# Artificial intelligence

Unsupervised Deep Learning

- Autoencoders
- Generative Adversarial Networks (GAN)

Carl McBride Ellis

carl.mcbride@u-tad.com

# Recommended reading

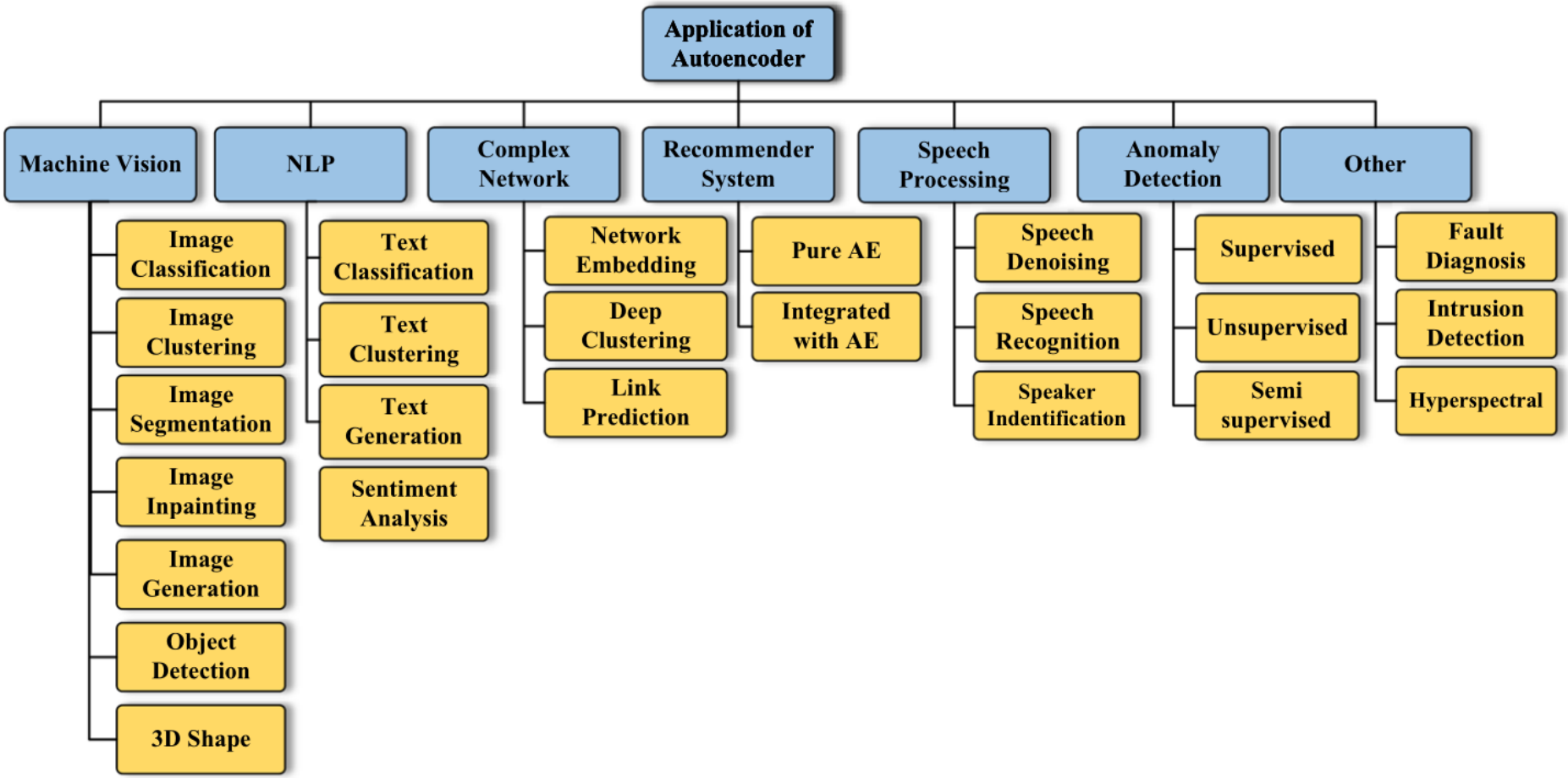
- Aurélien Géron "*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*", 3rd Edition O'Reilly Media (2022)
  – chapter 17
- Autoencoders and their applications in machine learning: a survey (2024)

Unsupervised learning is when the training data has no labels

Autoencoders are actually *self-supervised*:
an autoencoder learns to (more or less) replicate the target

Potential uses:

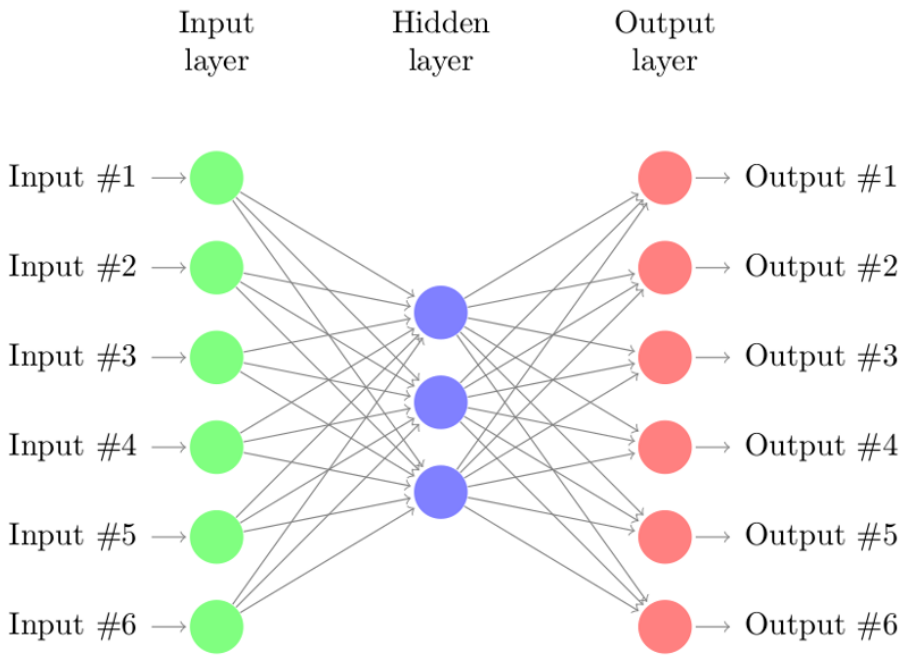
- reducing data dimensionality
- extract meaningful image features
- noise reduction: eliminating unwanted artifacts from images
- image inpainting: reconstructing missing or corrupted parts of an image.
- learn text embeddings
- anomaly detection




(Source: "Autoencoders and their applications in machine learning: a survey" (2024))

