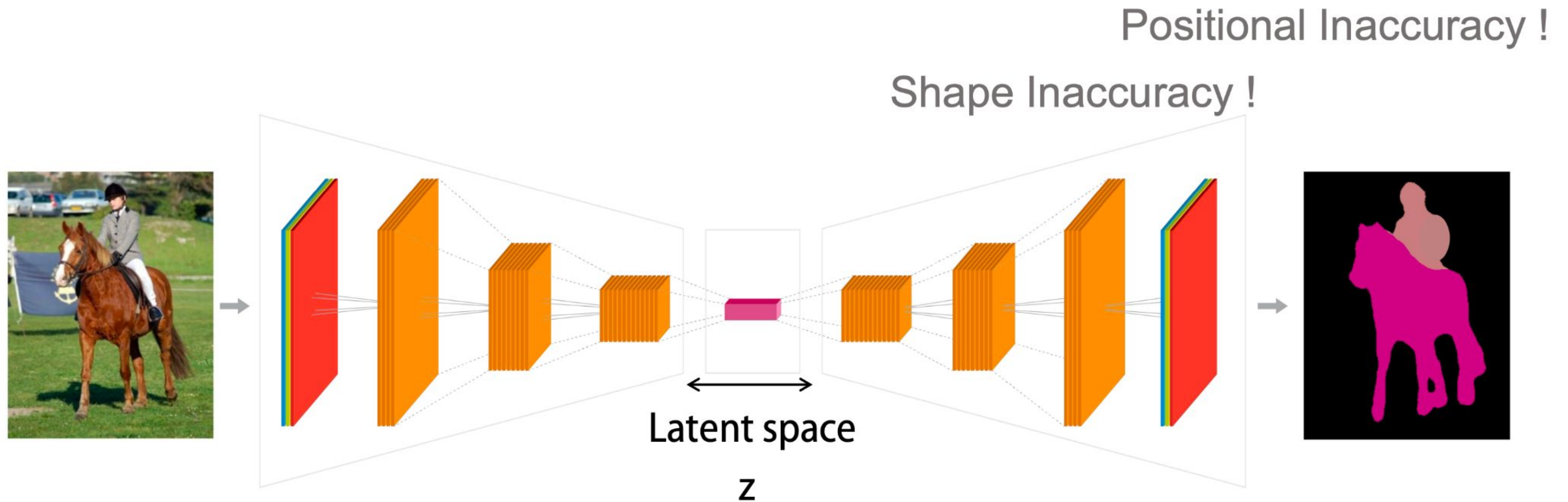
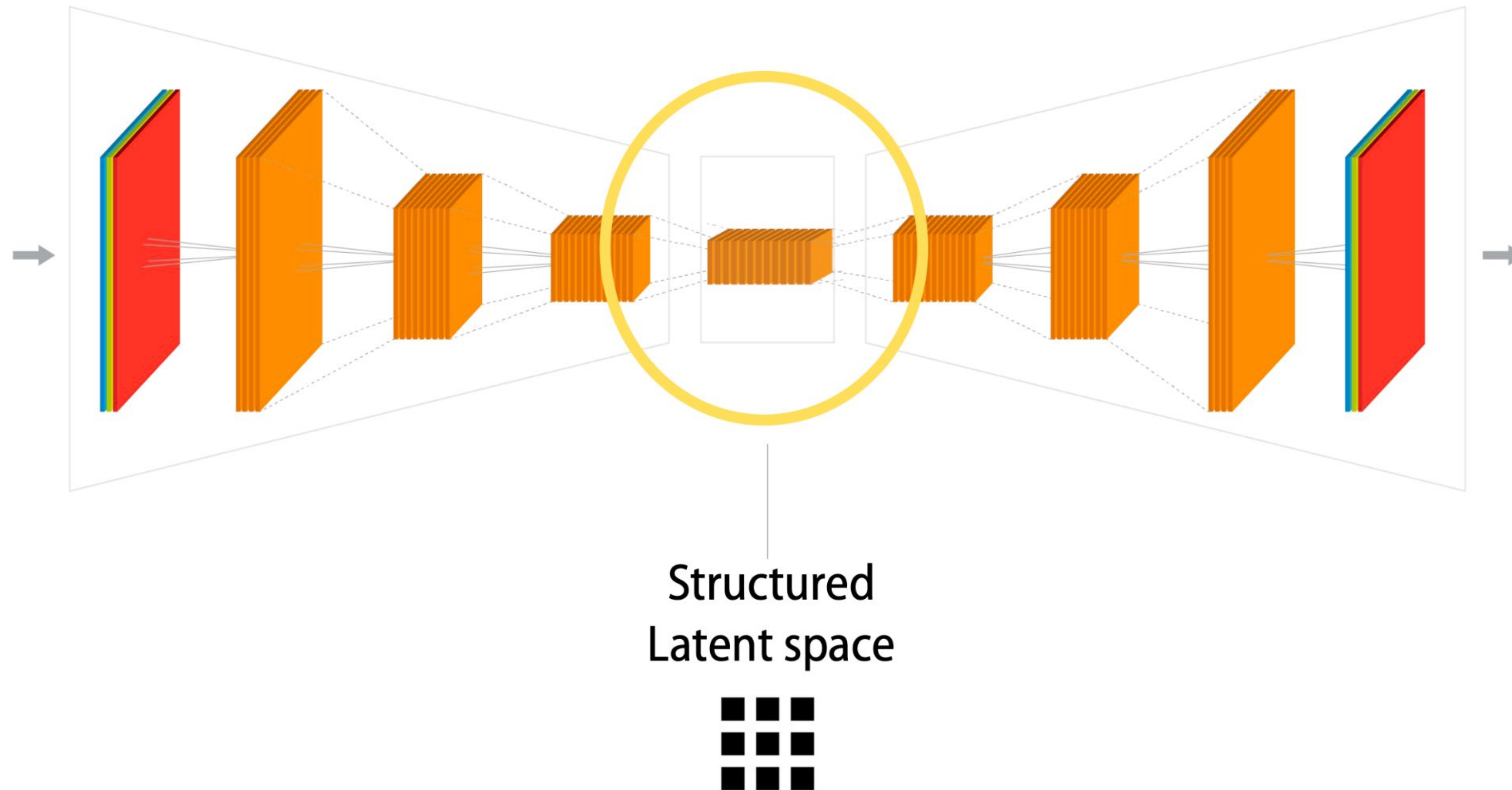


