AI for the Media

Week 7, Latent spaces & VAEs

~ Vít Růžička

Today's Motivation

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

3x3 conv

$\longrightarrow$

Manual filter: **edge detection**

>>> setosa.io/ev/image-kernels/ <<<

Overview

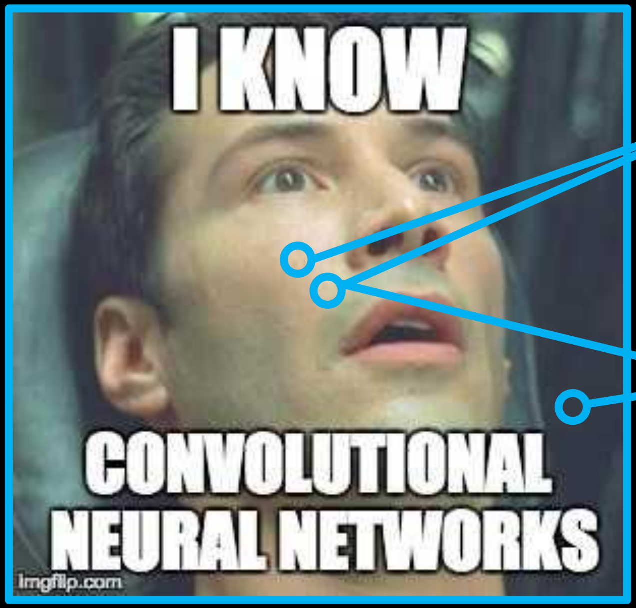
Latent spaces & VAEs (*pre-recorded lecture*):

- **Convolutions** for working with image data and **Convolutional Neural Networks**
- Latent spaces and **AutoEncoder architecture**

Practical session (*during the live session*):

- **Code**: Convolutional Variational Autoencoder for images

Images have locality



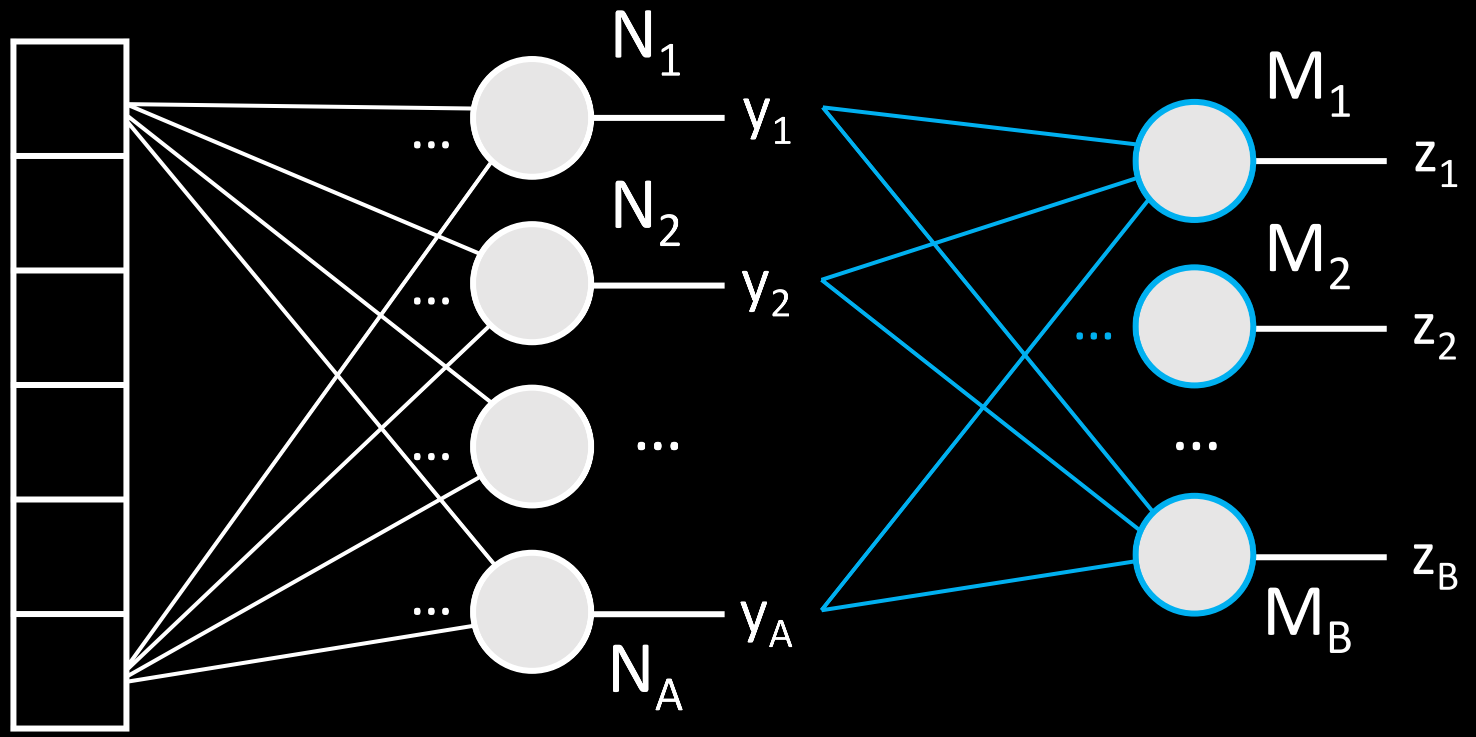
These two pixels will likely contain something relevant to each other.

While these two pixels won't likely have that much in common.

^^^ *Let's imagine any image in here*

- *In 1962 a study by Hubel and Wiesel explored human visual system and discovered two types of cells – **localized cells** processing details and **complex cells** covering larger areas.*

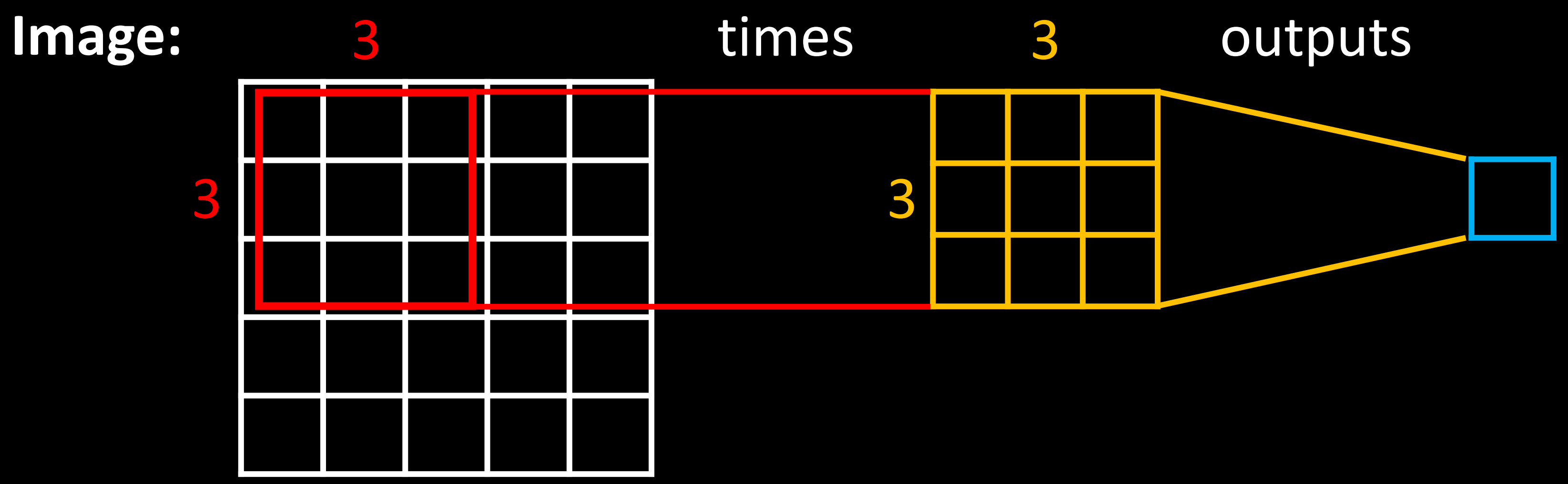
Fully-connected neuron layers:



< In code these are often named Dense()

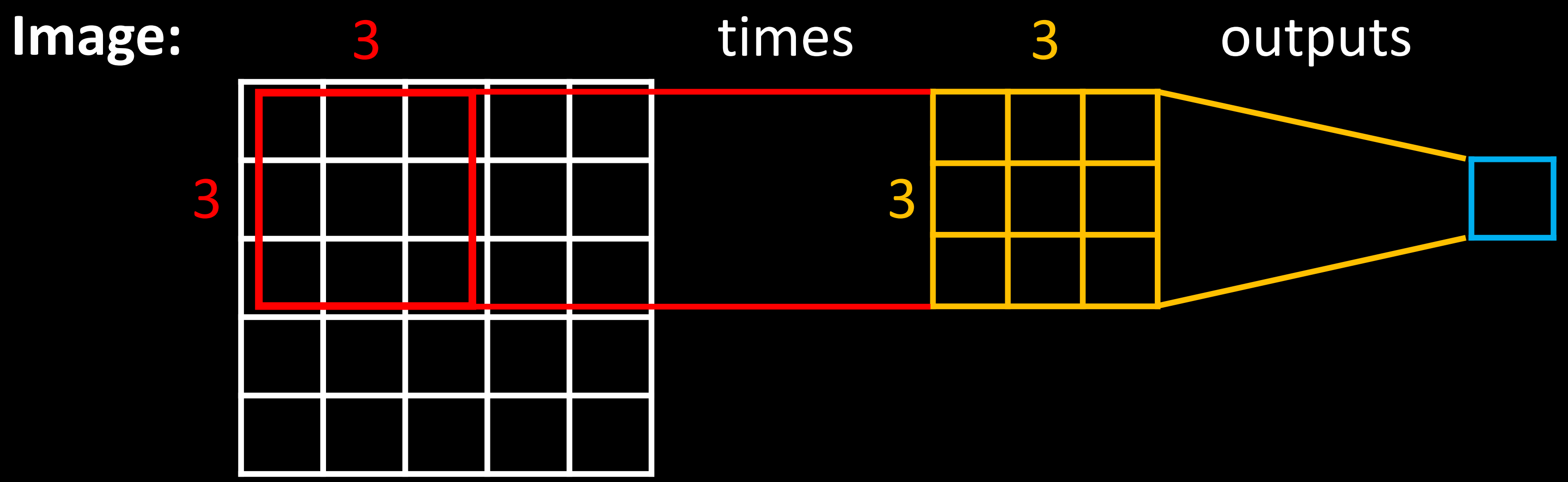
- Each neuron in its layer will be **connected to every output of the previous layer**.

Convolution

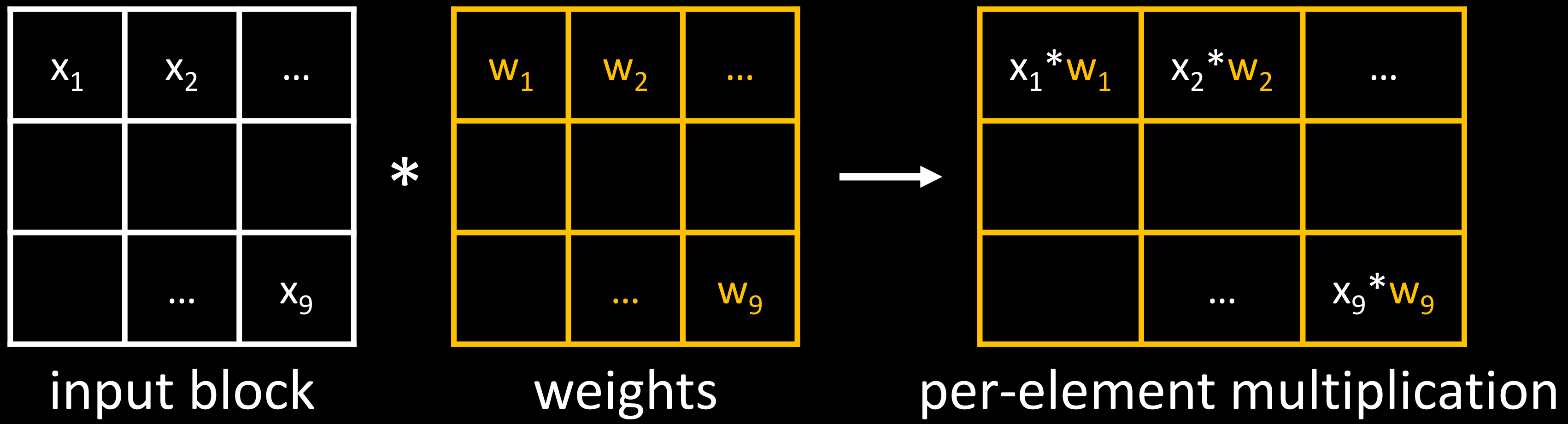


- **Convolution** is an operation applied on an image …

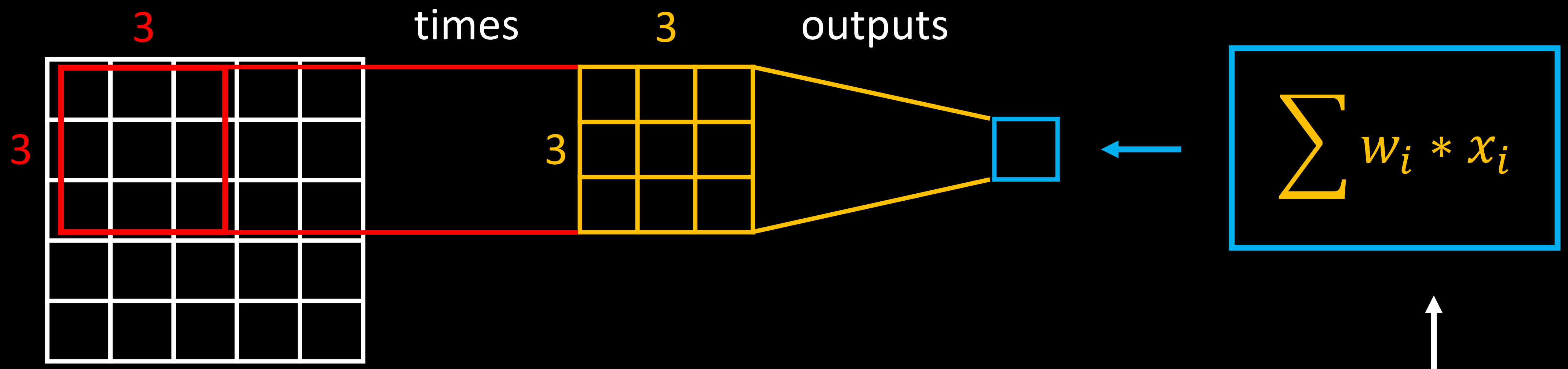
Convolution



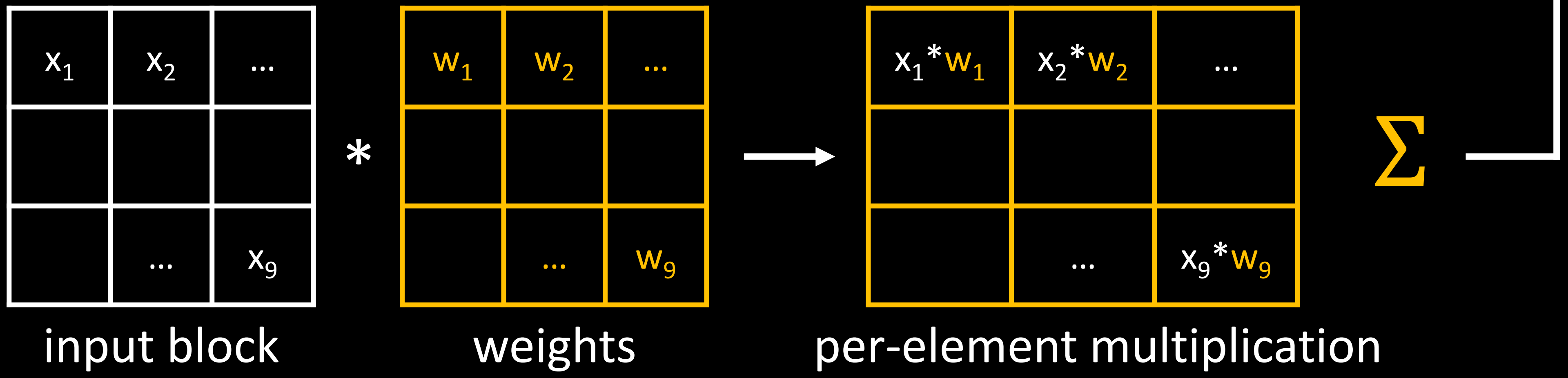
- **Convolution** is an operation applied on an image …