- Undercomplete autoencoders:
  - Deep (or stacked) autoencoder
  - Convolutional autoencoder
- Overcomplete autoencoders:
  - Denoising autoencoder
  - Variational autoencoders (VAE)

Undercomplete autoencoder
the middle layer is smaller than the input space



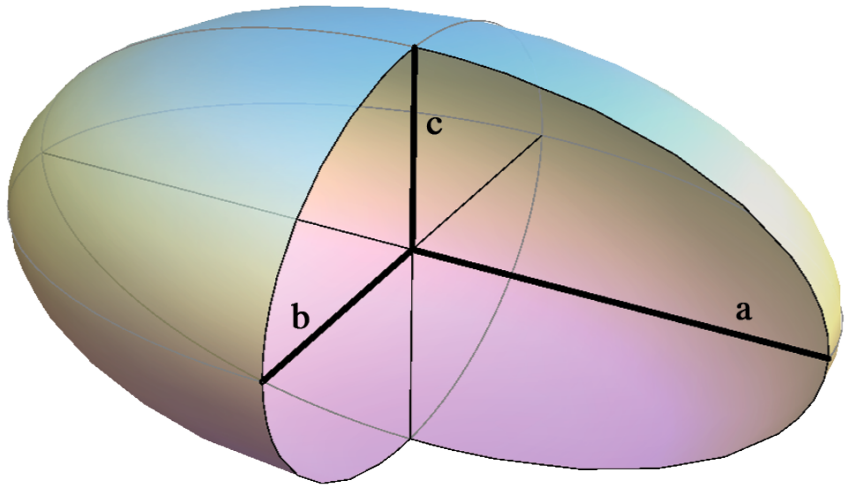
# Principal component analysis (PCA)

From Géron (p. 639)

## Performing PCA with an Undercomplete Linear Autoencoder

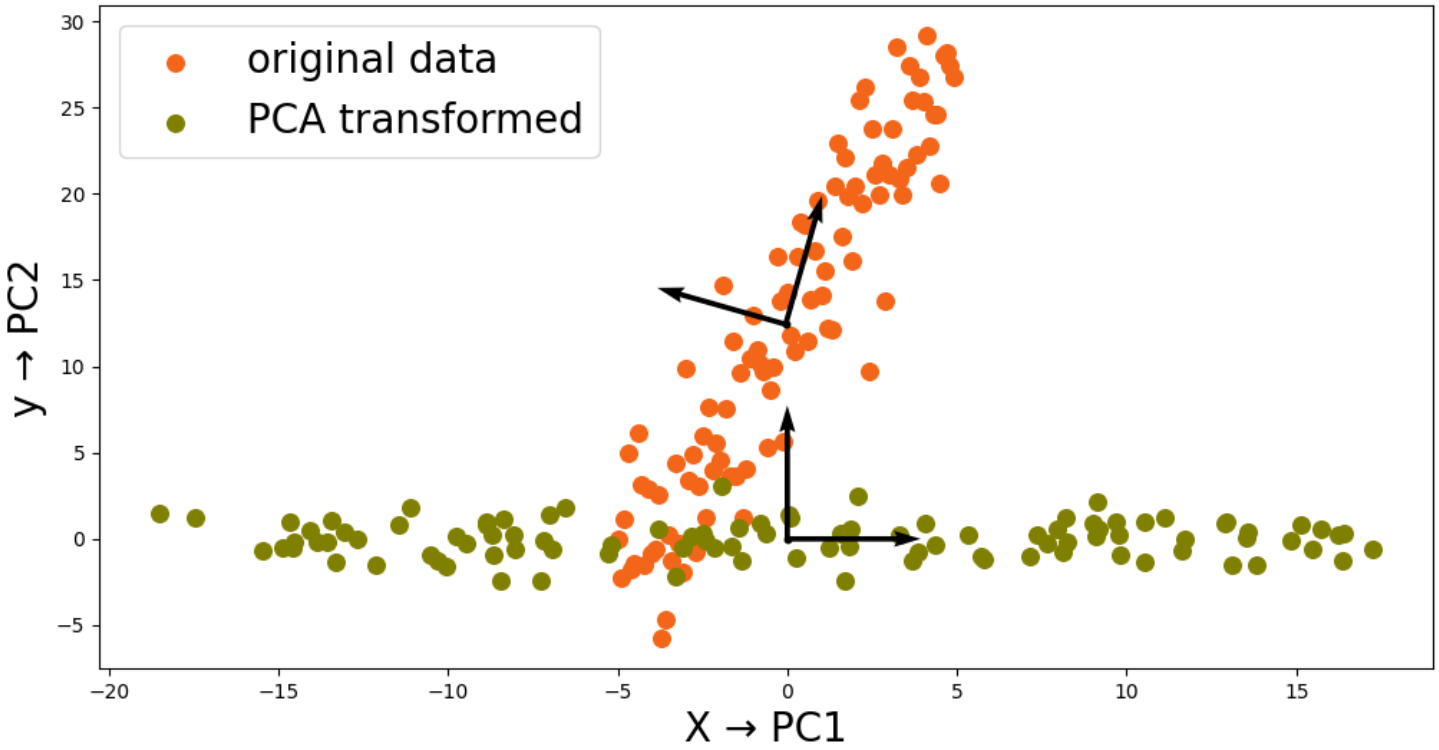
If the autoencoder uses only linear activations and the cost function is the mean squared error (MSE), then it ends up performing principal component analysis