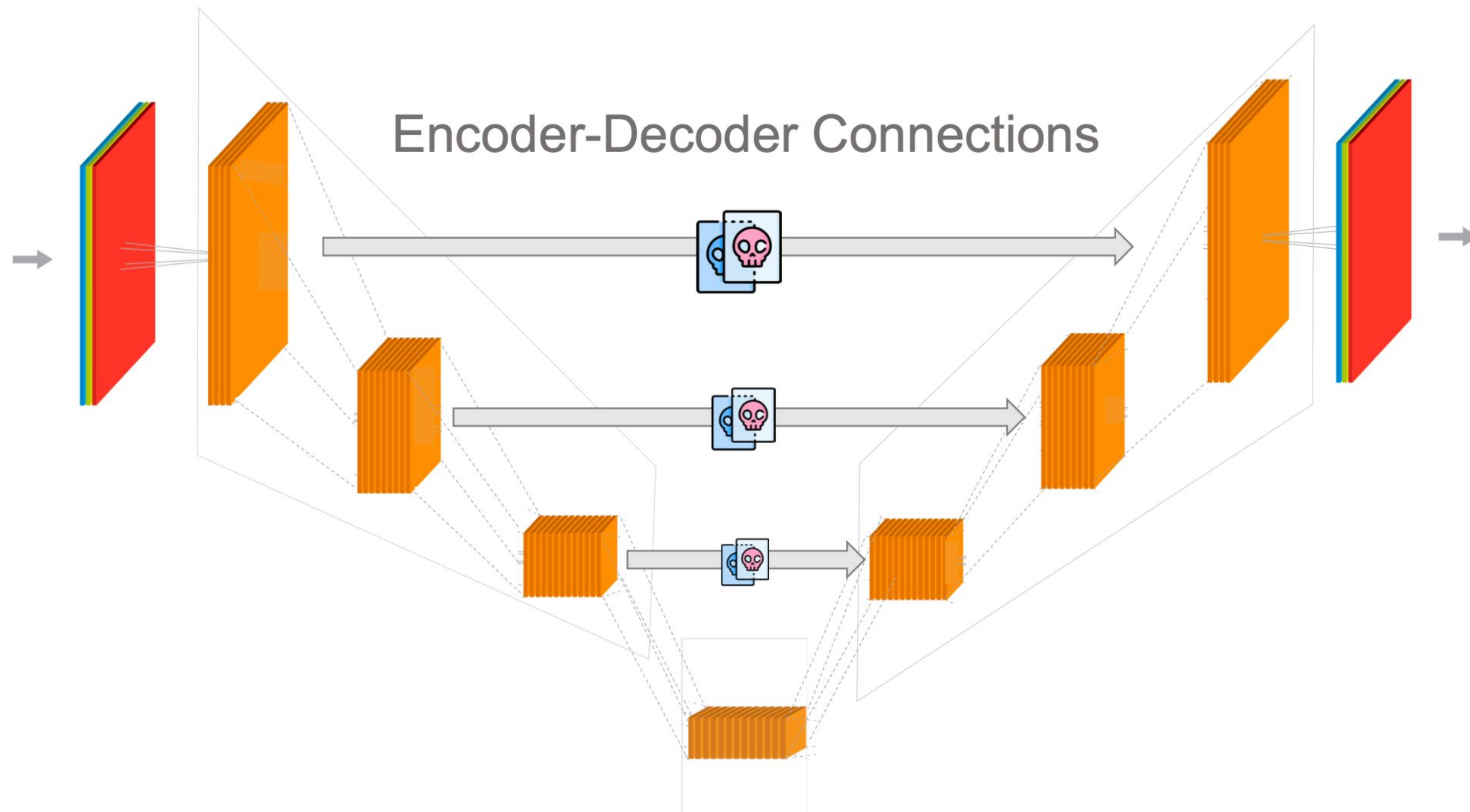
Image segmentation with a basic AE



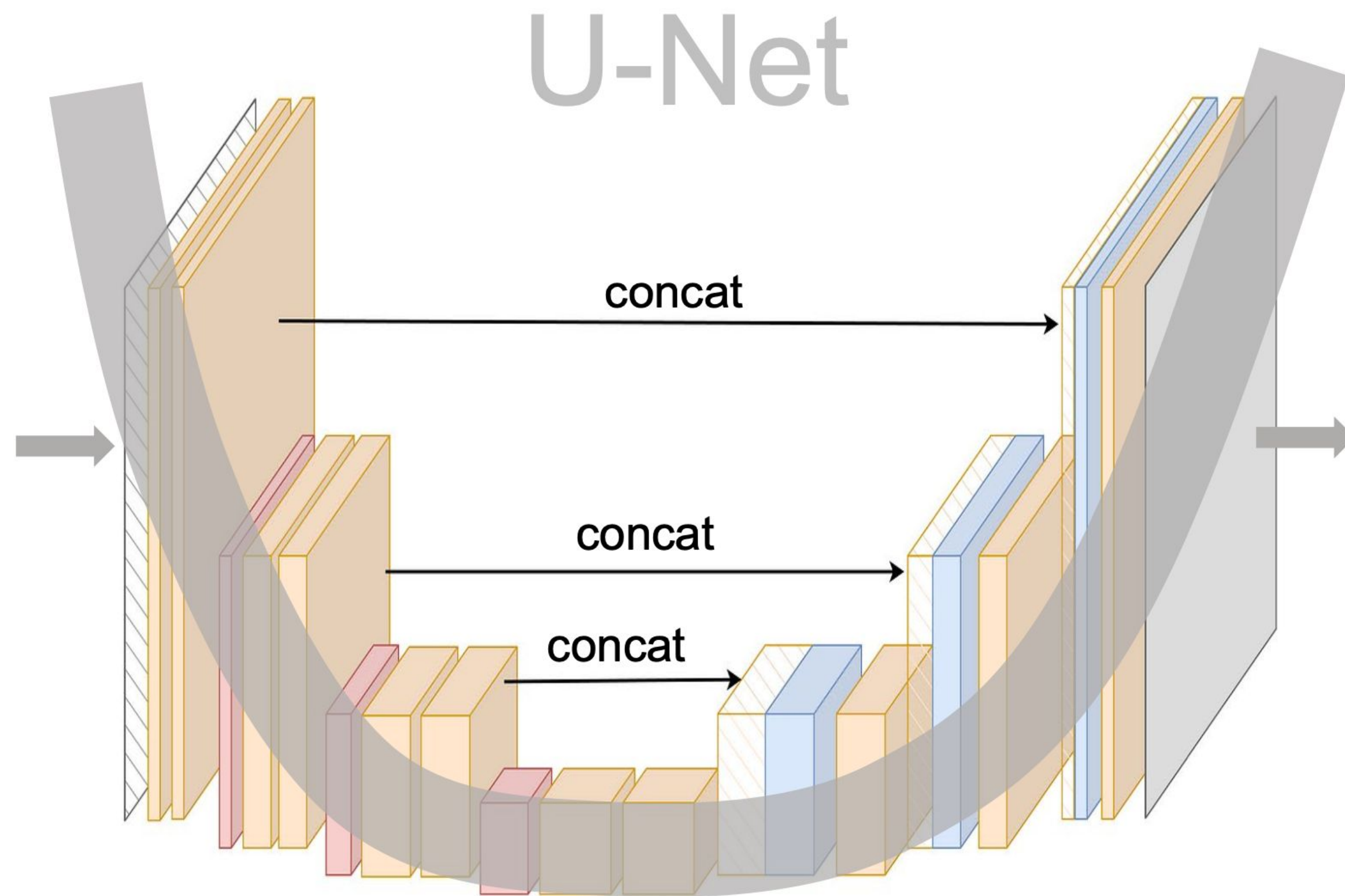
From AE to U-Net: Fully convolutional



From AE to U-Net: Fully convolutional

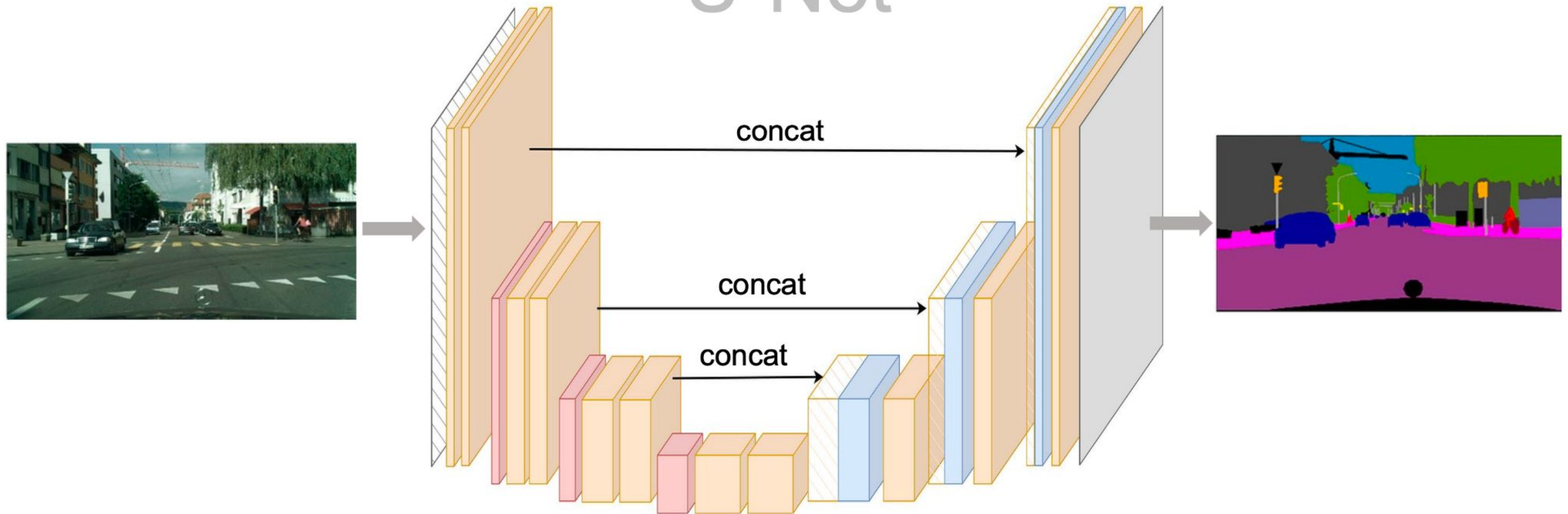


From AE to U-Net : Fully convolutional



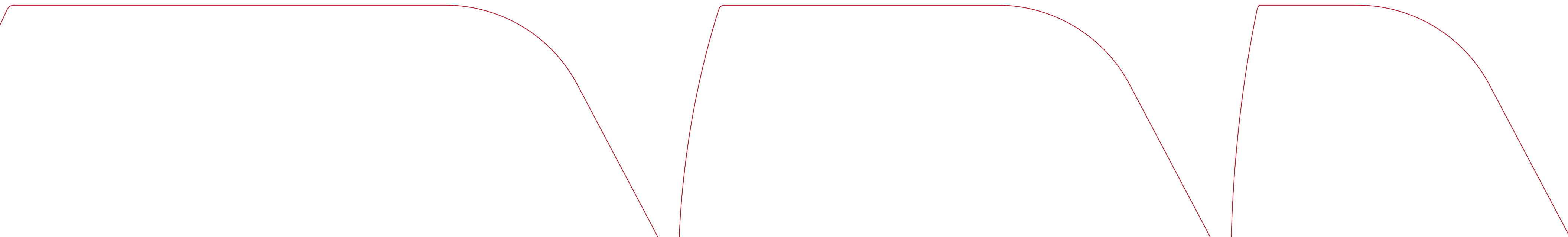
From AE to U-Net : Fully convolutional

U-Net



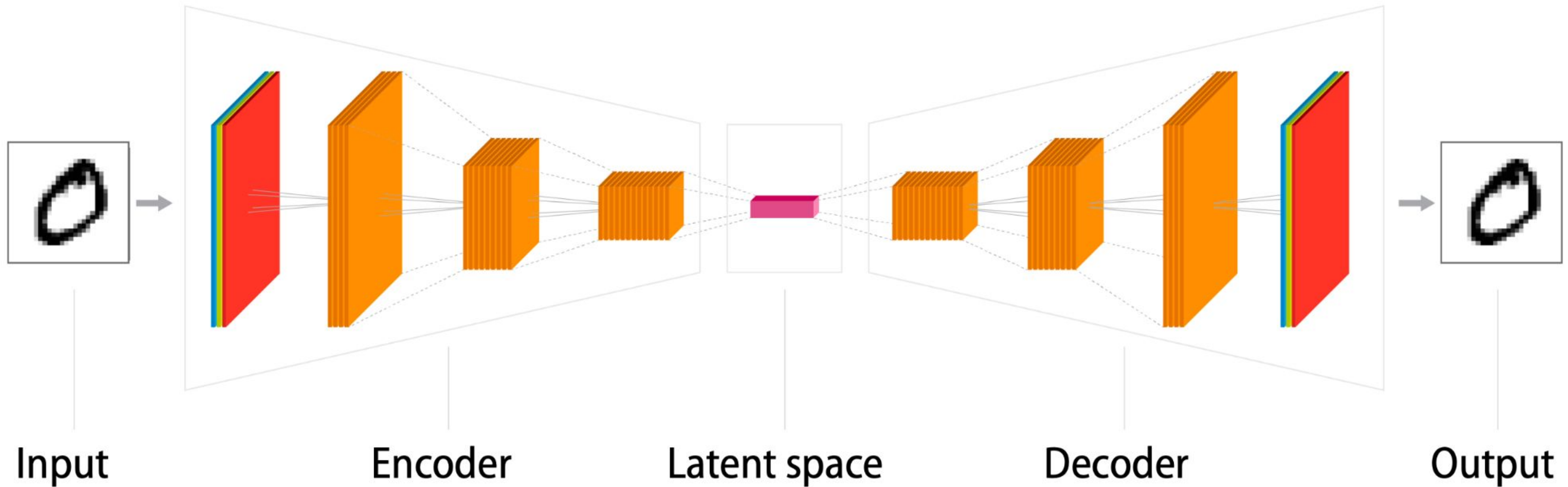


Variational Autoencodeur (VAE)



Autoencoders

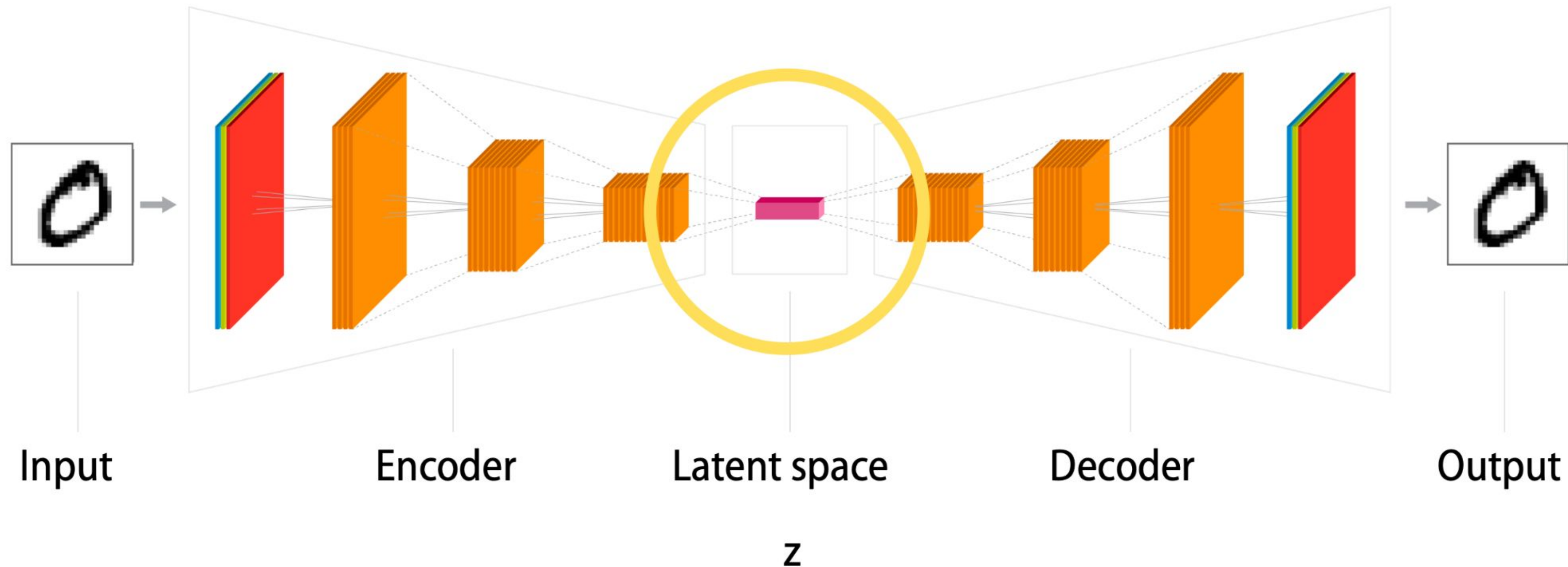
Autoencoders are trained to { minimise reconstruction errors.
retain essential data



Autoencoders


3

An autoencoders network have ~~2~~ parts :

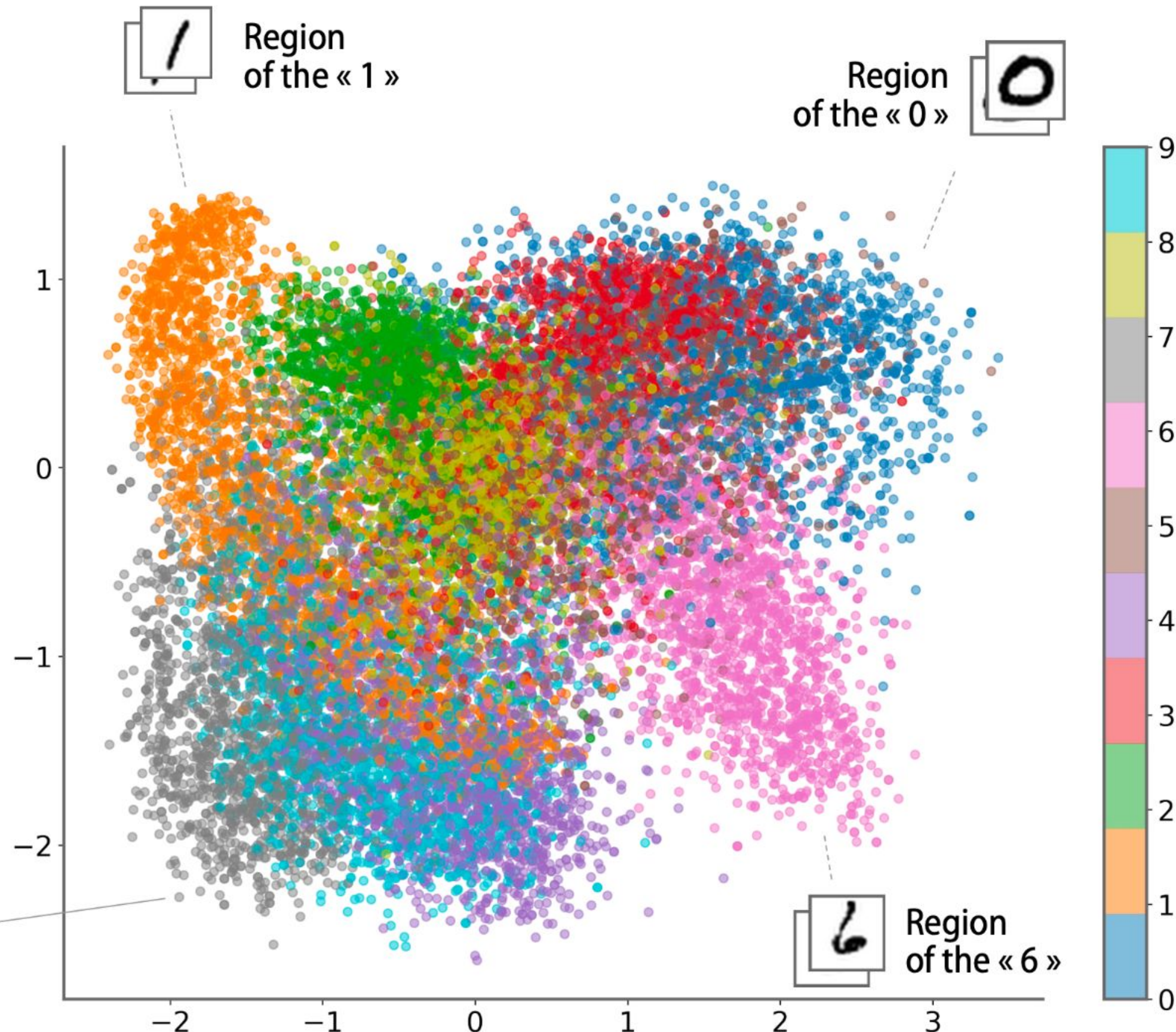


Autoencoders


Example of
MNIST
dataset
distribution
in its latent
space :