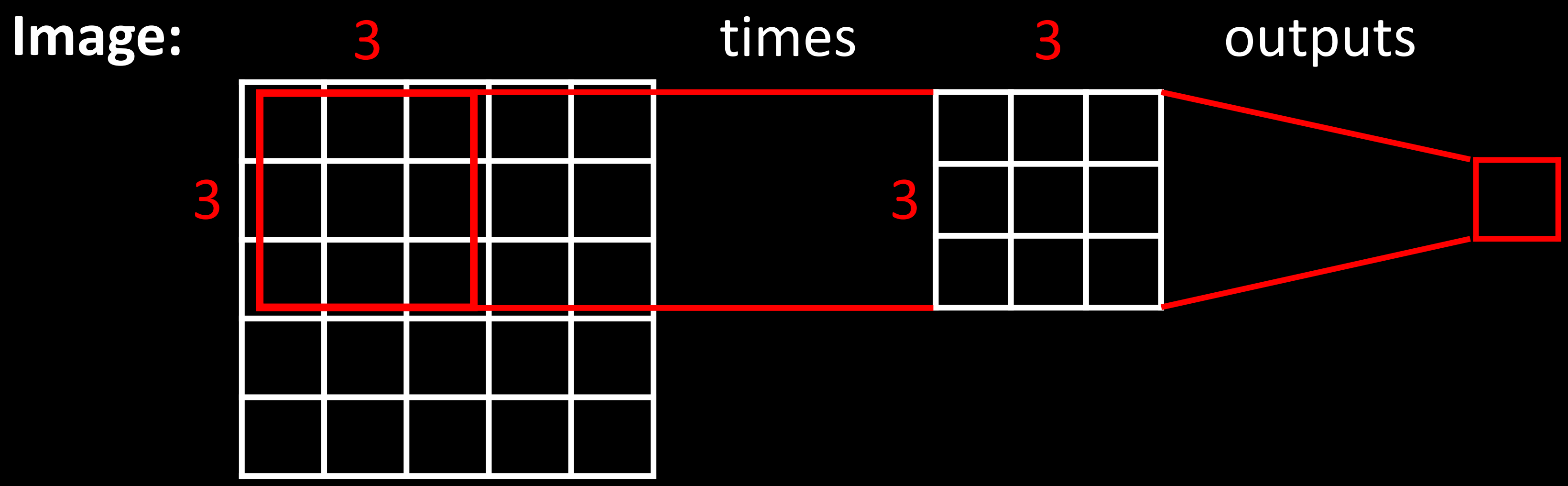
Convolution



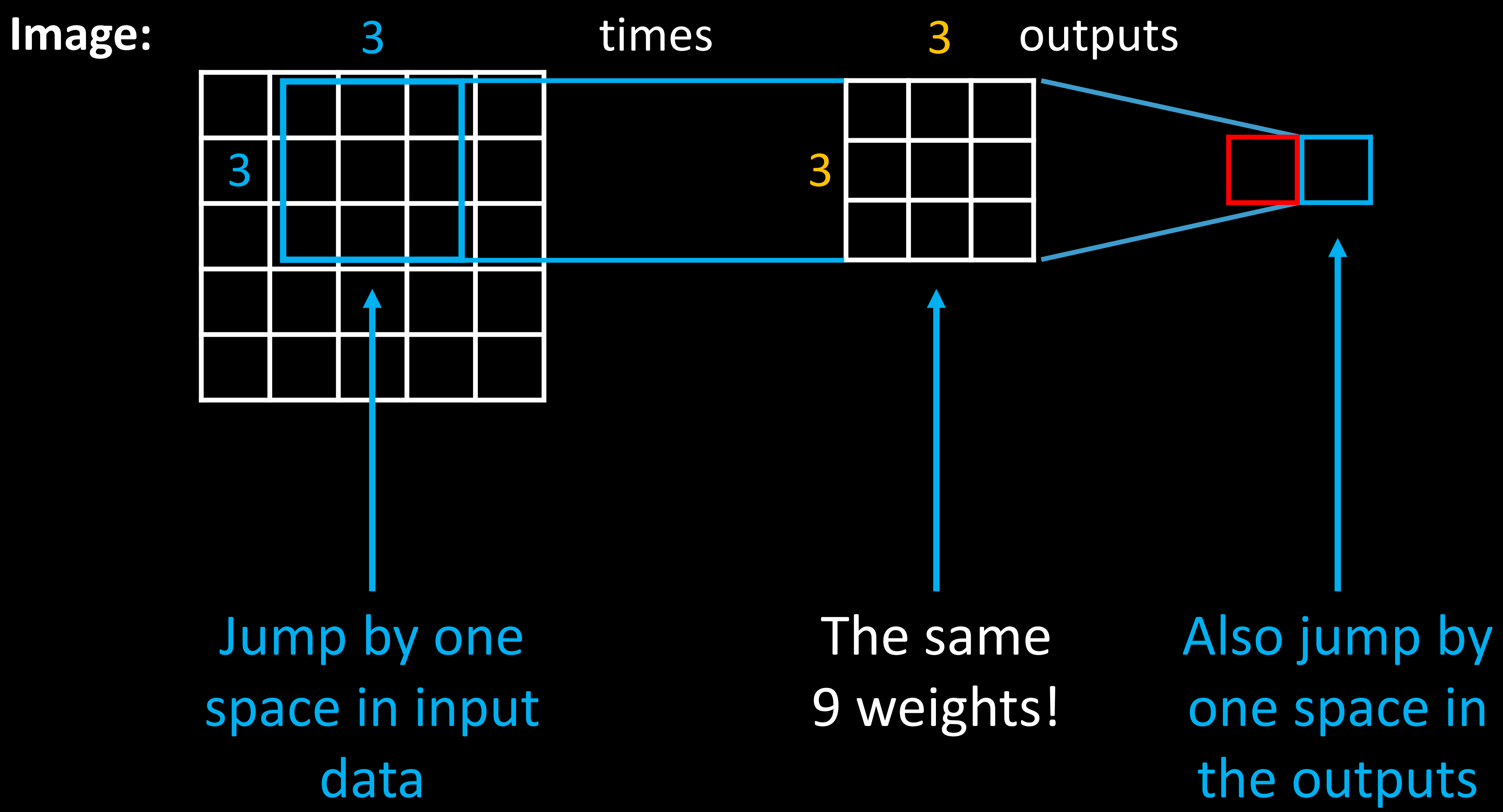
- **Convolution** is an operation applied on an image …



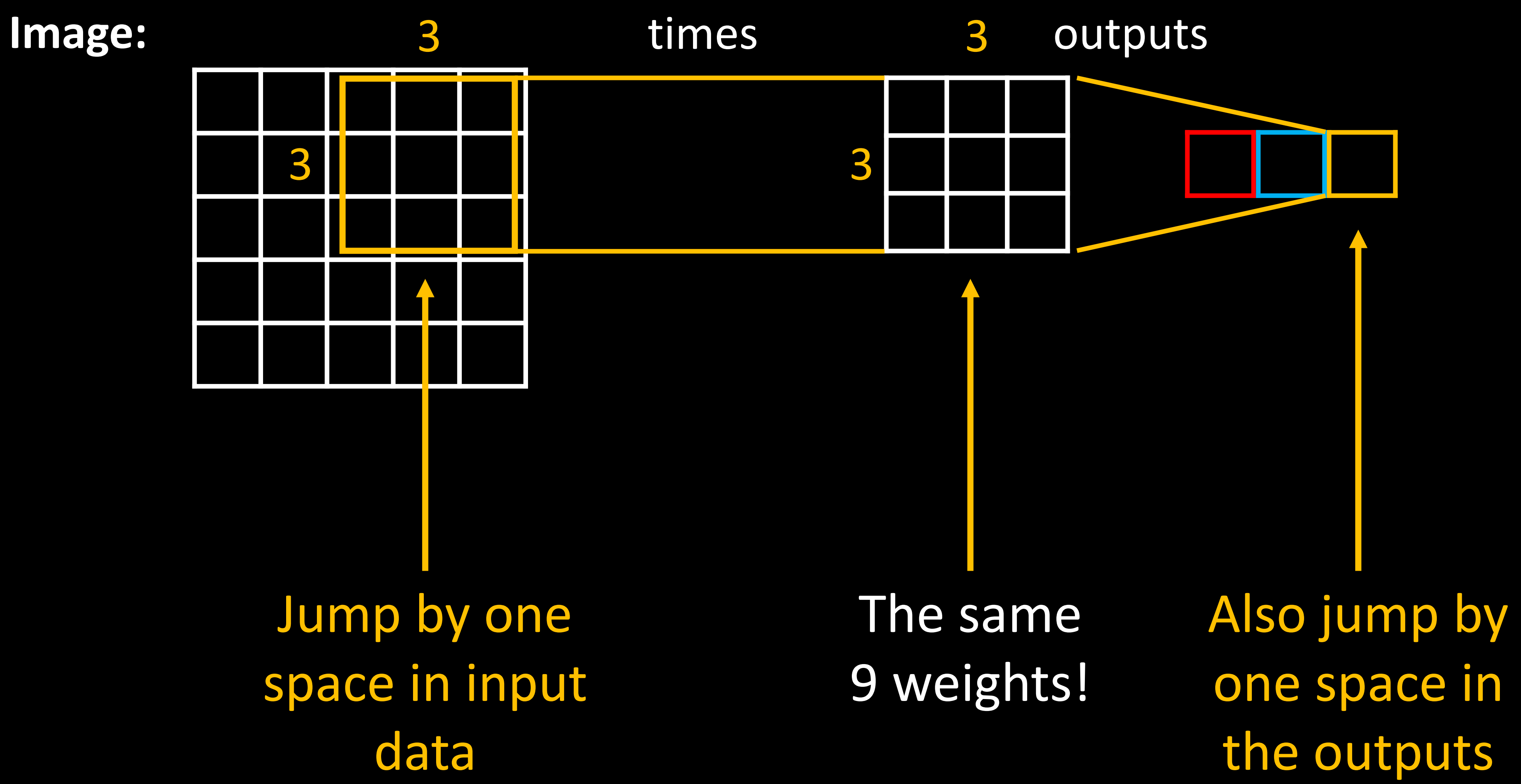
Convolution



Convolution

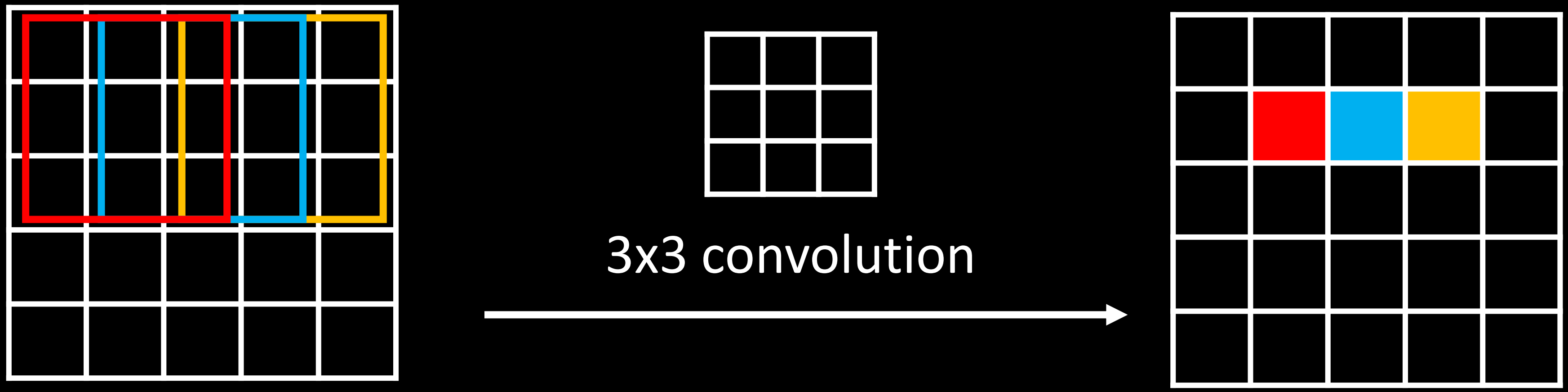


Convolution



Convolution

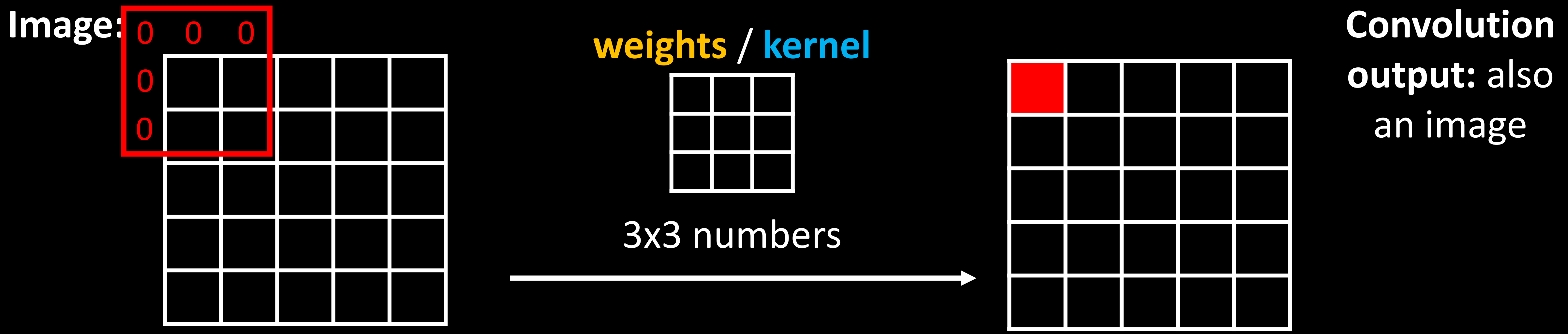
Image:



Convolution output: also an image

- The **same weights** are applied as a **filter**, jumping over the whole image … effectively producing an image on the output!
- Convolutions were used before neural networks as transformation functions to process images.

Convolution



- **Terminology**: we called these 3x3 numbers **"weights"**, but with convolutions they are also referred to as **kernel**
- **Detail**: To keep the same size of the output image, we need to extend the original image by 1 pixel – we pretend that it includes zeros (also called **"zero padding"**)

Manual convolution filters

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

3x3 conv

→

blur

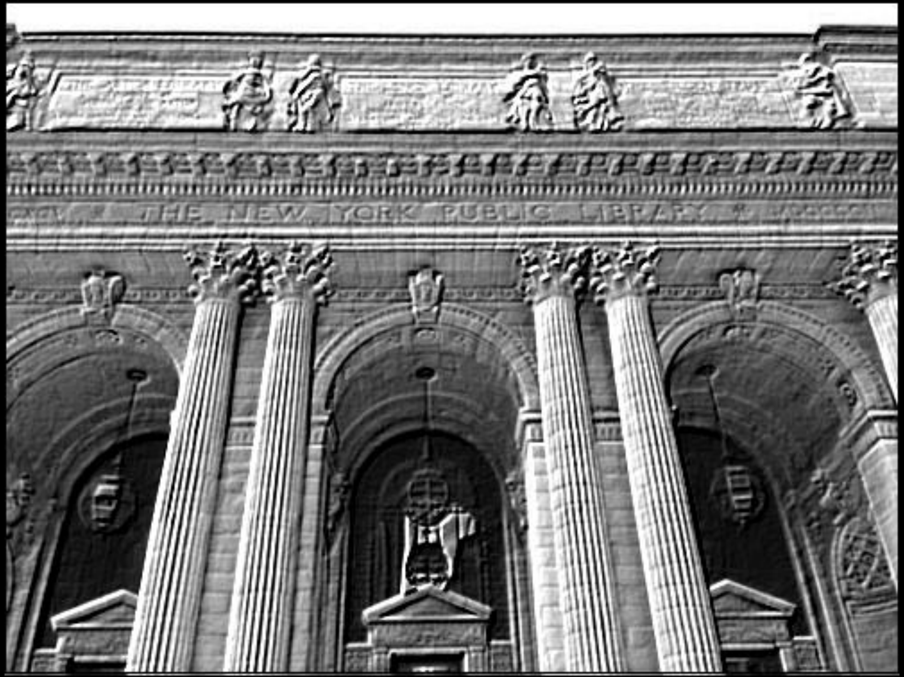
>>> setosa.io/ev/image-kernels/ <<<

Manual convolution filters

-2	-1	0
-1	1	1
0	1	2

3x3 conv

$\longrightarrow$



emboss

>>> setosa.io/ev/image-kernels/ <<<

Manual convolution filters

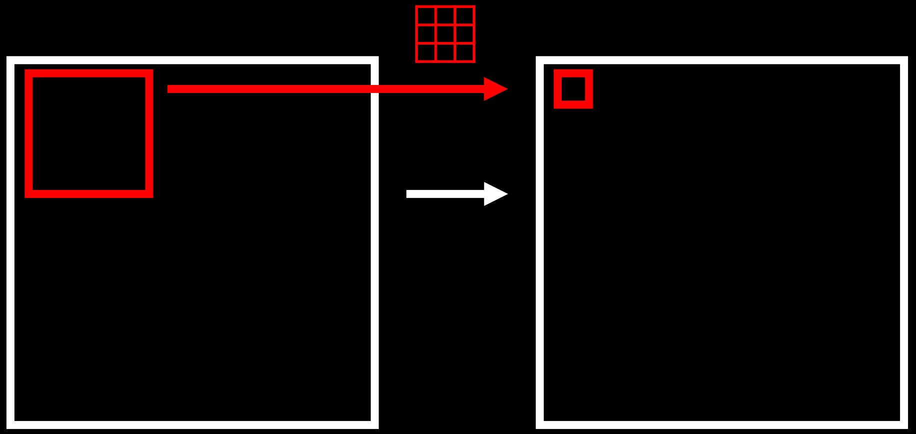
-1	-1	-1
-1	8	-1
-1	-1	-1

3x3 conv

outline

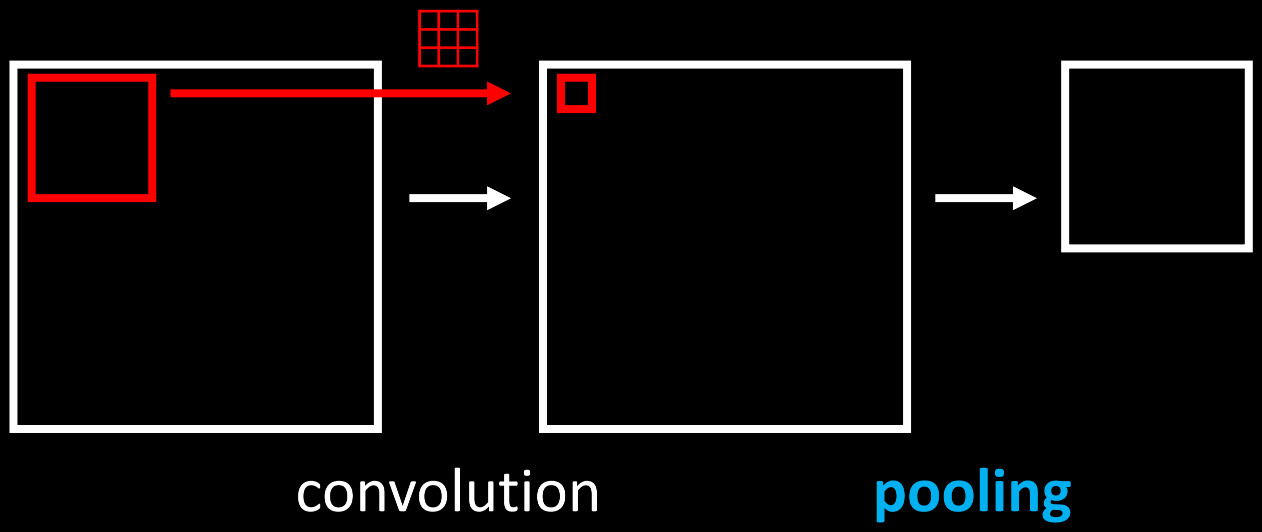
>>> setosa.io/ev/image-kernels/ <<<

Convolutions and Pooling



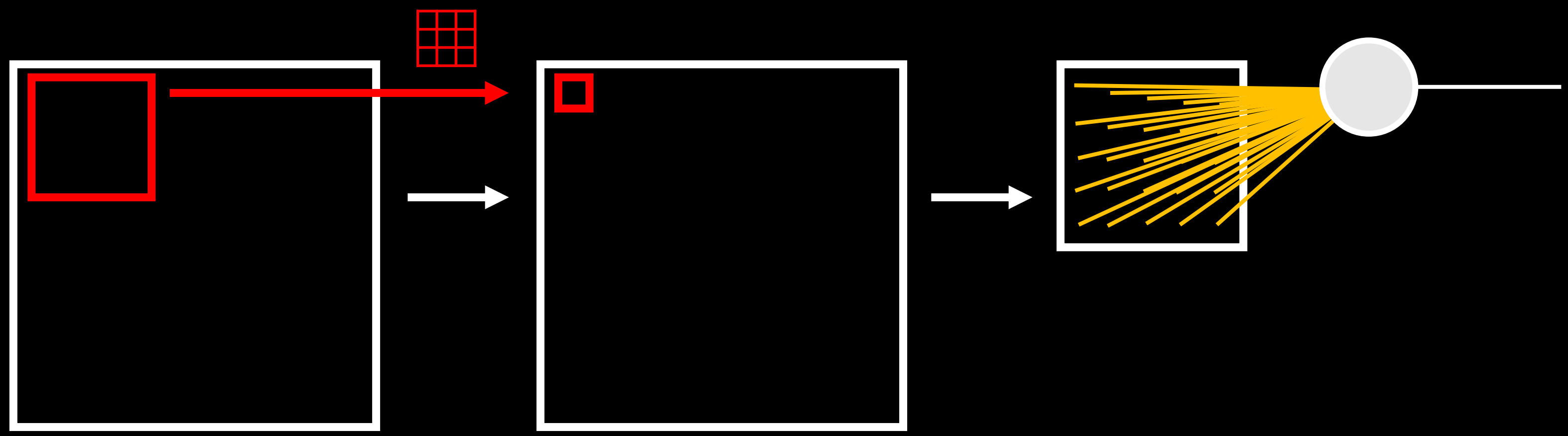
convolution

Convolutions and Pooling



- Convolutional layers are usually followed by **downsampling layers (pooling)** which rescale the image (less information)

Convolutions and Pooling

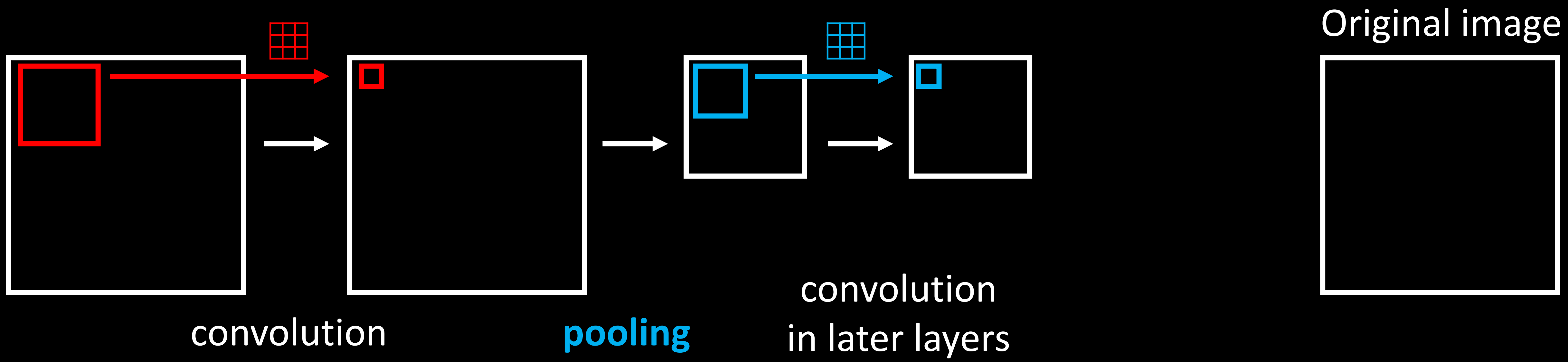


convolution **pooling**

Less information = Less connections in later *fully connected layers*!