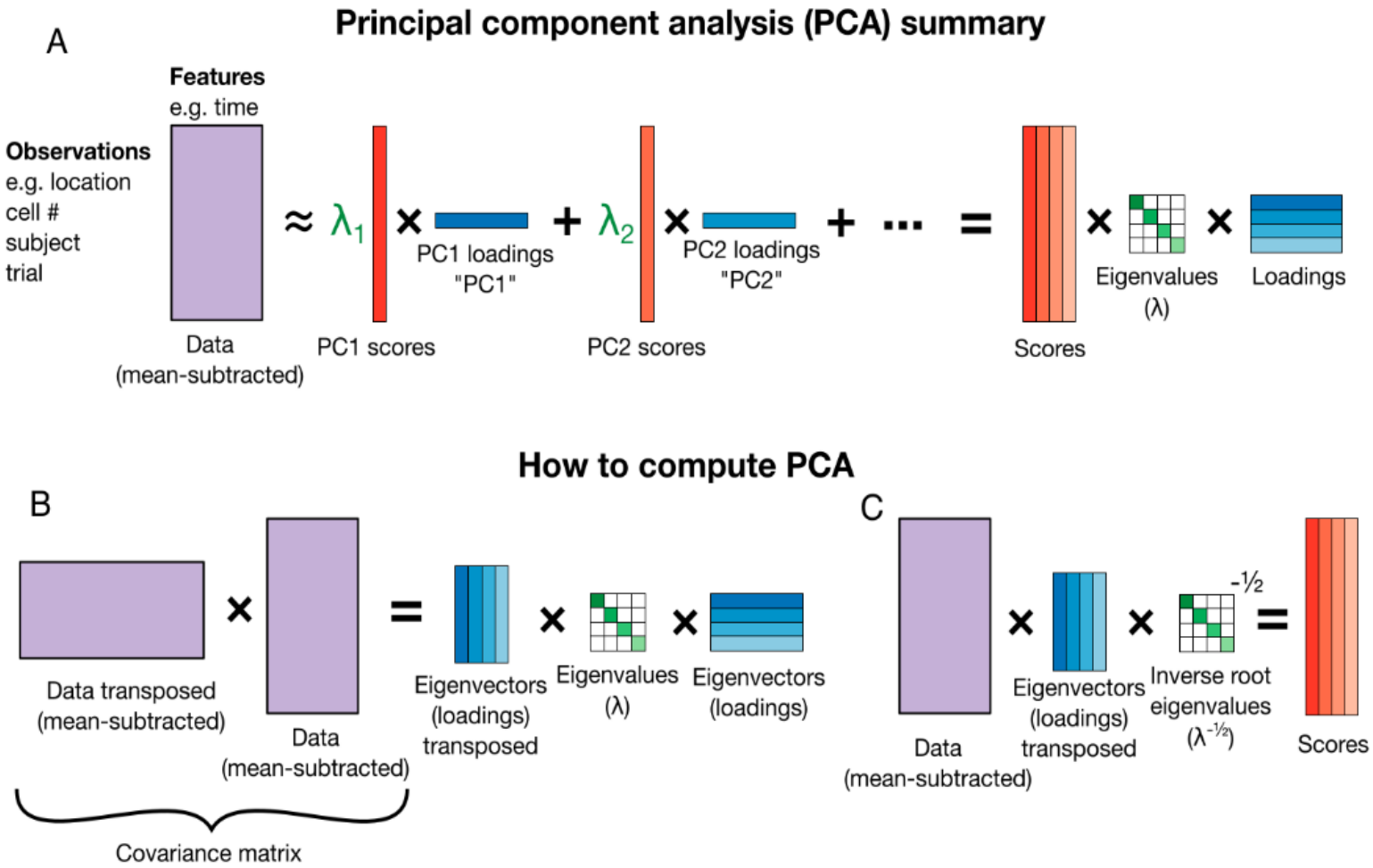
PCA is an old data science technique, invented in 1901 by Karl Pearson.




In mechanics one finds the principal moments of inertia from the eigenvalues and eigenvectors of the inertia tensor

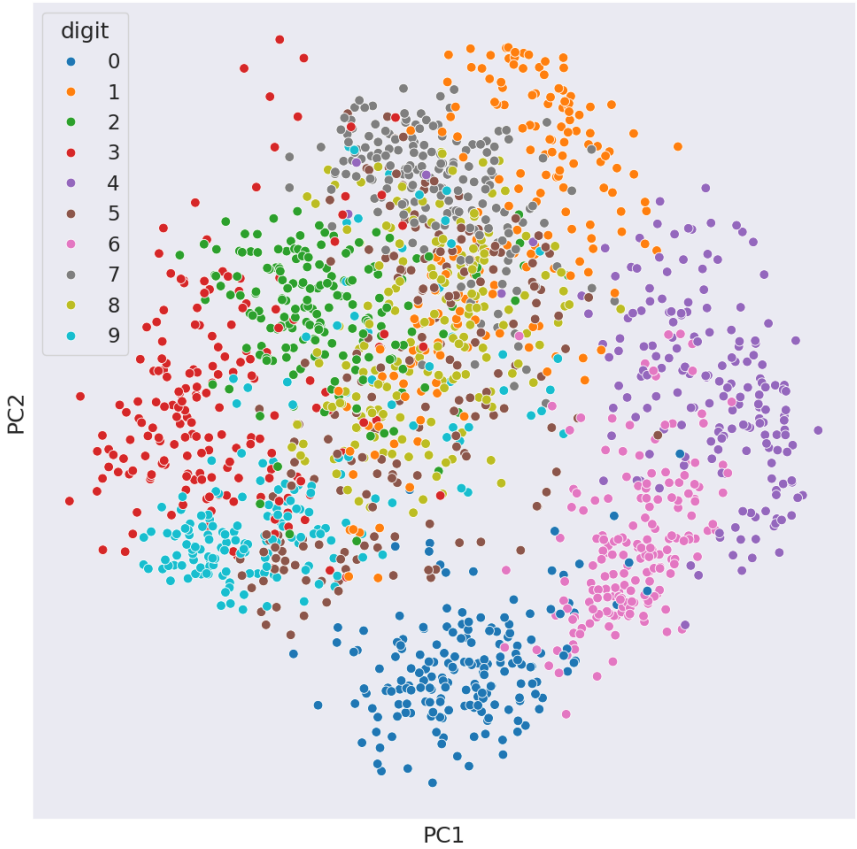
We can find the principal components of a dataset from the eigendecomposition of the covariance matrix





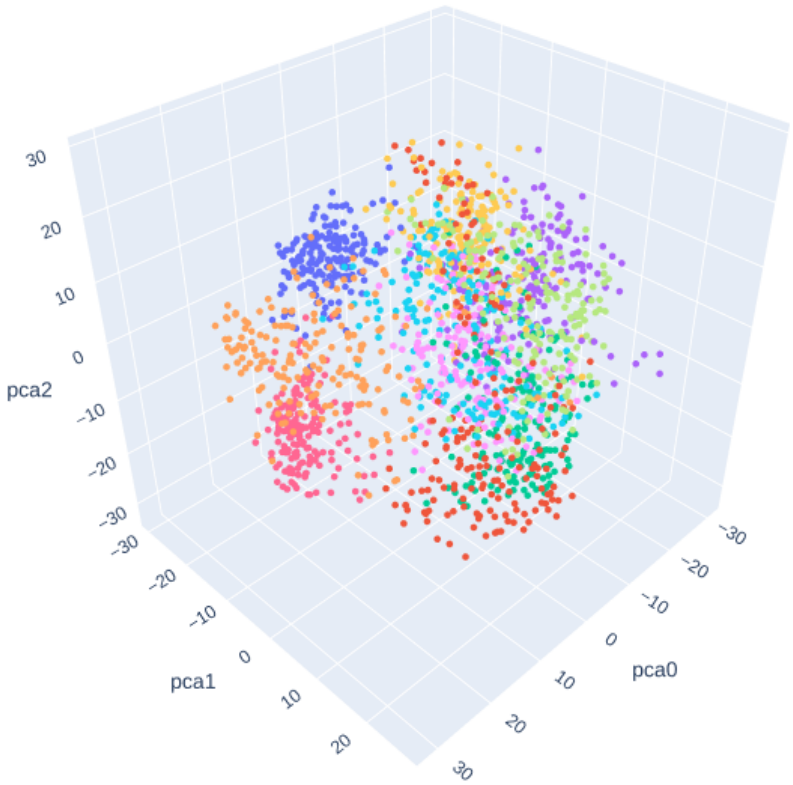

(Image source: "Phantom oscillations in principal component analysis")