 Only two
dimensions
are represented
in abscissa and
ordinate

$z = \text{encoder}(\text{inputs})$



Clusters
appear, but
many of them
are nested or
very spread out

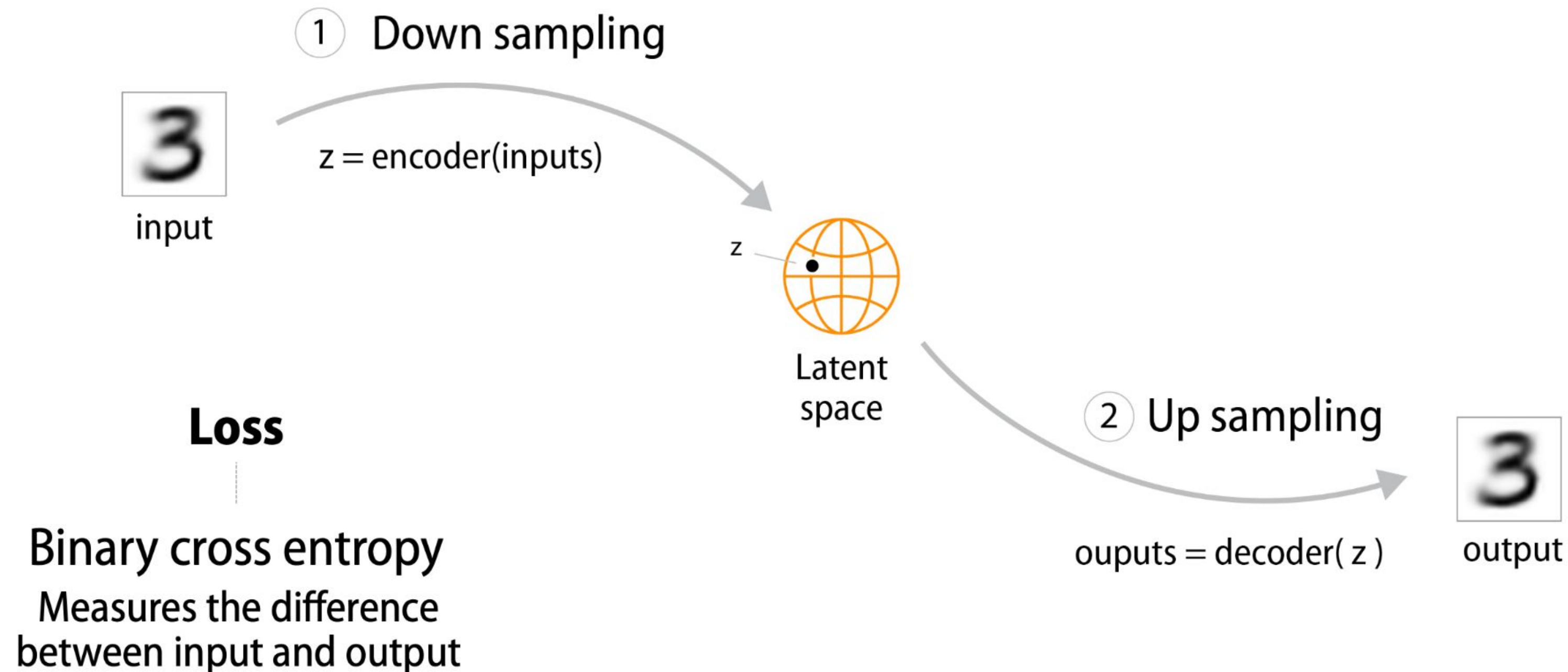
 How can
we make our
network better
separate the
different
clusters?

Autoencoders

z vector in latent space



An autoencoder performs a direct projection into the latent space and an upsampling from this latent space.



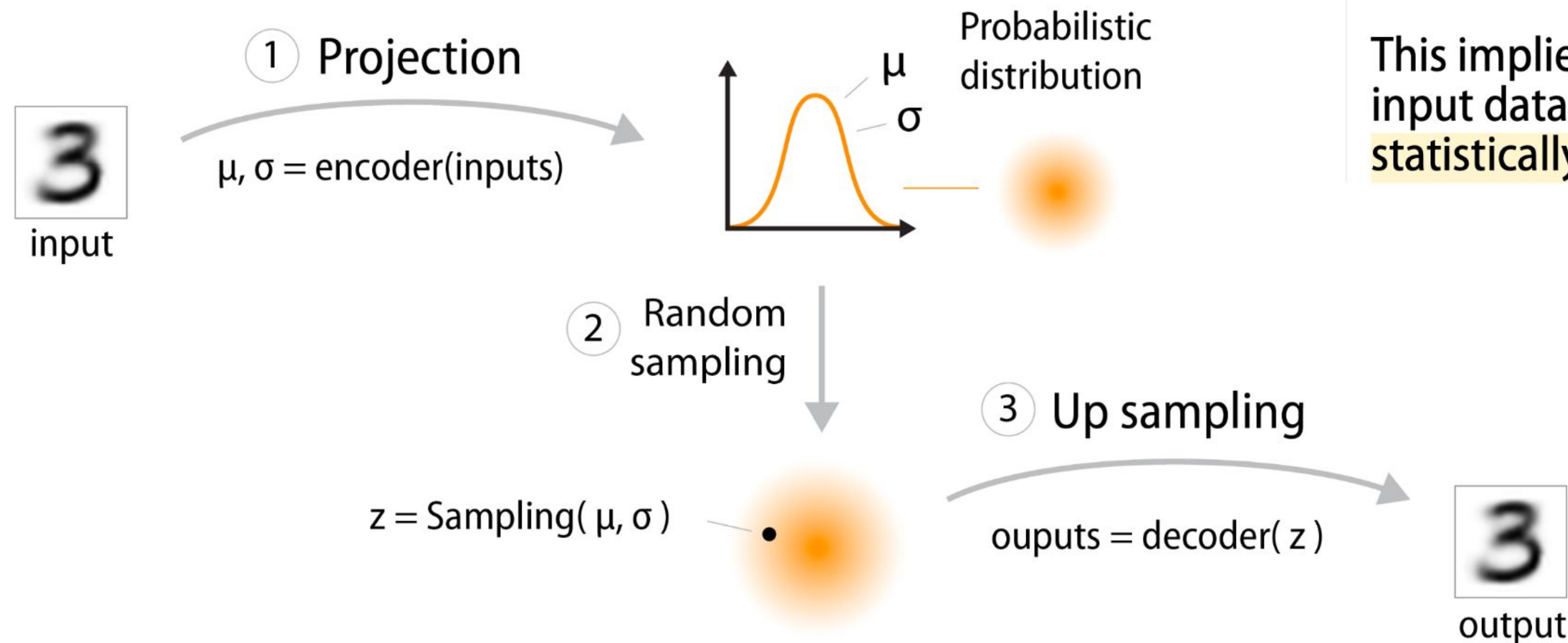
Variational Autoencoder (VAE)

μ is a mean
 σ is a variance
 z vector in latent space

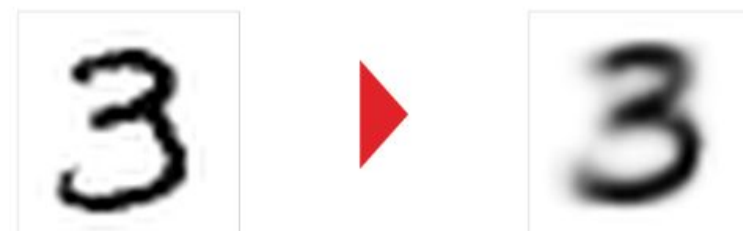
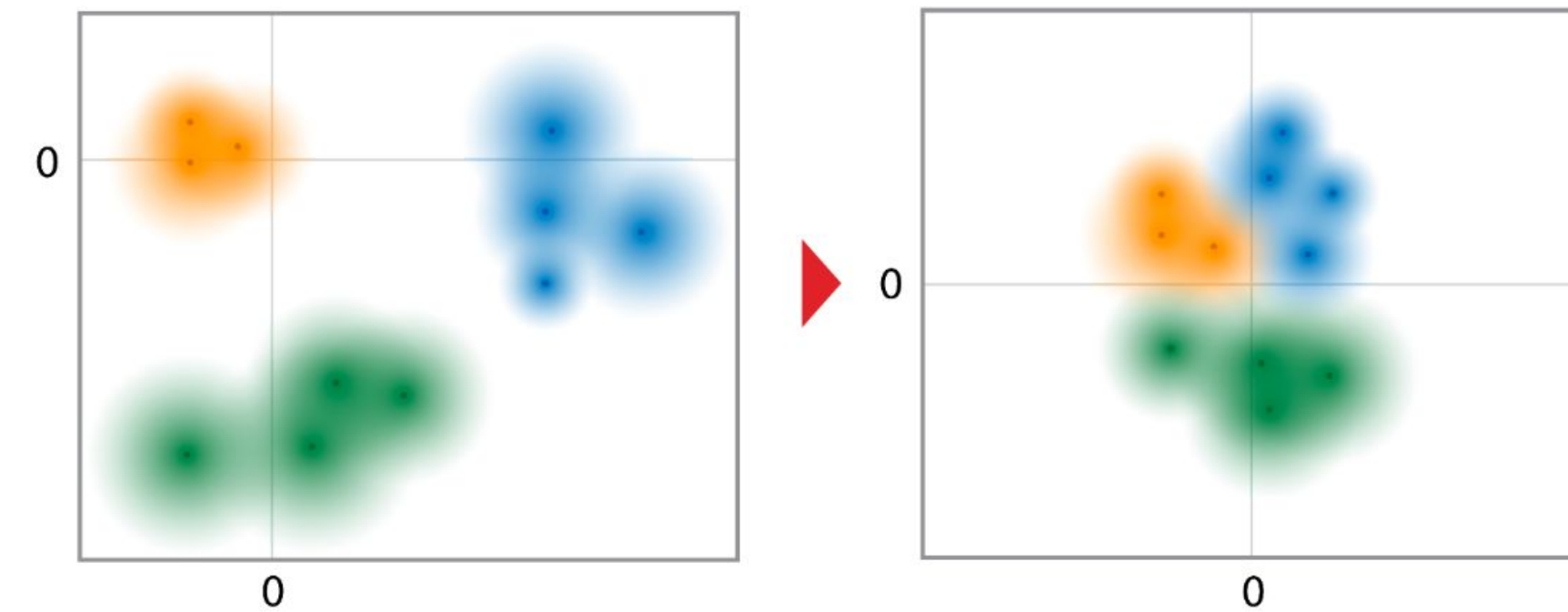


The stochastic approach encourages the network to generate relevant output throughout the distribution space.

This implies that the input data are statistically distributed.



Variational Autoencoder (VAE)



Reconstruction loss

Binary cross entropy
Measures the difference between input and output

Kullback-Leibler divergence*

$$D_{\text{KL}} = \frac{1}{2} \sum_{i=1}^k (\sigma_i^2 + \mu_i^2 - 1 - \ln(\sigma_i^2)) .$$

KL divergence between two (probability) distributions measures how much they diverge from each other

