

12 | Image Generation with DNN

Giordano De Marzo

<https://giordano-demarzo.github.io/>

Deep Learning for the Social Sciences

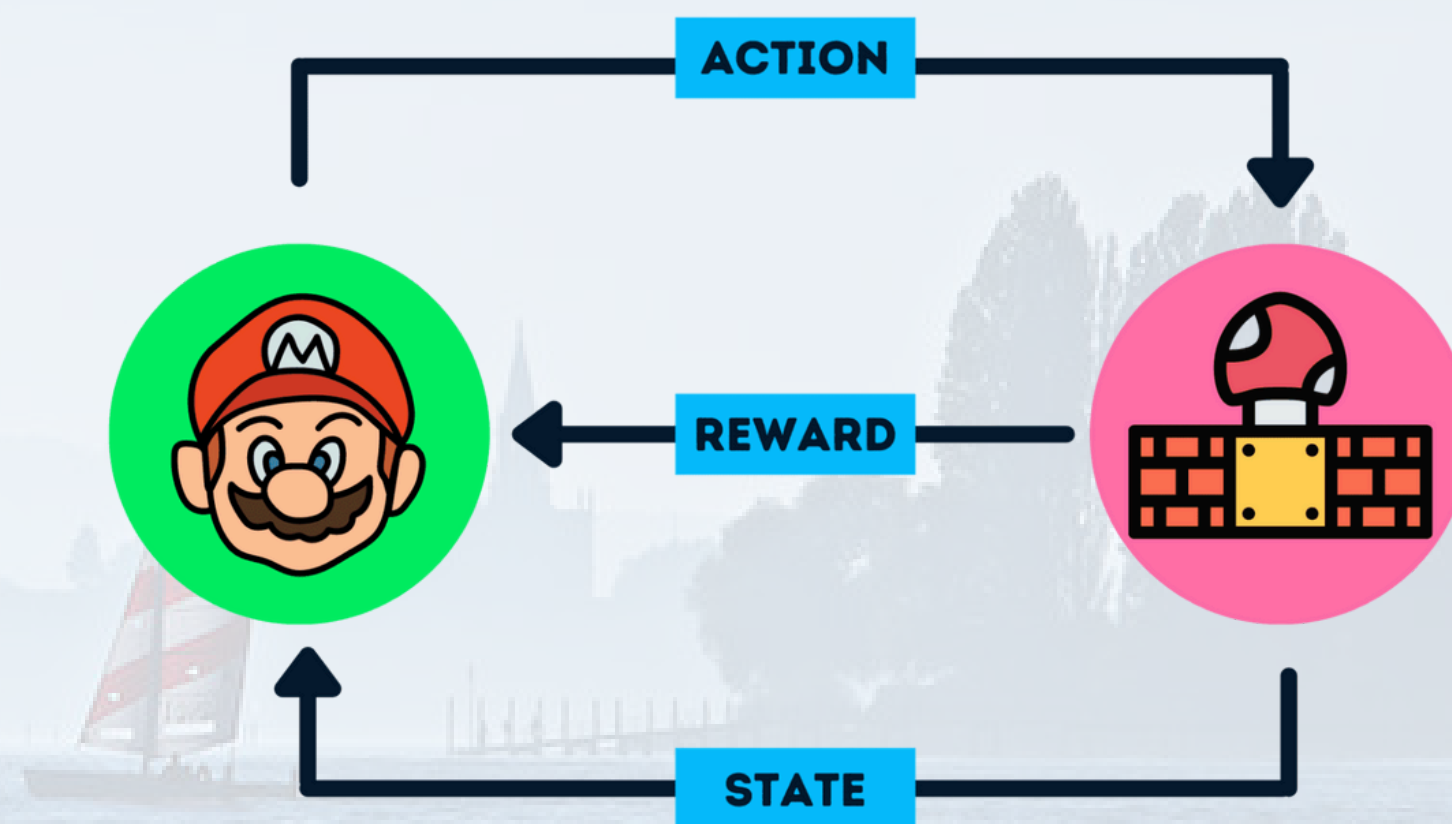


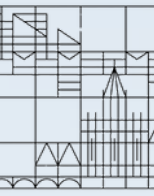
Reinforcement Learning Framework

The RL framework is structured around a set of key elements:

- **State:** The representation of the environment at any given time.
- **Action:** A set of possible moves the agent can make.
- **Reward:** Immediate feedback given to the agent to evaluate its last action.
- **Policy:** A strategy that the agent follows to determine its action at each state.

The goal of RL is to follow a policy or strategy that maximizes the reward





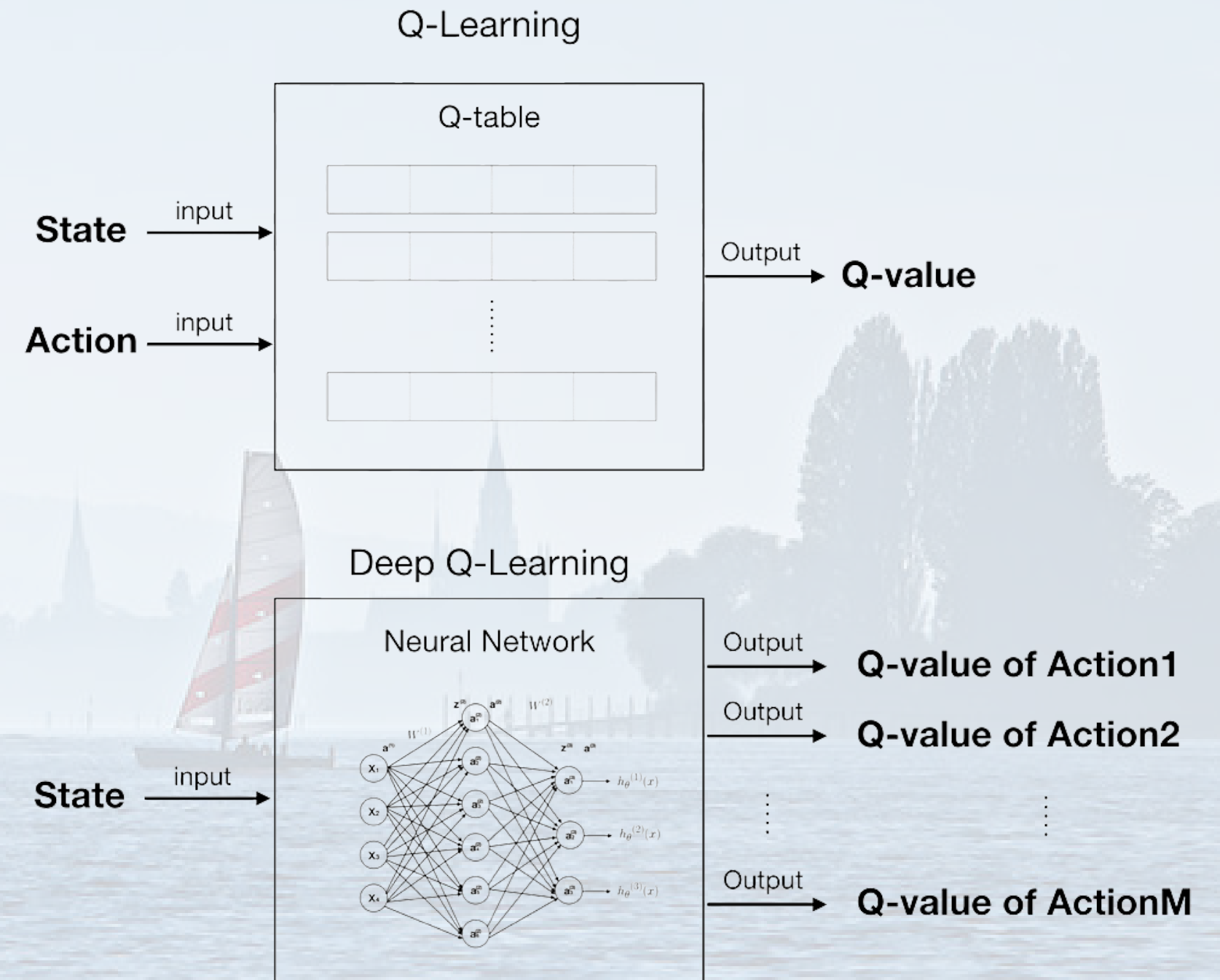
Deep Q-Learning Networks

Traditional Q-Learning has many limitations

- It relies on a Q-table, which becomes impractical in environments with vast numbers of state-action pairs.
- In high-dimensional state spaces it requires exponential data to adequately cover the state-action space.

Deep Q-Networks (DQN) utilize a neural network to approximate the Q-value function

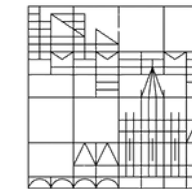
$$Q(s, a; \theta) \approx r + \gamma \max_{a'} Q(s', a'; \theta)$$



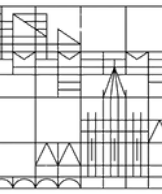


Outline

1. Generative Deep Learning
2. Variational Autoencoders
3. Generative Adversarial Networks
4. Diffusion Models



Generative Deep Learning

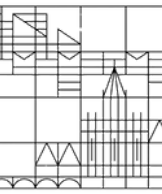


Generative Deep Learning

Generative deep learning focuses on creating models that can produce new, realistic data samples similar to their training data.