PCA is often (mostly?) used to project multidimensional data in 2D




See: notebook_PCA_MNIST.ipynb

We can also take the three largest PCA components and make an interactive 3D plot using the Plotly library



Disadvantages of using an autoencoder: it basically doesn't work very well

- the weights of the encoder and decoder are generally not the same
- the PCA features are often still correlated

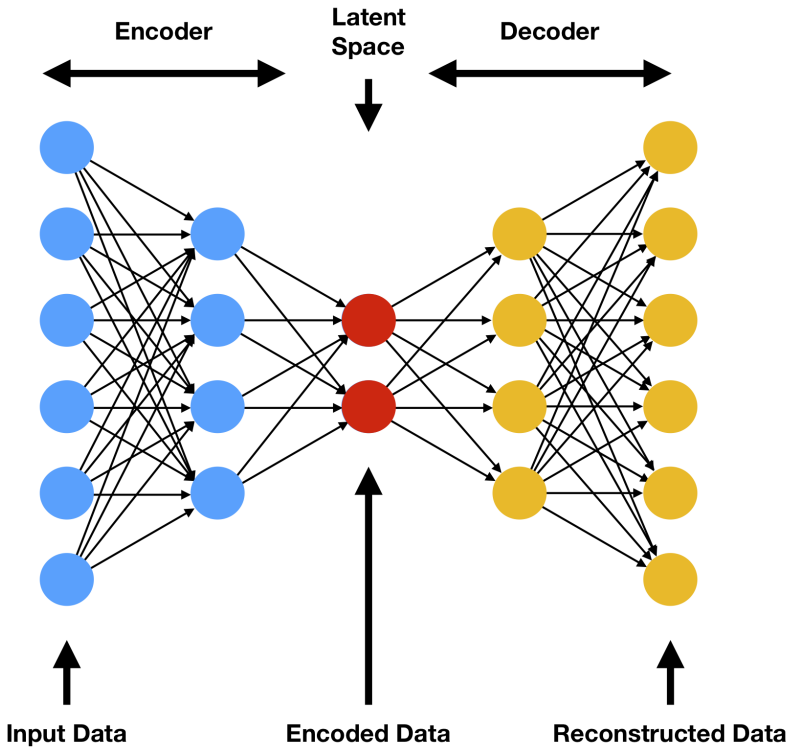
Advantage of using an autoencoder

- with a non-linear activation function we can perform non-linear-PCA (kernel PCA)

Let us take a look at

notebook_PCA_autoencoder.ipynb

# Deep (or stacked) autoencoder




(paper: "*Reducing the Dimensionality of Data with Neural Networks*" Hinton and Salakhutdinov (2006))

# Convolutional autoencoder

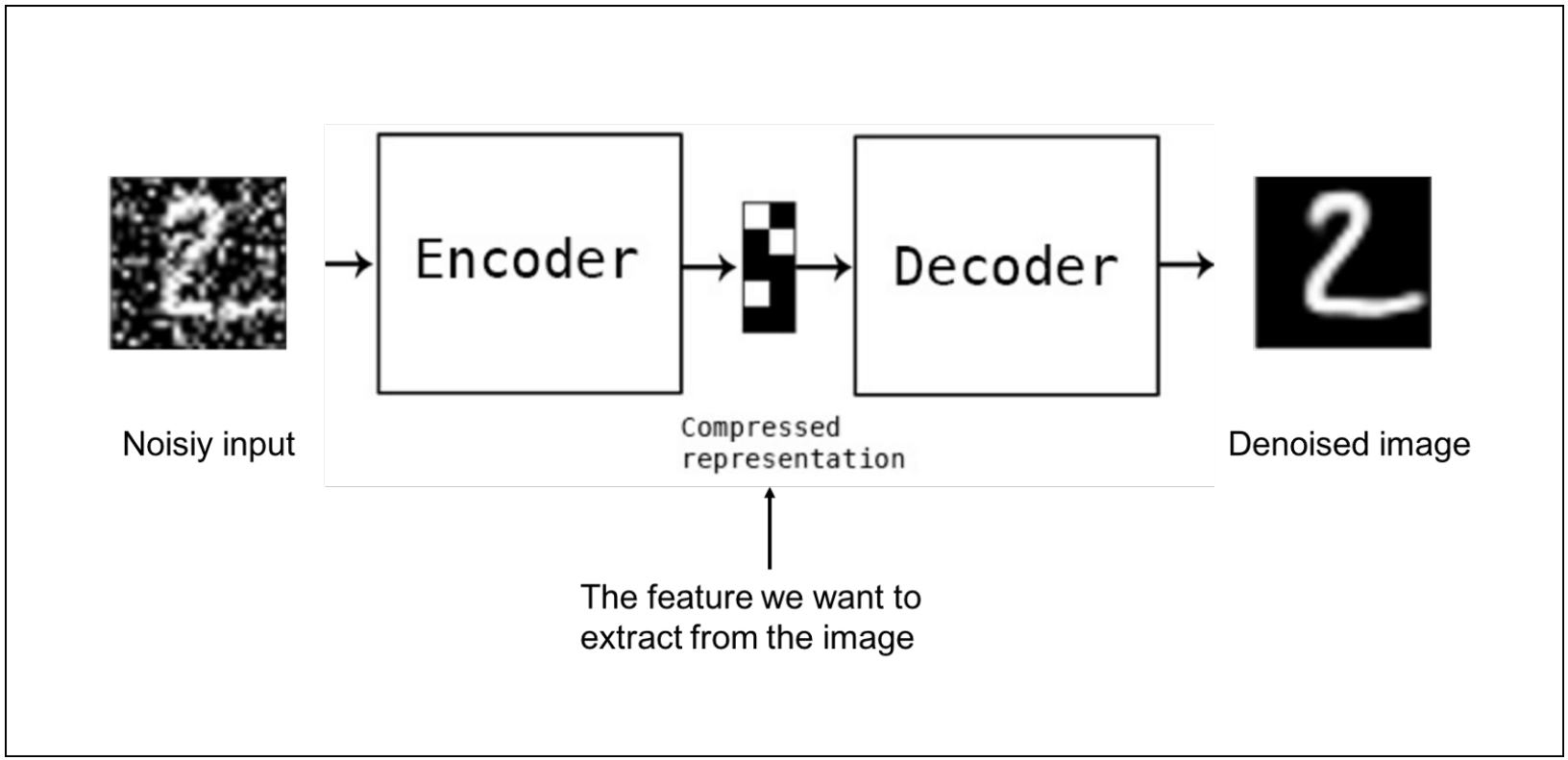
Convolutional autoencoders are used for image data composed of convolutional layers and pooling layers

```
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = keras.Model(input_img, decoded)
```

Notice the new `UpSampling2D` layer in the decoder.