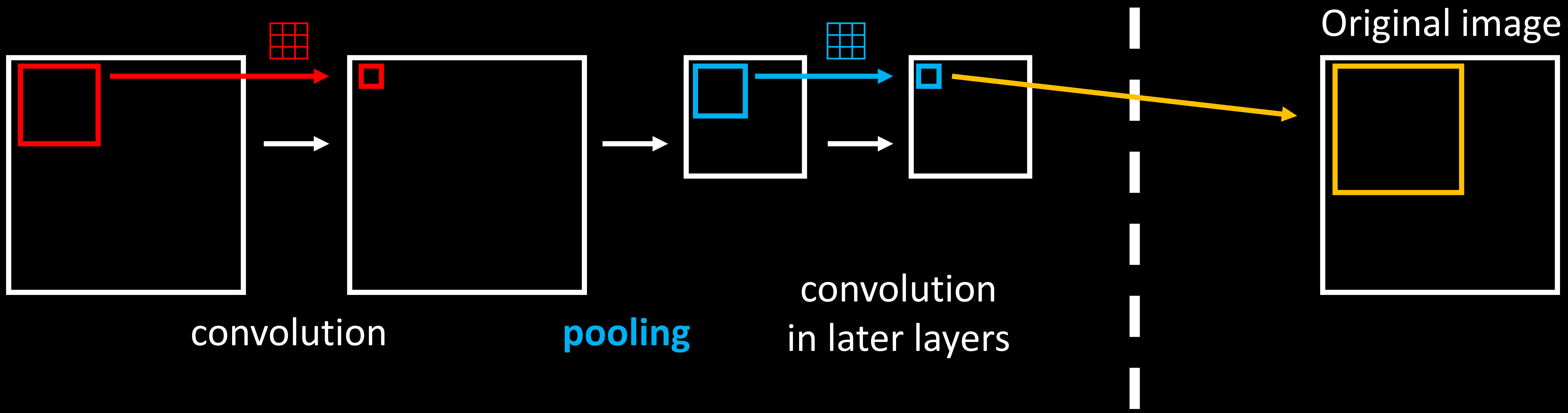
- Convolutional layers are usually followed by **downsampling layers (pooling)** which rescale the image (**less information**)

Convolutions and Pooling



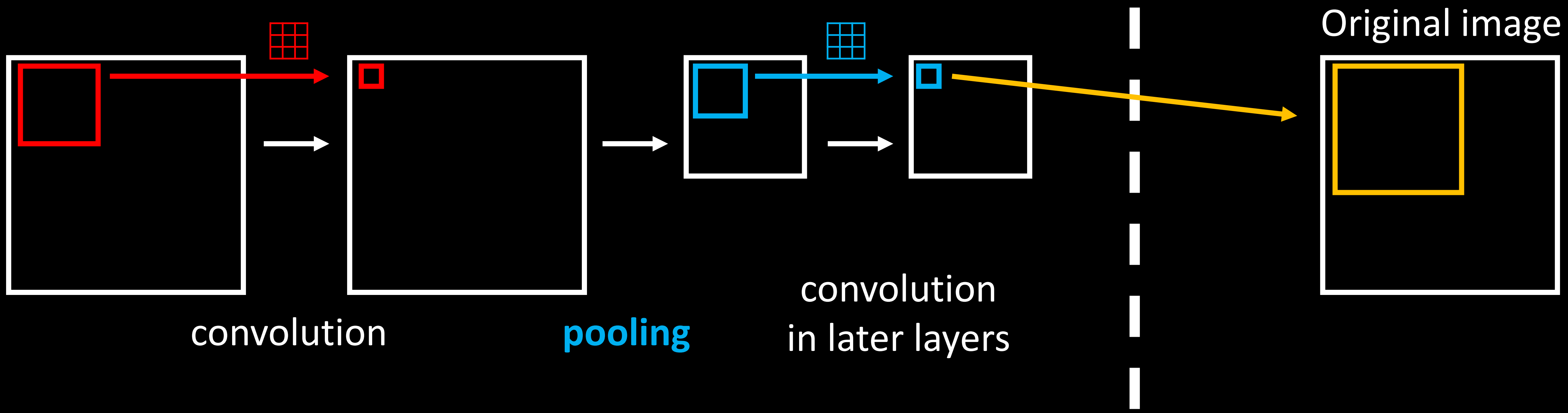
- Convolutional layers are usually followed by **downsampling layers (pooling)** which rescale the image (less information)
- "Later" convolutions are also looking at much **larger region in the original image** with their kernel (*receptive field*)

Convolutions and Pooling



- Convolutional layers are usually followed by **downsampling layers (pooling)** which rescale the image (less information)
- "Later" convolutions are also looking at much **larger region in the original image** with their kernel (*receptive field*)

Convolutions and Pooling



- This means, that the later convolutional layers can **specialize on processing different scales of the image** (This is huge!)

details   larger areas   … We will see that this means visually!

Specialized filters

- Local features
- Shapes, colors, edges, ...

kernel visualizations

- Higher level features, more general
- Objects, concepts ...

depth

Earlier layers

Later layers

Intuition for classification

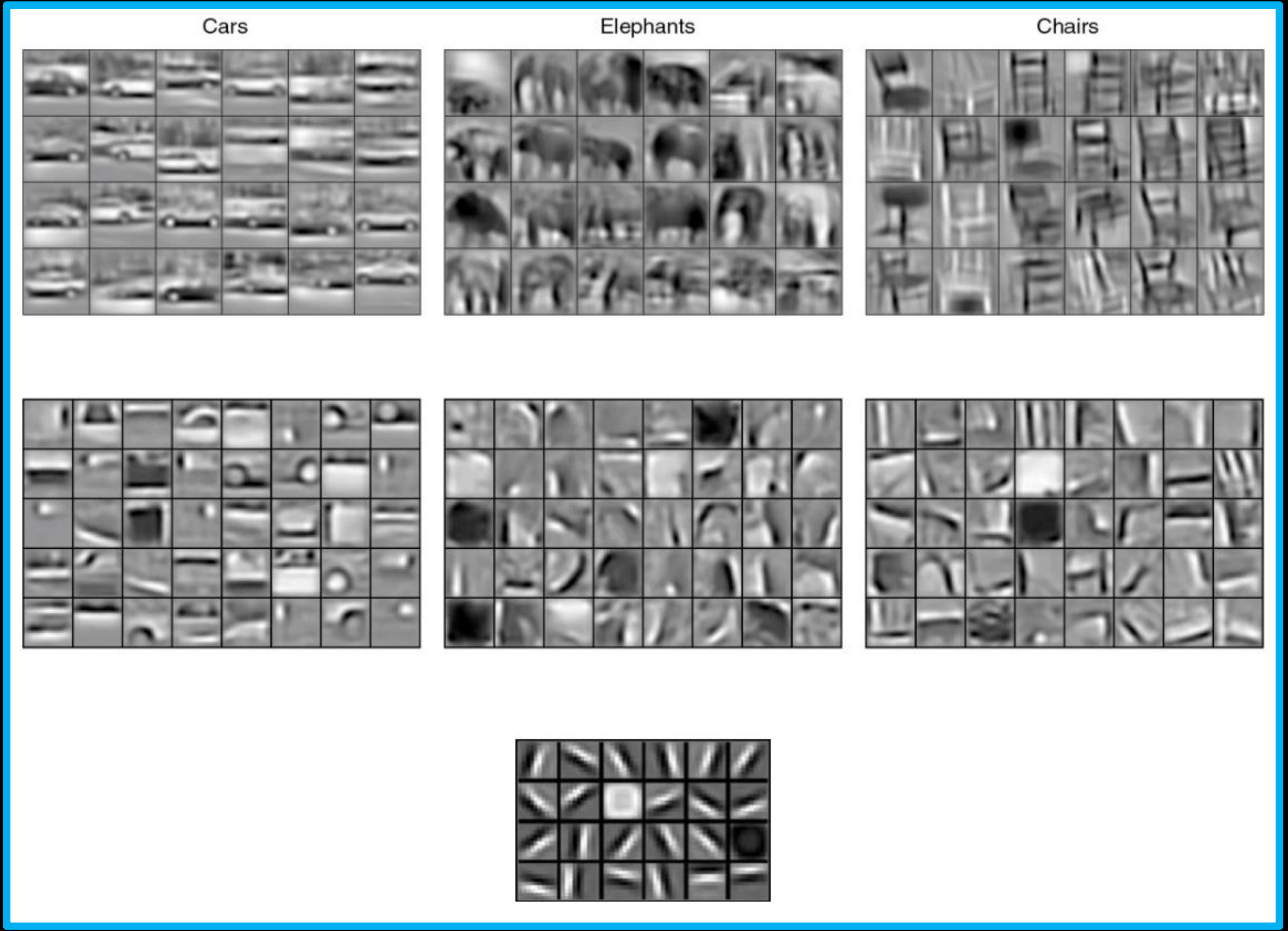
Classifications

High level features

Mid level features

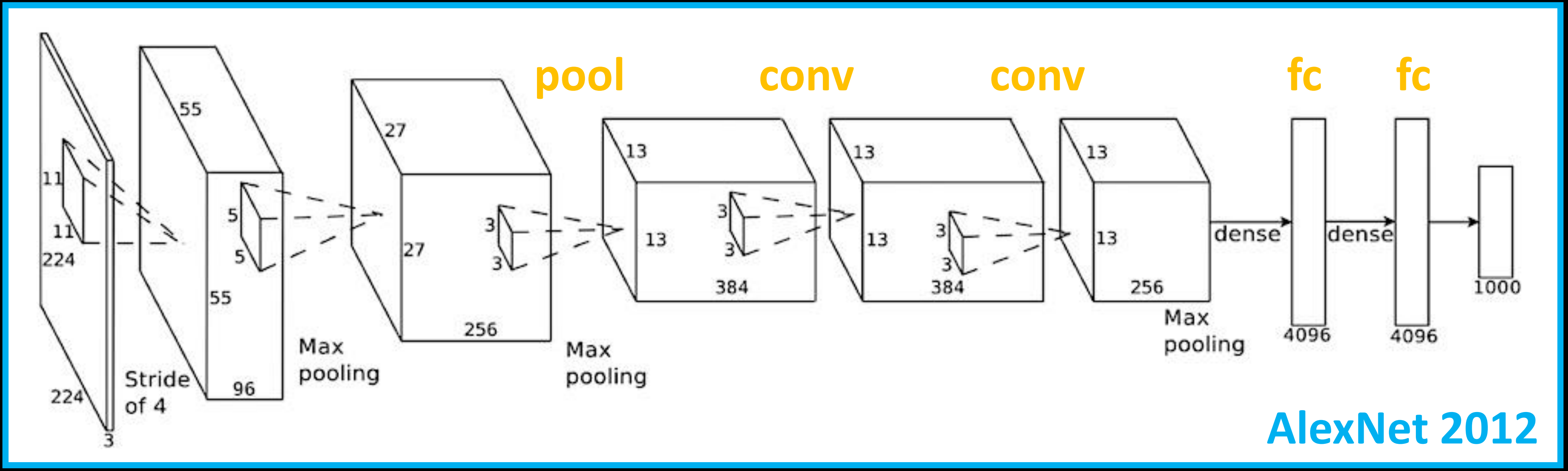
Low level features

Inputs



Bonus: [novel methods of visualizations]

Real world example: AlexNet

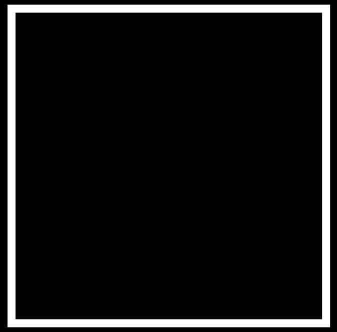


By now, we know *most of* the layers used in this CNN architecture!

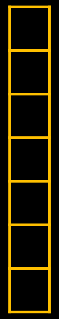
(One more detail: the Dropout layers)

General feature description

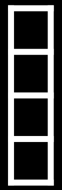
Image $\rightarrow$ **Feature extractor** $\rightarrow$ $\vec{v}$ $\rightarrow$ **Classification** $\rightarrow$ **label**



224x224x3 px



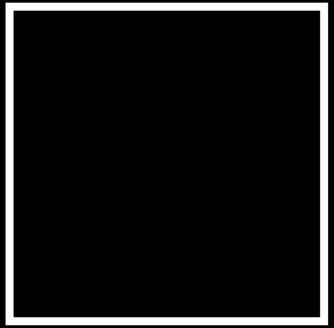
vector of 4096 real numbers

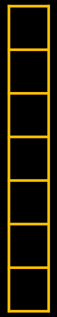


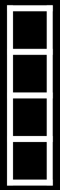
1000 classes

General feature description

Image → **Feature extractor** → $\vec{v}$ → **Classification** → **label**



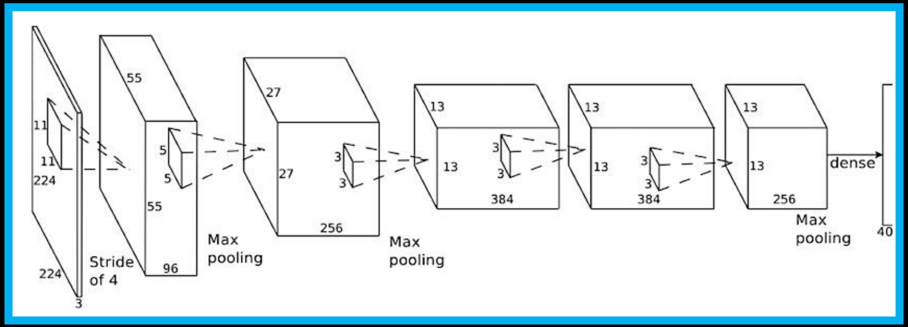


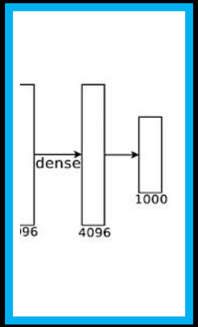


224x224x3 px

vector of 4096 real numbers

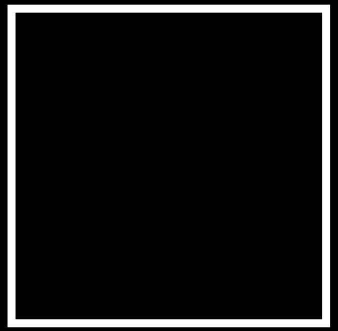
1000 classes



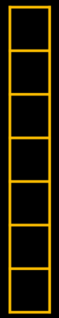


General feature description

Image → Feature extractor → $\vec{v}$



224x224x3 px



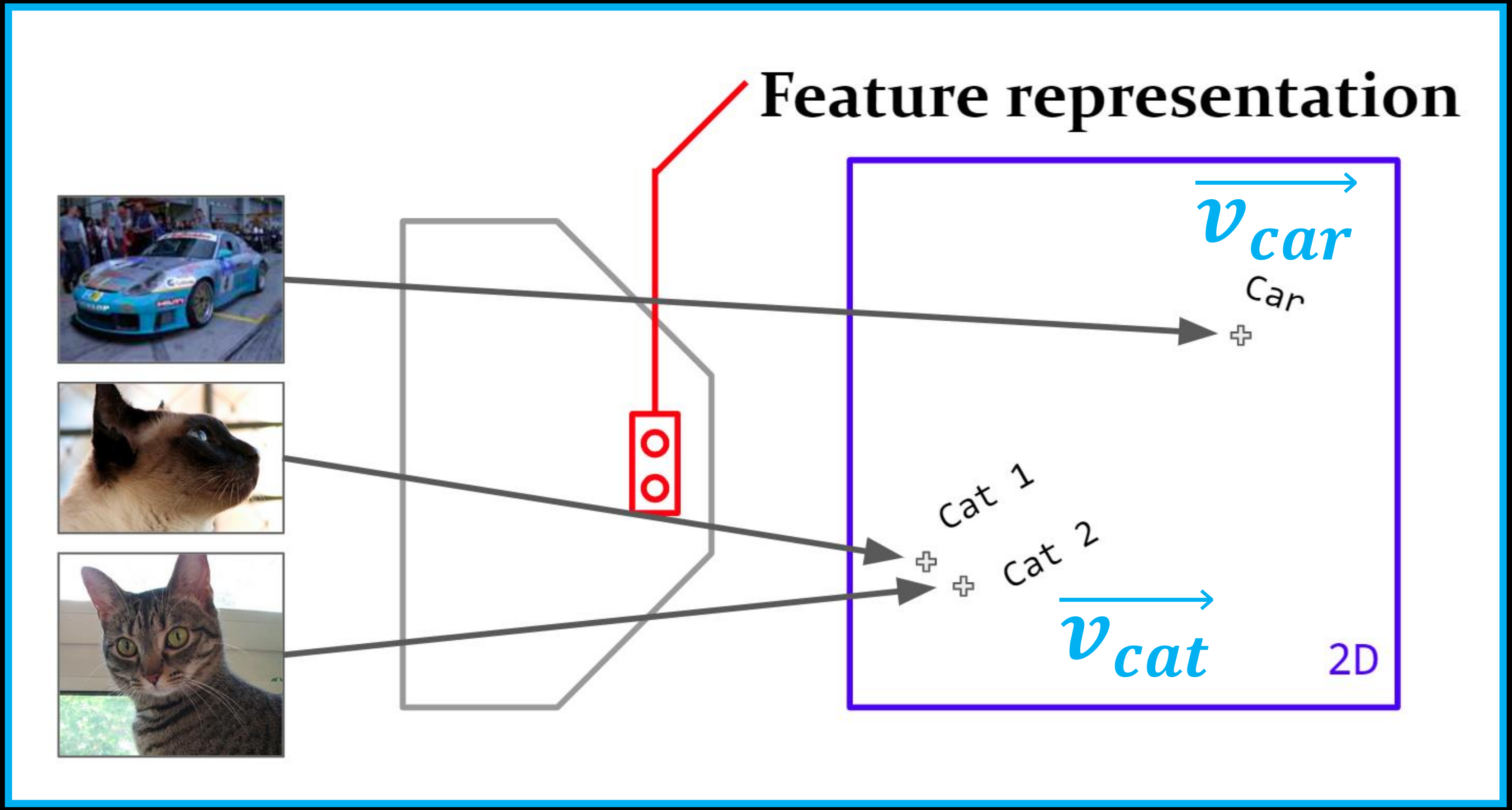
vector of 4096 real numbers

- This part of the model becomes useful as a general feature extractor (its learned **representation-ability is data driven** – depends on the dataset!)