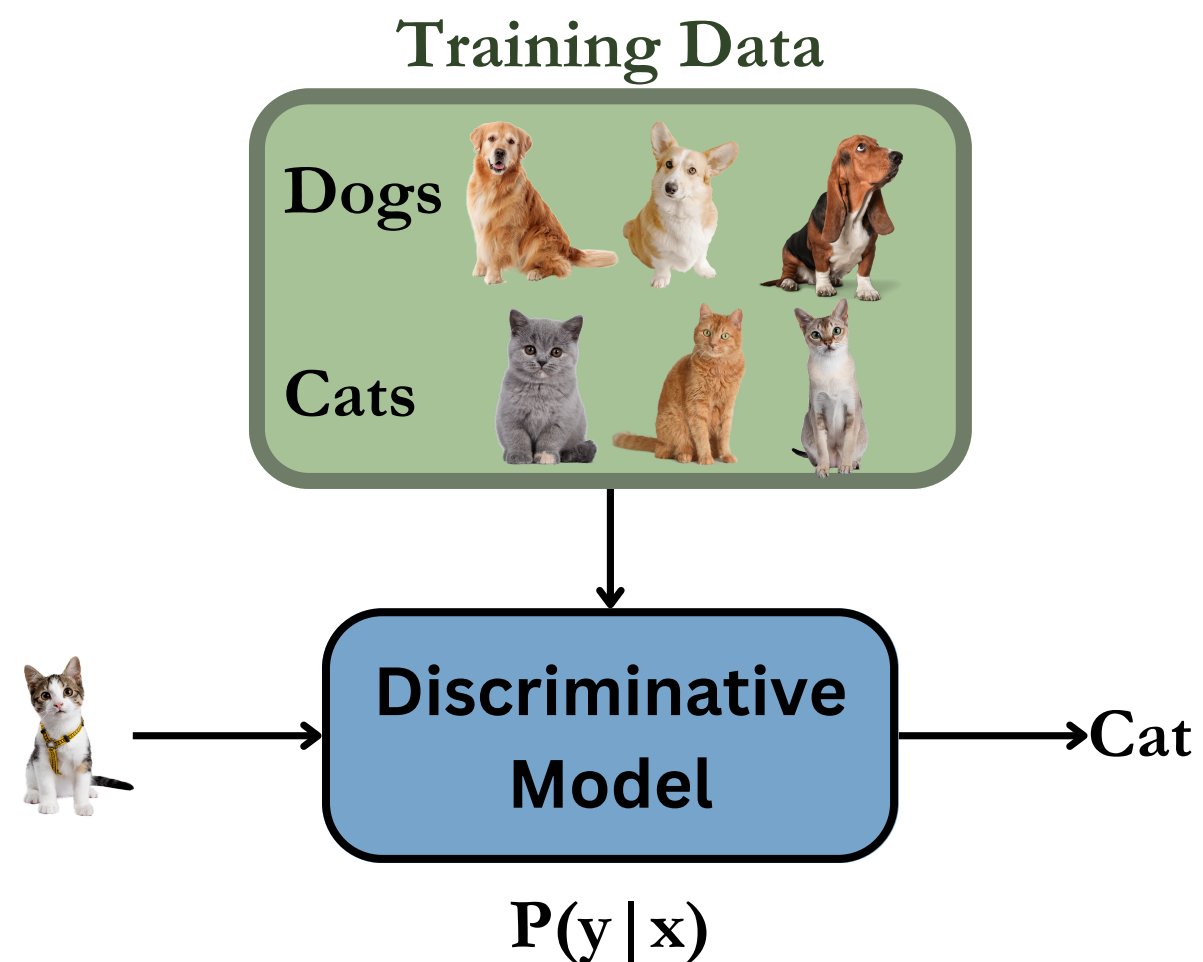
- **Core objective:** Learn to generate new examples that could plausibly come from the same distribution
- **Applications:** Text generation, image synthesis, music composition, code generation
 - Large Language Models are generative models specialized for text
- **Key insight:** Instead of just recognizing patterns, models learn to create new patterns
- **Examples:** GPT generates coherent text, DALL-E creates images, Codex writes code

Fundamentally, we train models to learn the probability distribution $P(x)$ where x represents the data (text, images ...)

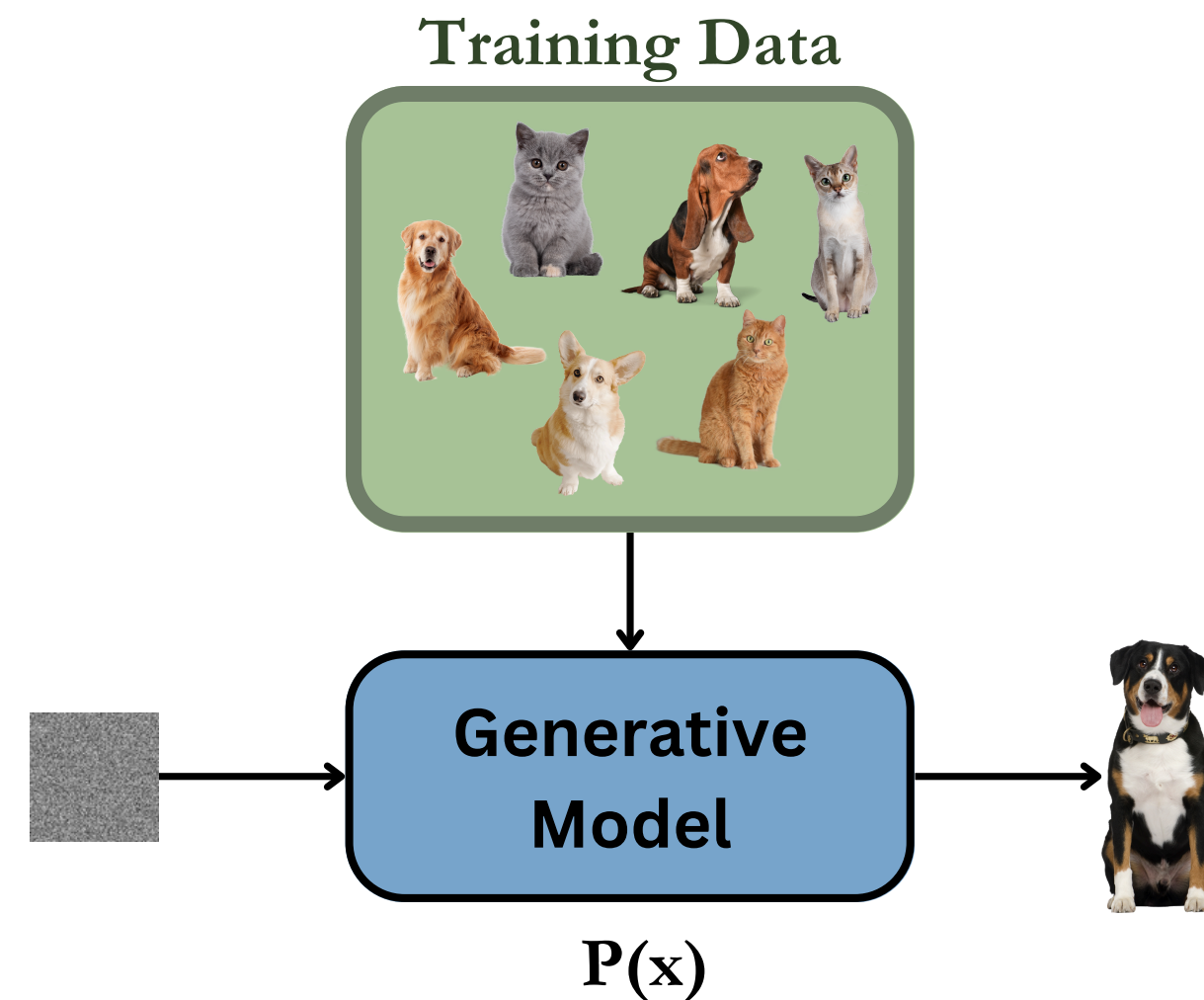


Discriminative vs Generative

Discriminative models assign labels to data: they reconstruct the probability $P(y | x)$ of the label (y) given the data (x)



Generative models create new data: they reconstruct the probability $P(x)$ of the data in our sample



Conditional Generative Models

The information about the label can be included in generative models getting the so called conditional generative models:

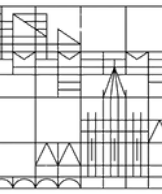
- they generate data given labels
- they reconstruct the conditional probability $P(x | y)$