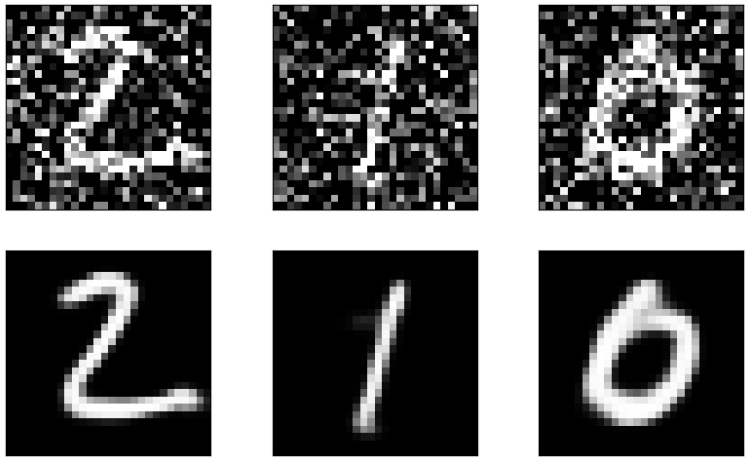
# Denoising autoencoder

We provide the autoencoder with noisy examples of the training data and it has to learn how to go from the noisy data (image) to the clean data (image)



# Practical session

Denoising MNIST/fashionMNIST



Notebook (hosted on Kaggle)

notebook_GAN_denoising_MNIST

also try to denoise the fashion-MNIST data

# Variational autoencoders (VAE)

This is an example of a generative technique

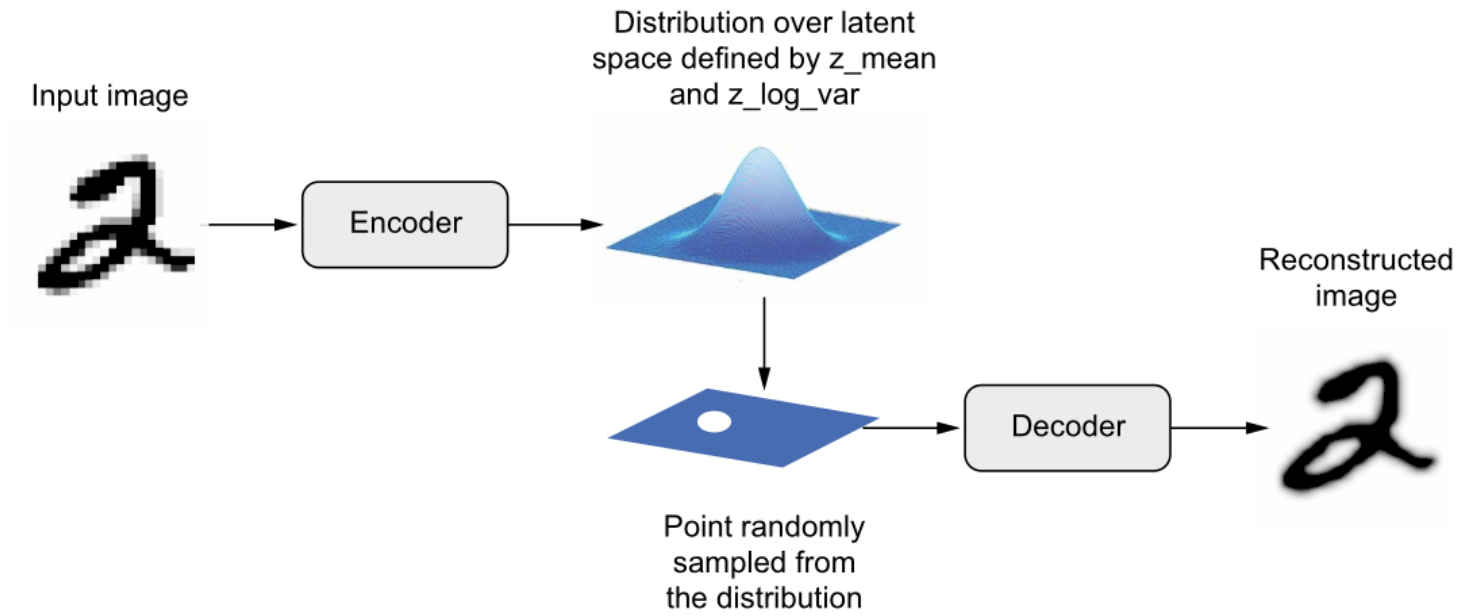



**Figure 12.17 A VAE maps an image to two vectors, `z_mean` and `z_log_sigma`, which define a probability distribution over the latent space, used to sample a latent point to decode.**

paper "Auto-Encoding Variational Bayes" (2013)

In a standard autoencoder we learn to encode the training data pixel by pixel.

In a VAR we also learn to encode the *distribution* of the training data using a loss function known as the *latent loss*:

$$\mathcal{L} = -\frac{1}{2}\sum_{i=1}^{N}\left(1 + \log(\sigma_i^2) - \sigma_i^2 - \mu_i^2\right)$$

Learning how the training data is distributed now permits us to now produce new data having a similar distribution by sampling from the learned distribution.