***)** PS: now it’s a good time to take a pause

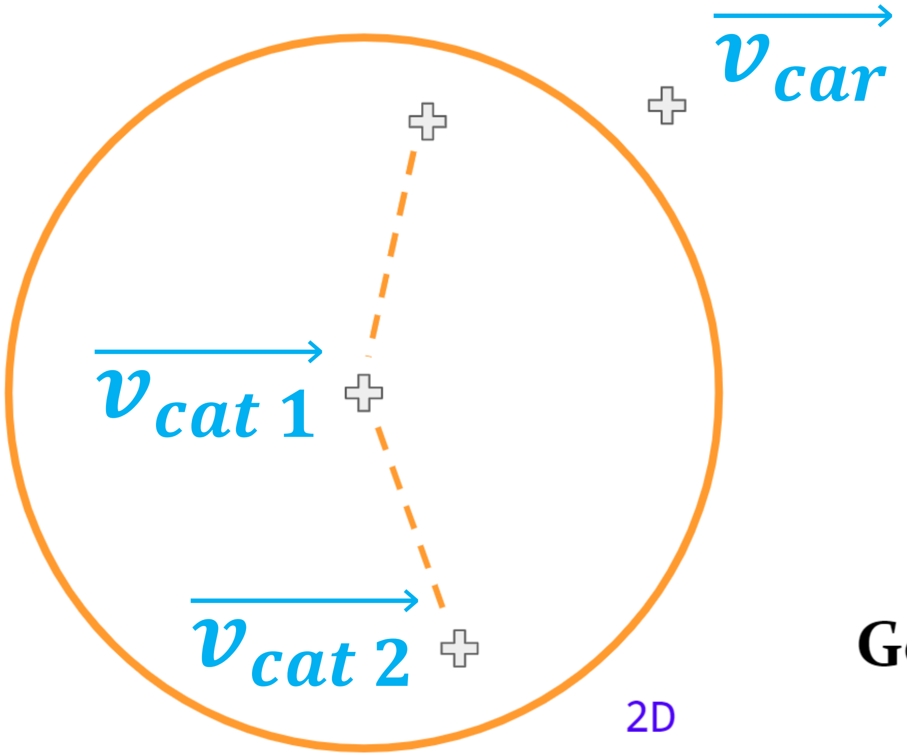
General feature description

Image → Feature extractor → $\vec{v}$ → Classification → label



General feature description

Image → **Feature extractor** → $\vec{v}$ → **Classification** → **label**

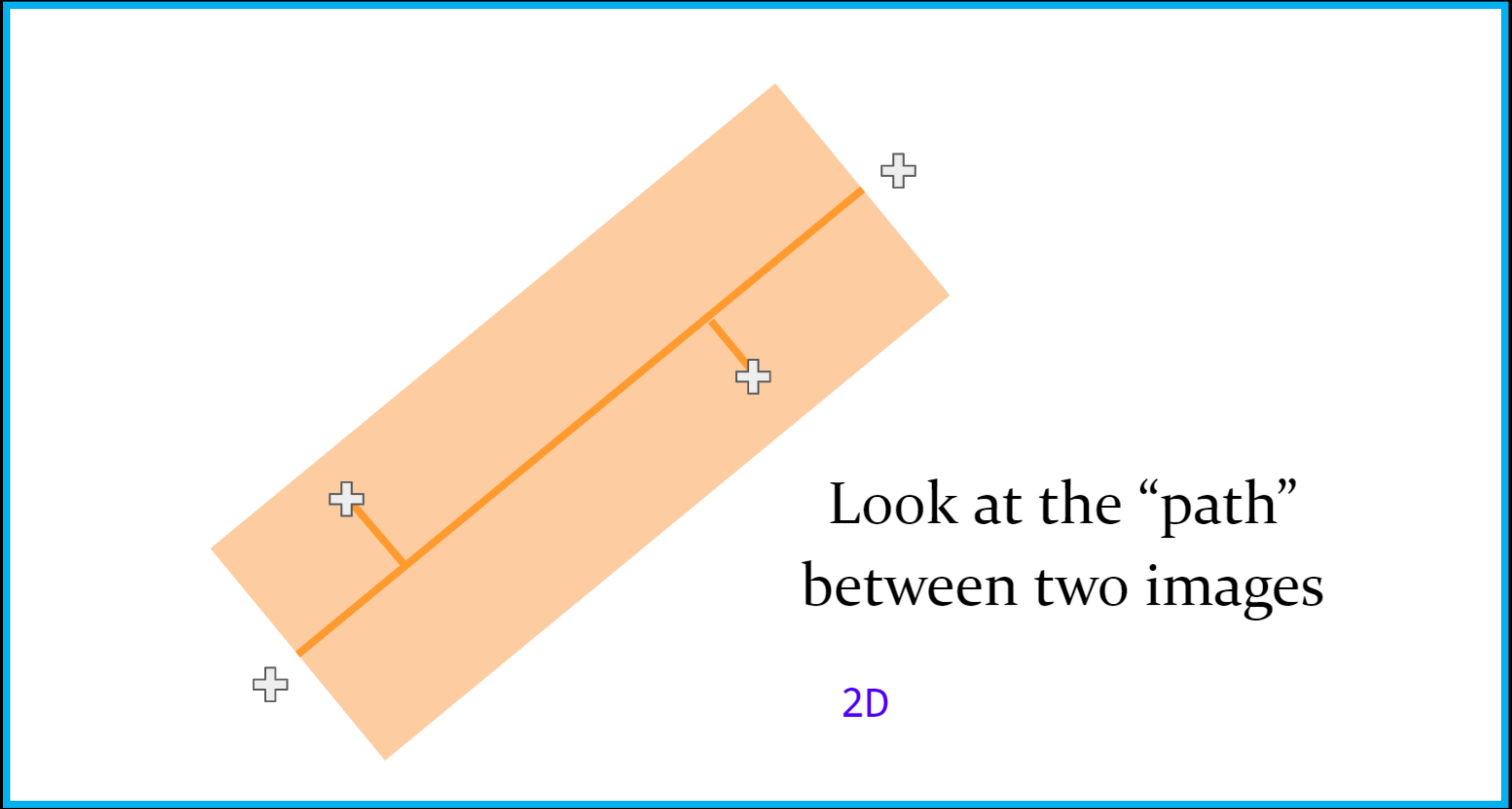


Search for
"nearby" images

Google Image search

General feature description

Image → **Feature extractor** → $\vec{v}$ → **Classification** → **label**



Mario Klingemann - X Degrees of Separation (2018)

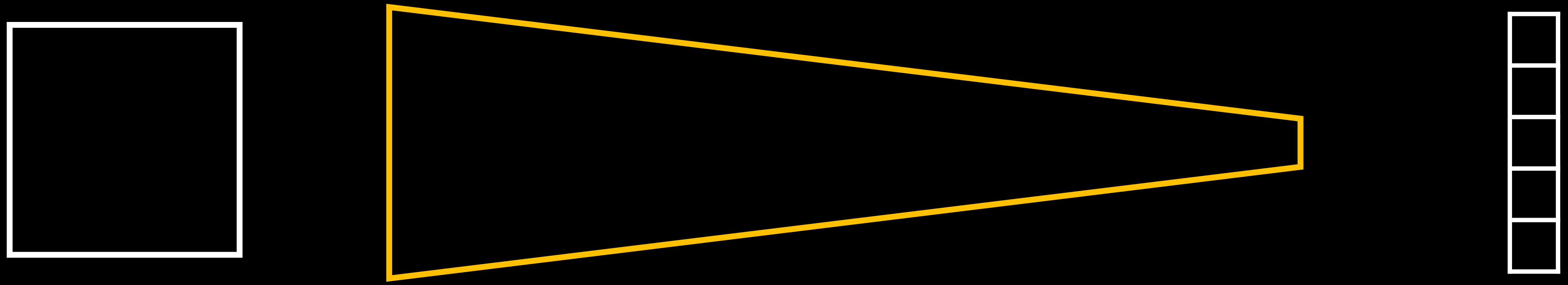
Image 1 → Feature extractor → $\vec{v_1}$

Vectors in the path ← Feature extractor ← Other images

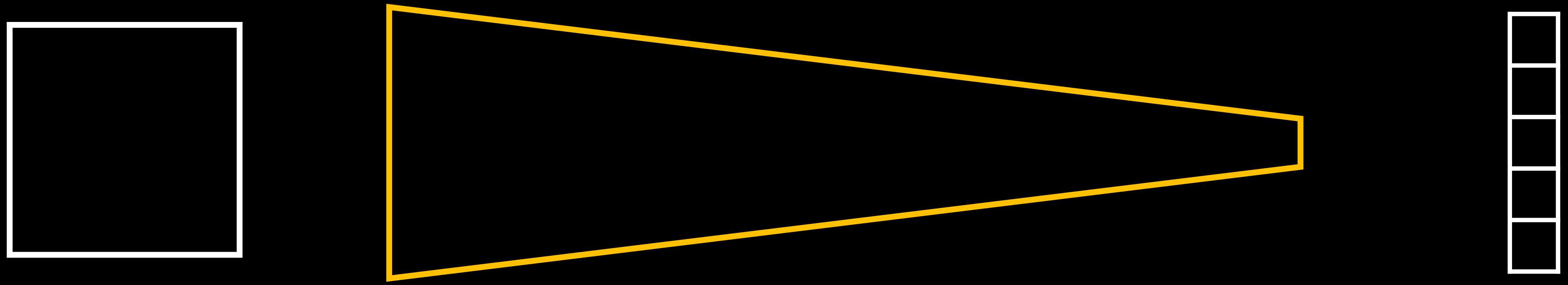
Image 2 → Feature extractor → $\vec{v_2}$

Mario Klingemann - X Degrees of Separation (2018)

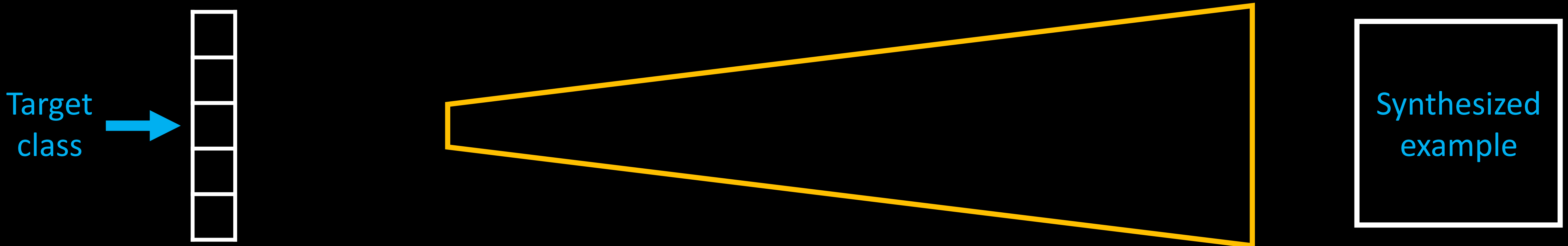
Classification



Classification

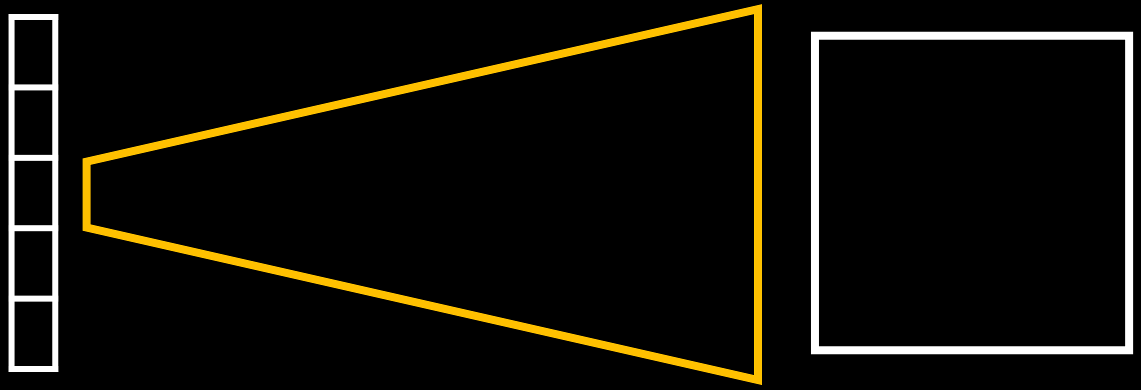


Generation



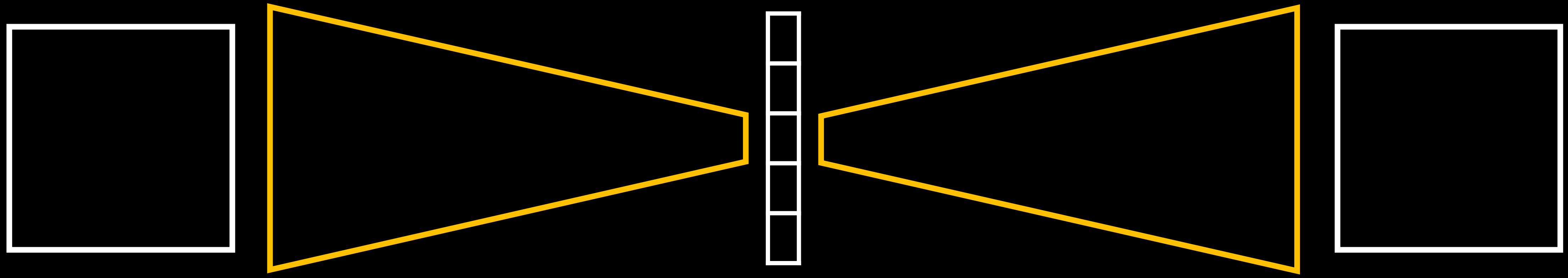
- What we *more or less* want is to **flip the task** – but it turns out this is too difficult task

Generative Architectures



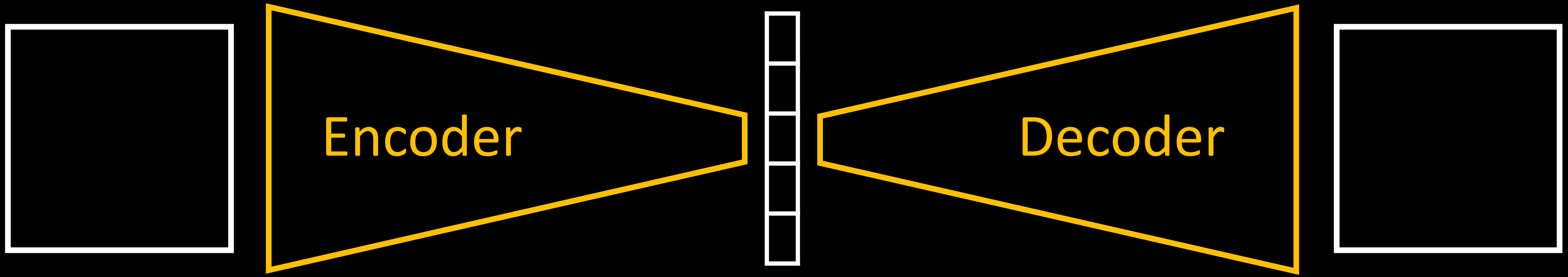
How to get
this model
to work?

AutoEncoders



- Rephrase it as an **identity operation**, we want the model to encode image into some intermediate lower-dimensional representation and decoder to undo that work.