KL loss

Total loss

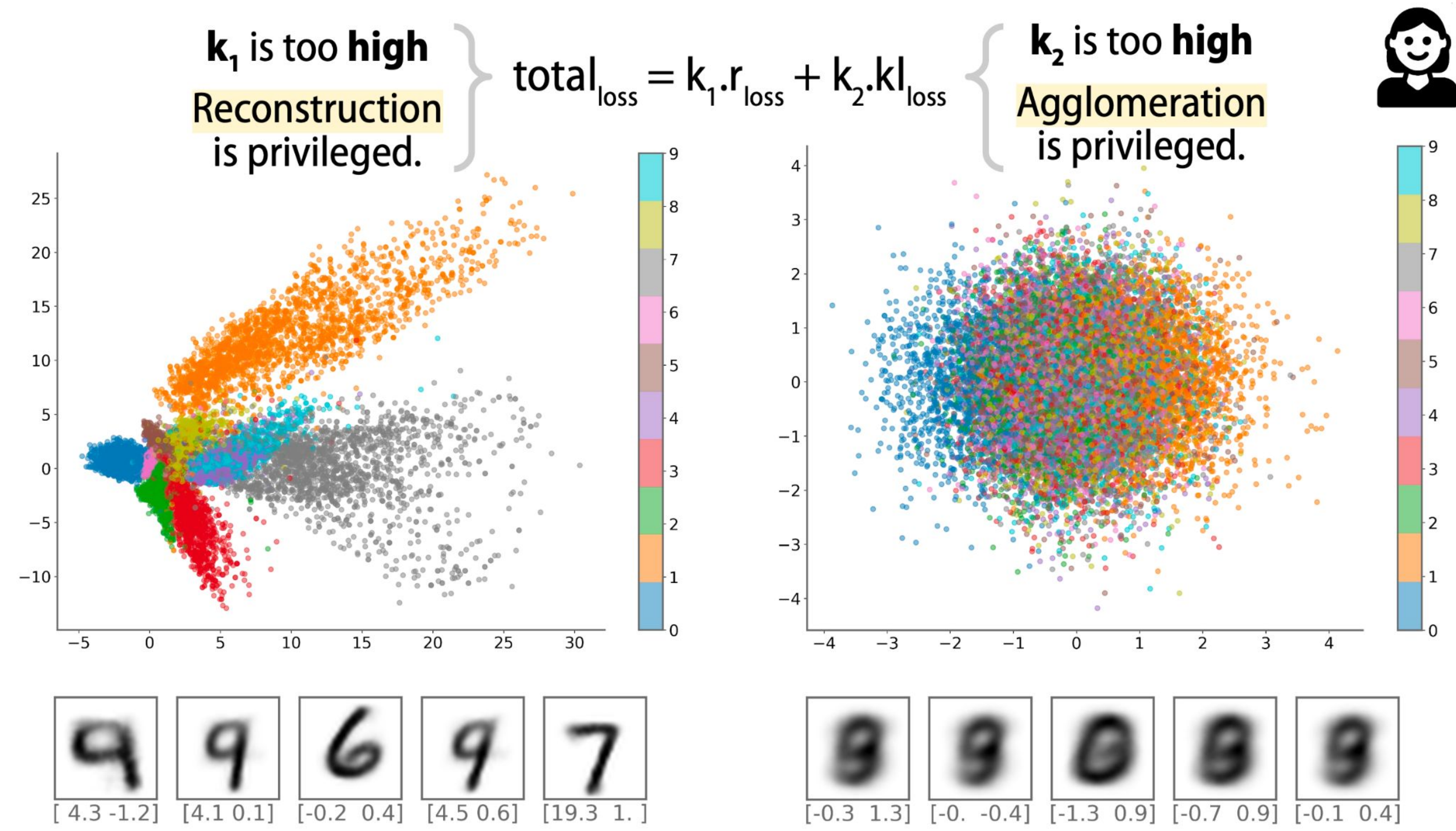
$$\text{total}_{\text{loss}} = k_1 \cdot r_{\text{loss}} + k_2 \cdot \text{kl}_{\text{loss}}$$



The trick is to find a nice compromise between these two components of the loss function.

(*) Special case for a standard normal distribution

Variational Autoencoder (VAE)



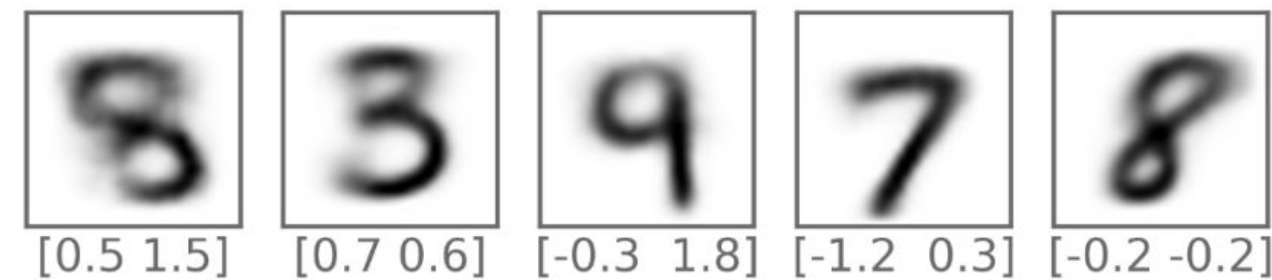
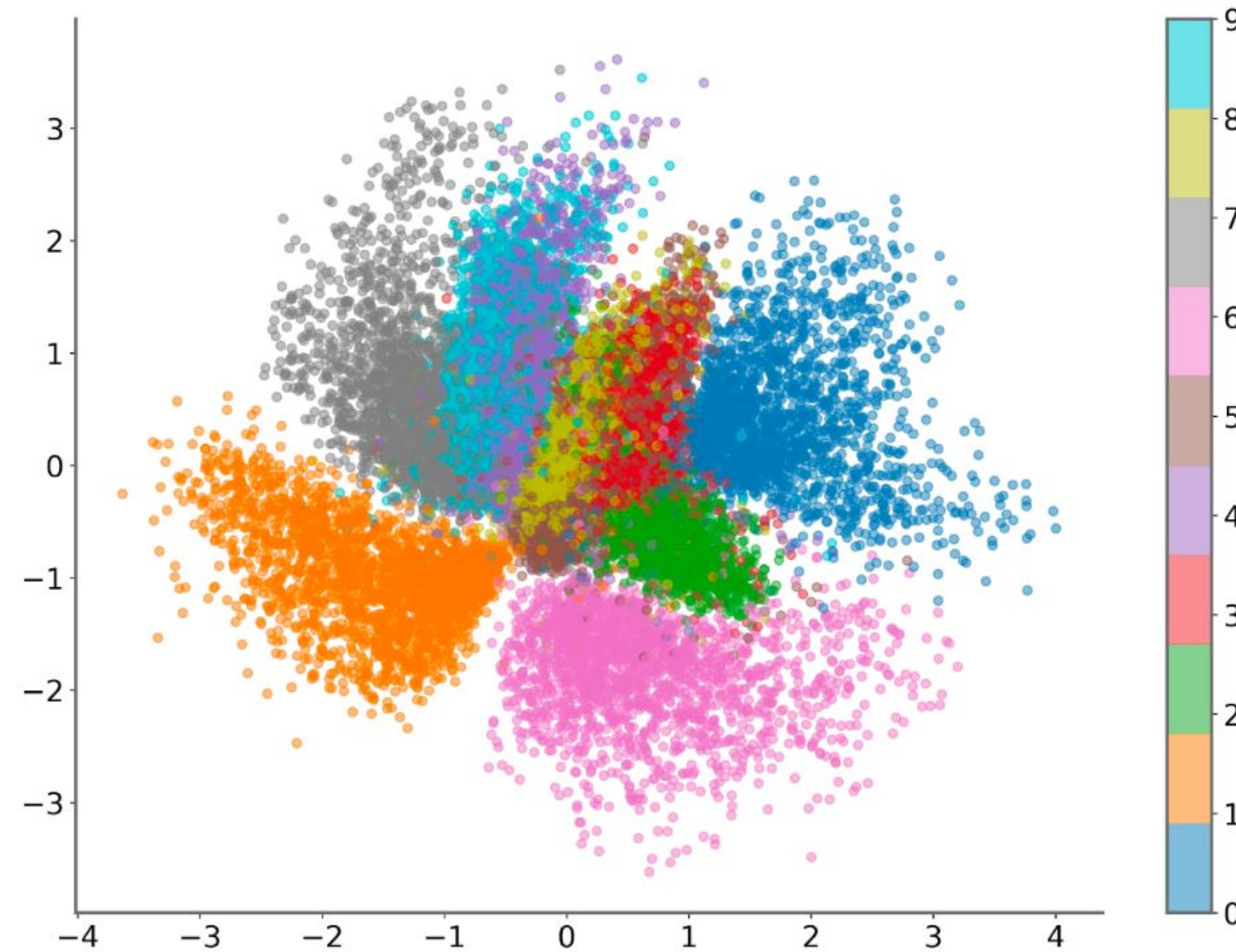
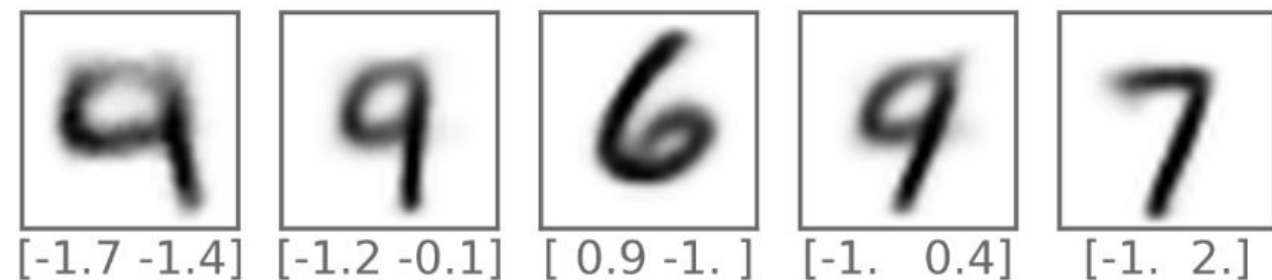
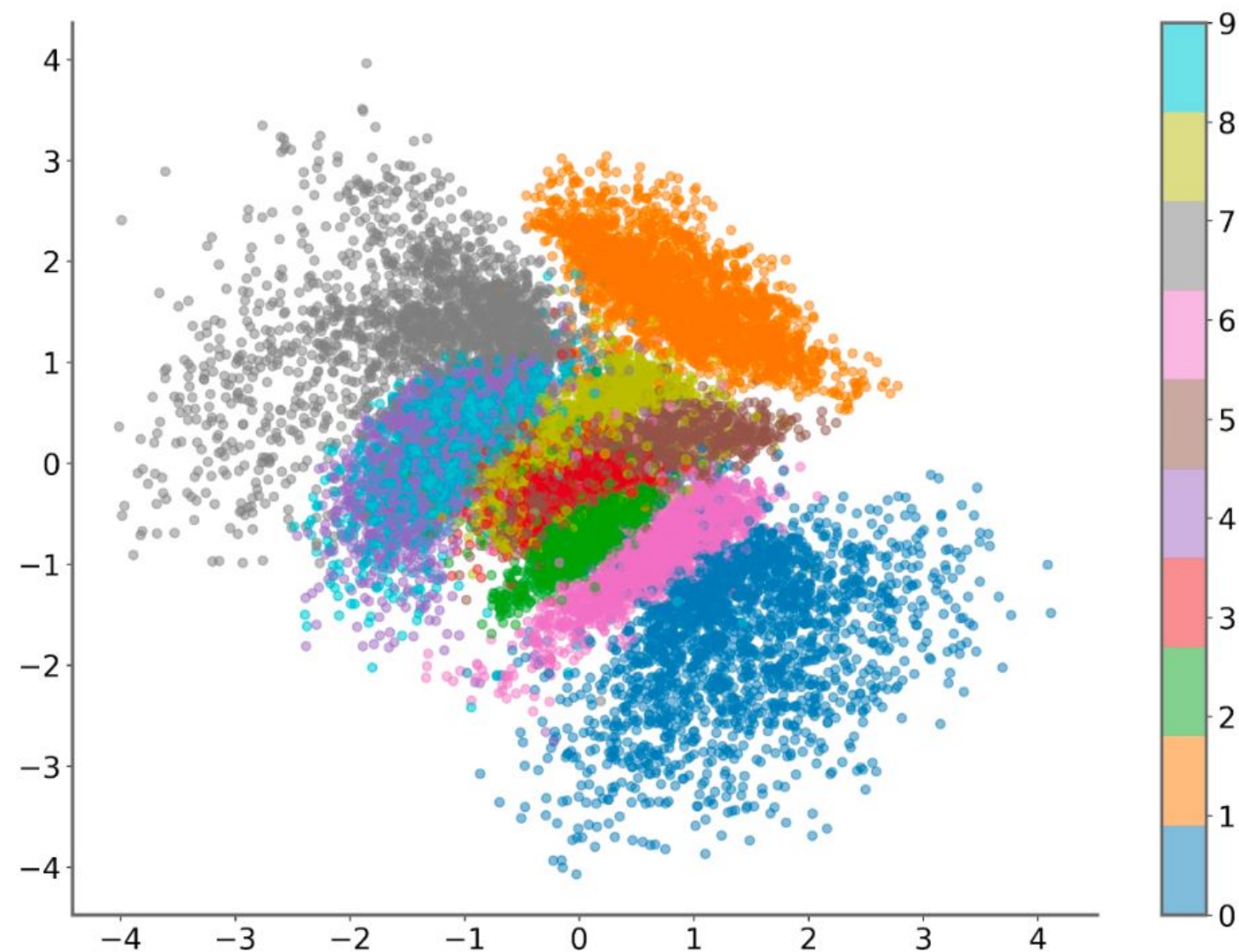
Variational Autoencoder (VAE)