The 3 classes of problems are related by Bayes' theorem

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

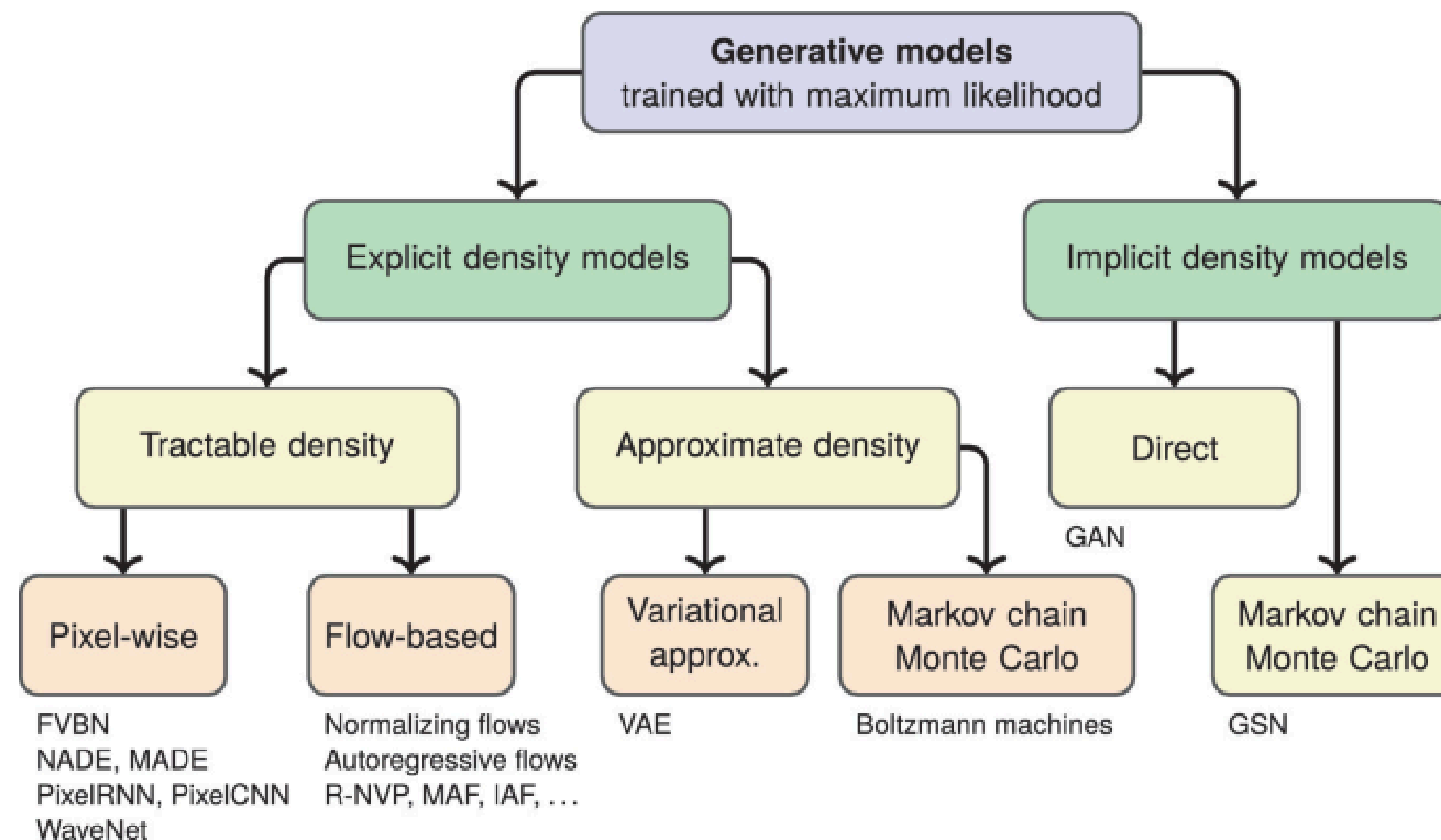


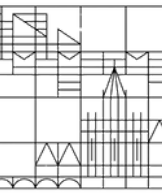
A photo-realistic picture of a sailboat on a calm lake at sunset



Taxonomy of Generative Deep Learning

There is a jungle of generative deep learning models depending on if and how they reconstruct the data probability $P(x)$. We will focus on VAEs, GANs and Diffusion Models.

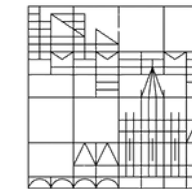




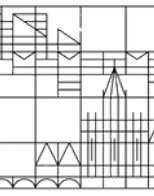
Strengths of Generative Deep Learning

Generative deep learning is an incredibly powerful technique

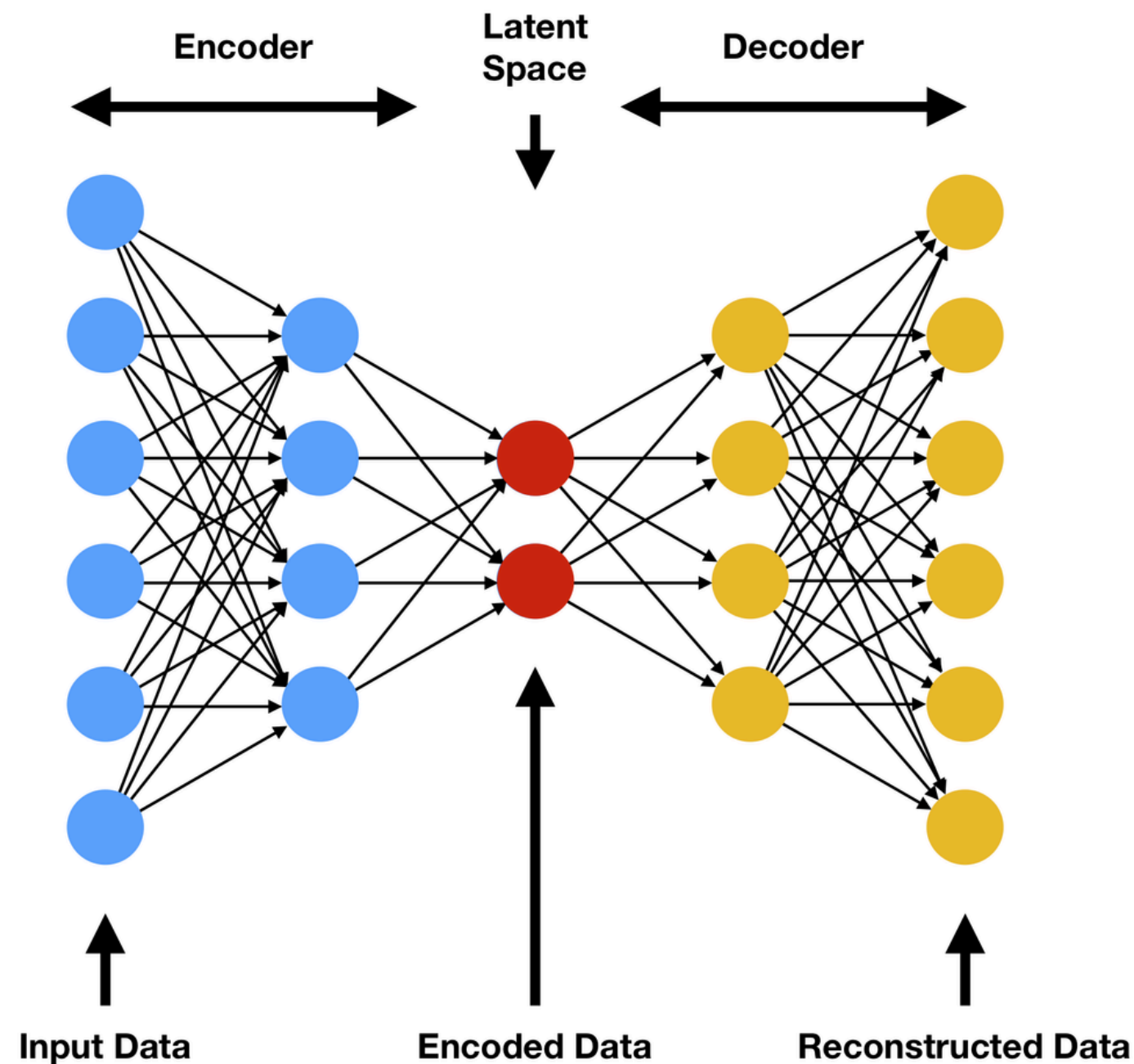
- Discriminative deep has a crucial weakness, the need of labelled data
- Generative deep learning only requires large amounts of unlabelled data
- Generative deep learning models can thus be trained on more data and have a larger number of parameters
- This allows generative deep learning models to better learn the hidden features and regularities of data
- The regularities learnt by such models can then be used for enhancing the performances of discriminative models



Variational Autoencoders



The Autoencoder



The Autoencoder is one of the most important MLP architectures for unsupervised learning. It is composed of three sections:

- **Encoder** Encodes the data into a latent representation
- **Latent Space** Space where the encoded data live (z)
- **Decoder** Convert back the data from the latent space to the standard representation