AutoEncoders



- Rephrase it as an **identity operation**, we want the model to encode image into some intermediate lower-dimensional representation and decoder to undo that work.
- Seemingly this is a useless task – we are making a machine for nothing. But as we saw in previous part (feature extraction), **parts of the models can also be useful!**

AutoEncoders

Layers:

Convolutional

Pooling

Fully Connected

UnPooling (scale up)

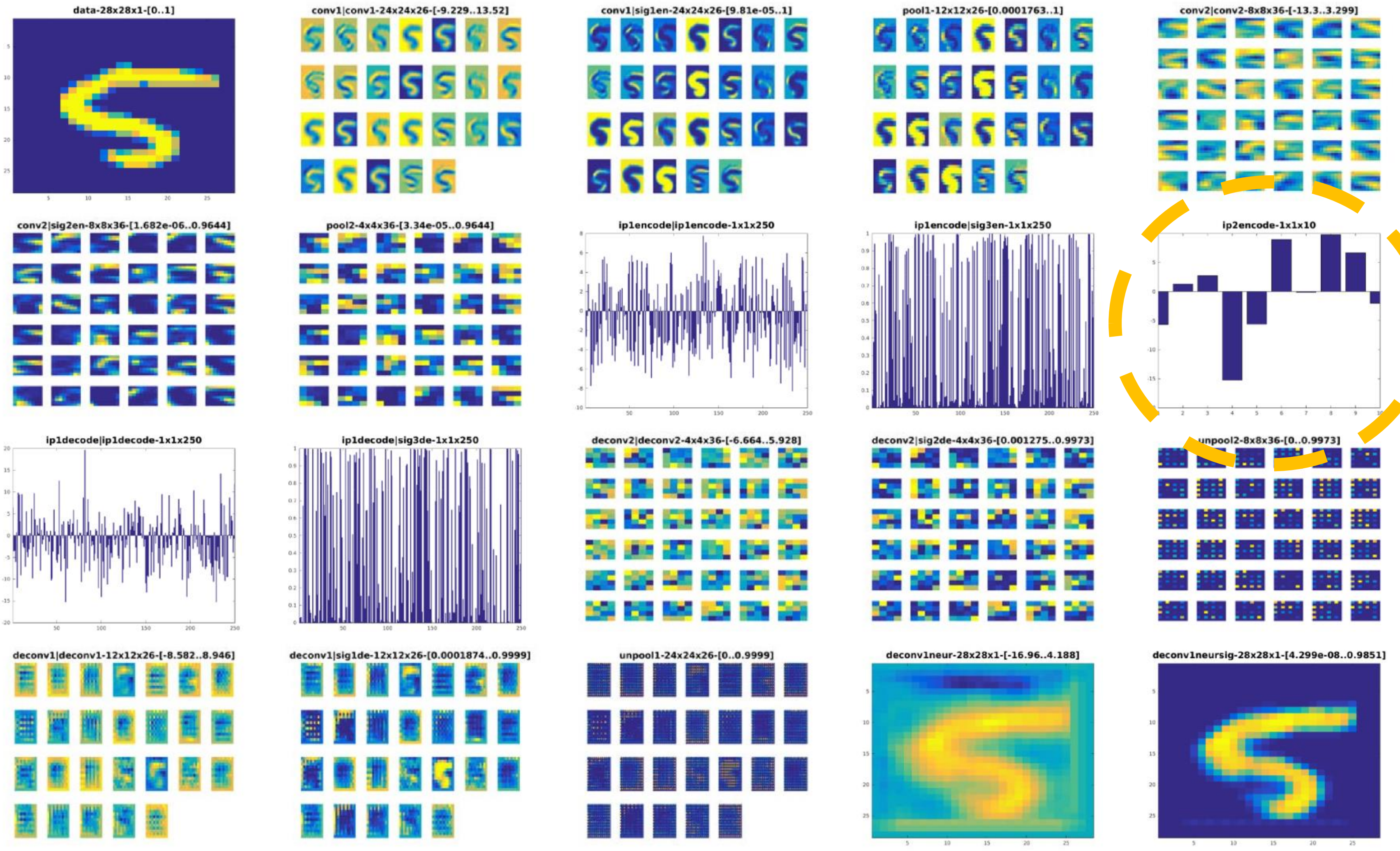
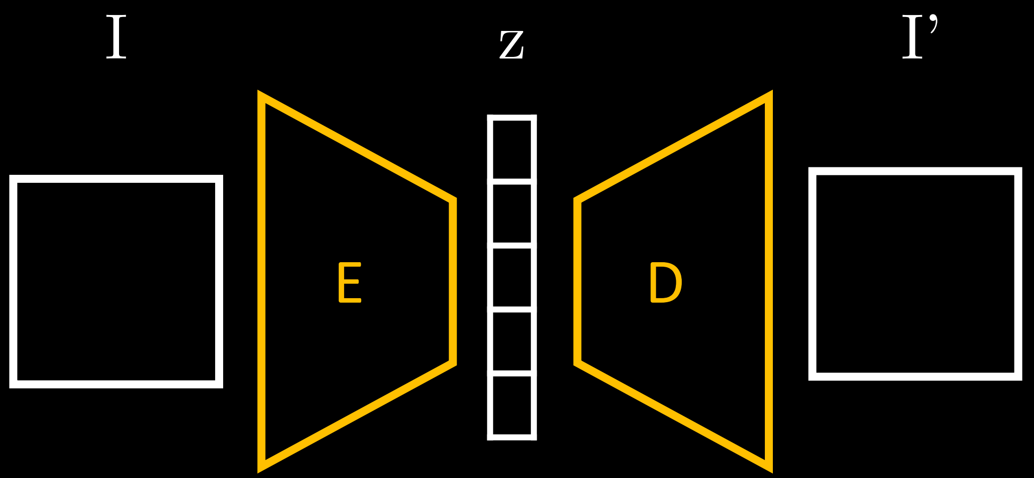


Fig. 11. Visualization of encoding and decoding digit '5' by 10D Model 4.
Each panel visualizes the output of appropriate layer of Model 4. The title of each panel describes a type of the layer (e.g. conv1, pool1, conv2, pool2, and so on), a size of the layer (e.g. 24x24x26 means 26 feature maps with 24x24 elements each) and a range of the layer's output [min..max]. See more explanation in the next to last paragraph of Section 4.3.

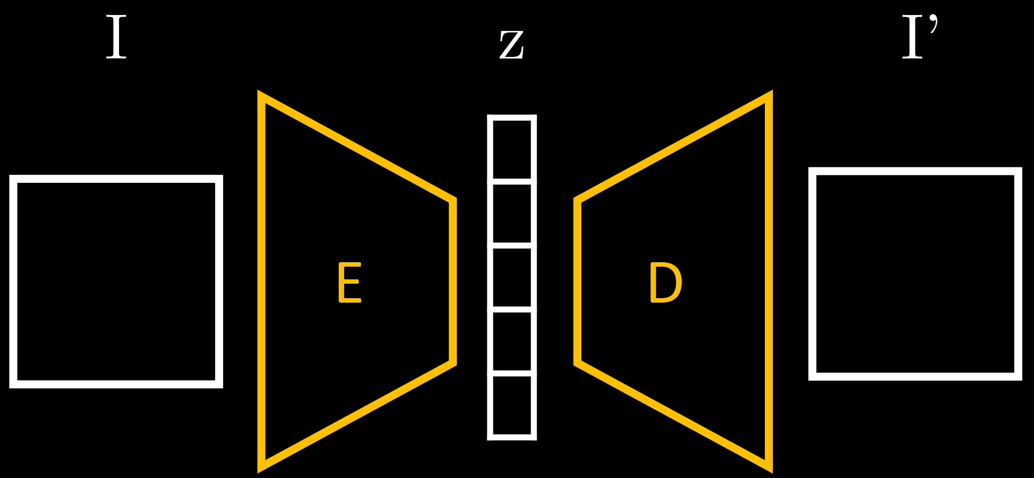
Latent vector: 10 numbers

AutoEncoder



< Training

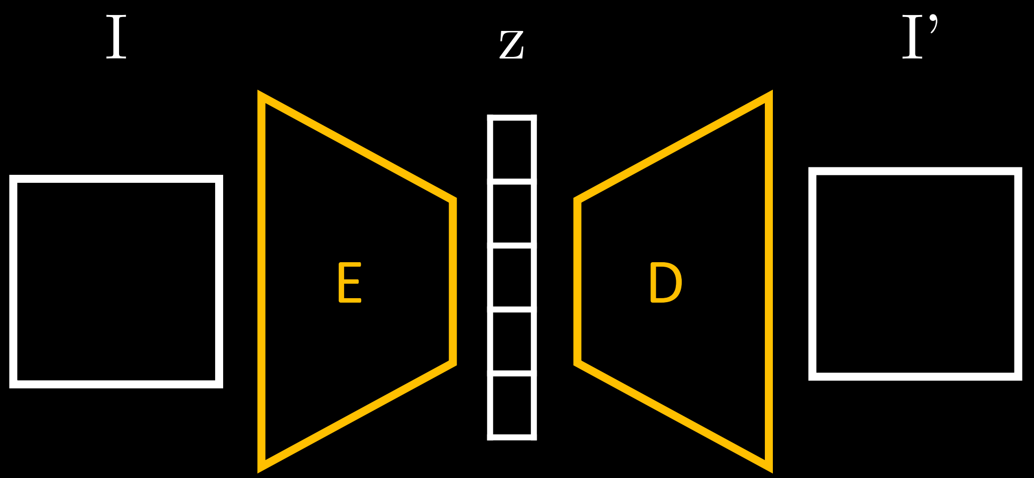
AutoEncoder



- Reconstructed image

$$I' = D\big(\underbrace{E(I)}_{z}\big)$$

AutoEncoder



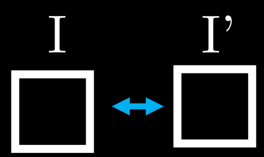
- Reconstructed image

$$I' = D(\underbrace{E(I)}_{\mathrm{z}})$$

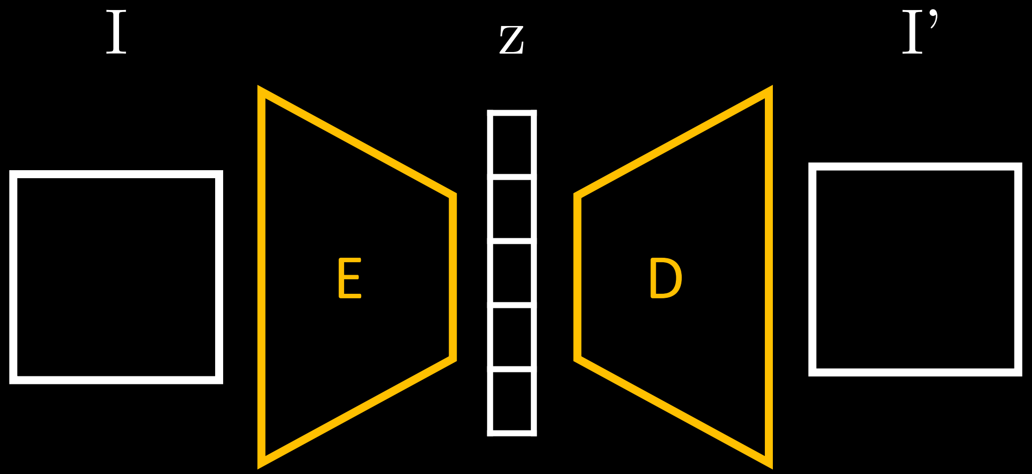
- Training with reconstruction loss:

$$\boldsymbol{distance(D(E(I)), I)}$$

(for example: MSE)



AutoEncoder



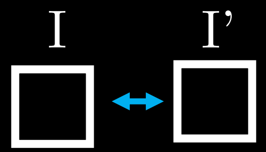
- Reconstructed image

$$I' = D(\underbrace{E(I)}_{z})$$

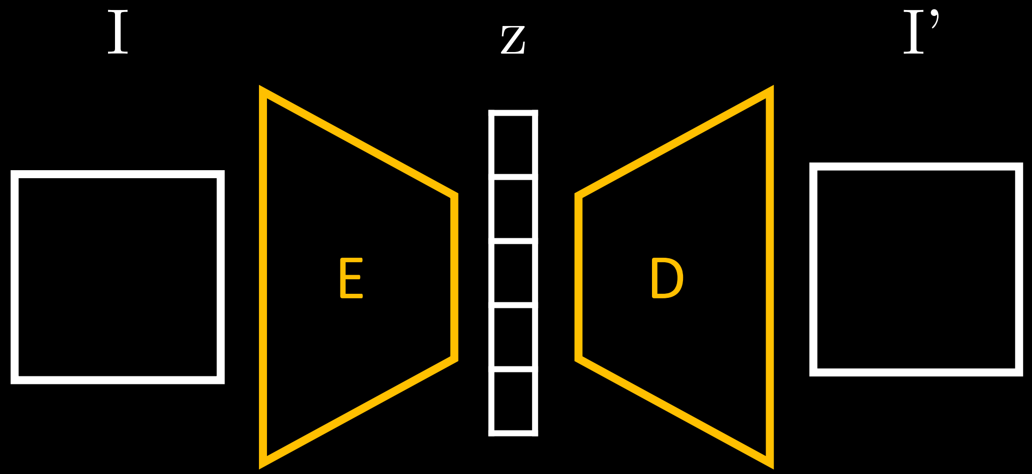
- Training with reconstruction loss:

$$\boldsymbol{distance(D(E(I)), I)}$$

(for example: MSE)

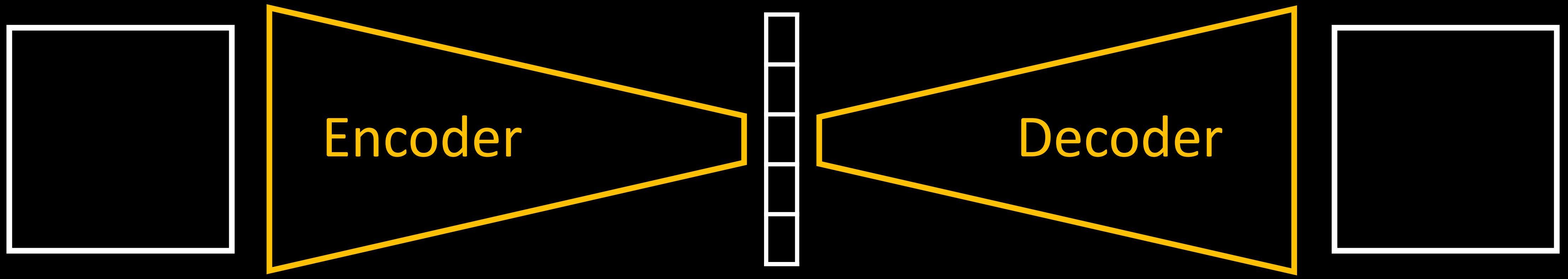


Variational AE



- We impose **additional restrictions on the latent representation** (z), such as that it has a Gaussian distribution: $z \sim N(0,1)$
- The loss function is more complicated, it comes from probabilistic theory (for further read see a blog)

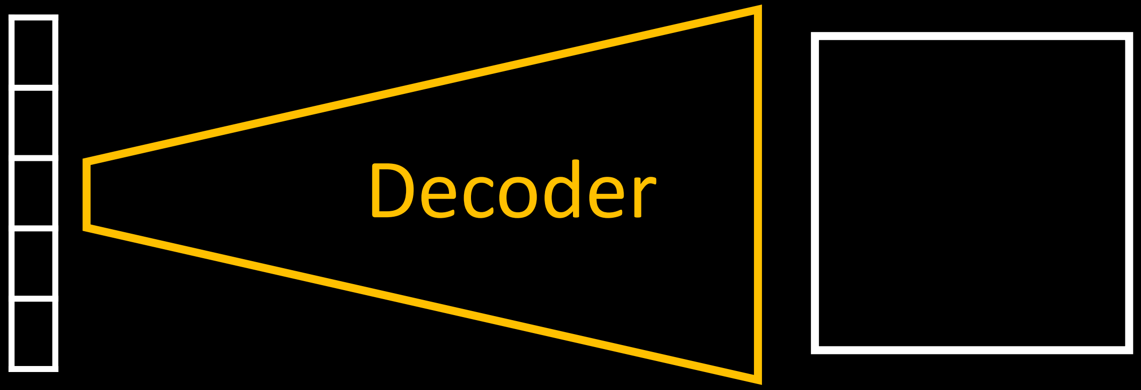
Interaction



- With a trained AutoEncoder model

Interaction

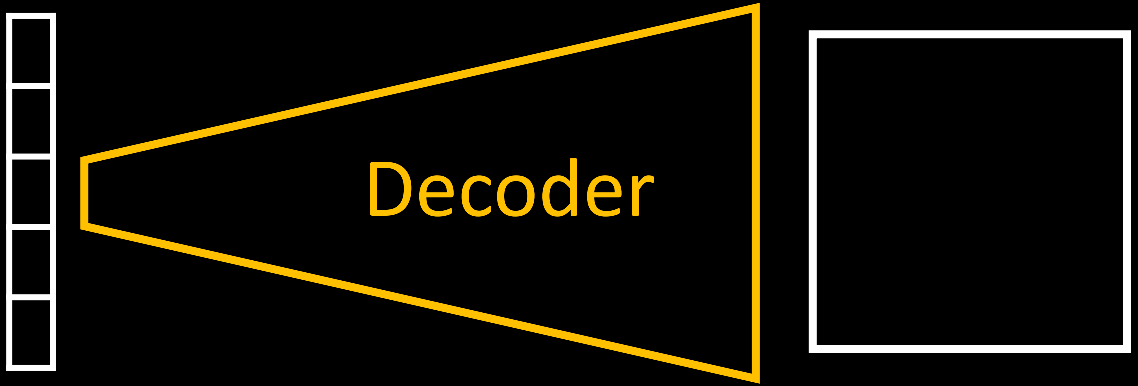
We want to **generate new images**:



Interaction

We want to **generate new images**:

- New random vector z_1 -> new random image I_1



$z_1 \in \mathbb{R}^{512}$ … z_1 is a vector of 512 real numbers

$\downarrow$

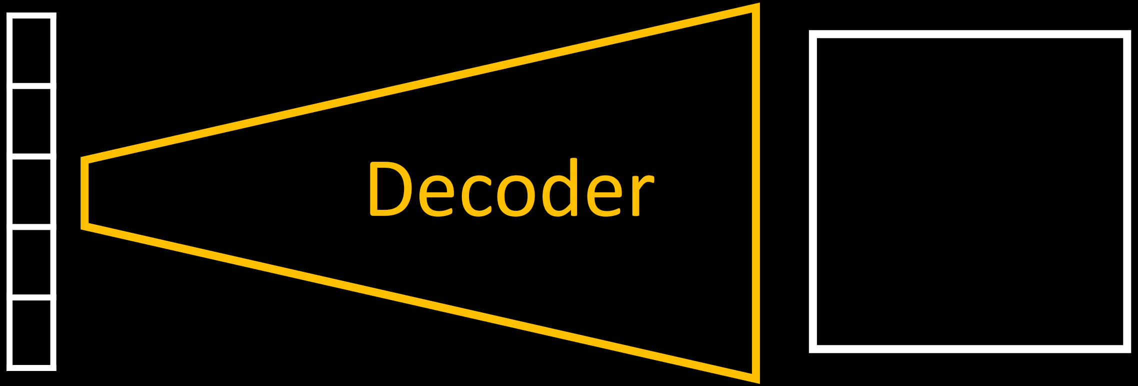
I_1

z1 = [0.1, 0.2, …, 0.9, -0.2, 0.3]

Interaction

We want to **generate new images**:

- New random vector z_1 -> new random image I_1
- Another random vector z_2 -> another random image I_2



$z_1 \in \mathbb{R}^{512}$

$\downarrow$

I_1

$z_2 \in \mathbb{R}^{512}$

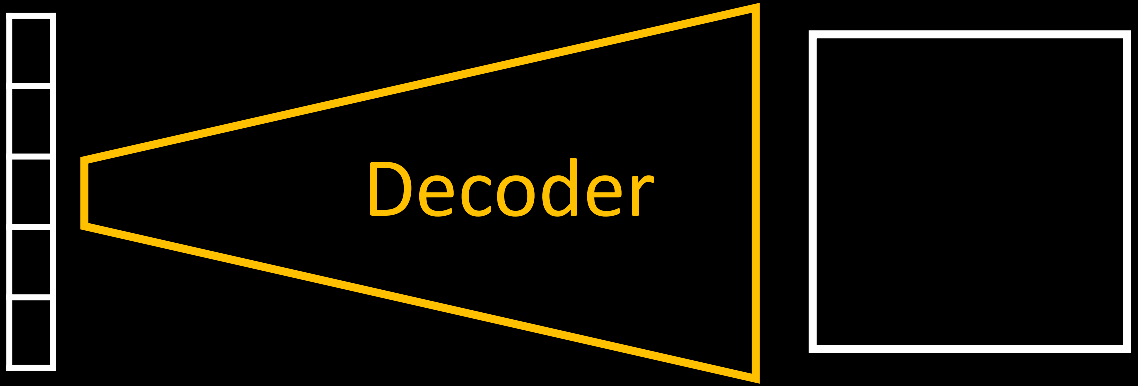
$\downarrow$

I_2

Interaction

We want to **generate new images**:

- New random vector z_1 -> new random image I_1
- Another random vector z_2 -> another random image I_2



$z_1 \in \mathbb{R}^{512}$ - - - - - - -> $0.4^* z_1 + 0.6^* z_2$ <- - - - - - - - - - - - - $z_2 \in \mathbb{R}^{512}$

$z_1 \in \mathbb{R}^{512} \rightarrow I_1$

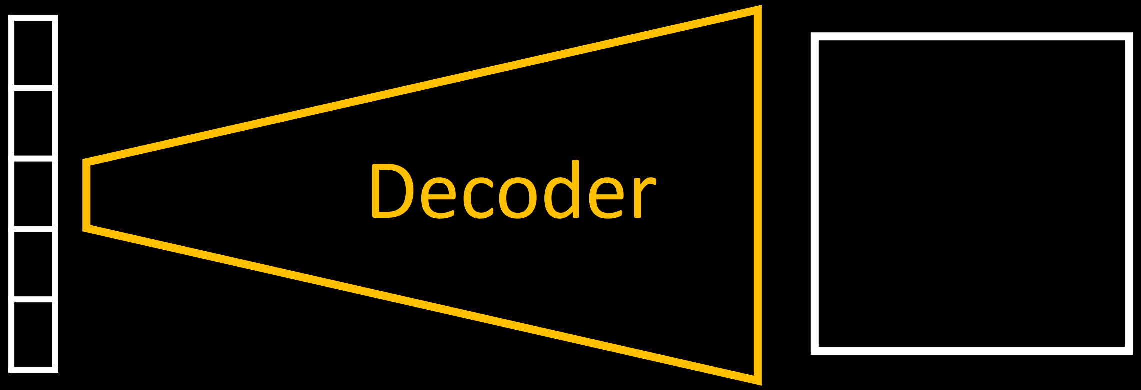
$z_2 \in \mathbb{R}^{512} \rightarrow I_2$

We can easily mix these vectors and get a new vector of 512 numbers.

