MACHINE LEARNING

MEI/1

University of Beira Interior,
Department of Informatics

Hugo Pedro Proença,
hugomcp@di.ubi.pt, 2025/2026

Machine Learning

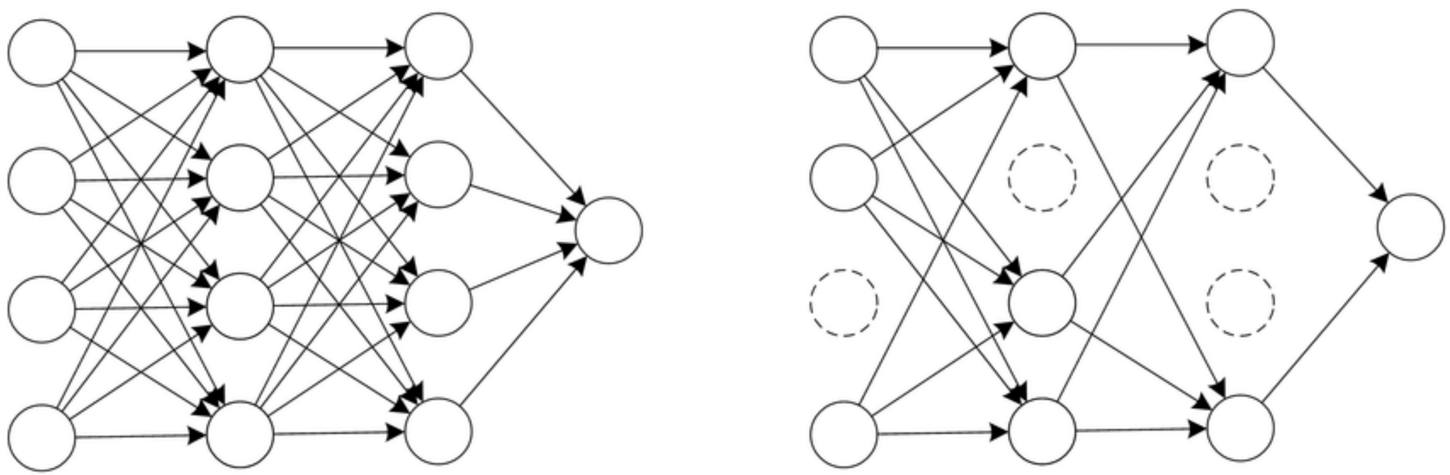
[07]

Syllabus

- Deep Learning Architectures
- Self-Supervised Learning

CNNs: Other Layers

- **Dropout** Layers. This kinds of layers drops out units of a neural network <u>during the learning phase</u>.
  - Typically, a proportion (0, 1) of neurons is randomly chosen and not considered for a particular "**forward**/**backward**" pass.
  - Dropout is an approach to regularization in neural networks which helps to avoid interdependent learning amongst the neurons.
  - Recall that regularization is way to **prevent over-fitting**, by adding a penalty to the loss function.
  - It is applied exclusively to the fully connected layers of a CNN model.

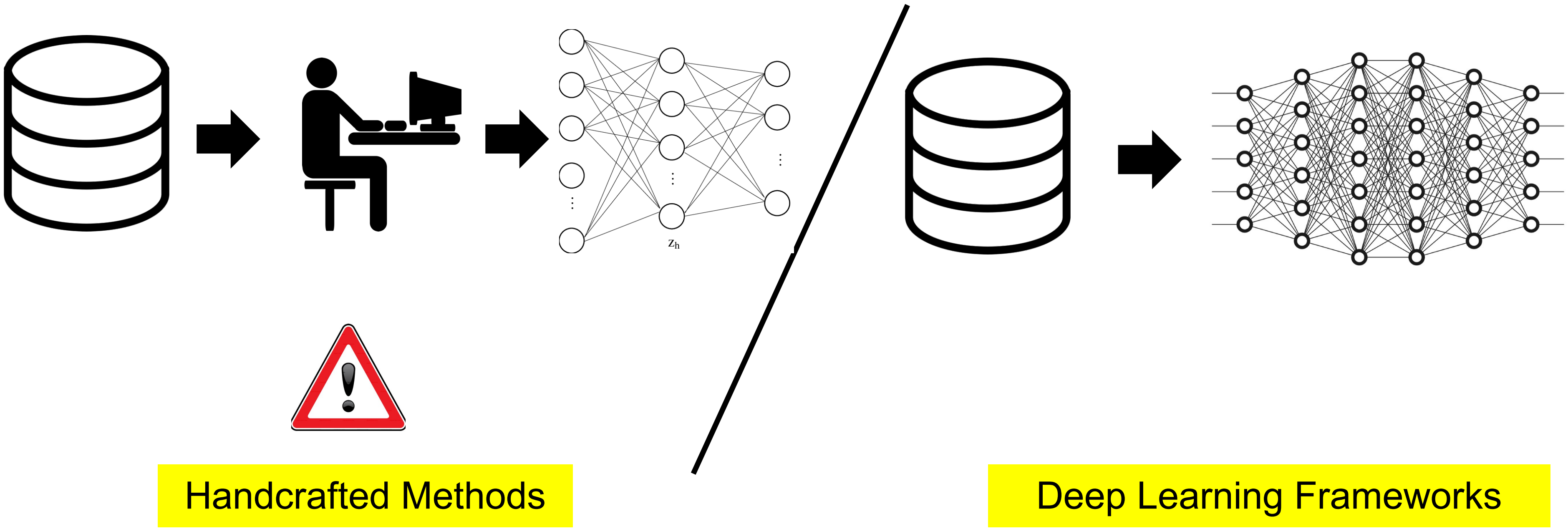


CNNs: Other Layers

- **Batch Normalization** Layers. To increase the stability of a neural network, this kind of layers normalizes the output of a previous layer by **subtracting** the batch mean and **dividing** by the batch standard deviation.
- This kind of layer can be added both after fully connected layers, but also after convolutional layers.
- Typically, using batch normalisation: 1) allows **higher learning rates;** 2) makes weights **easier to initialise**, helping to reduce the sensitivity to the initial starting weights.
- As the activations of one layer are the inputs of the next one, each layer in the neural network receives – at each iteration – diferente input distributions. This is problematic because it forces each layer to continuously adapt to its changing inputs.
- Using Batch Normalization allows the layer to learn on a more stable distribution of inputs (close to a standardized Gaussian distribution) and accelerates the training of the network.