The Autoencoder is trained in an unsupervised way using a reconstruction loss

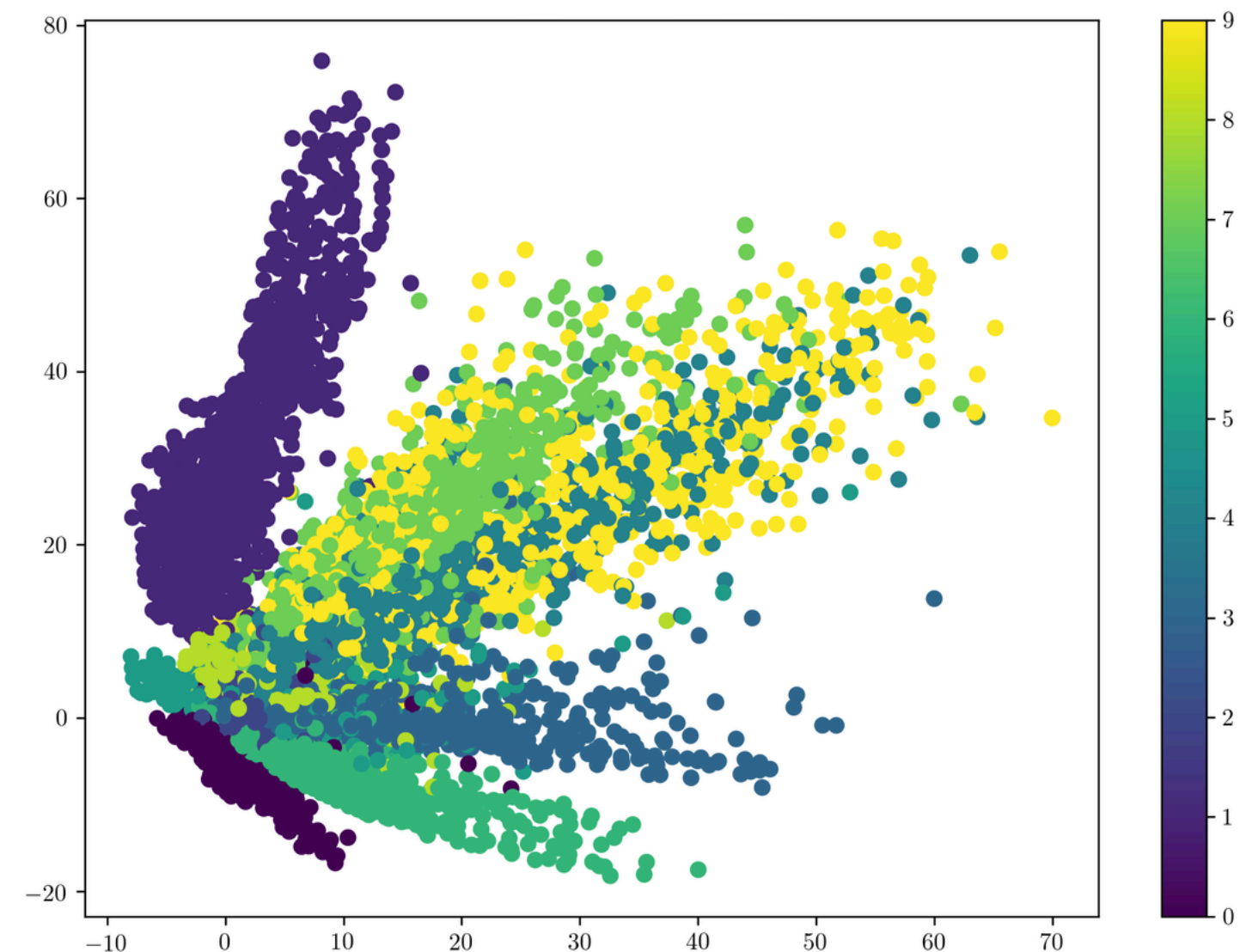
- it is trained to reconstruct in output exactly what it is given in input
- the loss quantifies how different is the output from the input

Dimensionality Reduction

The Autoencoder architecture has a bottleneck,
the latent space:

- if the reconstruction loss is low, the autoencoder can successfully reconstruct the input
- this means that the latent space contains enough information to reconstruct the input
- the latent space contains a low dimensional representation of the input data

Therefore we can train an autoencoder and then use its encoder to get a dimensionality reduction of the data (similar to PCA or T-SNE)



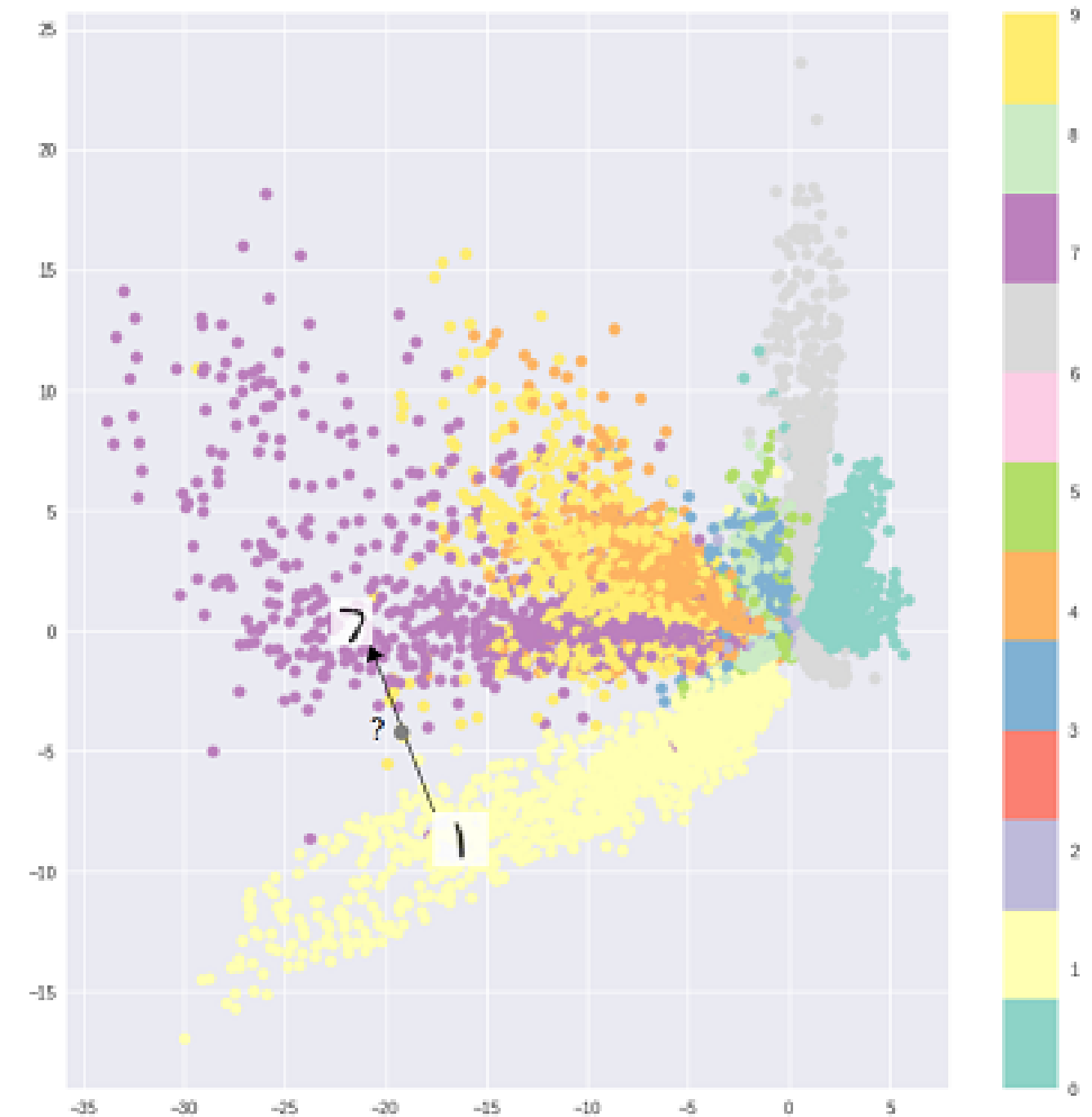
Problems of Generating Data with AEs

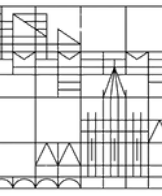
One possible approach to data generation involve an AE:

- we train the autoencoder as usual
- we then generate new samples by using the decoder section
 - we select a random point in the latent space
 - we input it in the decoder
 - we collect the output

This is in theory a valid approach, but fails due to the structure of the latent space

- there is no order
- there are large empty regions
- it is impossible to interpolate between training points

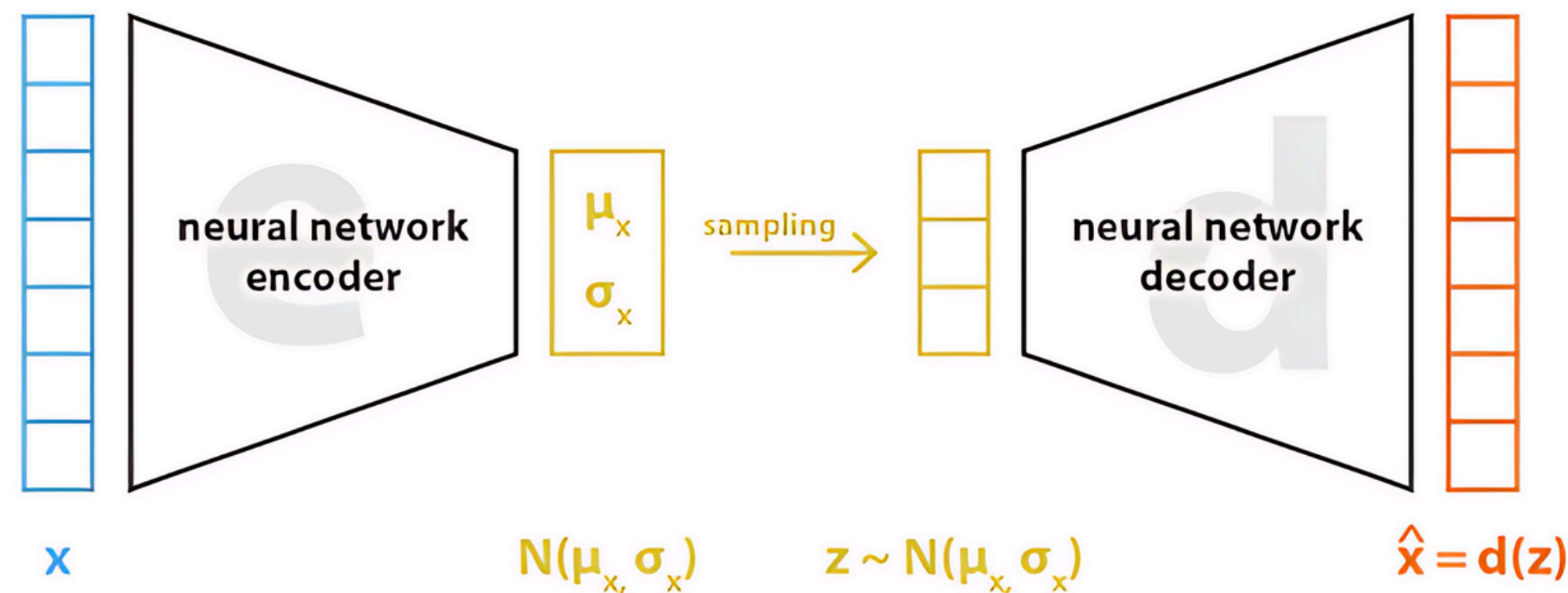


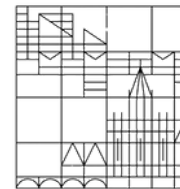


The Variational Autoencoder

The Variational Autoencoder (VAE) adds a twist of stochasticity to the standard AE