Interaction

We want to **generate new images**:

- New random vector z_1 -> new random image I_1
- Another random vector z_2 -> another random image I_2



$z_1 \in \mathbb{R}^{512}$ $\downarrow$ I_1

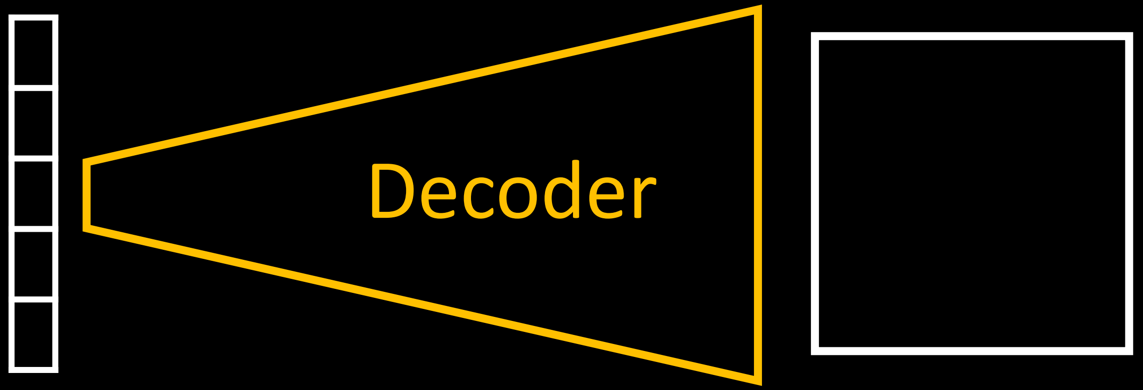
$0.4 * z_1 + 0.6 * z_2$ $\downarrow$ Mix 0.4 of I_1 and 0.6 of I_2

$z_2 \in \mathbb{R}^{512}$ $\downarrow$ I_2

Interaction

We want to **generate new images**:

- New random vector z_1 -> new random image I_1
- Another random vector z_2 -> another random image I_2



$z_1 \in \mathbb{R}^{512}$ $\qquad$ $a^*z_1 + (1.0\text{-}a)^*z_2$ $\qquad$ $z_2 \in \mathbb{R}^{512}$

$\downarrow$ $\qquad$ **Interpolation!** $\qquad$ $\downarrow$ $\qquad$ $\downarrow$

I_1 $\qquad$ Mix of images I_1 and I_2 $\qquad$ I_2

Interpolation

- Mixing in the **pixel space**:

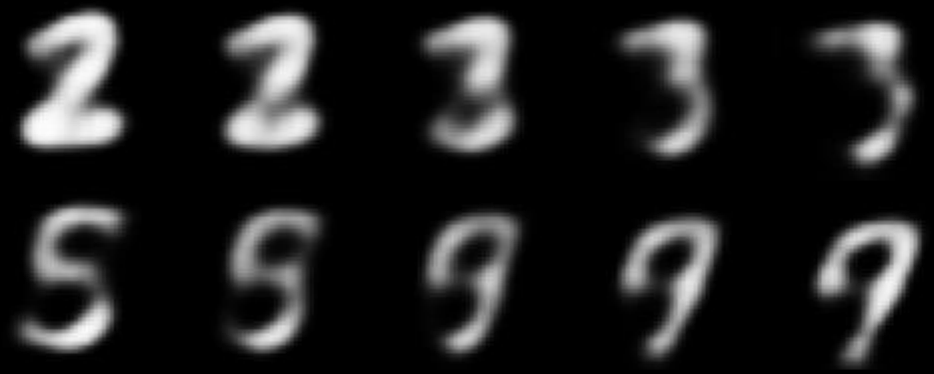
$$a^*I_1 + (1.0\text{-}a)^*I_2$$

… where each image is a matrix

- Mixing in the **latent space**:

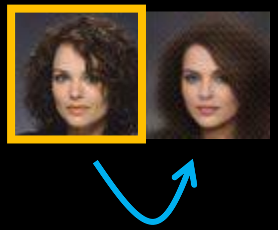
$$a^*z_1 + (1.0\text{-}a)^*z_2$$

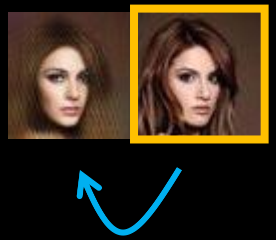
… where each image is generated from the mixed vector



AutoEncoder specific interaction I.

- **Encoding** **real samples** into their latent representations

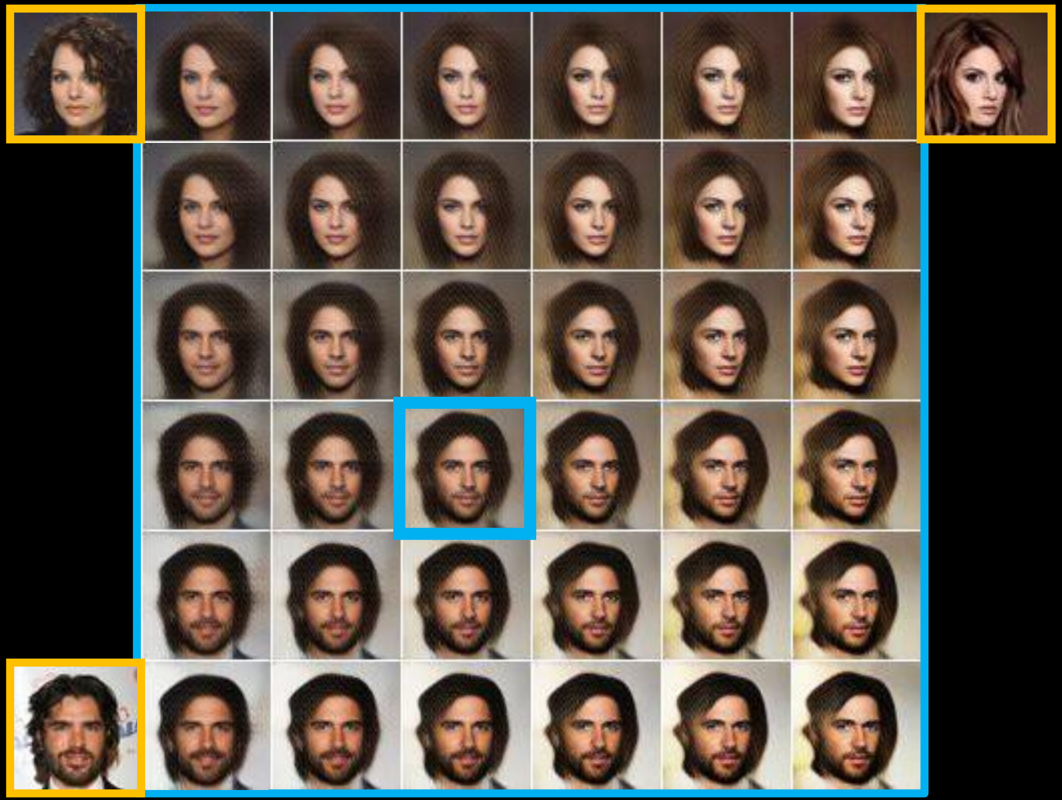




PS: the encoded image is often not reconstructed perfectly (we can notice a sort of blurry approximation)

AutoEncoder specific interaction I.

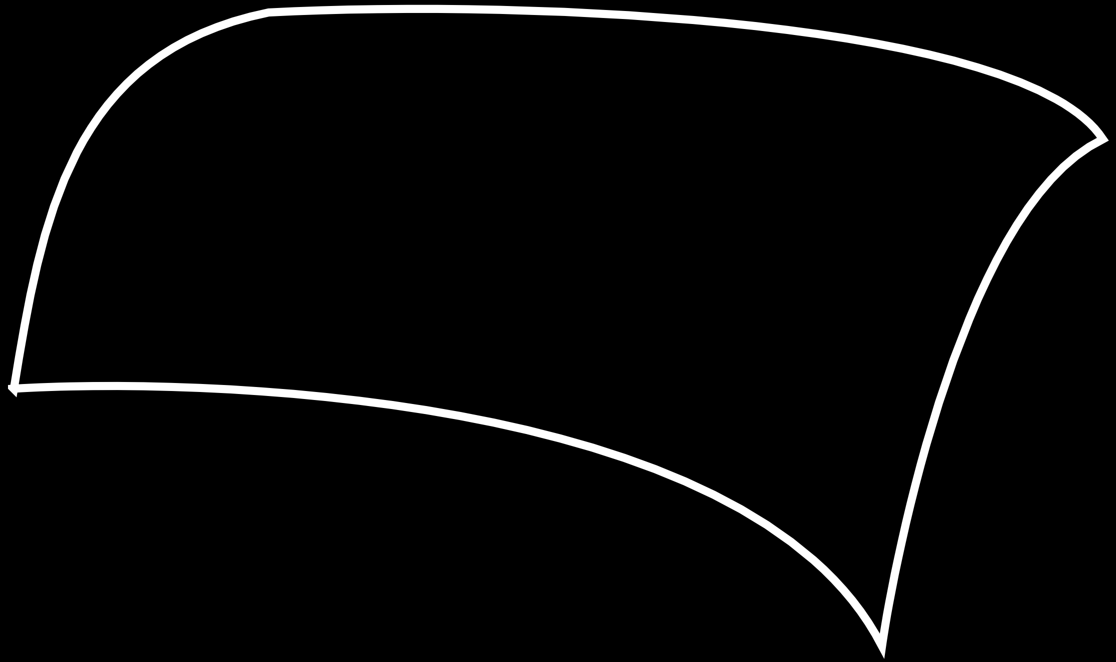
- **Encoding real samples** into their latent representations



$$\mathbf{z}_{\mathbf{mix}} = a^*z_1 + b^*z_2 + c^*z_3$$
$$a+b+c = 1.0$$

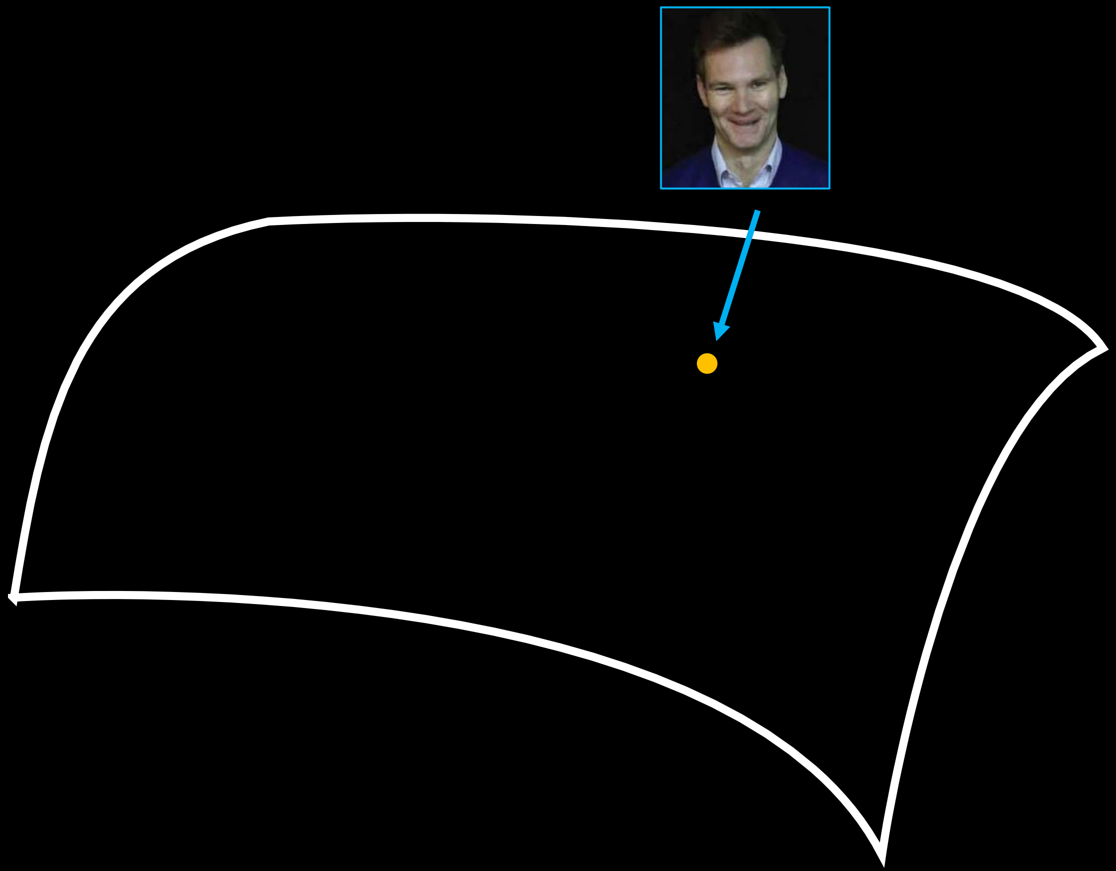
- Then using these, we can **generate interpolations**

AutoEncoder specific interaction II.



This is a **high dimensional space** (like this *cloth* shape in 3D, but in 512D instead), in which we can explore relations between data.

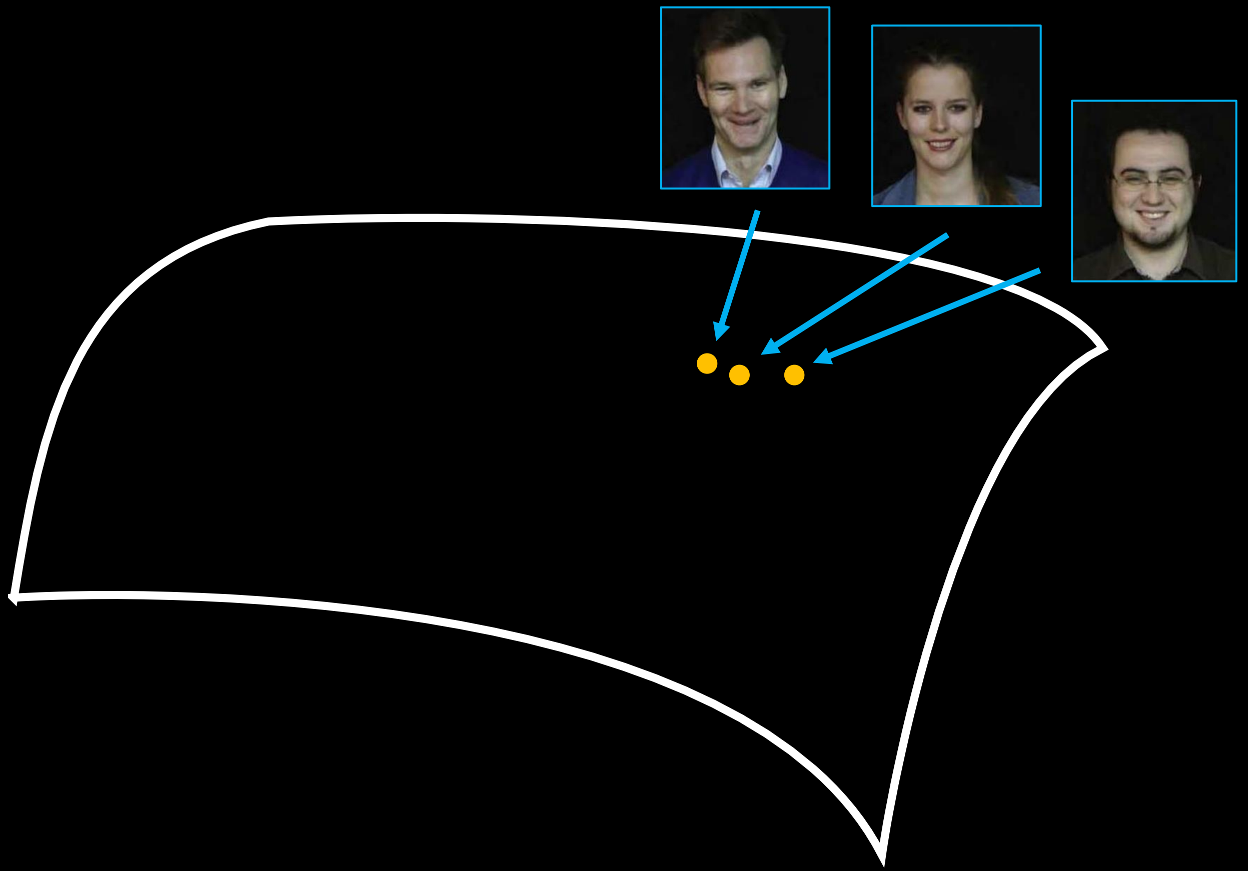
AutoEncoder specific interaction II.



This is a **high dimensional space** (like this *cloth* shape in 3D, but in 512D instead), in which we can explore relations between data.