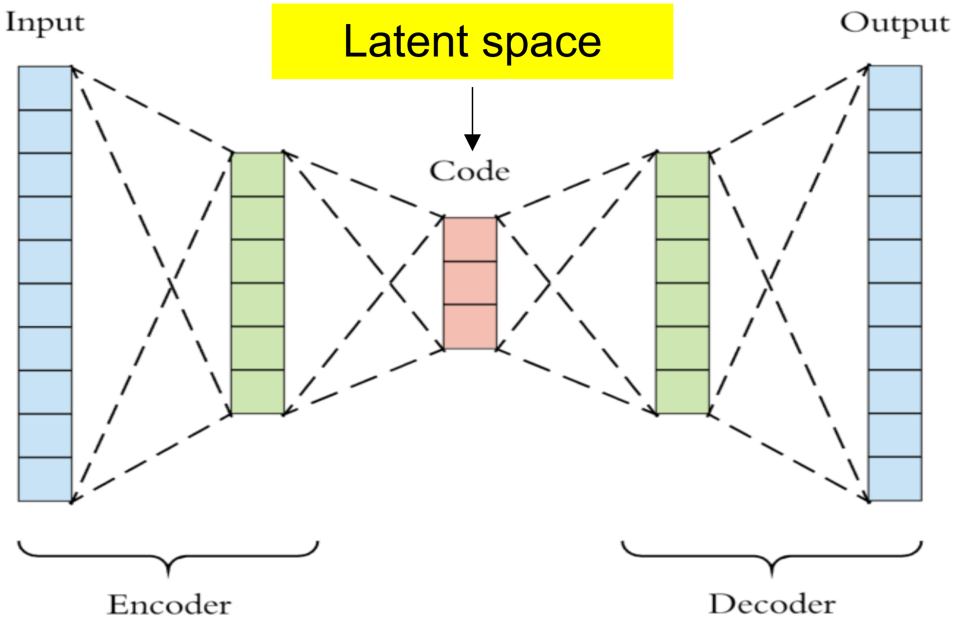
Deep Learning Architectures

- Deep Learning architectures are now "in the eye of the hurricane", and have been advancing the state-of-the-art in multiple Machine Learning problems (if not all…)
- Recall that the main advantage of Deep Learning-based solutions with respect to handcrafted approaches, is that this new generation of models also carries out the feature extraction phase in an automatic way.



Auto-Encoders

- Autoencoders are a class of Neural Networks that try to reconstruct the input itself. They are unsupervised in nature.
- Typically, the general structure of an auto-encoder has two parts:
  - The **Encoder** sub-network, that receives the original data and obtains a “latent space representation”;
  - The **Decoder** sub-network, that receives the latent code and attempts to reproduce the original data.

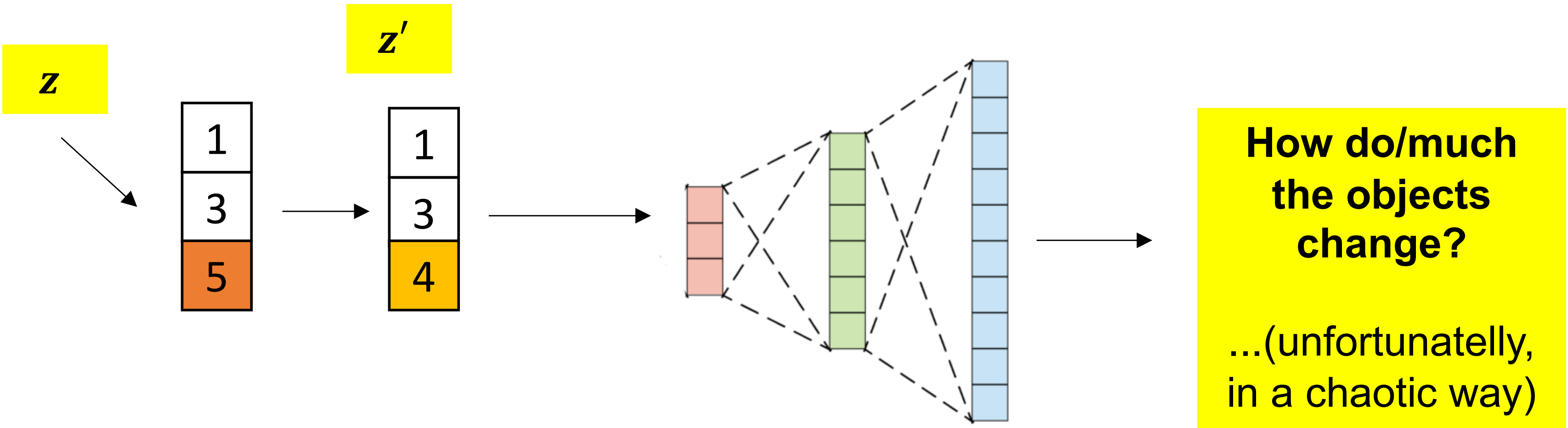


Auto-Encoders

- The first obvious application of auto-encoders is "**Data Storage and Transmission**"
  - Starting from a high-volume amount of information (size m), the latent code $\boldsymbol{z} \in \mathbb{R}^n$ is able to reconstruct the original data only with minor differences;
  - Obviously, n << m
- A second obvious application of using auto-encoders is to obtain a **compact feature representations** that can be used by Machine Learning models, for classification, regression or clustering purposes.
  - For such, it is assumed that a similarity between $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ (e.g., in terms of Euclidean/Co-sine distances) corresponds directly to the similarity of the corresponding original data