- same architecture, but now the encoder output consists of a mean vector and a variance vector
- the value of the latent variable z is computed sampling from a multivariate gaussian whose parameters are defined by the encoder's output
- the latent variable z is feed into the decoder, whose output is trained to be equal to the input

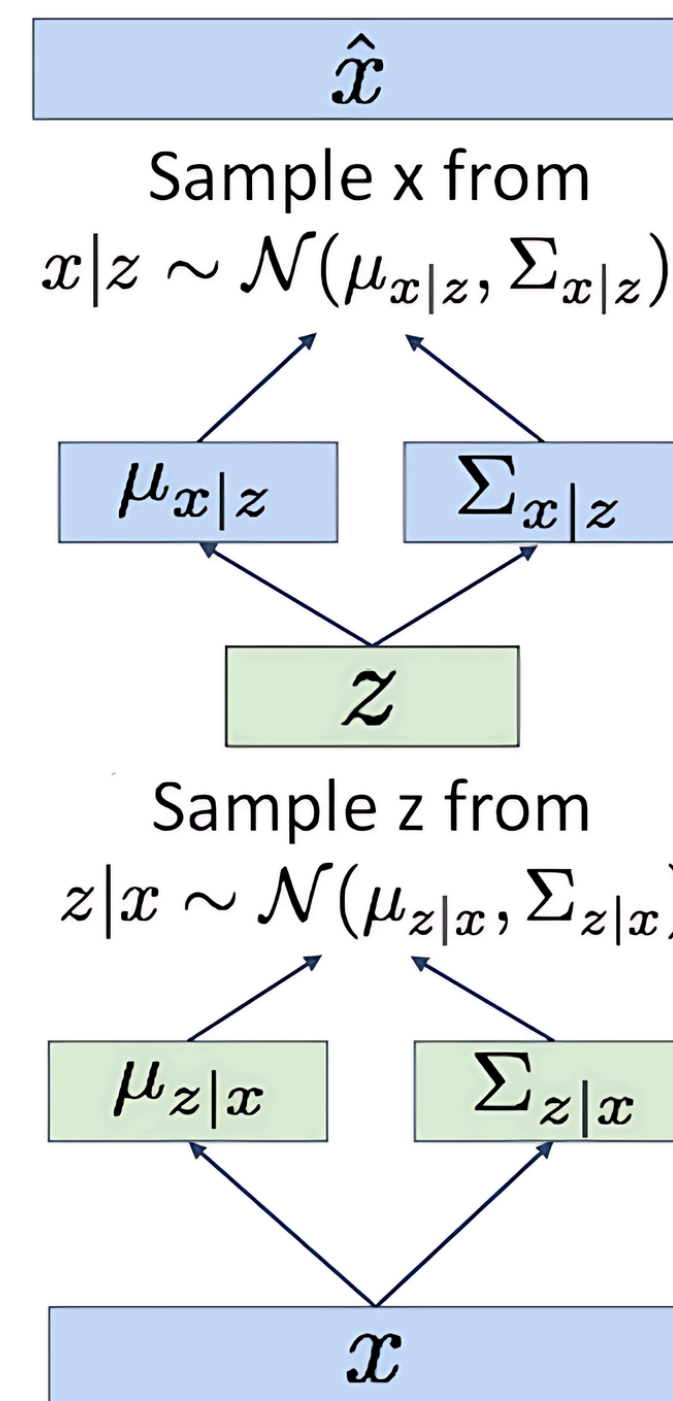


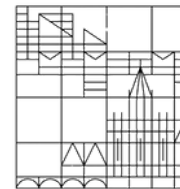


Probabilistic and Deterministic Decoder

Like the encoder, also the decoder can be either deterministic or probabilistic

- a deterministic decoder works as in the standard AE, it simply outputs a vector or an image depending on the tasks
- in a probabilistic decoder the output consists of a mean and a variance vector
 - each component of the output vector is obtained by random sampling
 - the sampling is done using gaussians with mean values and variances defined by the output of the decoder
 - a deterministic encoder is a probabilistic encoder with null variance

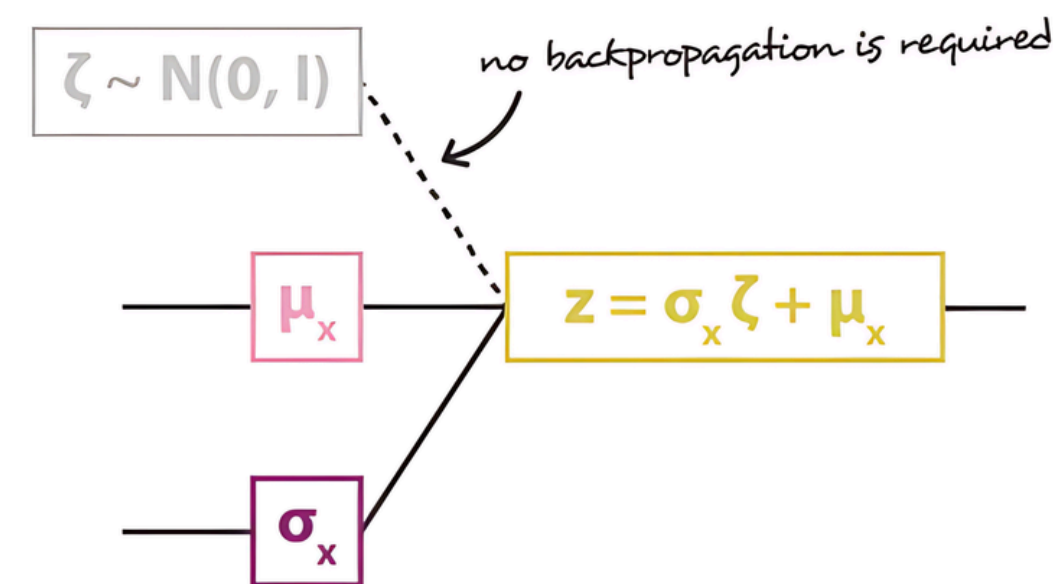
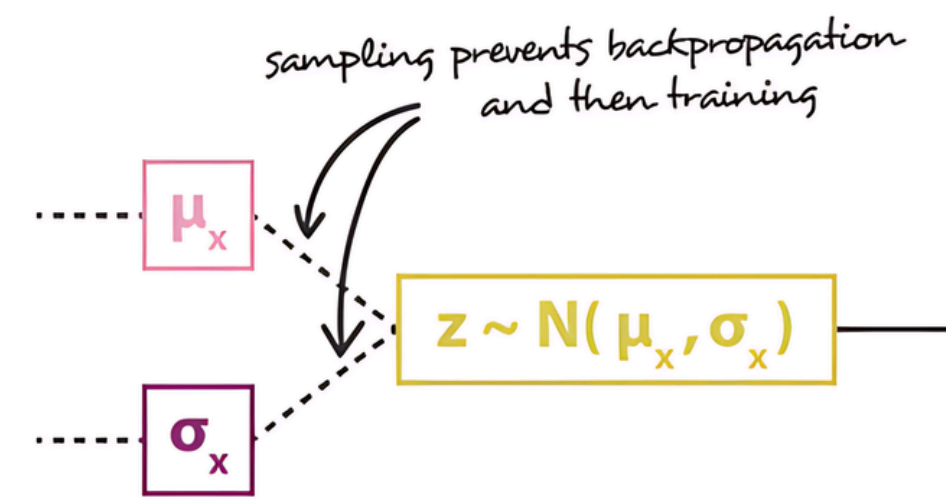


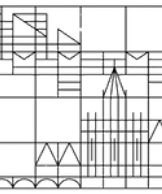


Reparametrisation Trick

The introduction of stochasticity in the VAE architecture poses some challenges:

- there is no direct way to backpropagate through a stochastic variable
- in order to solve this issue we can exploit the reparametrisation trick
 - we sample from a fixed standard gaussian (mean 0 and variance 1)
 - we multiply the result for the standard deviation and we sum the mean value
 - in this way we can backpropagate since we can compute the derivative





Bayesian Formulation

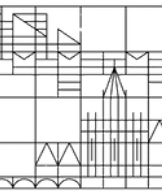
Autoencoders have a well grounded mathematical foundation