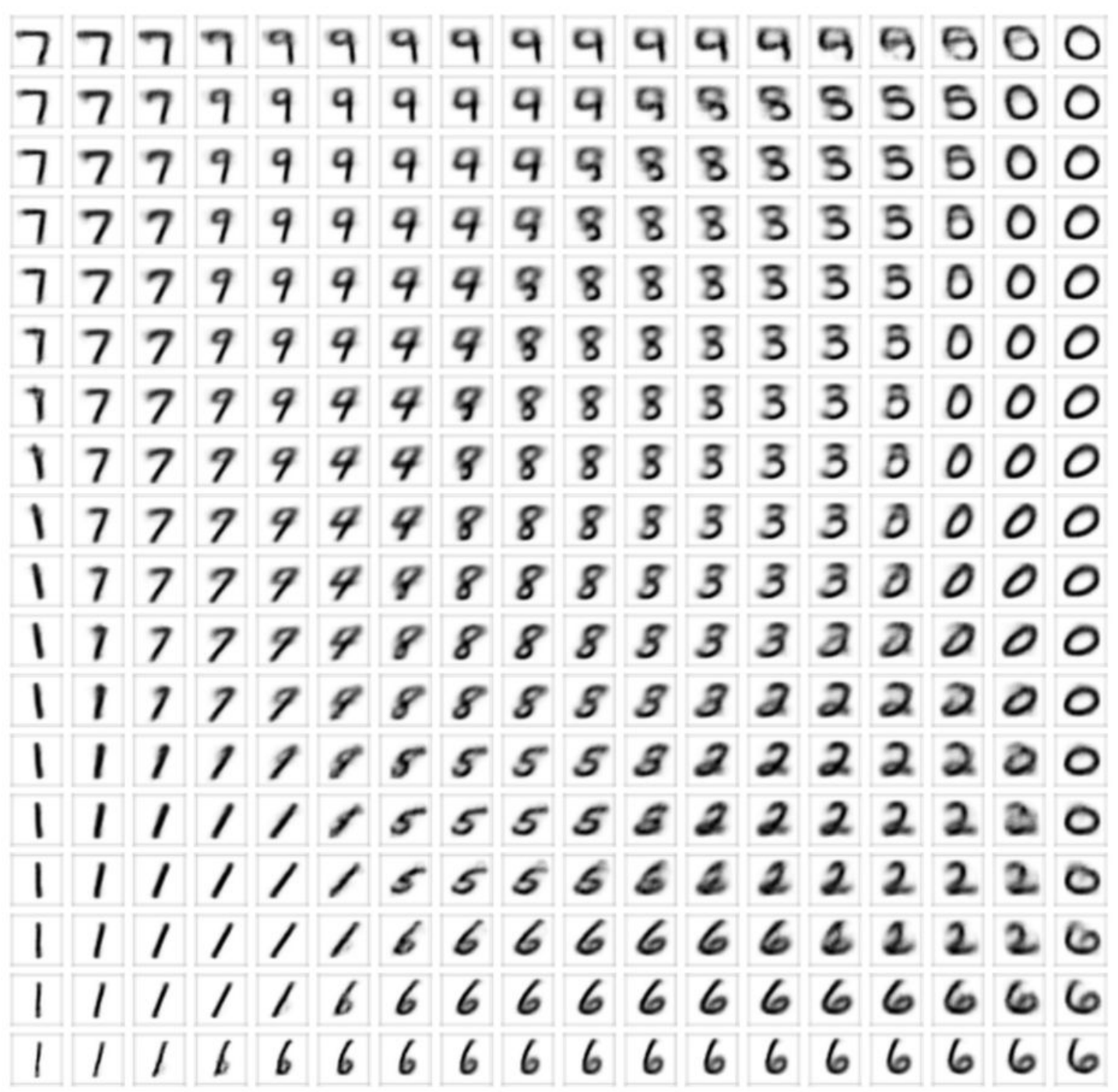
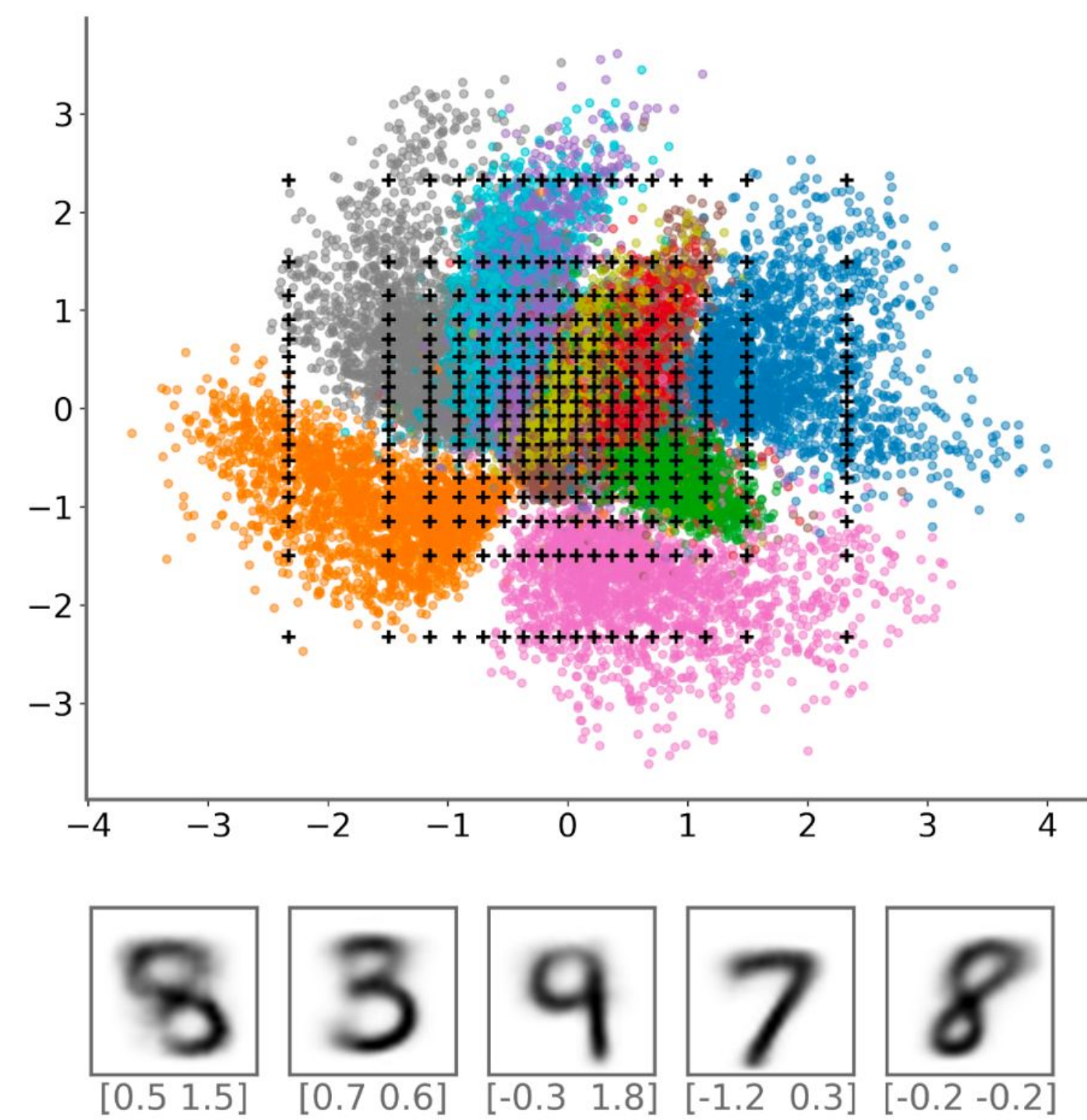
VAE2, with :
 $k_1, k_2 = [1, 5e-4]$

$$\left. \begin{array}{l} \text{VAE2, with :} \\ k_1, k_2 = [1, 5e-4] \end{array} \right\} \text{total}_{\text{loss}} = k_1 \cdot r_{\text{loss}} + k_2 \cdot \text{kl}_{\text{loss}}$$

VAE2, with :
 $k_1, k_2 = [1, 1e-3]$



Variational Autoencoder (VAE)

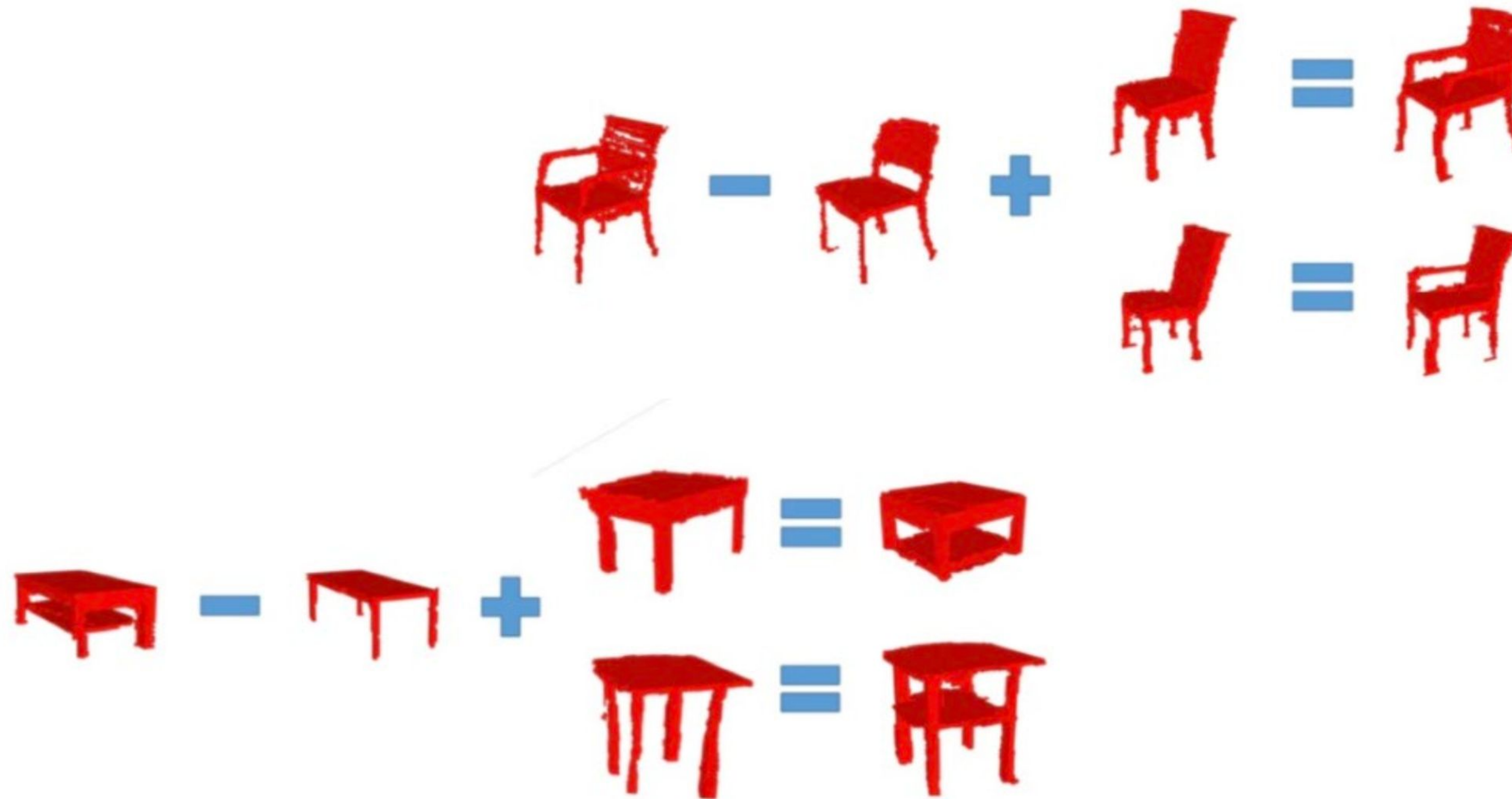


Tha's great ! We can generate data in profusion !!!

Latent space mixing

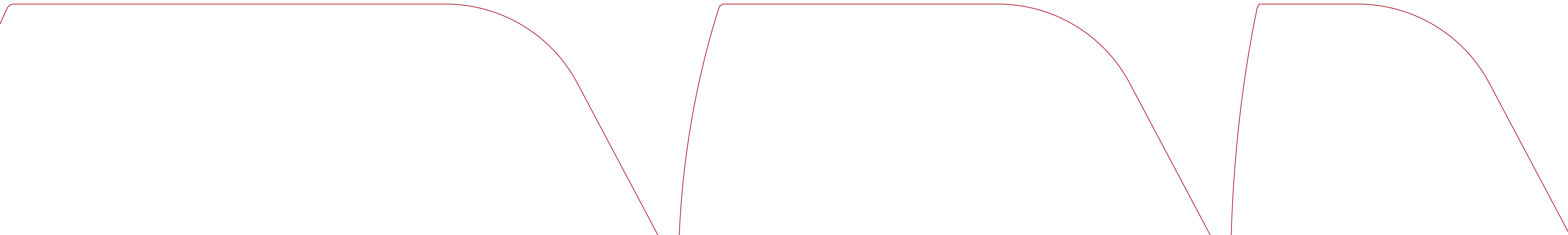


Latent space arithmetic

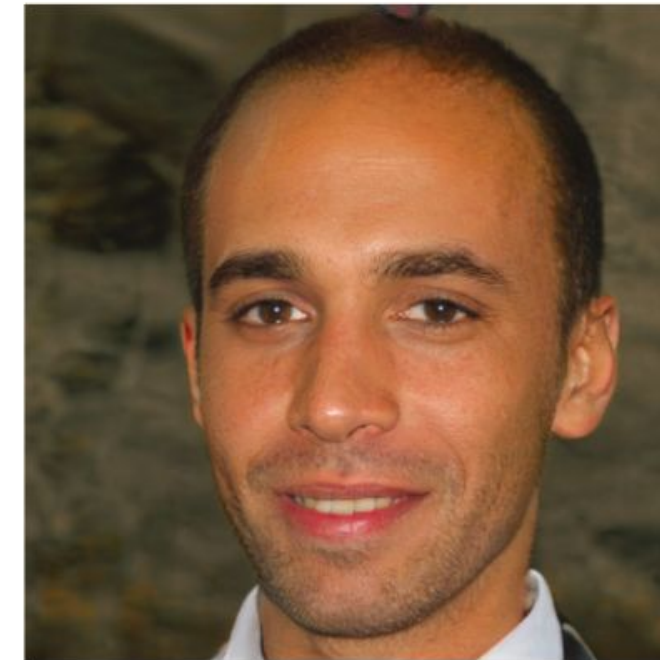
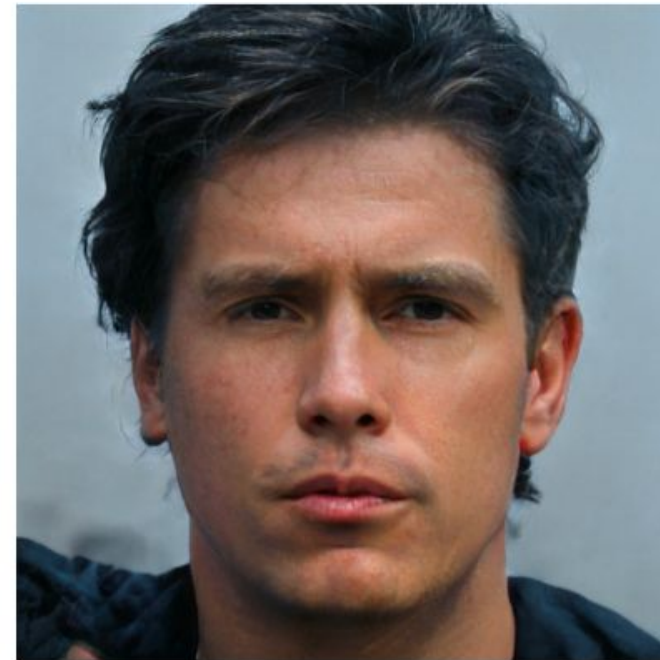