Let us take a look at the notebook (hosted on Kaggle)

`notebook_GAN_VAE_faces`

(Note: the notebook took $4\frac{1}{2}$ hours to run)

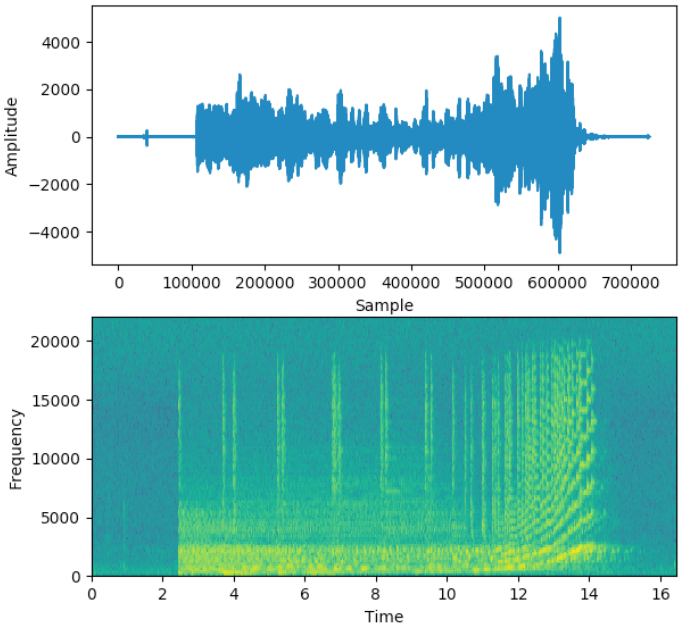
## Autoencoder use case: anomaly detection

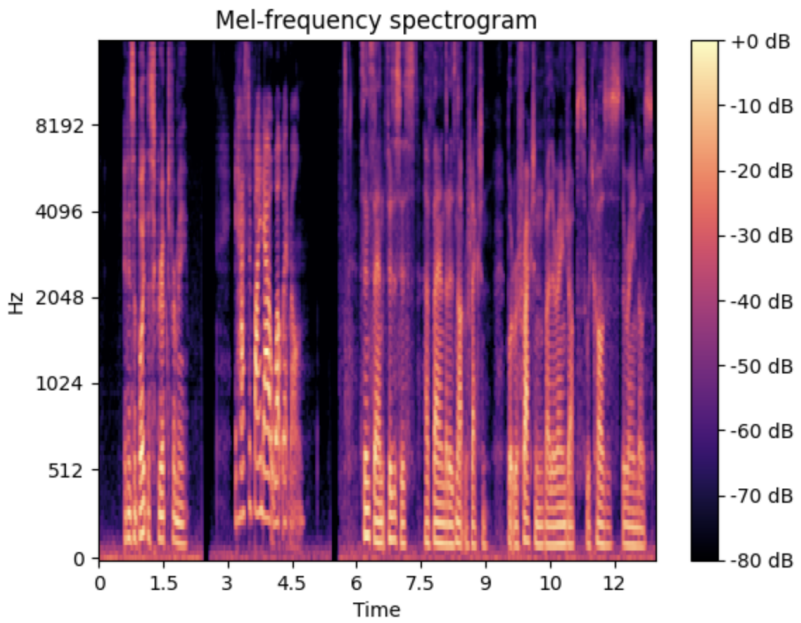
In unsupervised tasks, the idea is to train AEs on only sample data of one class (majority class). This way the network is capable of re-constructing the input with good or less reconstruction loss. Now, if a sample data of another target class is passed through the AE network, it results in comparatively larger reconstruction loss, a threshold value of reconstruction loss (anomaly score) can be decided, larger than that can be considered an anomaly

## Use case: cleaning speech

In the denoising AE framework, the model is trained using noisy speech samples, with the noisy speech serving as the input and the corresponding clean speech as the target. Through this training, the AE becomes adept at reconstructing noise-free speech from noisy inputs, enabling it to effectively denoise unseen speech signals.

Trick: one can turn audio data into an image using a mel sprectogram, and then use a CNN