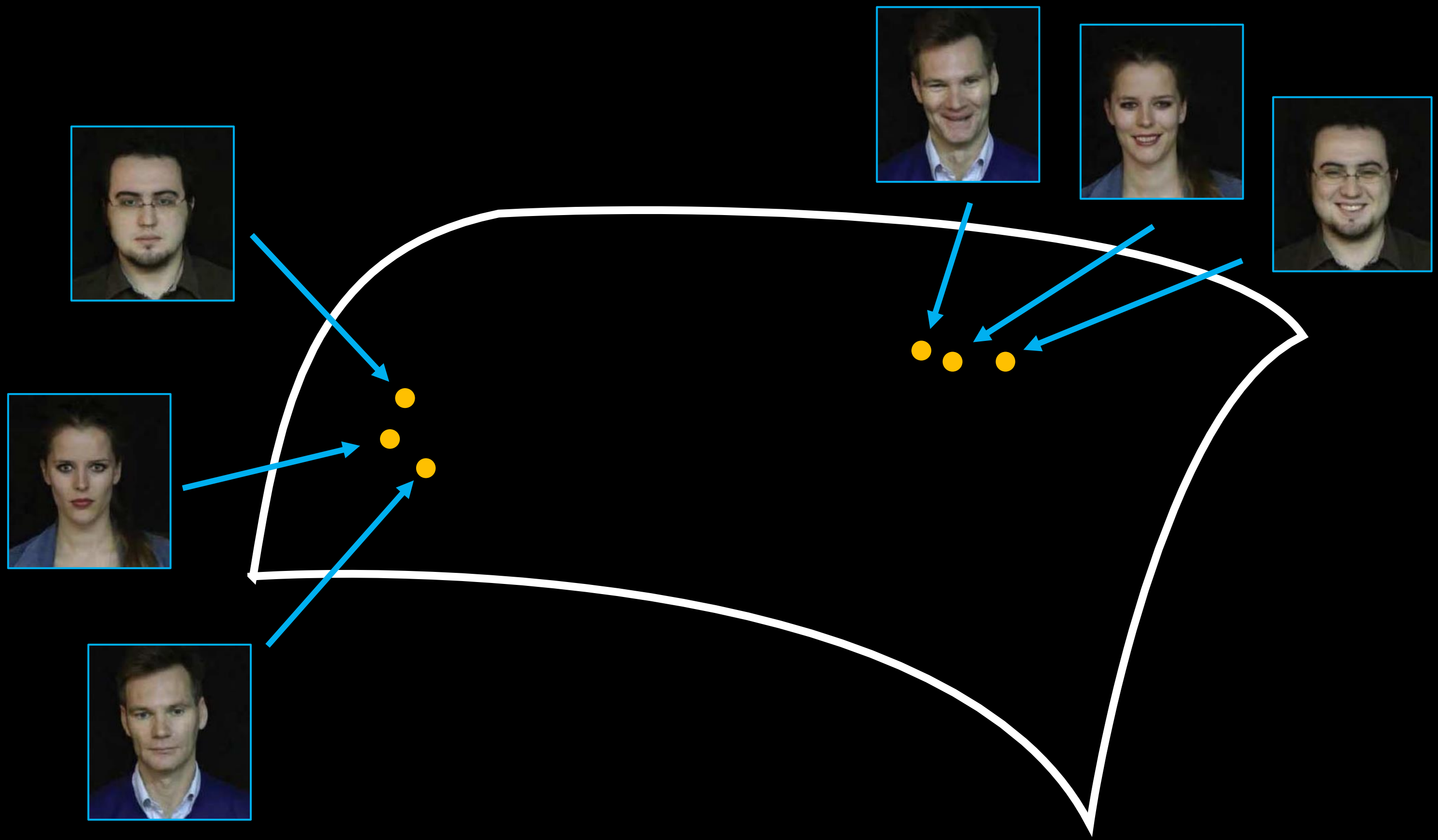
Each **point** in this space **is a vector of 512 numbers** (and so each point is an encoded real sample – and can also be used to generate an image)

AutoEncoder specific interaction II.



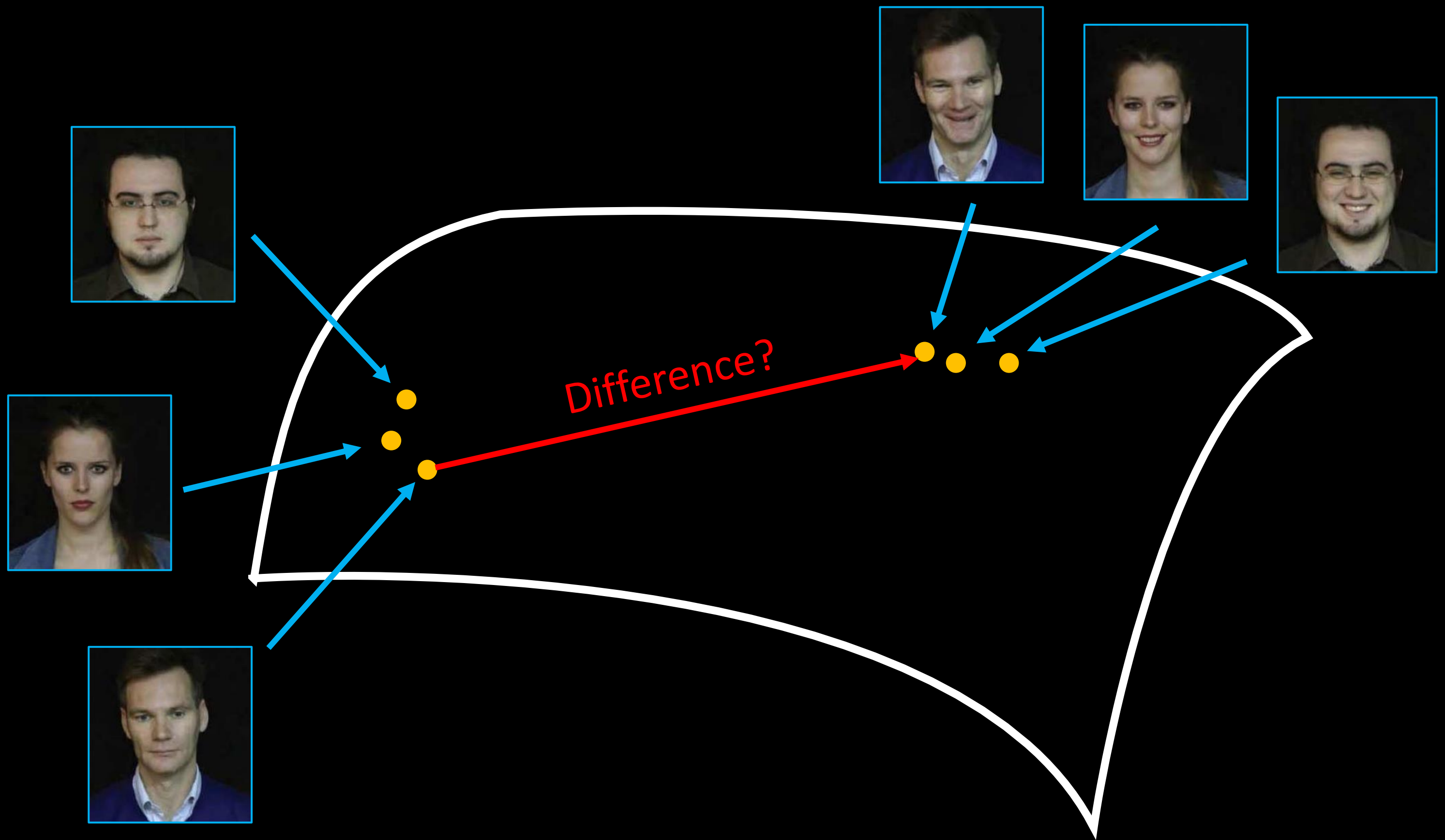
If we encoded images *with some property* and *without this property* (for example: **smiling / neutral** expression) ...

AutoEncoder specific interaction II.



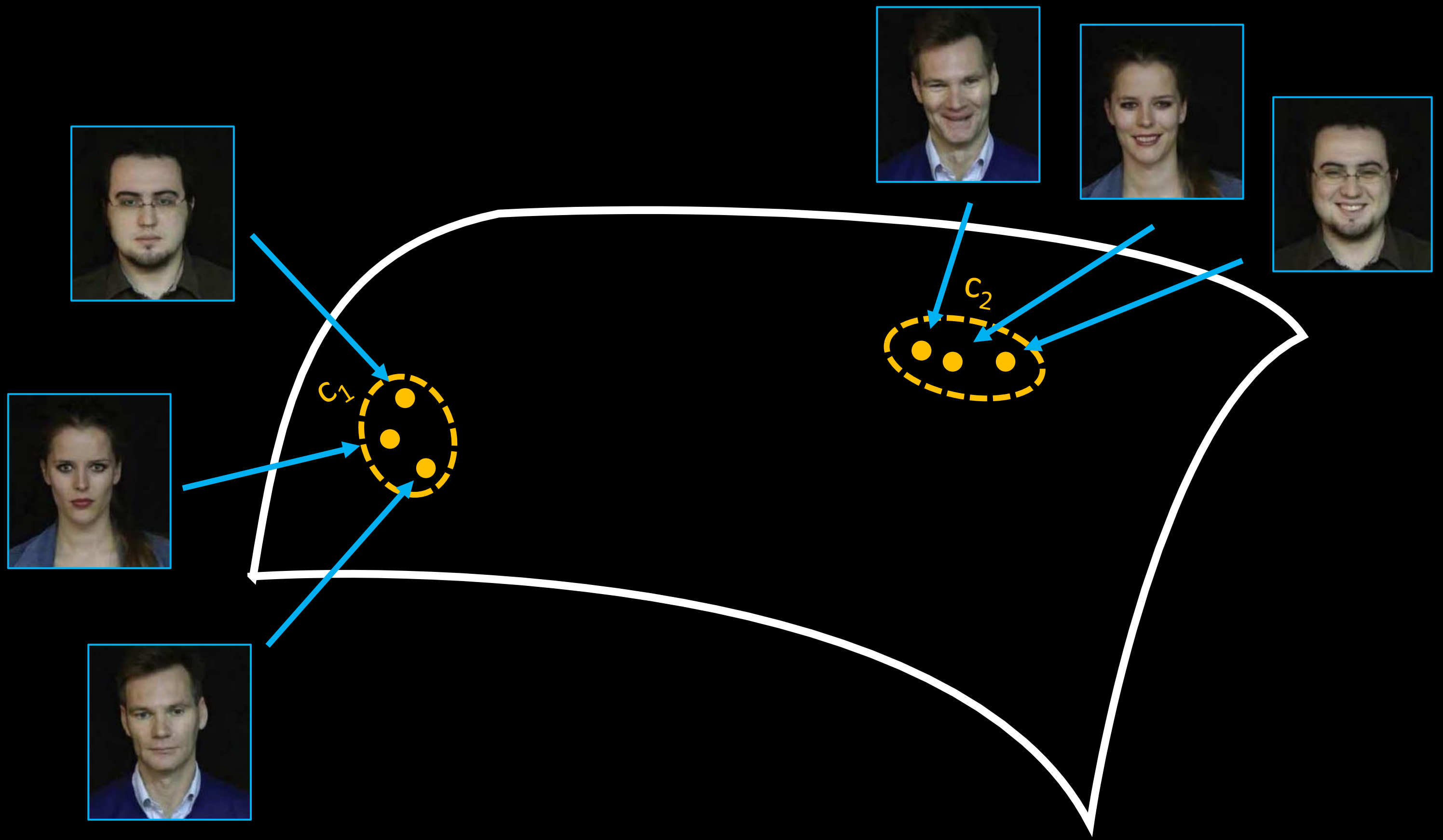
If we encoded images *with some property* and *without this property* (for example: **smiling / neutral** expression) …

AutoEncoder specific interaction II.



If we encoded images *with some property* and *without this property* (for example: **smiling / neutral** expression) …

AutoEncoder specific interaction II.

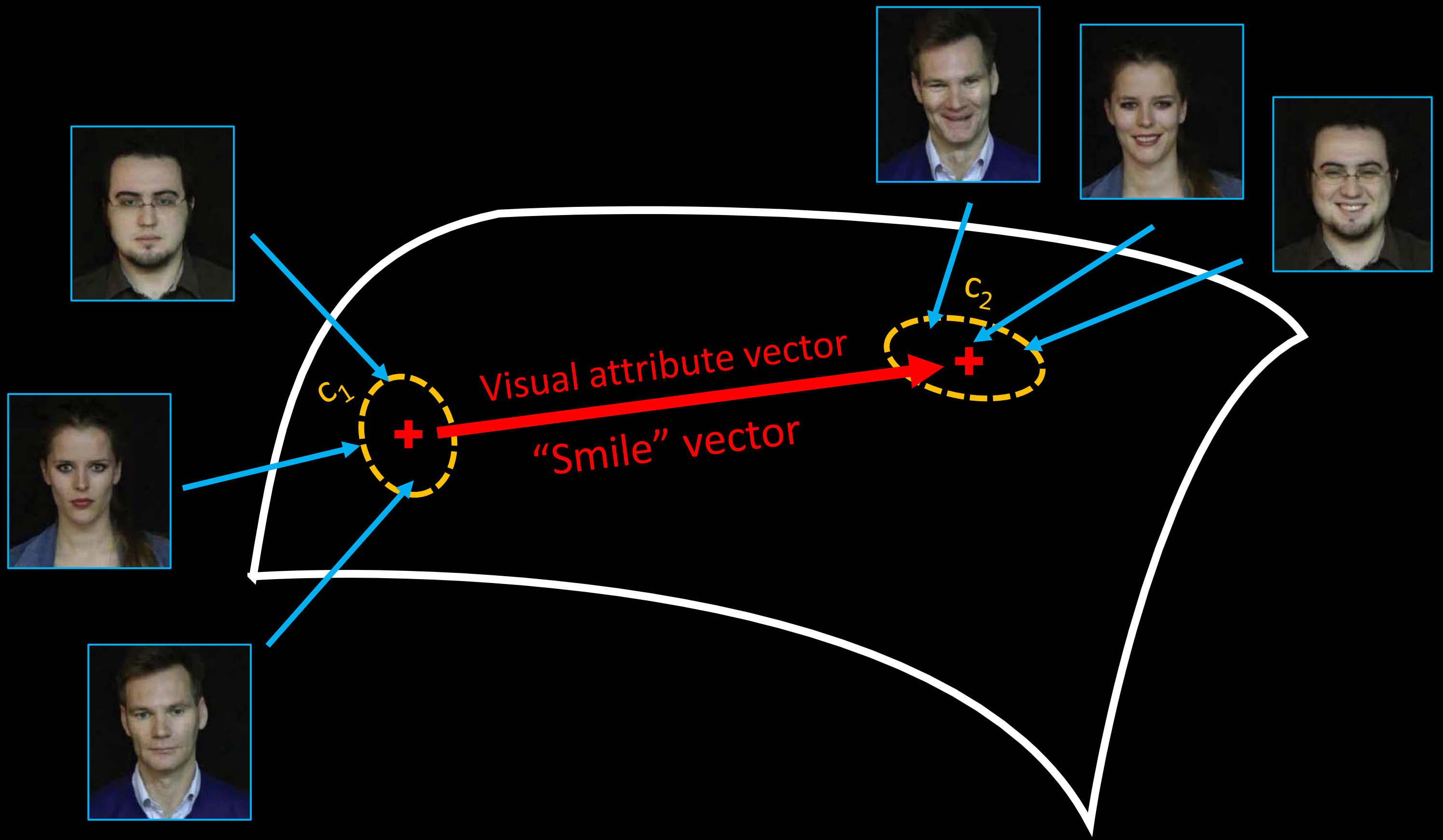


If we encoded images *with some property* and *without this property* (for example: **smiling / neutral** expression) …

We can find clusters and check their relative positions:

v = centroid of c_1 – centroid of c_2

AutoEncoder specific interaction II.

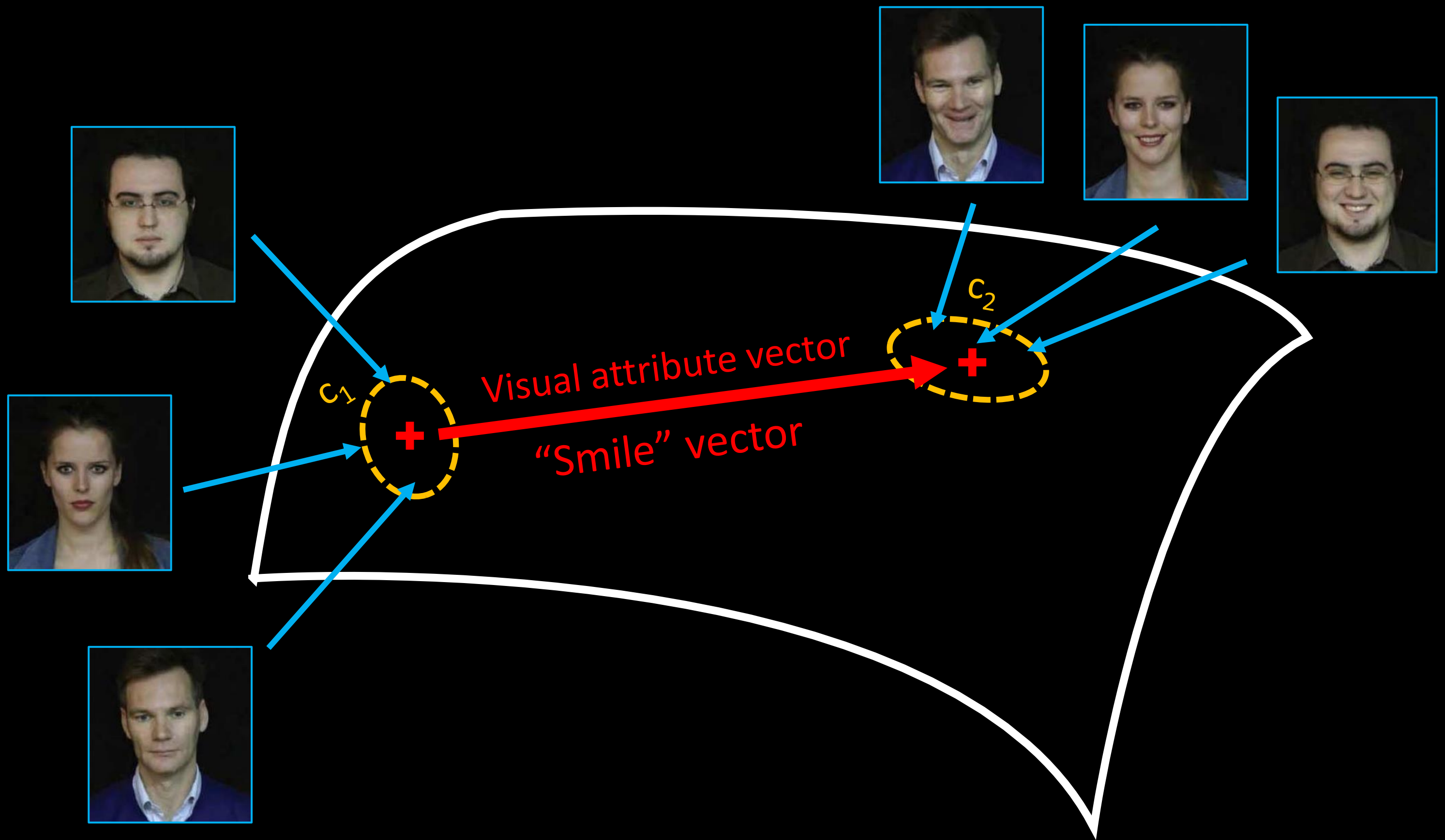


If we encoded images *with some property* and *without this property* (for example: **smiling / neutral** expression) …

We can find clusters and check their relative positions:

v = centroid of c_1 – centroid of c_2

AutoEncoder specific interaction II.