Generative Adversarial Networks (GAN)



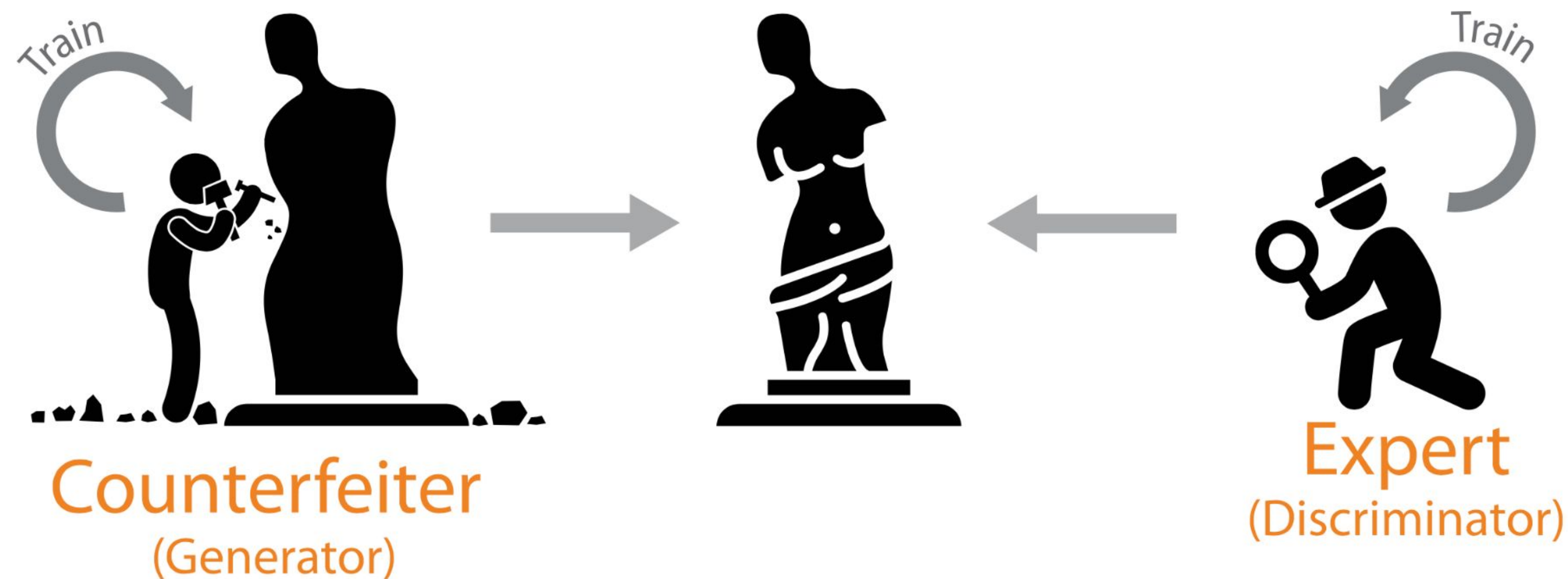
Which face is real?



<https://www.whichfaceisreal.com>

Sophie J. Nightingale and Hany Farid, « AI-synthesized faces are indistinguishable from real faces and more trustworthy », PNAS, <https://doi.org/10.1073/pnas.2120481119>, February 2022

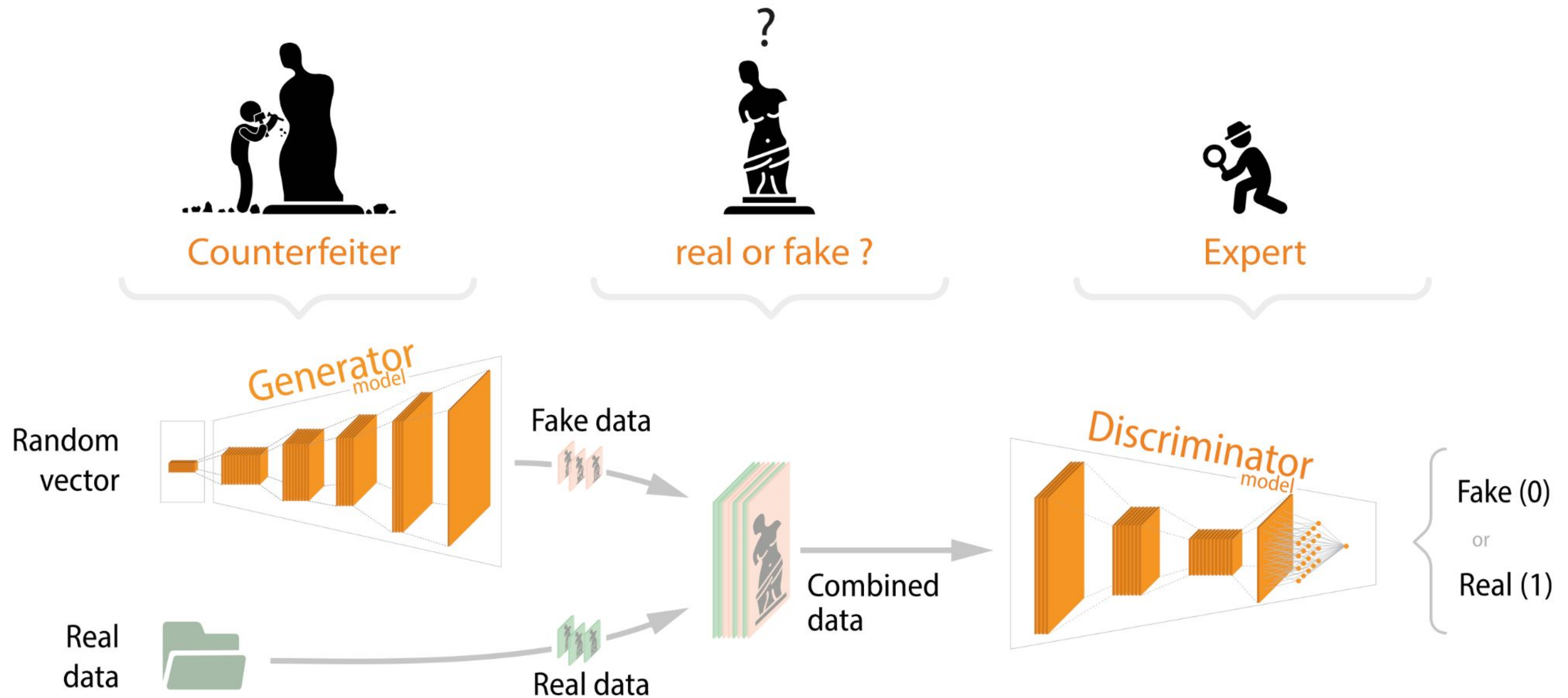
« (...) synthetically generated faces are more trustworthy than real faces. This may be because synthesized faces tend to look more like average faces which themselves are deemed more trustworthy. (...) »



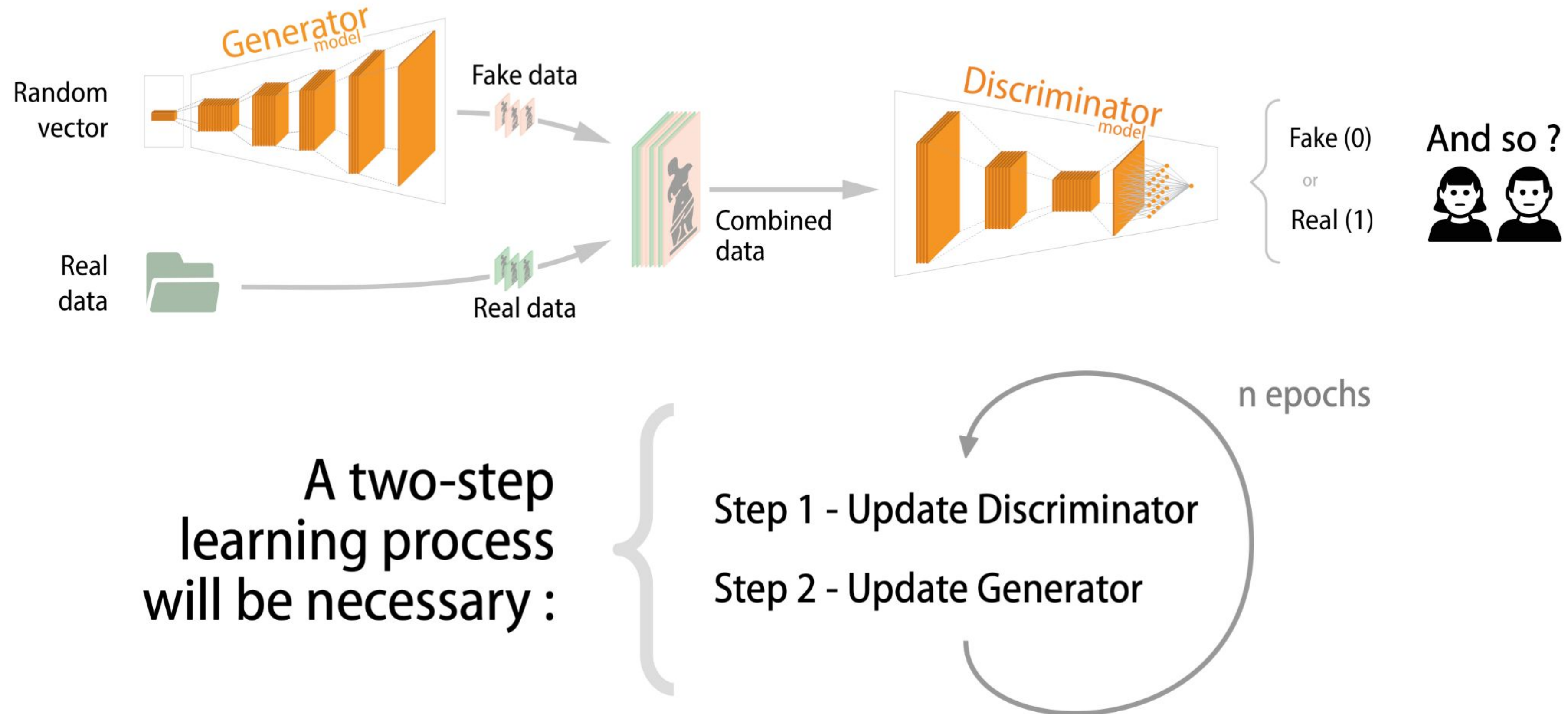
Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville et Yoshua Bengio, « Generative Adversarial Networks », in Advances in Neural Information Processing Systems 27, 2014

<https://arxiv.org/abs/1406.2661>

Principle



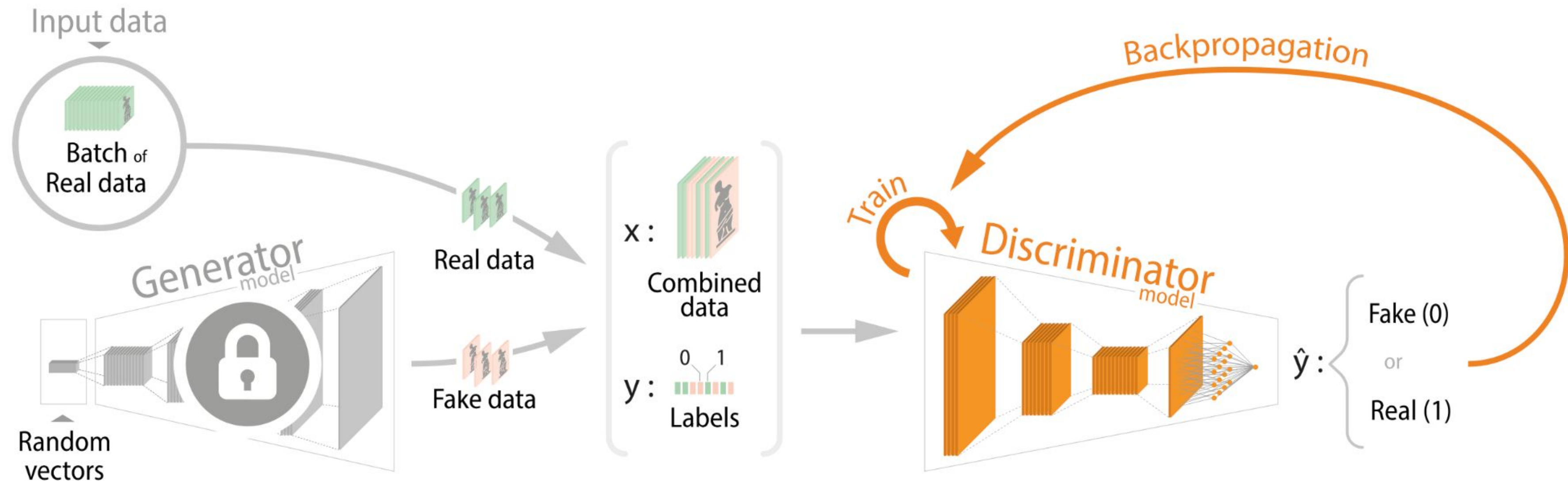
How to train a GAN ?