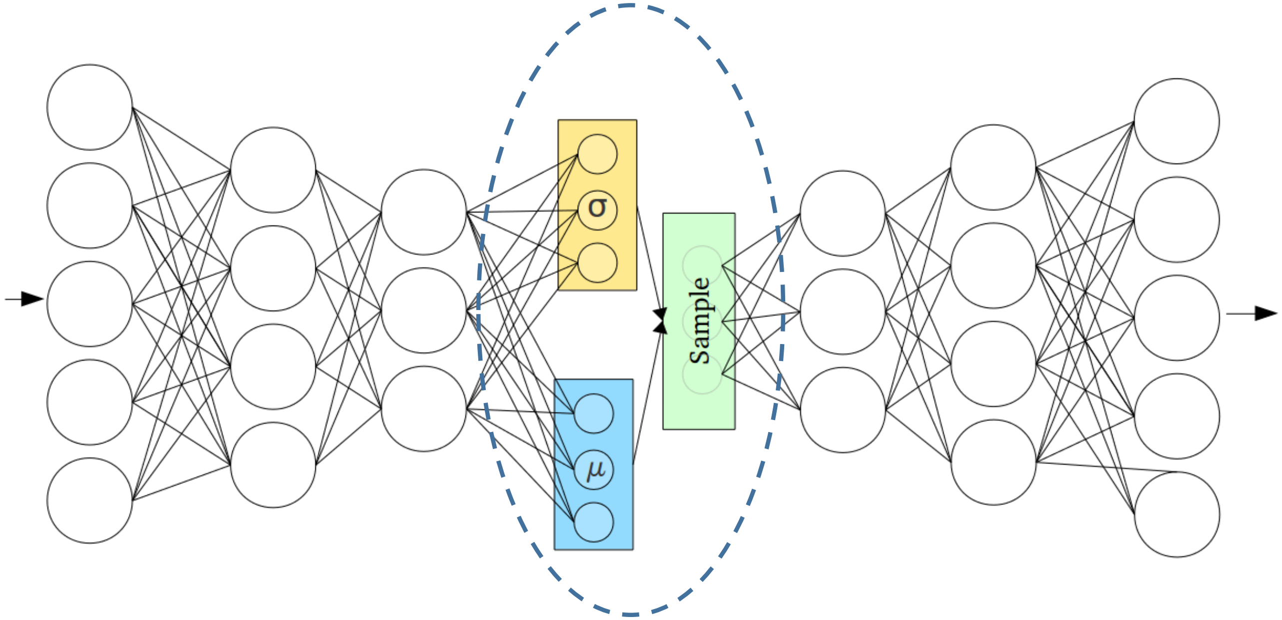
Auto-Encoders

- Subsequently, another ingenious application for auto-encoders was to "**Generate Data**"
  - There is a "**Generative**" paradigm of Machine Learning/Pattern Recognition models that attempts to model the phenomena to be handled
    - i.e., obtain an approximation of **p(C,I),** with "I" representing the input data and C the corresponding desired response.
  - This is in opposition to the "**Discriminative**" family of methods, which typically attempt to infer **p(C|I)**
- The idea in auto-encoders was to change some components in the latent code, to perceive the corresponding changes in the reconstructed data.



Variational Auto-Encoders

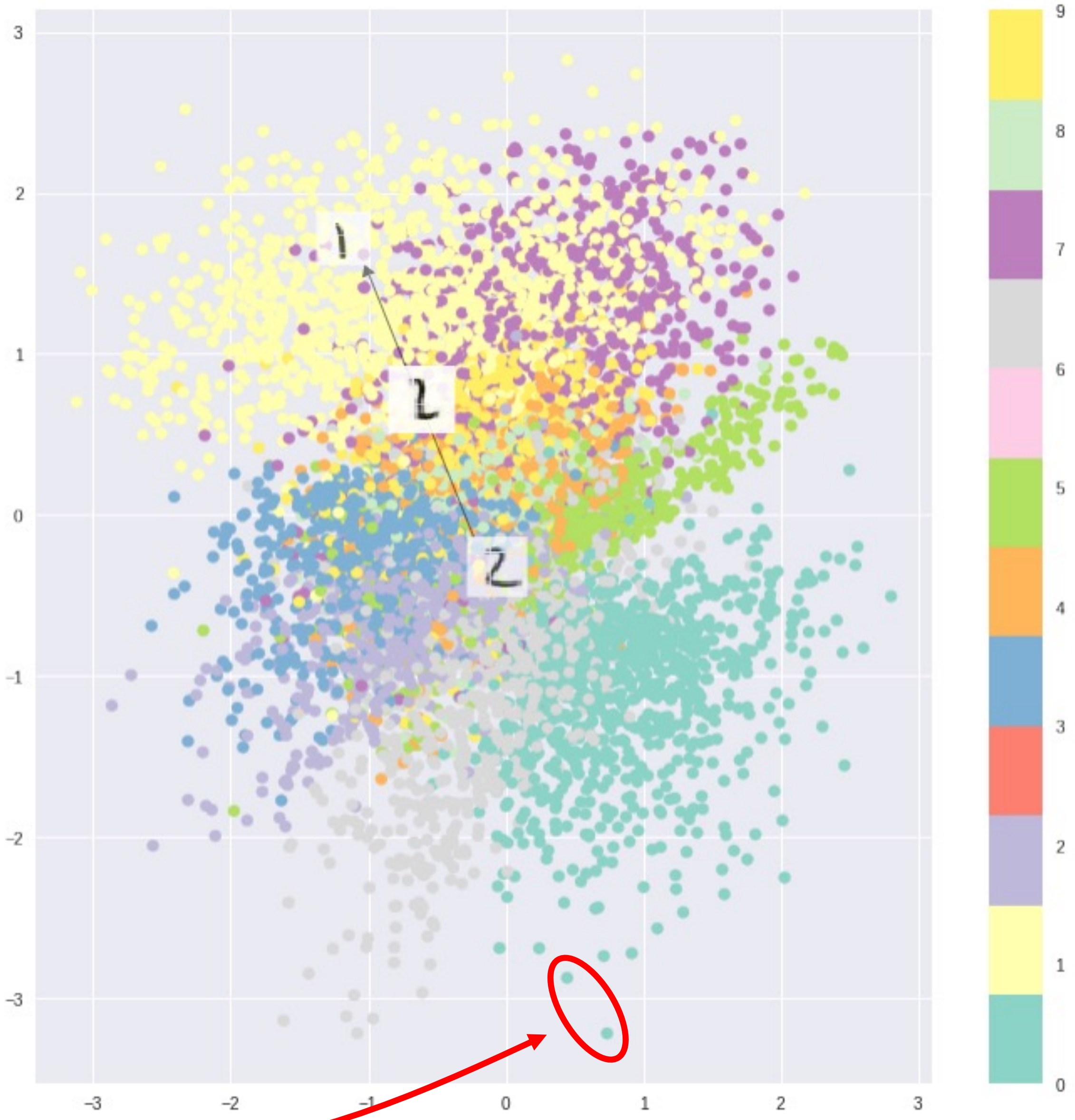
- This kind of models have arisen upon the difficulties in controlling the appearance/features of the reconstructed data .
  - Standard autoencoders can obtain compact representations $\boldsymbol{z}$ and reconstruct their inputs well.
  - However, the main problem, for generation, is that the latent space they convert their inputs to and where their encoded vectors lie, **may not be continuous**, or **allow easy interpolation**.
- The key novelty in variational auto-encoders is a layer that explicitly encodes **means** and standard deviations of the latent representations, which are sampled to generate a reconstructed sample.