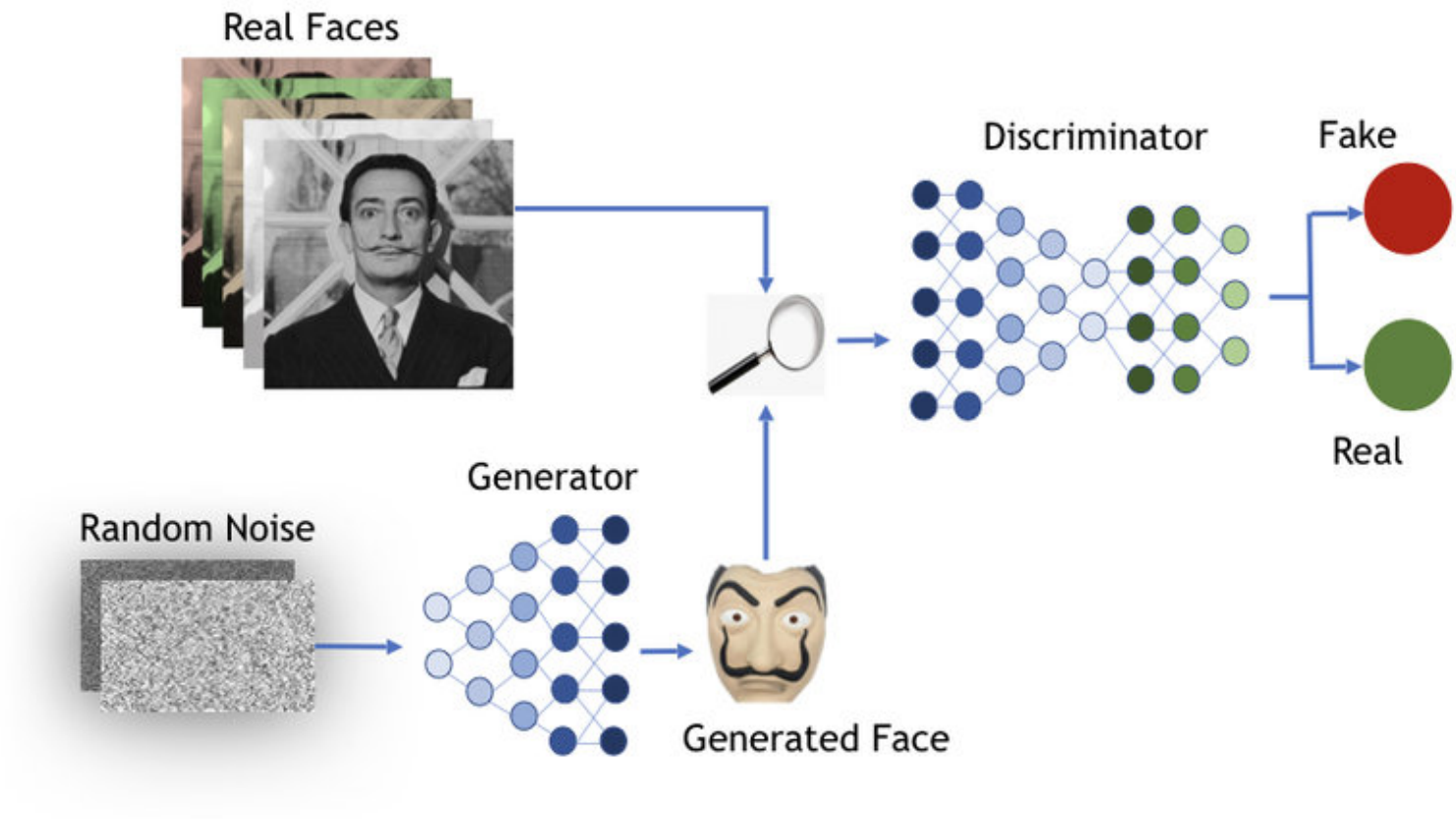


# Generative Adversarial Networks (GAN)

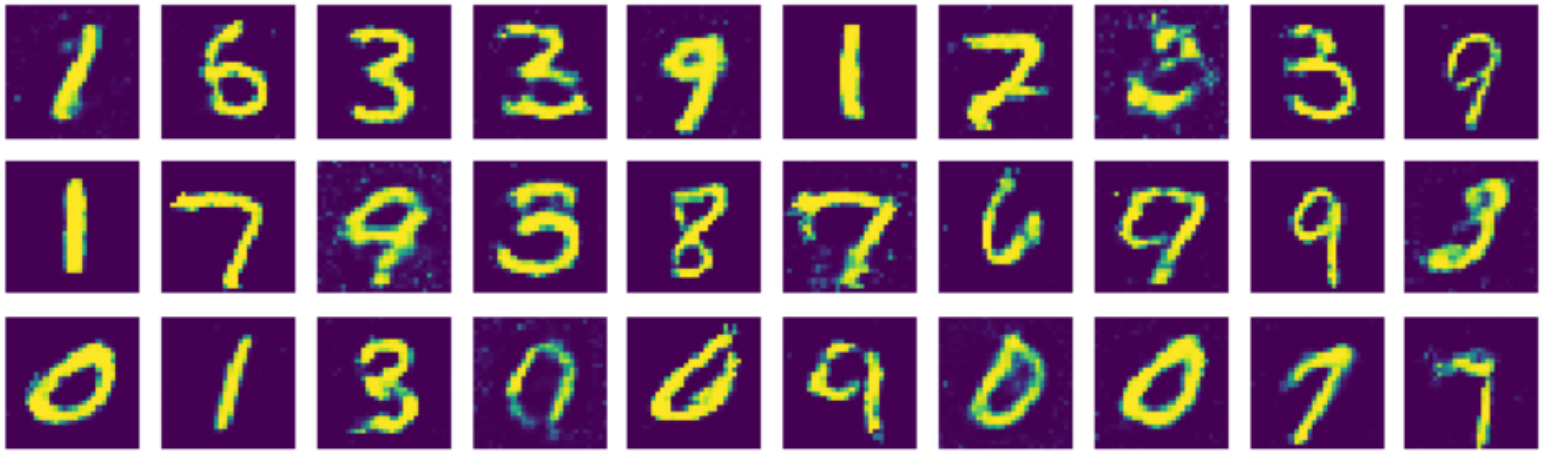
*We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model $G$ that captures the data distribution, and a discriminative model $D$ that estimates the probability that a sample came from the training data rather than $G$. The training procedure for $G$ is to maximize the probability of $D$ making a mistake. This framework corresponds to a minimax two-player game.*

Source: Generative Adversarial Nets (2014)




The generator ($G$) is akin to a (variational) decoder
The discriminator ($D$) is akin to a binary classifier

Using a GAN to make new MNIST images

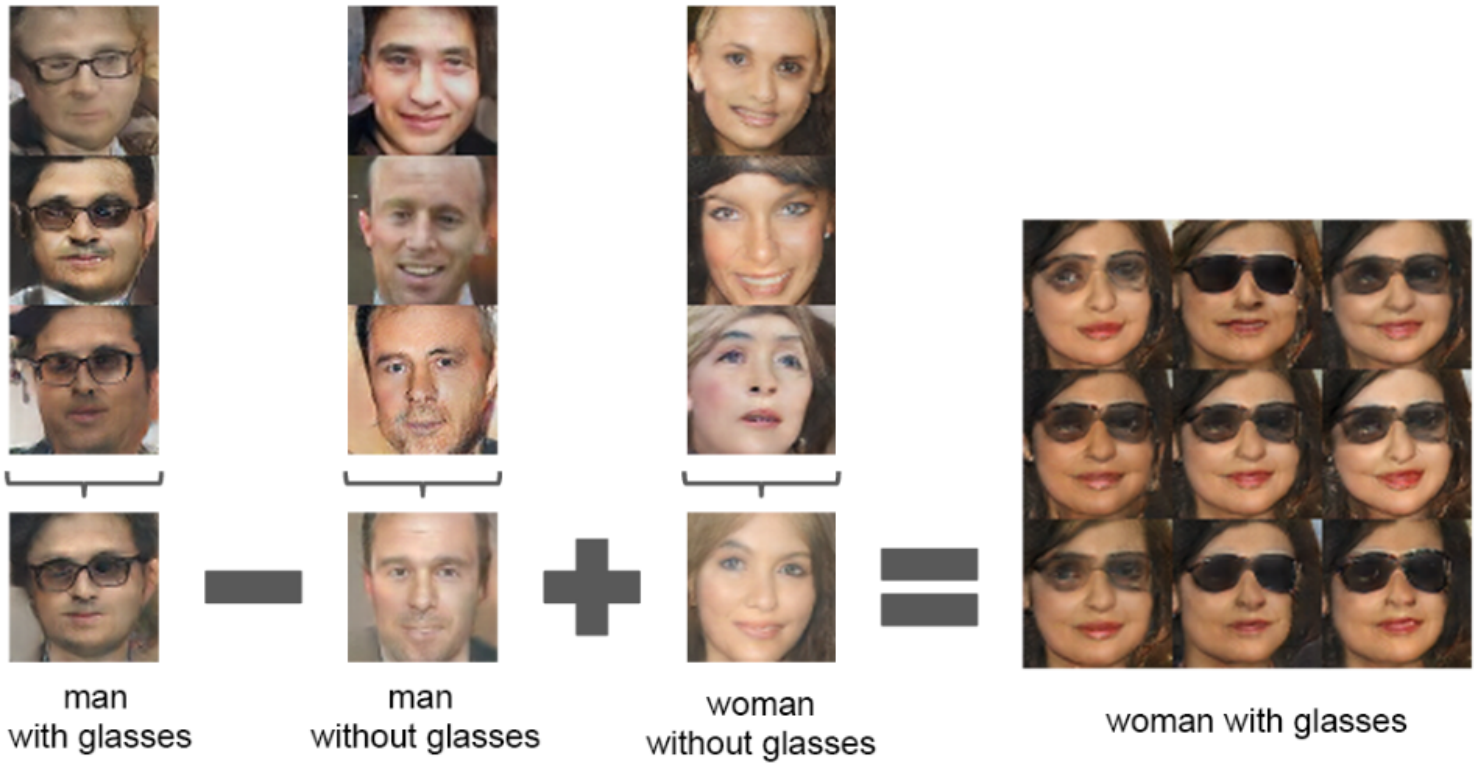


Notebook (hosted on Kaggle)

notebook_GAN_MNIST

Adapt the notebook to use the Fashion MNIST dataset.