- the assumption is that data x are generated starting from some hidden latent variable z
 - ex: for faces these could be gender, age, expression etc
- the distribution of the latent variables $P(z)$ are independent gaussians
 - there is no correlation between the features
 - each feature will have a typical value with small fluctuations

We can use Bayes theorem to write the probability of real data as

$$P(x) = \frac{P(x|z)P(z)}{P(z|x)}$$

- $P(x|z)$ is the decoder
- $P(z)$ is the normal sampling of the latent variable
- $P(z|x)$ is not directly available, but we train a neural network (the encoder) to approximate it as a function $Q(z|x)$



Loss and ELBO

The Bayesian formulation allows to define a loss function for training the VAE

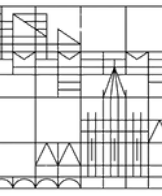
- we want to maximize the probability of the observed data point $p(x)$, so we use the log-likelihood as our loss

$$\text{Loss} = \frac{1}{N} \log P(\{x_i\}_{i=1}^N) = \frac{1}{N} \log \left(\prod_{i=1}^N P(x_i) \right) = \frac{1}{N} \sum_{i=1}^N \log P(x_i) = \mathbb{E}_{x \sim \text{data}} [\log P(x)]$$

- however this loss is impossible to compute directly, so we have to look at a lower bound, the ELBO (Evidence Lower Bound)

$$\text{ELBO} = \left[\mathbb{E}_{z \sim Q(z|x_i)} [\log P(x_i|z)] - D_{KL} (Q(z|x_i), P(z)) \right]$$

- the first term is the expectation value of the probability of the generated data computed over the latent variable (generated from the encoder)
- the second term is the distance between the distribution of the encoder and the prior of the latent variable $P(z)$ that is by definition a gaussian distribution

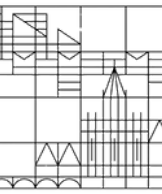


Latent Space and Regularization

This seems very complex, but the concept is very easy:

- the first term quantifies how well we can reconstruct the probability of the real data. For a deterministic decoder it is just the reconstruction error of the data
- the second term is a regularization that makes the latent space more regular

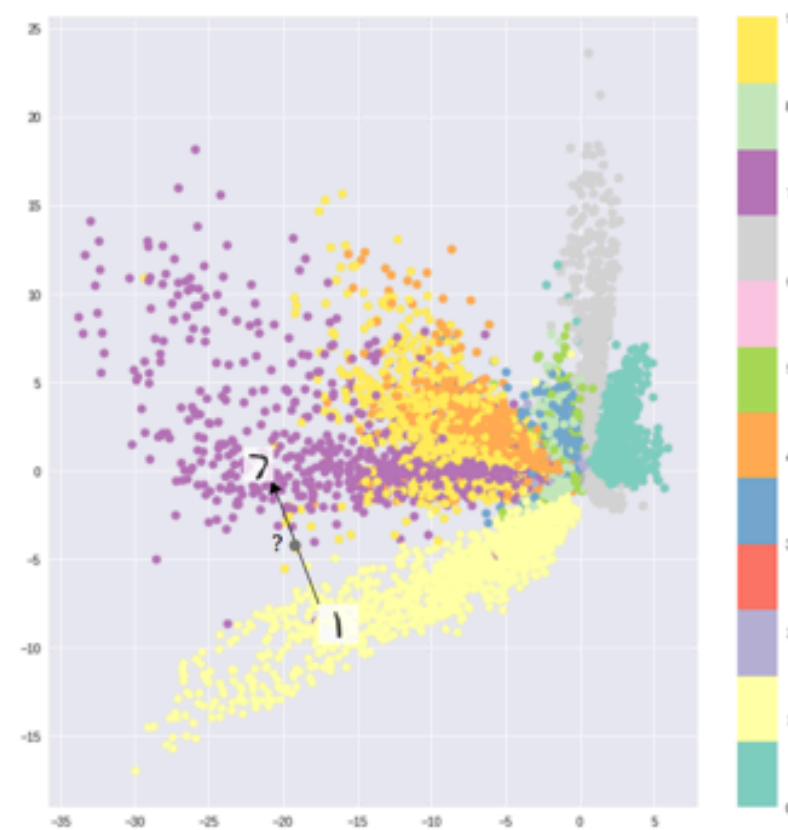




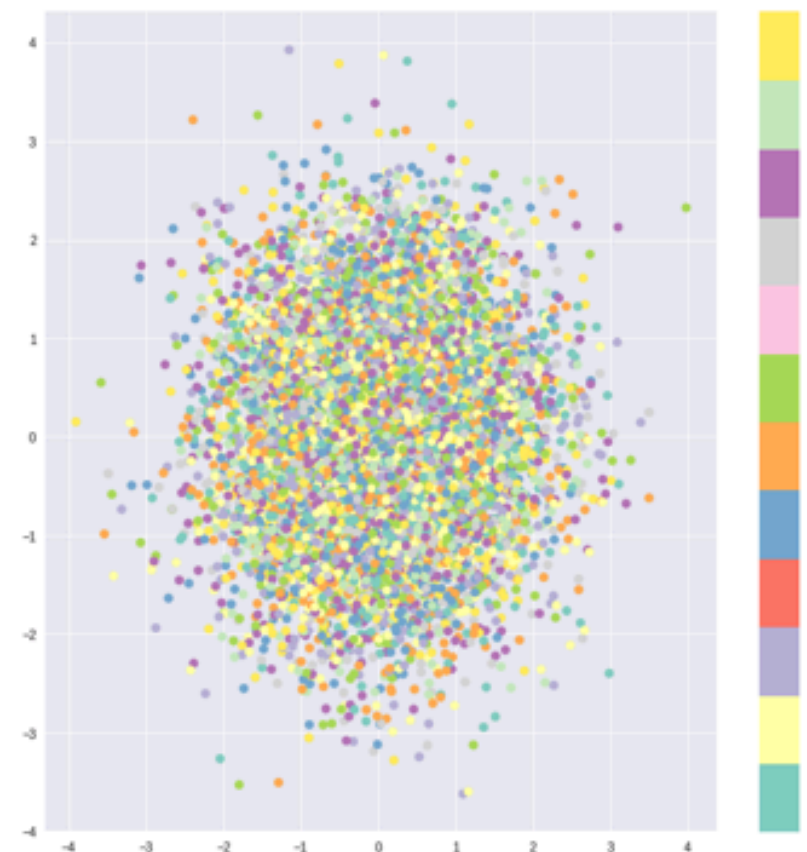
AE vs VAE Latent Space

The reconstruction term would like all points to be very far in the latent space to be easier to distinguish. The regularization term would like to force the points in the latent space to follow a gaussian, so to be very concentrated in the middle.