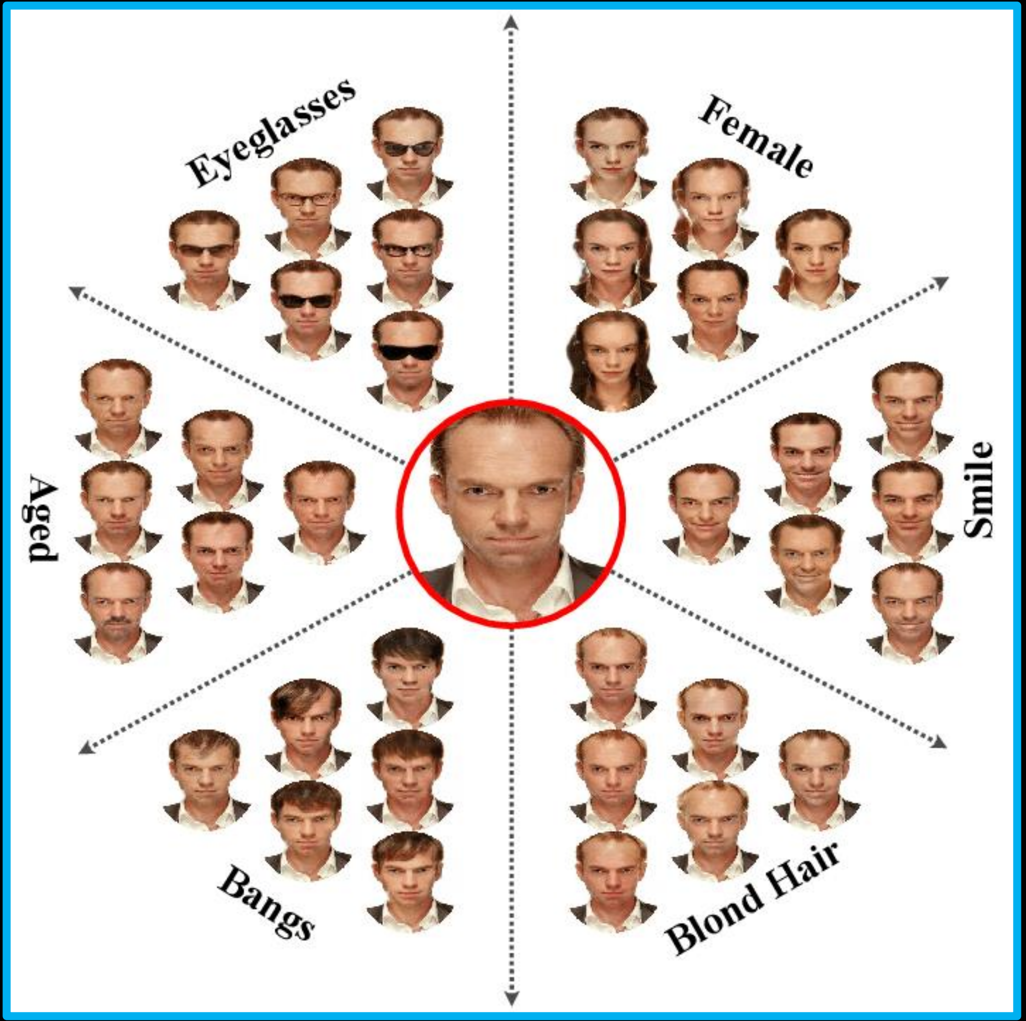
If we encoded images *with some property* and *without this property* (for example: **smiling / neutral** expression) …

We can find clusters and check their relative positions:

v = centroid of c_1 – centroid of c_2

- With labeled datasets we can extract **visual attribute vectors**
- We can call this **latent space arithmetic**

Visual attribute vectors



- **Visual attribute vectors:**
  - Eyeglasses vector
  - Smile vector
  - Blond hair vector
  - Bangs vector
  - Age vector
  - “Female” vector

- *PS: Heavily depends on what we labelled “smile”, “blond” … or “female” …*

>> See Animation link <<

PS2: this is secretly a special implementation of a GAN (link) , but it illustrates this point nicely …

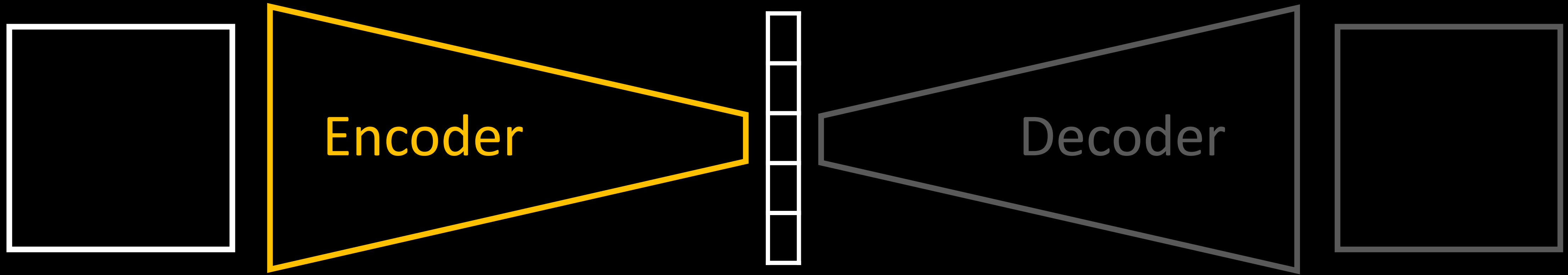
Side note: Diversity of datasets …

- **Celeb A** dataset
  - Diversity of samples
  - Choice of categories (planes of division)
  - Equal numbers of samples …

  - (etc … etc …)

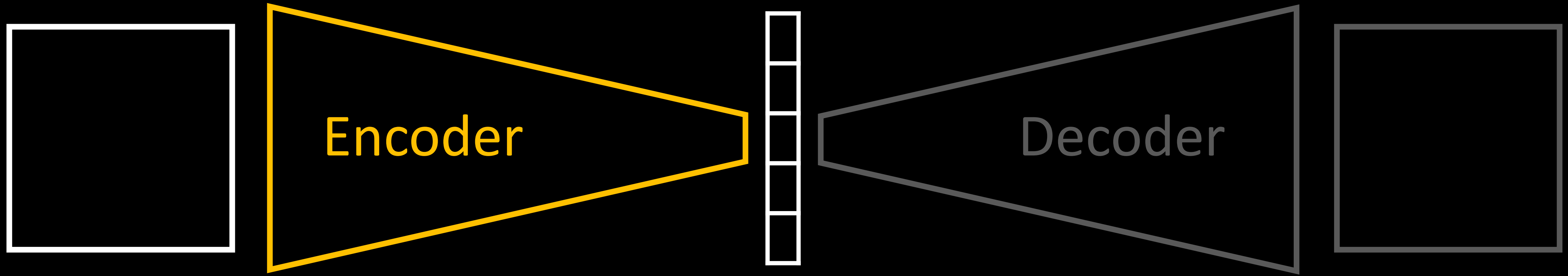
Weird uses of AutoEncoders I.

- When we need an **arbitrary feature extractor / transformer** of image -> feature



Weird uses of AutoEncoders I.

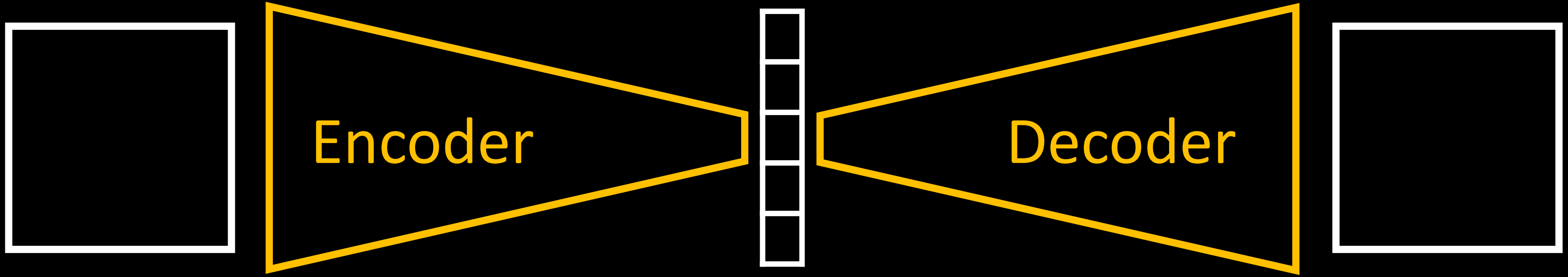
- When we need an **arbitrary feature extractor / transformer** of image -> feature
  - If we had labels (**supervised scenario**), we could used something similar like AlexNet …
  - Without labels (**unsupervised scenario**) we can instead use these methods – AEs or GANs



Intuition: if the encoder can create a “good enough” representation, that it can be used for reconstruction -> then we hope that it will be good in other scenarios as well

Weird uses of AutoEncoders II.

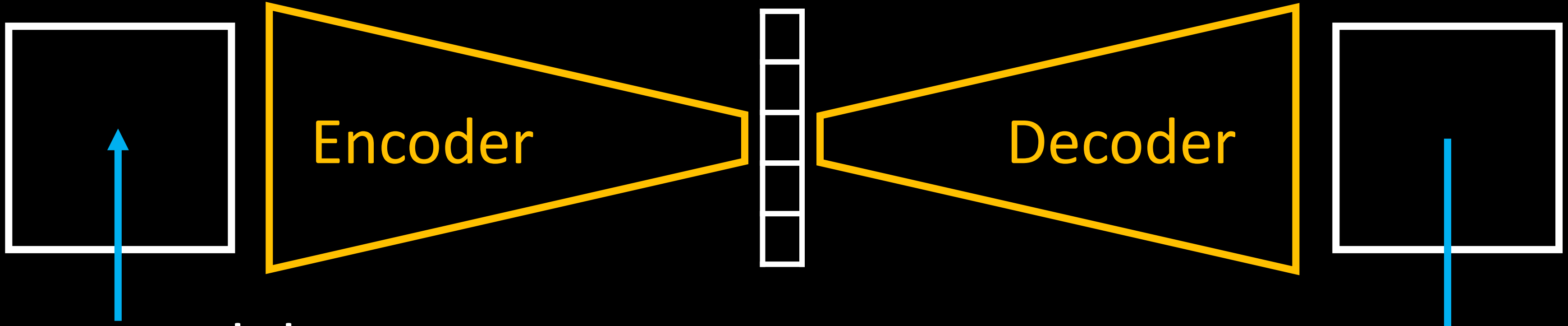
Train on a first dataset:



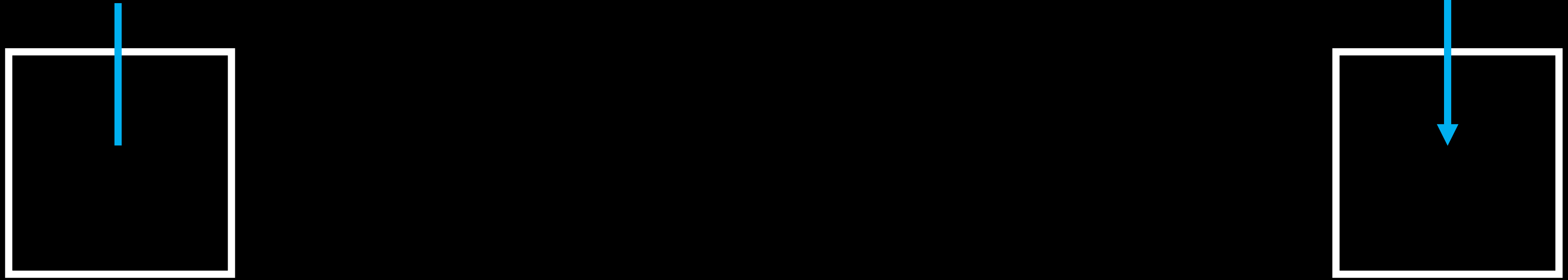
Intuition: **reinterpret material** with shapes/details/imagery of another material

Weird uses of AutoEncoders II.

Train on a first dataset:



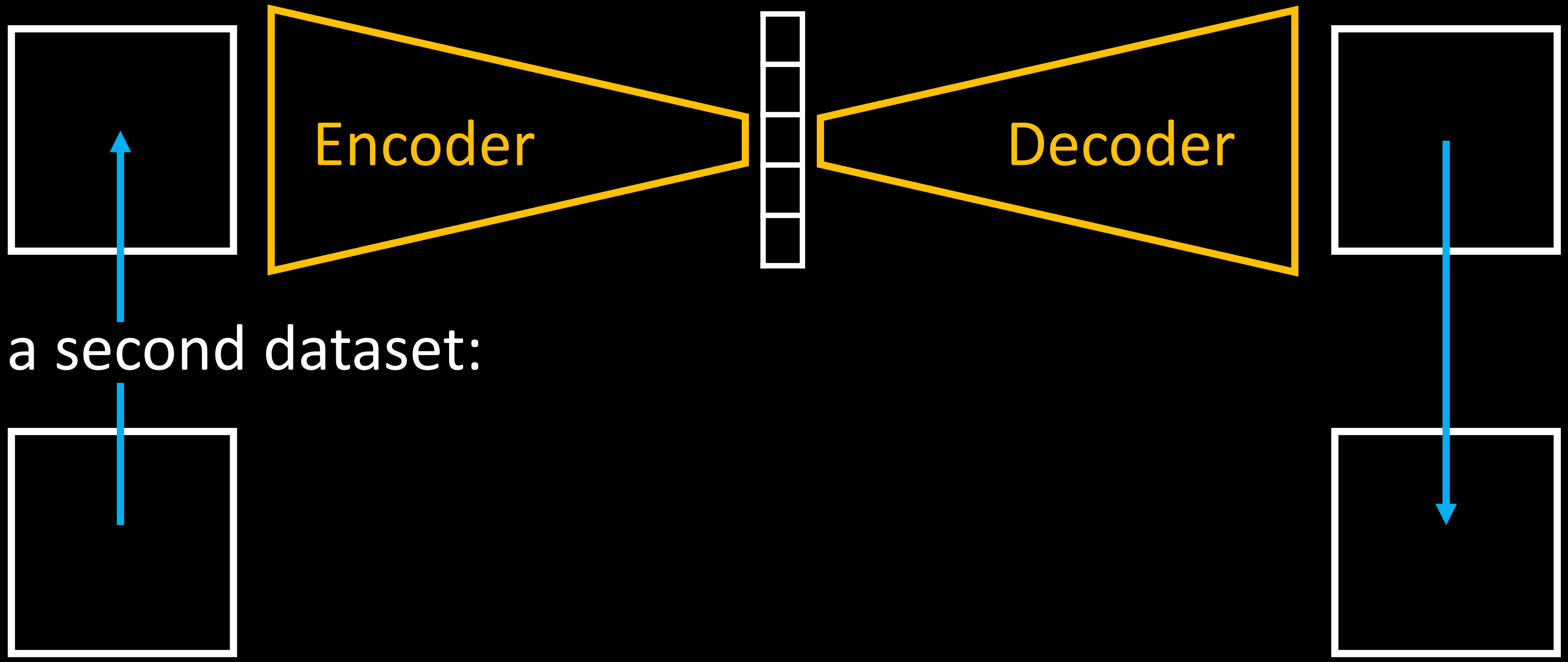
Use with a second dataset:



Intuition: **reinterpret material** with shapes/details/imagery of another material

Weird uses of AutoEncoders II.

Train on a first dataset:



Example [video 1]

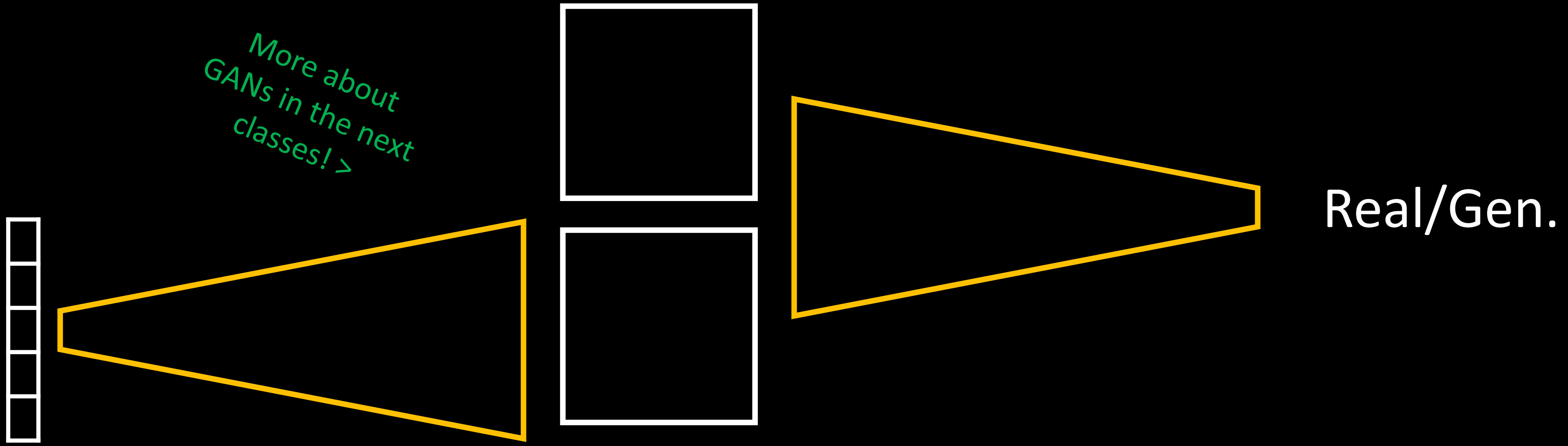
Use with a second dataset:

Example [video 2]

Intuition: **reinterpret material** with shapes/details/imagery of another material

More on: blog

Spoilers: Generative Adversarial Networks



- Rephrase it as a **two-player game**. One network is supposed to generate images – and the other one is responsible for discriminating if they are real or generated.

Spoilers : Differences between GANs and AEs

AE

-> **Blurry** results

Error from the square distance

GAN

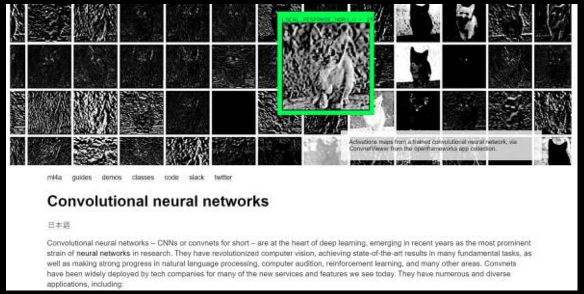
-> **Sharper**, more details

Error is propagated from G/D

- **But**: In the default version GANs don't have an Encoder (*image to latent vector translation* network) = we can't encode real samples

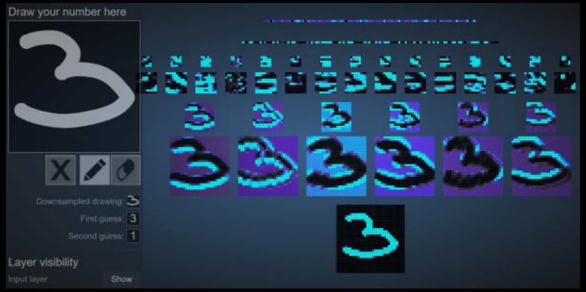
< There are cool hybrid models that can though!

Links and additional readings:



Convolutional Neural Networks

- Convnets on ML4A: ml4a.github.io/ml4a/convnets/
- Interactive Convolutional Neural Network (runs in the browser and has good visualization): cs.cmu.edu/~aharley/vis/conv/flat.html



AutoEncoders

- Details about types of VAEs: lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html
- VAE visualizations of latent space: hackernoon.com/latent-space-visualization-deep-learning-bits-2-bd09a46920df

End of the lecture

***)** PS: follows material for the practical session …