# Deep convolutional GAN




(Source: "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks" (2015))

# StyleGAN


(Source: "A Style-Based Generator Architecture for Generative Adversarial Networks" (2018))

# Practical session

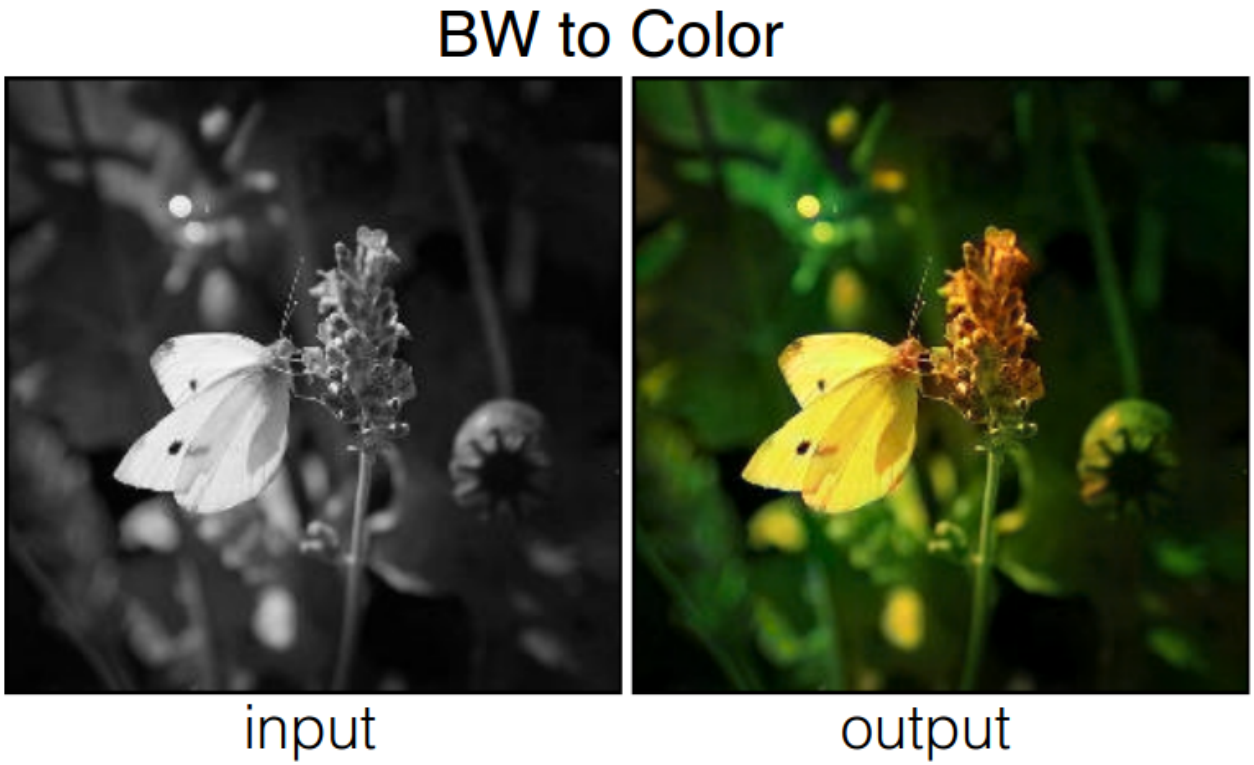
We shall play with the StyleGAN2-ADA package by NVIDIA Research Projects Notebook (hosted on Kaggle)

`notebook_GAN_StyleGAN2`

we shall generate images with different 'truncations' (`trunc`).

(here truncation is analogous to the `temperature` parameter we saw with the softmax function in RNN)

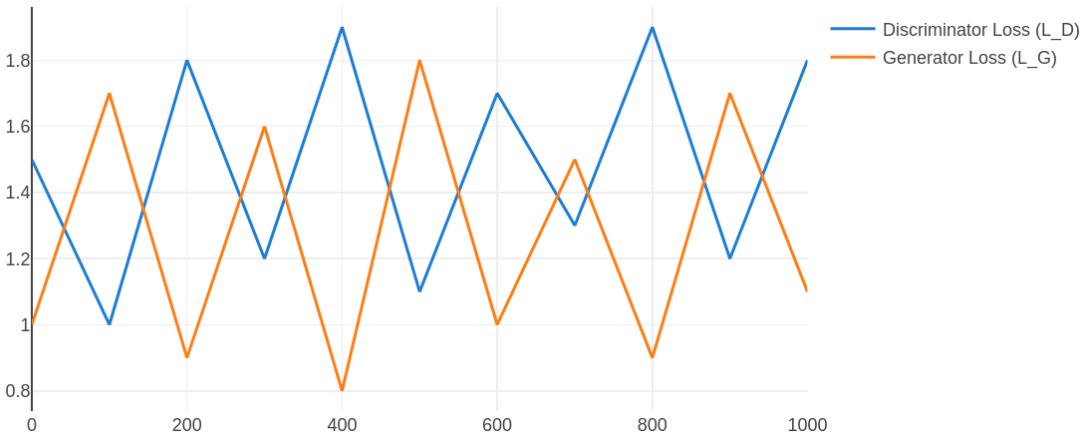
# Image colorization: `pix2pix`




(Source: "Image-to-Image Translation with Conditional Adversarial Networks" (2016) GitHub )

GAN problem: mode 'collapse'

- the generator discovers some image(s) that always fools the discriminator and then only produces these (few) images (this will especally happen if the discriminator learns slowly)
- Oscillations: quickly flipping between two images



# GenAI problem: model 'collapse'

**Abstract**

Trained on massive amounts of human-generated content, AI-generated image synthesis is capable of reproducing semantically coherent images that match the visual appearance of its training data. We show that when retrained on even small amounts of their own creation, these generative-AI models produce highly distorted images. We also show that this distortion extends beyond the text prompts used in retraining, and that once poisoned, the models struggle to fully heal even after retraining on only real images.

(Source: "Nepotistically Trained Generative-AI Models Collapse" (2023))

also for text data


(Source: Futurism - June 2025)

*LLM Brain Rot Hypothesis: continual pre-training on junk web text induces lasting cognitive decline in LLMs.*

(Source: LLMS CAN GET "BRAIN ROT"! - October 2025)

# More Articles Are Now Created by AI Than Humans