$\mu = [0.5, 1.2]$
$\sigma = [0.1, 0.2]$
Sample two values from:
$[X_1 \sim N(0.5, 0.1),\ X_2 \sim N(1.2, 0.2)]$

Variational Auto-Encoders

- The $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ values allow a continuity in the latent space, that can be used to generate synthetic elements according to some **pre-defined properties** and appearance features



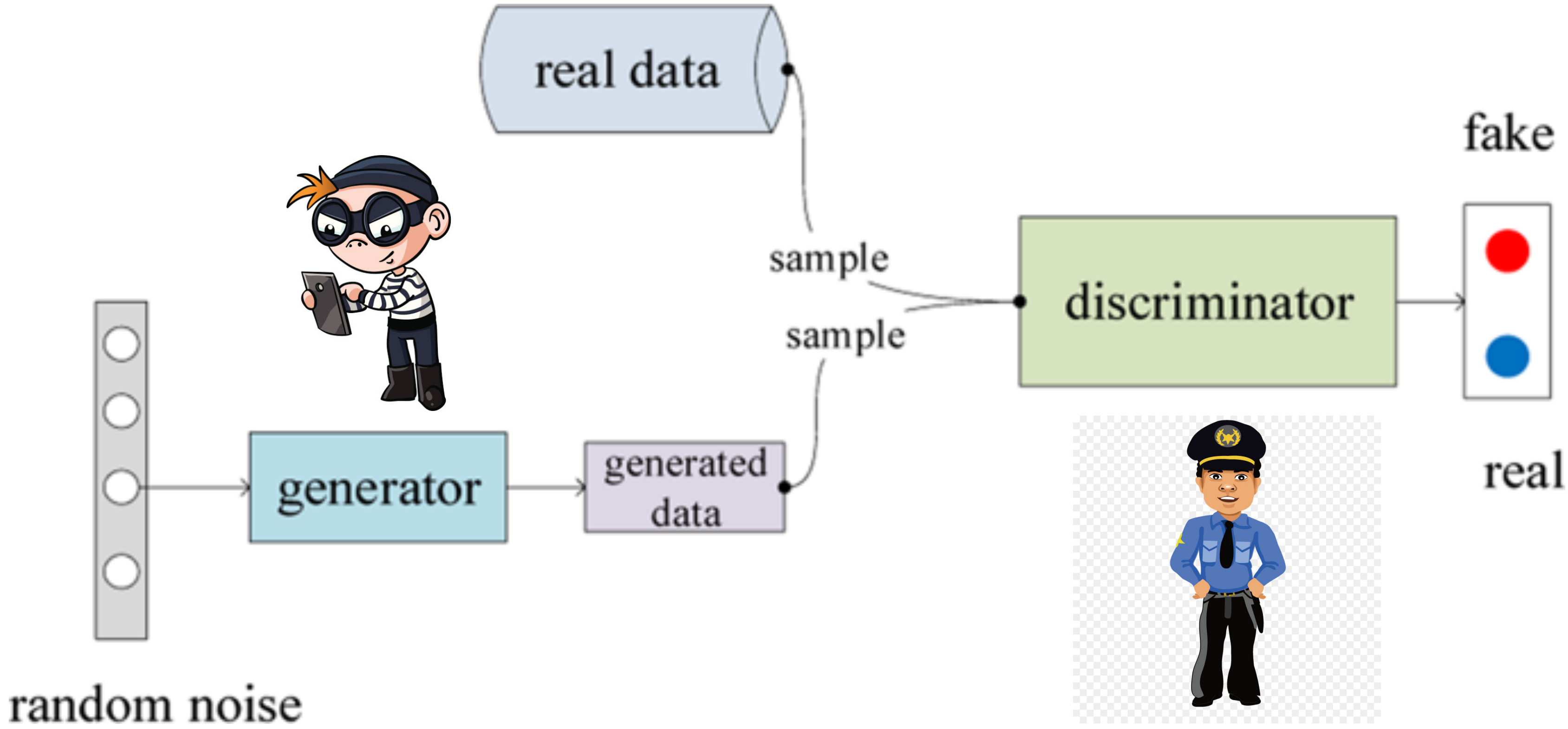
In practice terms, it is assured that neighbor elements in the latent space correspond to similar instances in the image space

Adversarial Learning

- Facebook's AI research director **Yann LeCun** called adversarial training “*the most interesting idea in the last 10 years in Machine Learning*”.
- **Generative Adversarial Networks** (GANs) are architectures that use two neural networks, competing one against the other (thus the “adversarial”) in order to generate new, synthetic instances of data that can pass for real data.
  - GANs were introduced in a paper by Ian Goodfellow and other researchers at the University of Montreal, including Yoshua Bengio, in 2014.
- GANs’ potential for both good and evil is huge, because they learn to mimic **any distribution of data**.
- GANs can be taught to create worlds eerily like our own in almost any domain: images, music, speech, prose…

GANs

- The basic idea in GANs is to have one network (**Generator**) trying to fool the other one, while the later (**Discriminator**) tries not to be fooled.
- This can be seen as a **Police Officer**←→ **Thief** game that, according to **Nash Game Theory**, typically converges into an equilibrium state.