How to train a GAN ?

Step1 - Update Discriminator

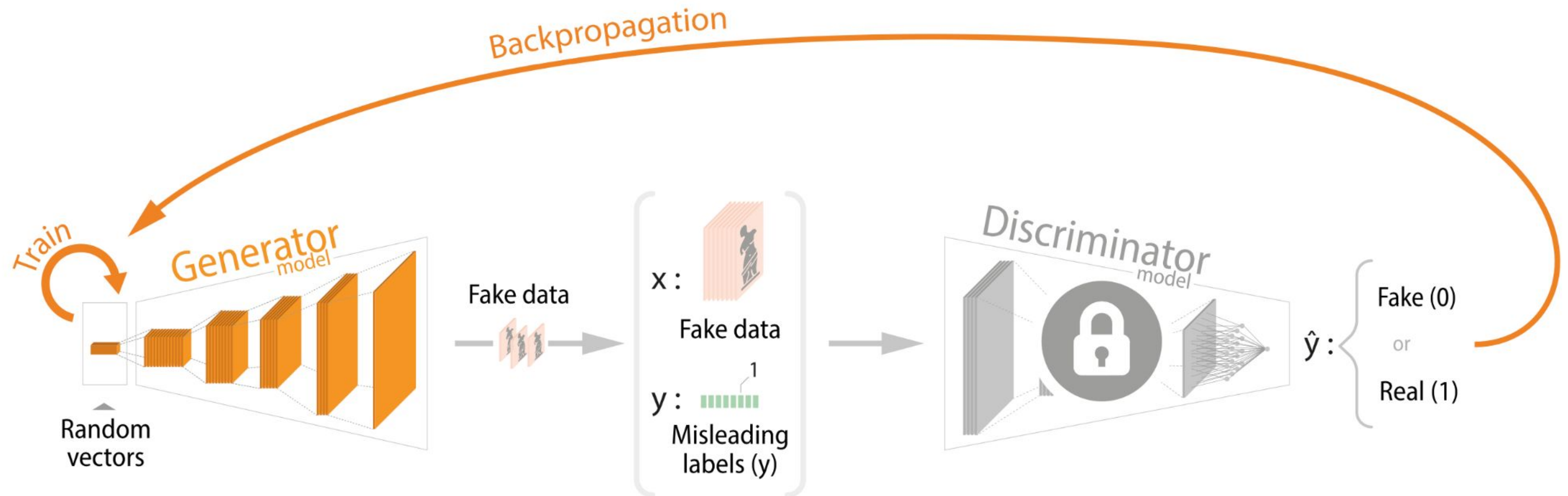
Generator is locked



How to train a GAN ?

Step2 - Update Generator

Discriminator is locked



Original GAN 2014

Categorical crossentropy : $H(y, \hat{y}) = -\frac{1}{n_{\text{obs}}} \sum_{i=1}^{n_{\text{obs}}} \sum_{c=1}^{n_{\text{class}}} y_c \cdot \log \hat{y}_c$

$$L_D = \frac{1}{m} \sum_{i=1}^m \left[\log(D(x^{(i)})) + \log(1 - D(G(z^{(i)}))) \right]$$
$$L_G = \frac{1}{m} \sum_{i=1}^m \left[\log(1 - D(G(z^{(i)}))) \right]$$

GAN loss functions

- $x^{(i)}$ is a real data from a set of m values
- $z^{(i)}$ is a latent vector from a set of m values
- $D(x)$ Output of the discriminator for a real image x
ie : probability that a real image is considered as real.
- $G(z^{(i)})$ Output of the generator from an input z, from latent space
so, $G(z)$ look like an X image, but is really fake...
- $D(G(z^{(i)}))$ Output of the discriminator for a fake image.
ie: probability that a fake image is considered as real.

Discriminator will try to **maximize** L_D :

$$\forall x \in X, D(x) \approx 1$$

$$\forall z \in Z, D(G(z)) \approx 0$$

Generator will try to **minimize** L_G :

$$\forall z \in Z, D(G(z)) \approx 1$$

Where : $\begin{cases} \text{fake} : 0 \\ \text{real} : 1 \end{cases}$

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, « Generative Adversarial Networks », <https://arxiv.org/abs/1406.2661>, 2014

Original GAN 2014

Categorical crossentropy : $H(y, \hat{y}) = -\frac{1}{n_{\text{obs}}} \sum_{i=1}^{n_{\text{obs}}} \sum_{c=1}^{n_{\text{class}}} y_c \cdot \log \hat{y}_c$

$$L_D = \frac{1}{m} \sum_{i=1}^m \left[\log(D(x^{(i)})) + \log(1 - D(G(z^{(i)}))) \right]$$
$$L_G = \frac{1}{m} \sum_{i=1}^m \left[\log(1 - D(G(z^{(i)}))) \right]$$

GAN loss functions



Limitations :

Gradient vanishing problem.
Risk of blocking in the early stages of
GAN learning, when the discriminator
work is very easy.

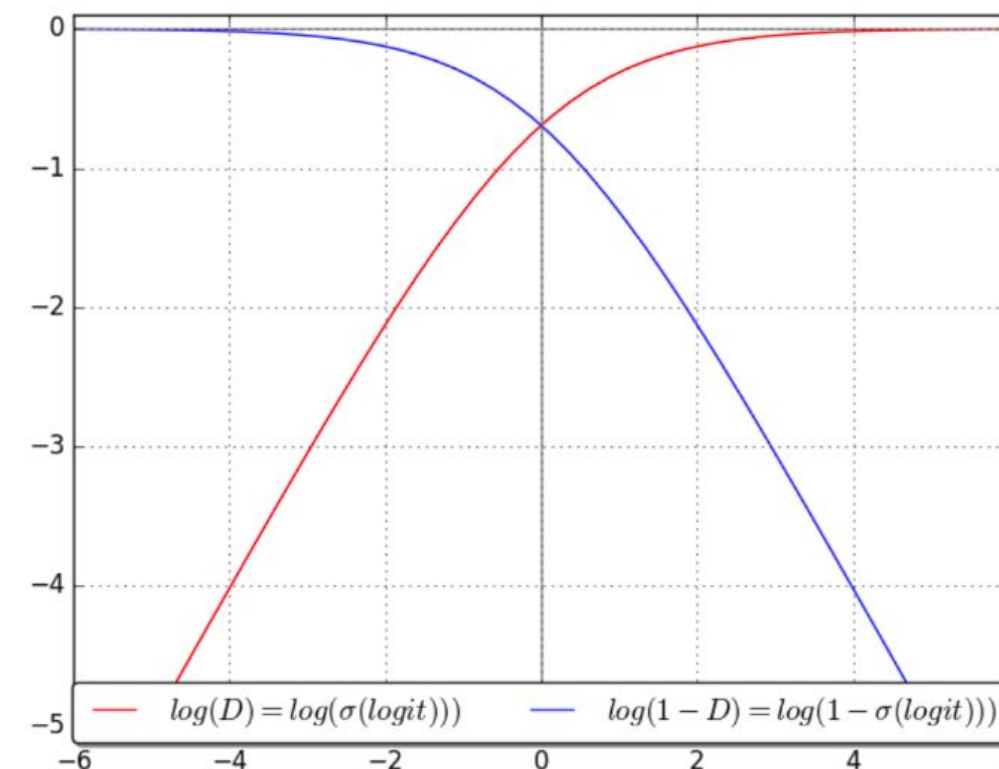


Figure 6: Illustration of vanishing gradient in the negative quadrant when using the original loss formulation $\log(1-D)$ (blue curve). The x-axis is D's logit (output of last layer before sigmoid activation). Minimising $\log(1-D)$ yields the same solution as maximising $\log(D)$ (red curve) but the red curve exhibits stronger gradients.

<https://developer.nvidia.com/blog/photo-editing-generative-adversarial-networks-1/>

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, « Generative Adversarial Networks », <https://arxiv.org/abs/1406.2661>, 2014

Wasserstein GAN (2017)

→ Proposition of a more theoretical approach to the learning strategy [1]:



What is the best way to measure the distance between two distributions ?

Optimization problem [2]

Total Variation distance
Kullback-Leibler divergence
Jensen-Shannon divergence
Earth-Mover Distance 2] / Wasserstein-1

Distance
du terrassier

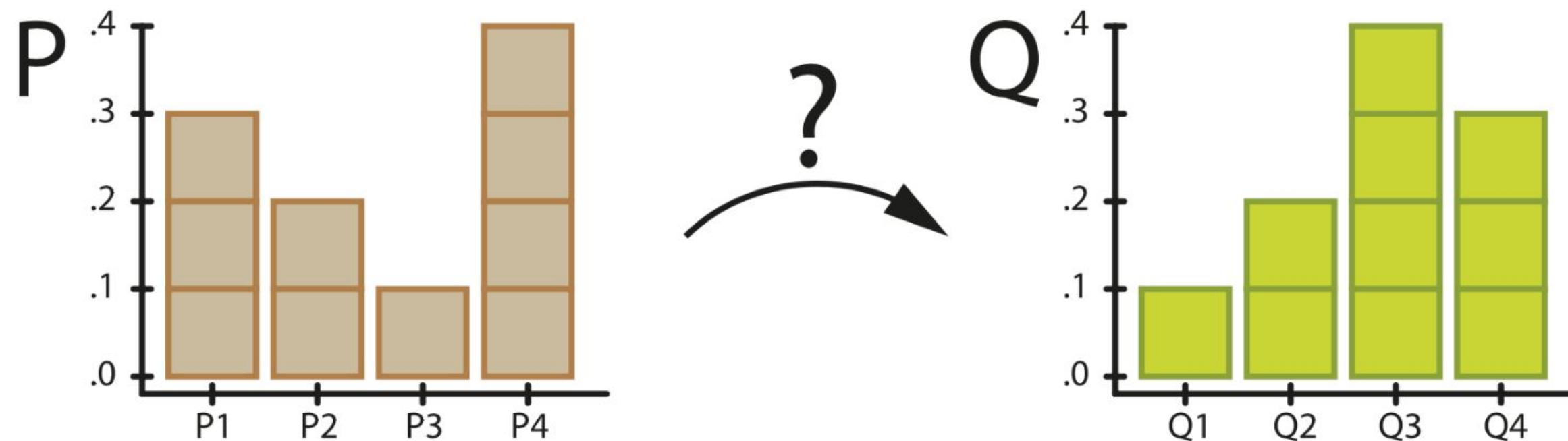
[1] Martin Arjovsky, Soumith Chintala, Léon Bottou, « Wasserstein GAN », <https://arxiv.org/abs/1701.07875>, 2017

[2] Cédric Villani, « Optimal transport, old and new », Springer, 2008
https://cedricvillani.org/sites/dev/files/old_images/2012/08/preprint-1.pdf
<https://www.youtube.com/watch?v=zo46TEp6FB8>

Wasserstein GAN: Earth Moving Distance

Objective is to measuring the distance between 2 distributions

This distance can be interpreted by the minimal energy needed to move from one box distribution to another:



This energy can be calculated as:

$$W = \text{Number of boxes moved} \times \text{Moving distance of the boxes}$$

Wasserstein GAN: Earth Moving Distance

We can write :

$$W = \sum_i^{\text{class}} |\delta_i|$$

$$\text{with } \begin{cases} \delta_0 = 0 \\ \delta_{i+1} = \delta_i + P_i - Q_i \end{cases}$$

$$\delta_0 = 0$$

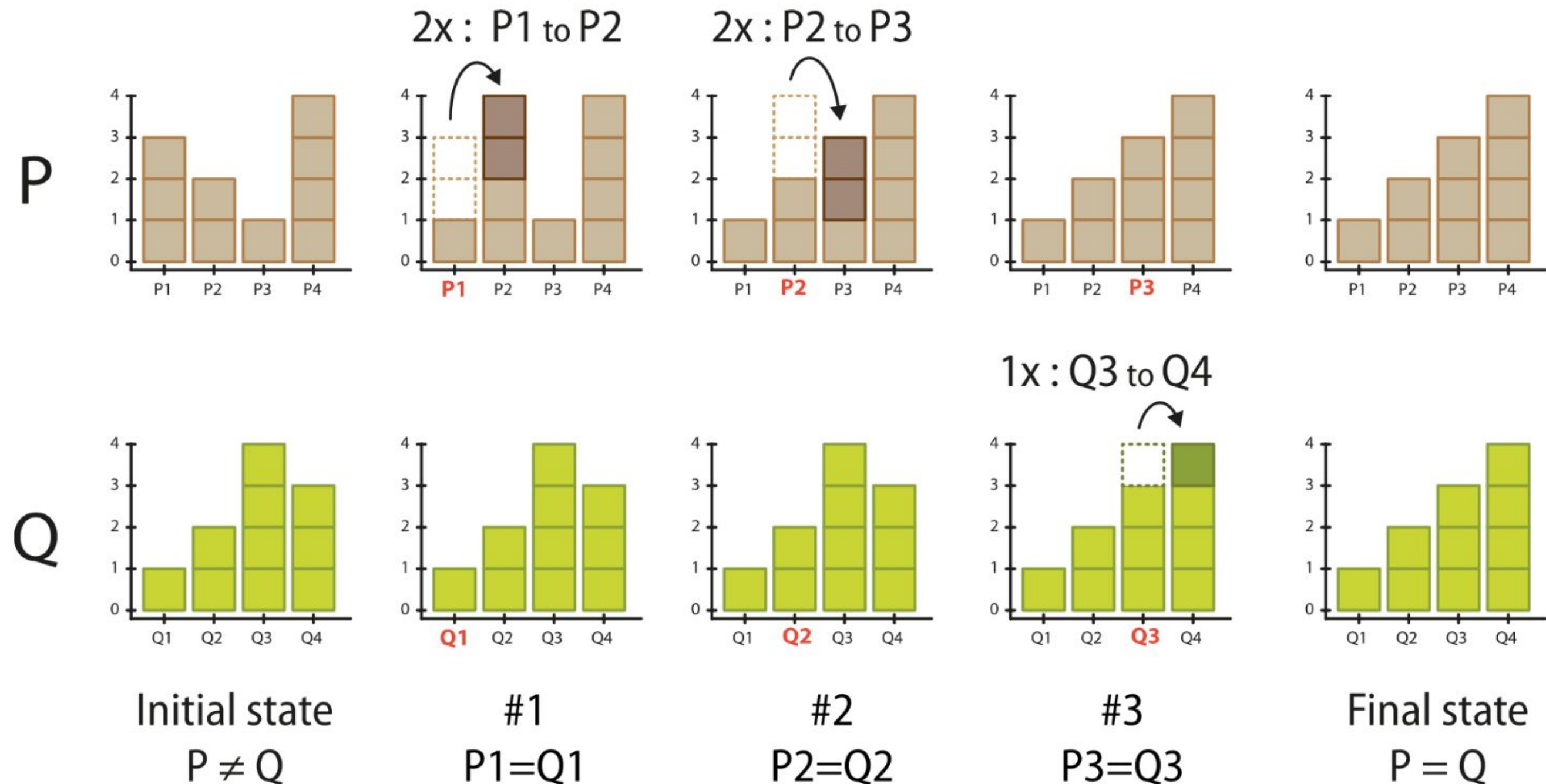
$$\delta_1 = 0 + 3 - 1 = 2$$

$$\delta_2 = 2 + 2 - 2 = 2$$

$$\delta_3 = 2 + 1 - 4 = -1$$

$$\delta_4 = -1 + 4 - 3 = 0$$

$$W = \sum_{i=1}^4 |\delta_i| = 5$$



Wasserstein GAN

→ Wasserstein GAN :

$$L_{\text{critic}} = \frac{1}{m} \sum_{i=1}^m \left[D(x^{(i)}) - D(G(z^{(i)})) \right]$$

$$L_{\text{generator}} = \frac{1}{m} \sum_{i=1}^m \left[D(G(z^{(i)})) \right]$$

WGAN
loss functions

- $x^{(i)}$ is a real data from a set of m values
- $z^{(i)}$ is a latent vector from a set of m values
- $D(x)$ Output of the discriminator for a real image x
ie : probability that a real image is considered as real.
- $G(z^{(i)})$ Output of the generator from an input z , from latent space
so, $G(z)$ look like an X image, but is really fake...
- $D(G(z^{(i)}))$ Output of the discriminator for a fake image.
ie: probability that a fake image is considered as real.

Critic / Discriminator

try to **maximize** L_{critic}

$$\forall x \in X, \forall z \in Z$$

$$D(x) \gg D(G(z))$$

Generator

try to **maximize** $L_{\text{generator}}$

Note : $\begin{cases} \text{fake : low score} \\ \text{real : high score} \end{cases}$

Martin Arjovsky, Soumith Chintala, Léon Bottou, « Wasserstein GAN », <https://arxiv.org/abs/1701.07875>, 2017

Wasserstein GAN

→ Wasserstein GAN :

$$L_{\text{critic}} = \frac{1}{m} \sum_{i=1}^m \left[D(x^{(i)}) - D(G(z^{(i)})) \right]$$

$$L_{\text{generator}} = \frac{1}{m} \sum_{i=1}^m \left[D(G(z^{(i)})) \right]$$

WGAN
loss functions

Critic / Discriminator
try to **maximize** L_{critic}

$$\forall x \in X, \forall z \in Z \\ D(x) \gg D(G(z))$$

Generator
try to **maximize** $L_{\text{generator}}$



According to the authors, the discriminator should **no longer be considered as a classifier** but more as a **critic**. The output of the discriminator will be greater for true instances than for false instances.

Concretely, a critic has **no sigmoid** function at its output

Note : $\begin{cases} \text{fake : low score} \\ \text{real : high score} \end{cases}$

Martin Arjovsky, Soumith Chintala, Léon Bottou, « Wasserstein GAN », <https://arxiv.org/abs/1701.07875>, 2017

Wasserstein GAN

Wasserstein GAN 2017 dec. 6

k-Lipschitz function :

$$\forall (x, y) \in E^2, |f(x) - f(y)| \leq k |x - y|$$

→ Wasserstein GAN :

$$L_{\text{critic}} = \frac{1}{m} \sum_{i=1}^m [D(x^{(i)}) - D(G(z^{(i)}))]$$

$$L_{\text{generator}} = \frac{1}{m} \sum_{i=1}^m [D(G(z^{(i)}))]$$

WGAN
loss functions

Critic / Discriminator
try to **maximize** L_{critic}

$$\forall x \in X, \forall z \in Z \\ D(x) \gg D(G(z))$$

Generator
try to **maximize** $L_{\text{generator}}$

! The calculation of the EMD distance is complex.
A simplified version is however possible if the discriminator is a **k-Lipschitz function**, implying a constraint on it.

The proposed solution is to clip the weights to contain them in a **limited interval** $[-c, c]$, with $c=0.01$

Note : $\begin{cases} \text{fake : low score} \\ \text{real : high score} \end{cases}$

Martin Arjovsky, Soumith Chintala, Léon Bottou, « Wasserstein GAN », <https://arxiv.org/abs/1701.07875>, 2017

Wasserstein GAN

→ Wasserstein GAN :

$$L_{\text{critic}} = \frac{1}{m} \sum_{i=1}^m \left[D(x^{(i)}) - D(G(z^{(i)})) \right]$$

$$L_{\text{generator}} = \frac{1}{m} \sum_{i=1}^m \left[D(G(z^{(i)})) \right]$$

WGAN
loss functions



Limitations :

« Weight clipping is a clearly terrible way to enforce a Lipschitz constraint (...) and we actively encourage interested researchers to improve on this method. » [2]



The calculation of the EMD distance is complex. A simplified version is however possible if the discriminator is a **k-Lipschitz function**, implying a constraint on it.

The **proposed solution** is to clip the weights to contain them in a **limited interval** $[-c, c]$, with $c=0.01$

Martin Arjovsky, Soumith Chintala, Léon Bottou, « Wasserstein GAN », <https://arxiv.org/abs/1701.07875>, 2017

Wasserstein GAN - GP

Wasserstein GAN - GP 2017 dec. 25

→ Wasserstein GAN with **Gradient Penalty** :

Here is 
the gradient
penalty

$$L_{\text{critic}} = \mathbb{E}_{\tilde{x} \sim \mathbb{P}_g} [D(\tilde{x})] - \mathbb{E}_{x \sim \mathbb{P}_r} [D(x)] + \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} [(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]$$
$$L_{\text{generator}} = \frac{1}{m} \sum_{i=1}^m [D(G(z^{(i)}))]$$

WGAN-GP
loss functions

x is a real data from a set of real values

$\tilde{x} = G(z)$ is a fake data, from the generator

\hat{x} mix fake / real data

λ penalty coefficient, proposed as $\lambda=10$

$z^{(i)}$ is a latent vector from a set of m values

$D(x)$ Output of the discriminator for a real image x

$G(z^{(i)})$ Output of the generator from an input z , from latent space

$D(G(z^{(i)}))$ Output of the discriminator for a fake image.

$$\hat{x} = \varepsilon x + (1 - \varepsilon)\tilde{x}$$


with $\varepsilon \sim U[0, 1]$

Critic / Discriminator
try to **minimize** L_{critic}

$$\forall x \in X, \forall z \in Z$$

$$D(x) \gg D(G(z))$$

Generator
try to **maximize** $L_{\text{generator}}$

The paper 
demonstrates that
the addition of the
penalty gradient
guarantees that the
loss function is 1-
Lipschitz.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville,
« Improved Training of Wasserstein GANs », <https://arxiv.org/abs/1704.00028>, 2017

GANs: a big family

Conditional GANs

Mehdi Mirza, Simon Osindero,
« Conditional Generative Adversarial Nets »,
<https://arxiv.org/abs/1411.1784>, 2014



Progressive GANs

Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen, « Progressive Growing of GANs for Improved Quality, Stability, and Variation »,
<https://arxiv.org/abs/1710.10196>, 2017



Image-to-Image Translation

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros,
« Image-to-Image Translation with Conditional Adversarial Networks »,
<https://arxiv.org/abs/1611.07004>, 2016



CycleGAN

Jun-Yan Zhu Taesung Park Phillip Isola Alexei A. Efros
« Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks »,
<https://arxiv.org/abs/1703.10593>, 2017



GANs: a big family

Text-to-Image Synthesis

Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas,
« StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks », <https://arxiv.org/abs/1612.03242>, 2016

« This smaller brown bird has white stripes on the coverts, wingbars and secondaries »



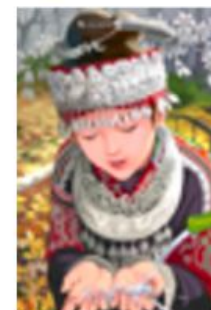
Semantic Image Inpainting

Raymond A. Yeh, Chen Chen, Teck Yian Lim, Alexander G. Schwing, Mark Hasegawa-Johnson, Minh N. Do,
« Semantic Image Inpainting with Deep Generative Models », <https://arxiv.org/abs/1607.07539>, 2016



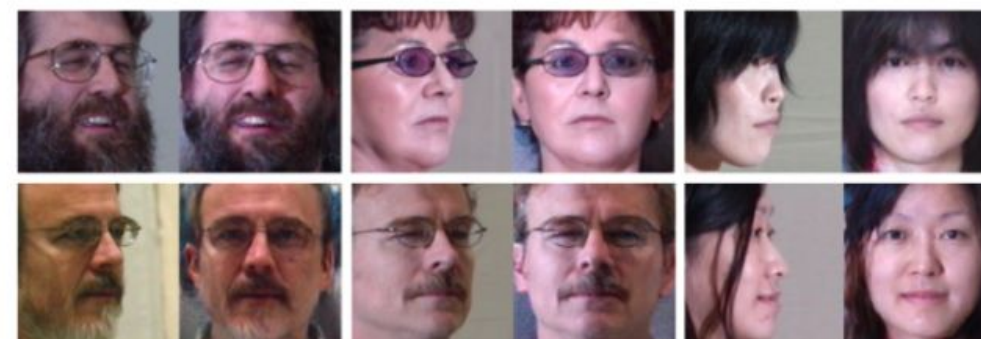
Super-Resolution

Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi,
« Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network », <https://arxiv.org/abs/1609.04802>, 2016



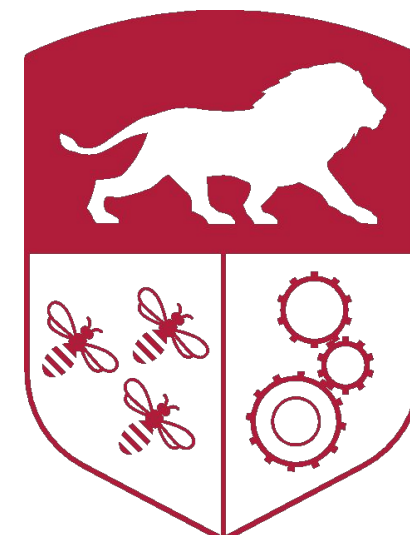
Face Frontal View Generation

Rui Huang, Shu Zhang, Tianyu Li, Ran He,
« Beyond Face Rotation: Global and Local Perception GAN for Photorealistic and Identity Preserving Frontal View Synthesis », <https://arxiv.org/abs/1704.04086>, 2017



Some useful references

- Fidle - Deep Learning Introduction (<https://www.fidle.cnrs.fr/w3/>)
- CS231n: Convolutional Neural Networks for Visual Recognition (<http://cs231n.stanford.edu>)
- Neural Networks and Deep Learning (<http://neuralnetworksanddeeplearning.com>)
- Deep Learning (<http://www.deeplearningbook.org>)
- PyTorch (<http://pytorch.org>)
- Weights & Biases (<https://wandb.ai/site/>)
- Hugging Face (<https://huggingface.co/>)



**CENTRALE
LYON**

36, avenue Guy de Collongue 69130 Écully
www.ec-lyon.fr | [@centralelyon](https://twitter.com/centralelyon)