GAN

- The **Discriminator** network is a typical binary classification CNN, that learns to distinguish between fake and real data.
- The **Generator** network receives one latent code (randomly generated, i.e., white noise) and produces one instance.
- The overall cost function is given by a two-player min-max game:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- That can be decomposed into:

Discriminator

$$\max_D V(D) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{“recognize genuine”}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{“recognize fakes”}}$$

Generator

$$\min_G V(G) = \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{“fool” D}}$$

GAN

- GANs are trained in an iterative way:

1. Generate a set of Fake data **F**

2. Train the Discriminator (with Real data **R** **(labelled 0)** and Fake Data **F** **(labelled 1)**) //Learns to distinguish R from F

3. Set Discriminator.trainable =FALSE

4. Train the GAN (with Fake Data F (**labelled 0**)) //Learns to fool D

5. Move to Step 1.

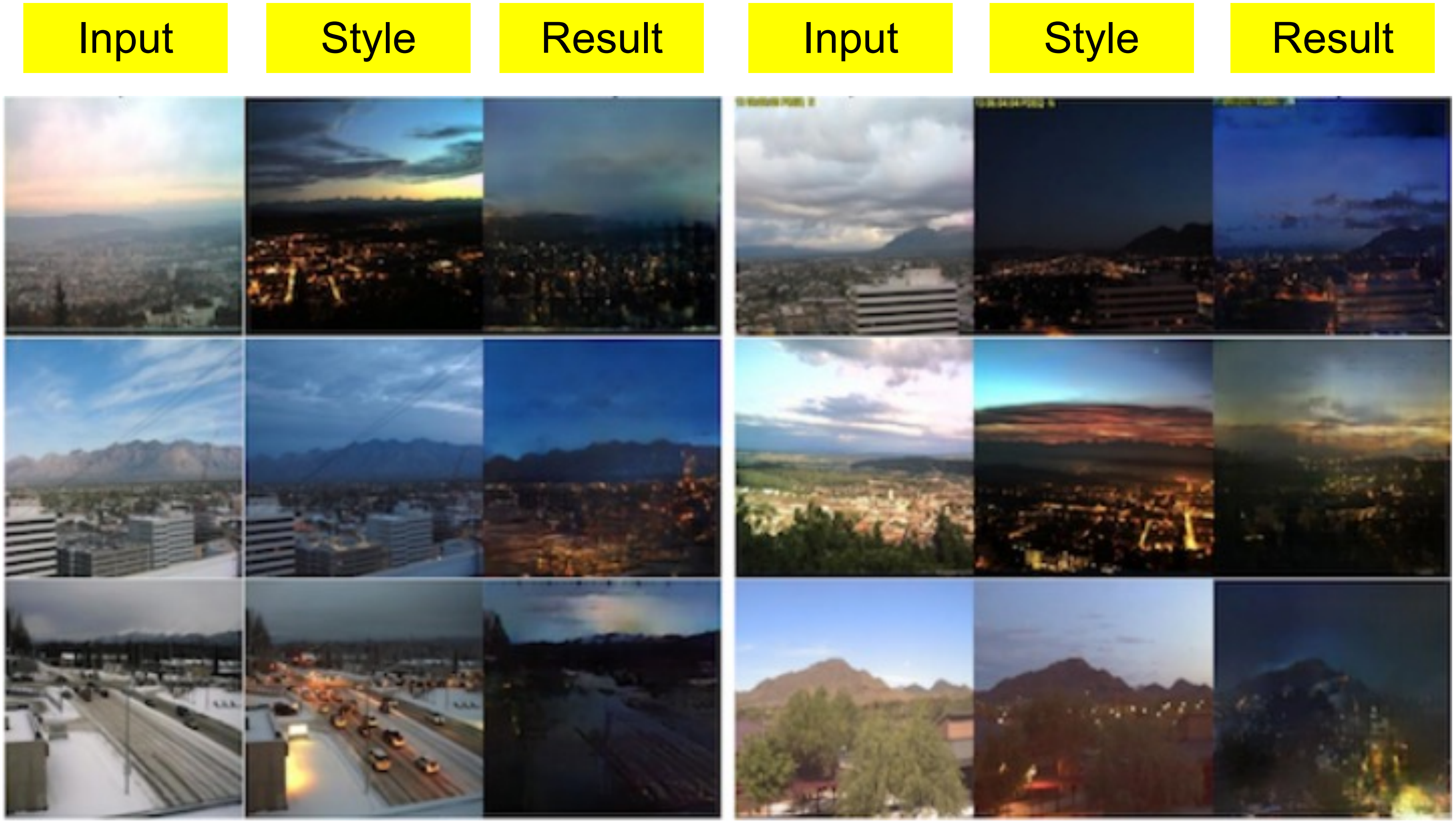
GANs Applications

- E.g., plausible realistic photographs of human faces:

These persons don't exist!!

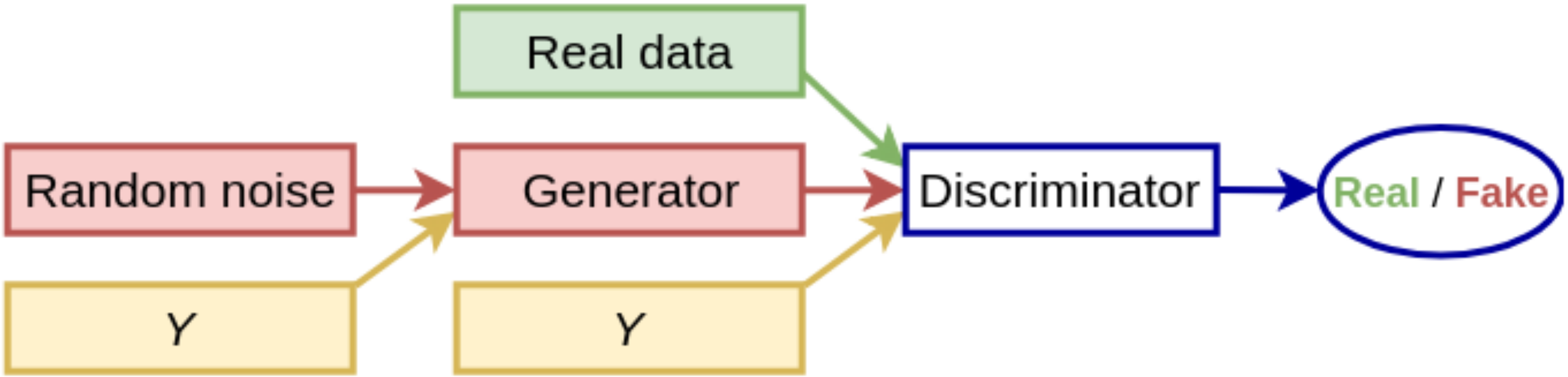
GANs Applications

- Image to Image Translation:



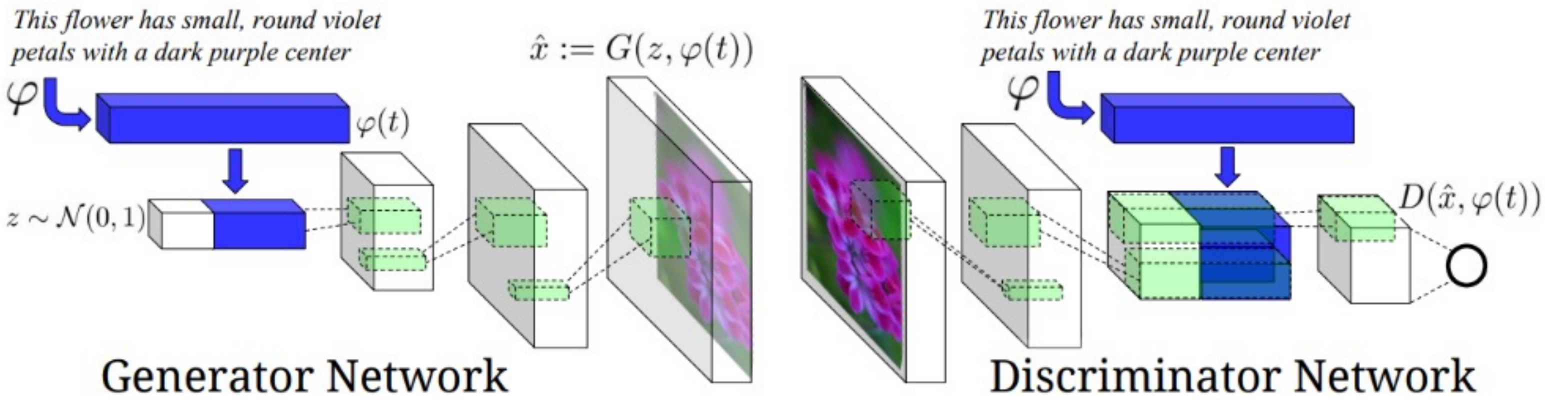
Conditional GANs

- Despite the remarkable effectiveness of GANs in generating synthetic (artificial) instances of one specific phenomenon, they provide a limited control over the specific features of the output.
  - Recall that the input is a random noise vector.
- Class-Conditional GANs (**cGANs**) introduce the label information to the learning architecture, enabling to produce instances of a specific class.
  - The Discriminator reports “1” only for genuine images with correct labels, and “0” for all other cases (genuine images with bad labels, and fake images with any label).



Conditional GANs (Applications)

- "*Text-To-Image Synthesis*": This is the problem of asking to a network, to generate images with specific features:



- "Style Transfer": Transferring style between different kinds of objects:

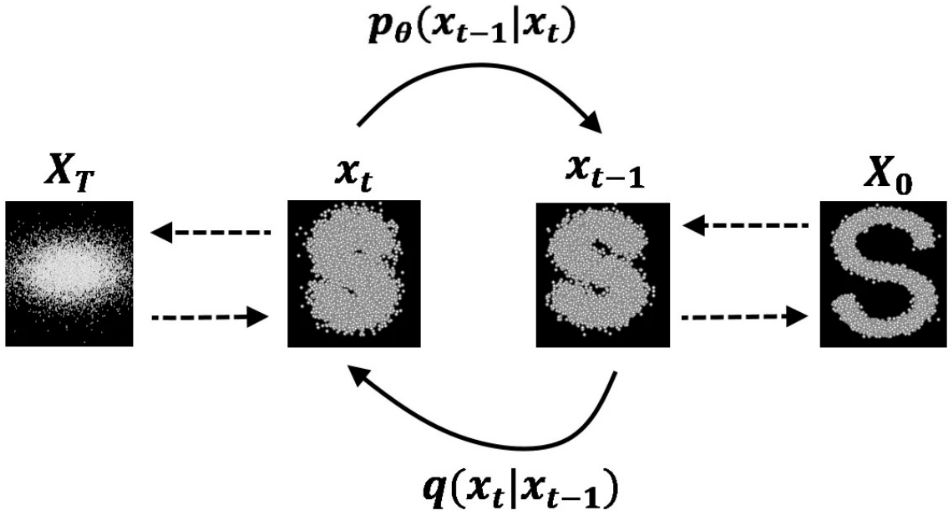
Diffusion Models

- Recently (>2021), Diffusion Models have been advocated as one of the most relevant advances in Machine Learning (Computer Vision) domains.
- As GANs and VAEs, they are a class of machine learning models that can generate new data based on training data.
- Coshesively, the rationale is to **degrade training data by adding noise** and **then learn to recover the data by reversing this noising process**.
  - As a result, this type of models learn to generate coherent images from noise.

Original Image   Some Noise Added   Noising Process Completed

Diffusion Models

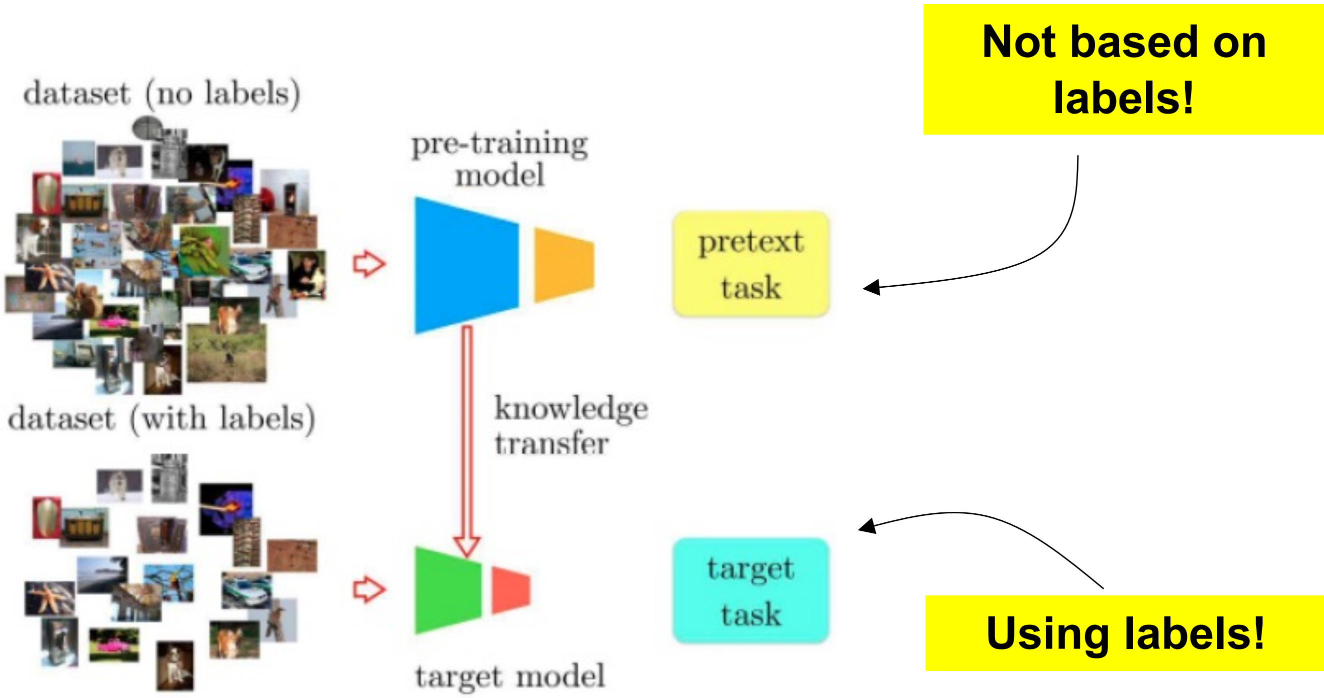
- The most common chain in Diffusion Models is composed of 2 directions:
  - **Reverse Diffusion**, that produces a (more) degraded (noisier) image, given an input image: $\boldsymbol{p_\theta(x_{t-1}|x_t)}$
  - **Forward Diffusion**, that tries to recover a less degraded image, from a noisier version of the data:$\boldsymbol{q_\theta(x_t|x_{t-1})}$



- The key is that if we learn a model that *understands* the systematic decay of information due to noise, then it should be possible to reverse the process and therefore, recover the information back from the noise.

Self-Supervised Learning

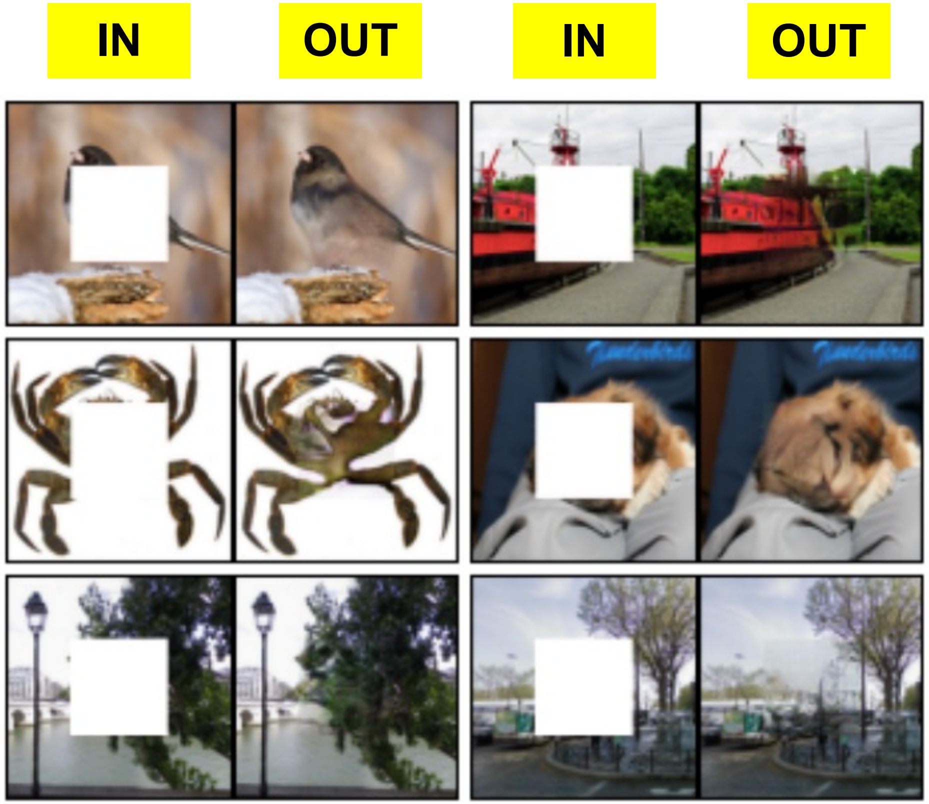
- Self-supervised learning is a relativelly recent type of machine learning that can be regarded as a midle point between supervised and unsupervised learning.
- It is a form of unsupervised learning where the model is trained on unlabeled data, but the goal is to **learn good representations** of the data that can belater used in a downstream supervised learning task.



Source: https://neptune.ai/blog/self-supervised-learning

Self-Supervised Learning

- At first, Self-supervised learning starts by training a model itself to learn one part of the input from another part of the input.
- This is known as **pretext learning**, which can assume different forms:
- For example, using unstructured 2D data, predict any part of the input from any other part:



By doing this, we force the model to “understand” the data

Self-Supervised Learning

- Still for unstructured 2D data, another very popular pretext task is to learn by solving Jigsaw puzzles:

IN

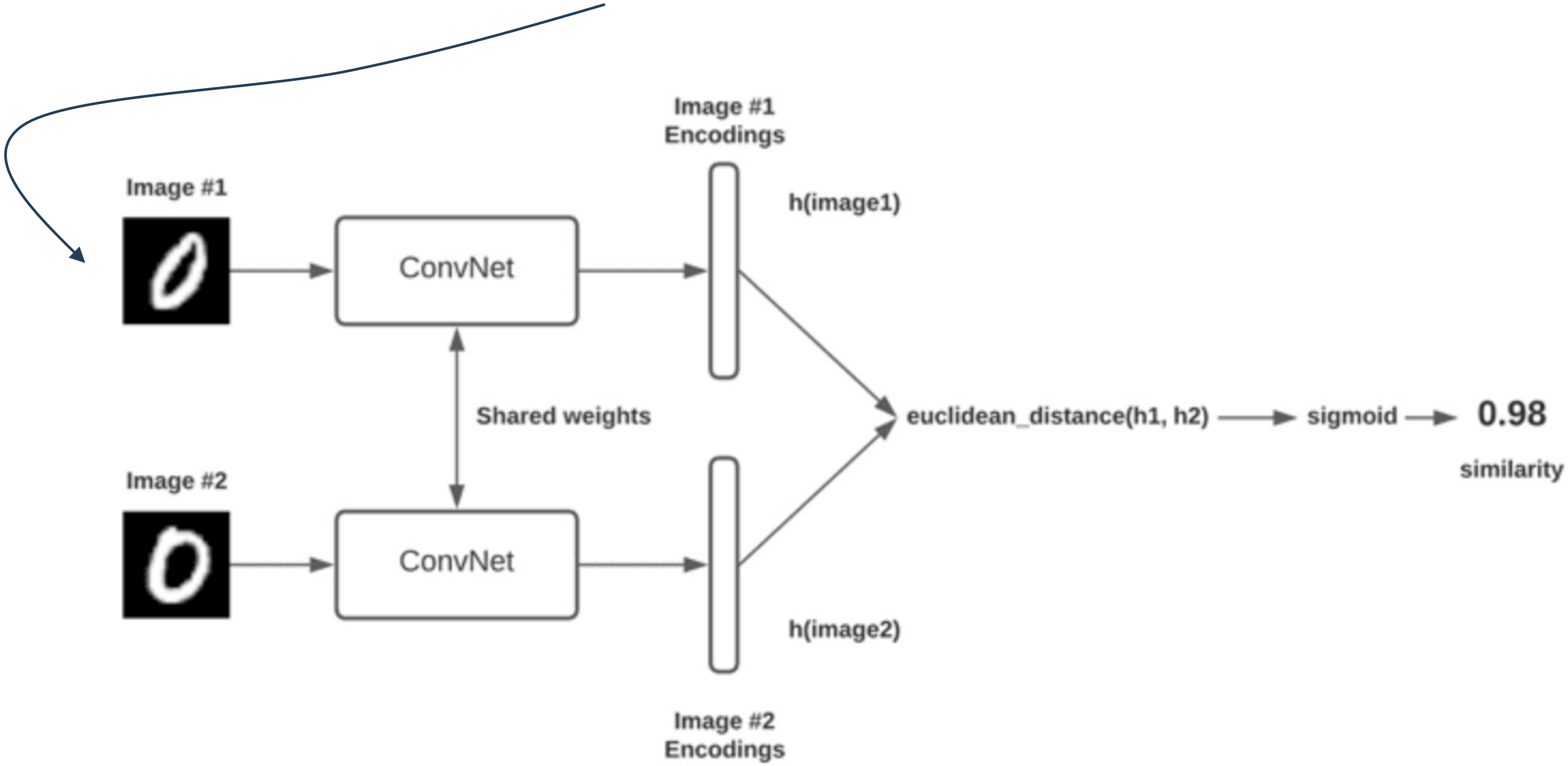
OUT

Again, the model is forced to understand each part of the input, in order to obtain a realistic output

Self-Supervised Learning

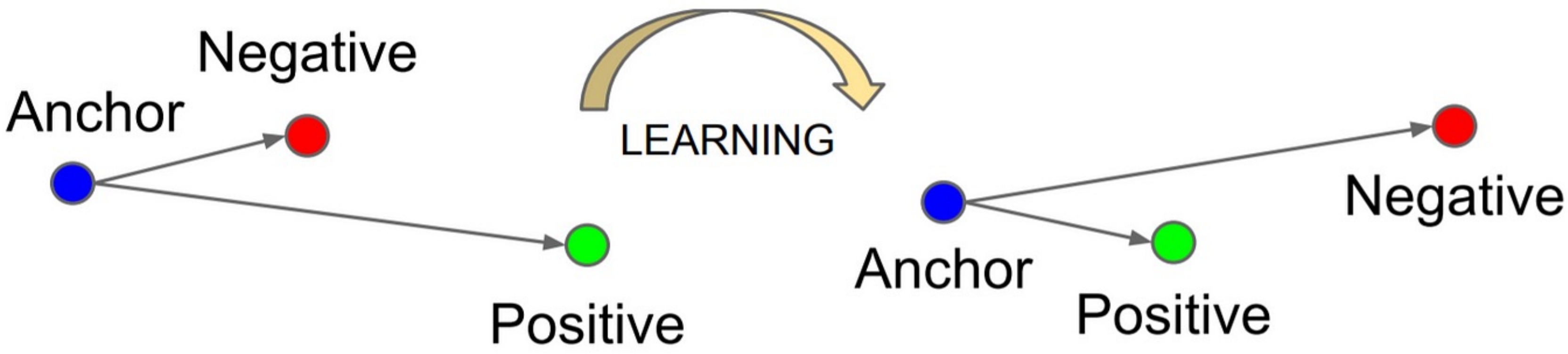
- It is also very common to use some Siamese architecture to obtain appropriate feature representations.

If both inputs are from the same image (not “class” in this case), the distance should be small. Otherwise, it should be large.



Self-Supervised Learning

- Another possibility is to use three images in the input: the Anchor (**A**) and the Positive (**P**) that are variations of the same image, and the negative (**N**), that regards a different image.



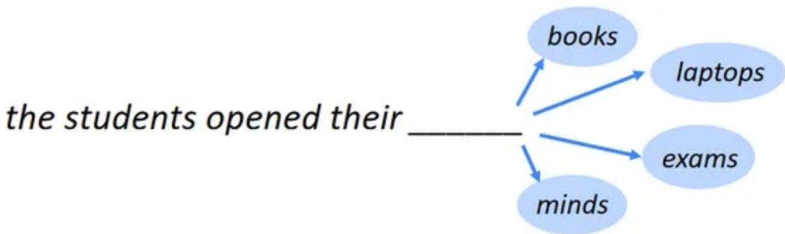
The Anchor and Positive should be near each other, while their distance to the Negative image should be large

$$\mathcal{L}(A, P, N) = \max(\| \mathrm{f}(A) - \mathrm{f}(P)\|_2 - \| \mathrm{f}(A) - \mathrm{f}(N)\|_2 + \alpha, 0)$$

Self-Supervised Learning

- In case of 3D unstructured data (video), one can predict the predict the future from the past/present, or predict the present from the future.

- In case of text data, the most obvious pretext task is to predict the next word, based in the last "k" words.



Self-Supervised Learning

- Once the pretext task is considered solved (i.e., the model stopped to learn), it is time to apply “Transfer Learning” techniques
- In practice, it consists in copying (and freezing ?) the weights from the earliest layers of the model into the new one.