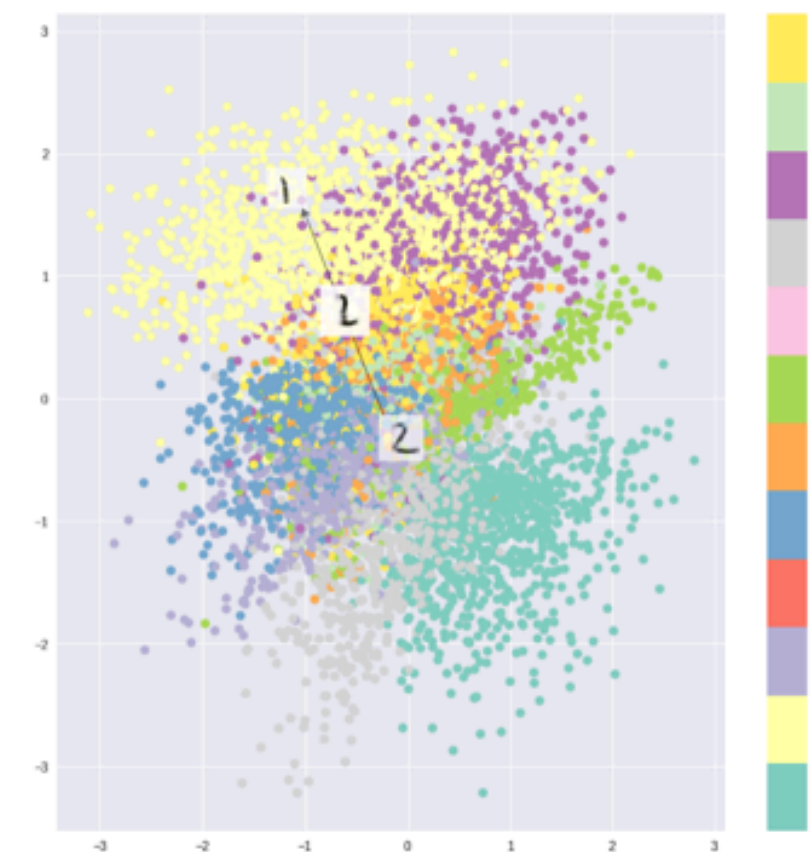
Only reconstruction loss

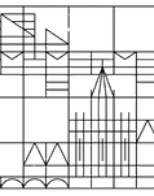


Only KL divergence



Combination

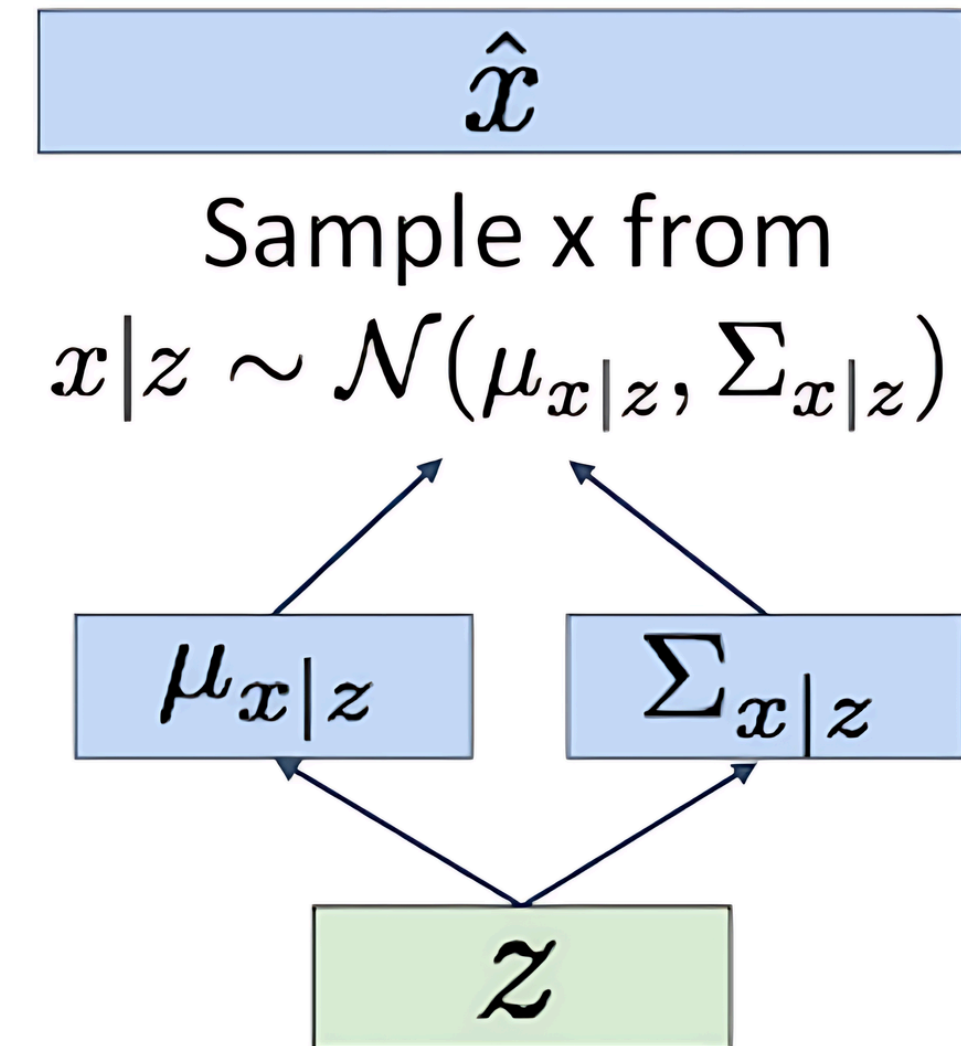




Generating New Data

Once we have trained a VAE using the ELBO we can use the decoder part for generating new data

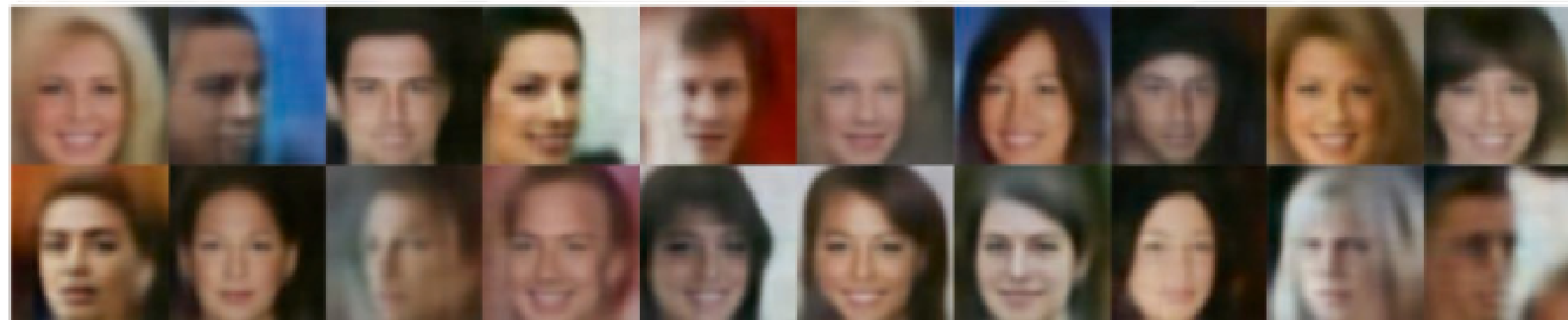
- we select a point in the latent space by sampling from a gaussian
- we process it using the decoder
 - if the decoder is deterministic that is our novel data point
 - if it is probabilistic we use its output to sample the new data
- since now the latent space is much more regular the neural network will be easy to interpolate



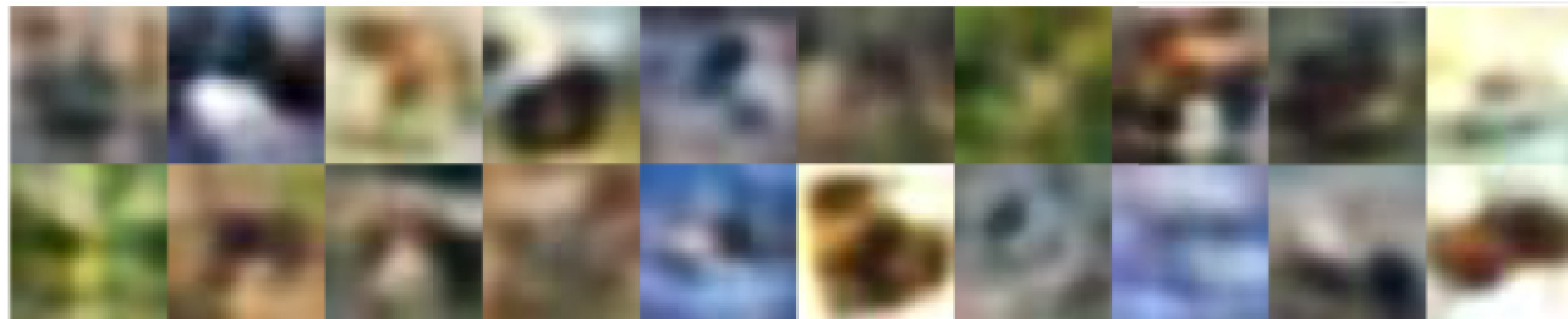
Examples of Generated Data

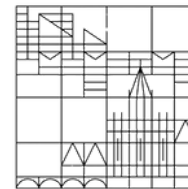
Human faces represent a standard benchmark for generative deep learning. We can very easily detect AI generated faces, while we may struggle with objects or animals.

Labelled Faces in the Wild Dataset



CIFAR-10 Dataset

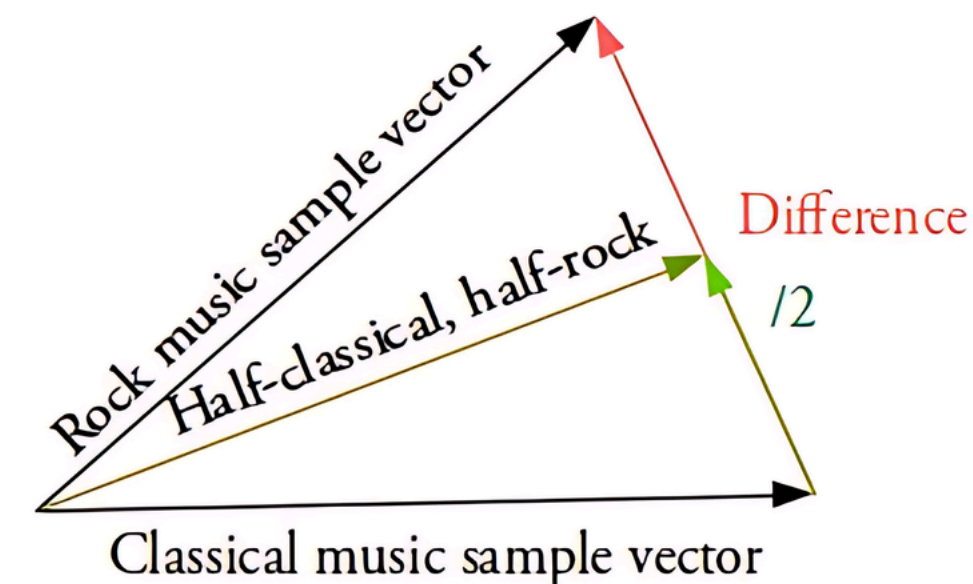




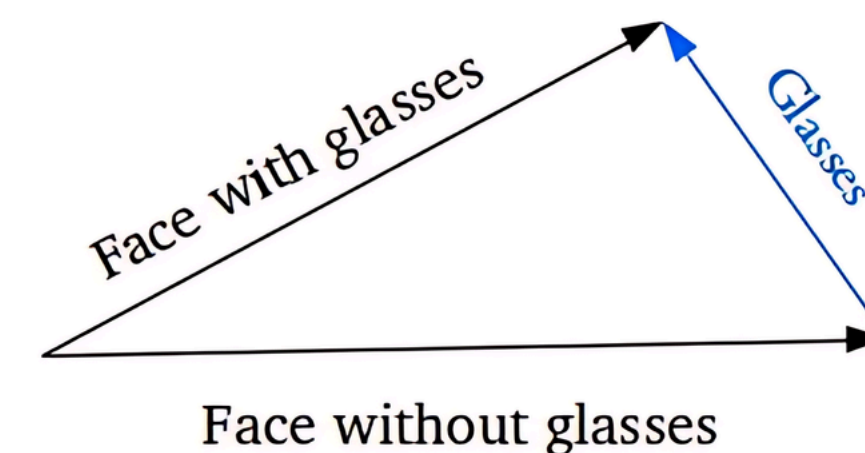
Vectors Arithmetic

The use of the KL divergence term in the loss not only regularizes the latent space, but makes it also very special:

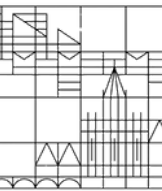
- we are forcing the different directions of the latent space to be orthogonal by using independent gaussians (the covariance is null)
- the result is something similar to a word embedding space, with different directions encoding different concept
- we can subtract two point obtaining the vector that transform one into the other
- we can gradually transition from a data point to another by moving along the line connecting them



Interpolating between samples



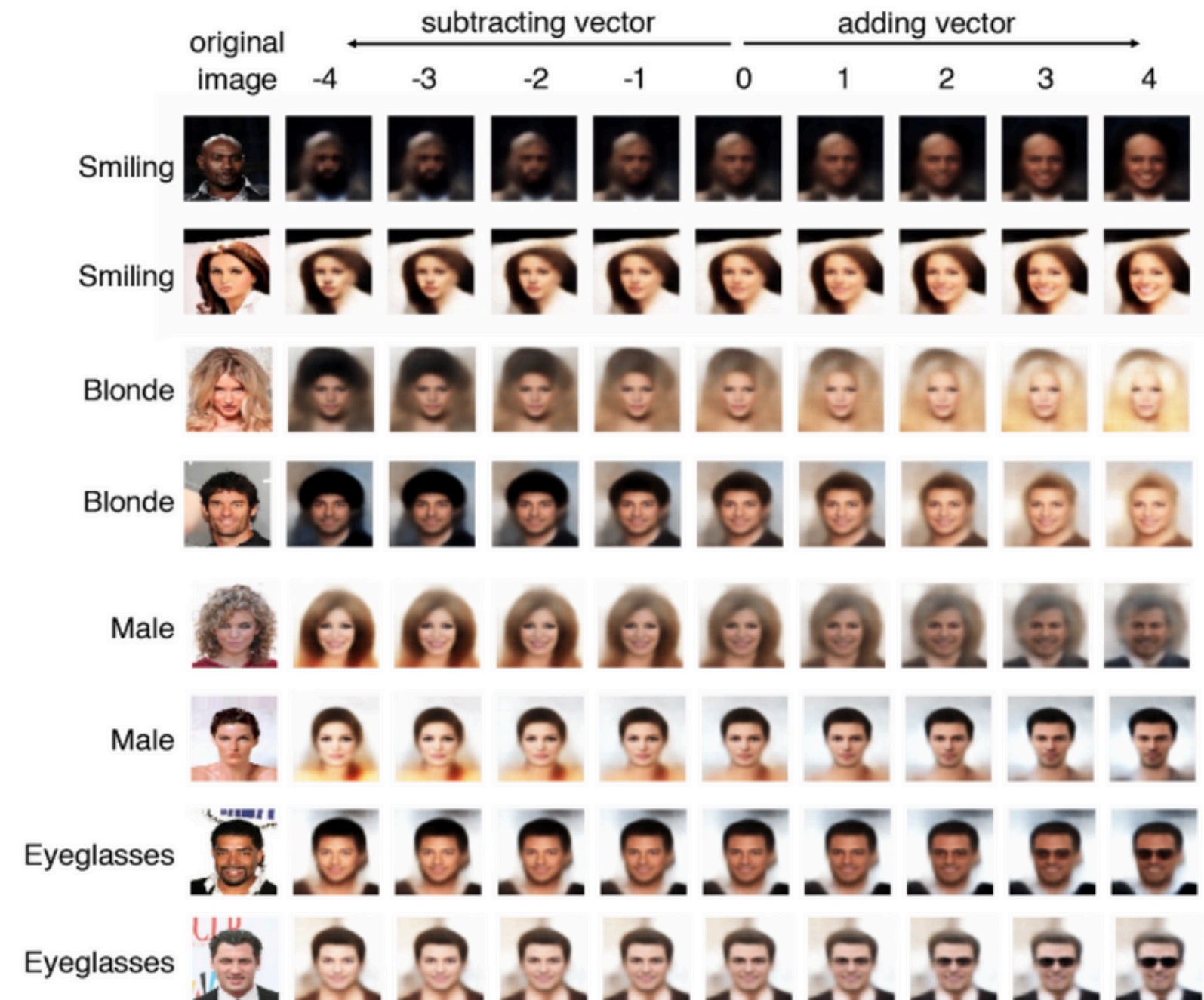
Adding new features to samples

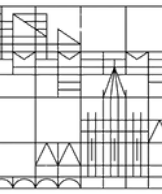


Example: Arithmetic on Faces

In the example we show how it is possible to add different features to images by summing an appropriate latent vector

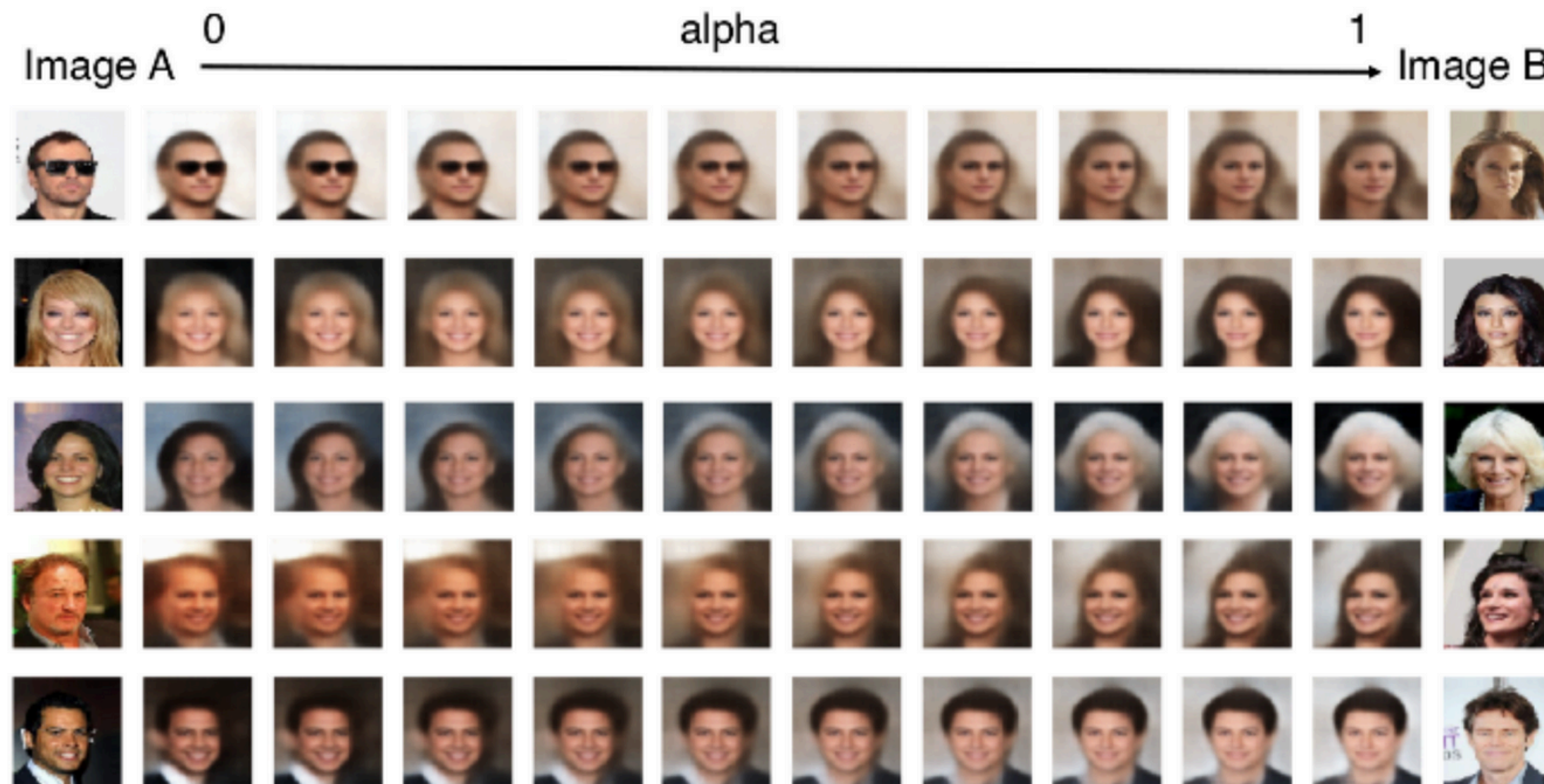
- we start from an original image
- we move along different directions corresponding to different properties
- the resulting output is similar to the original image, but modified including the new feature
- we can modulate the vector to weaken or strengthen the feature



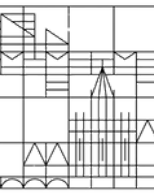


Example: Face Morphing

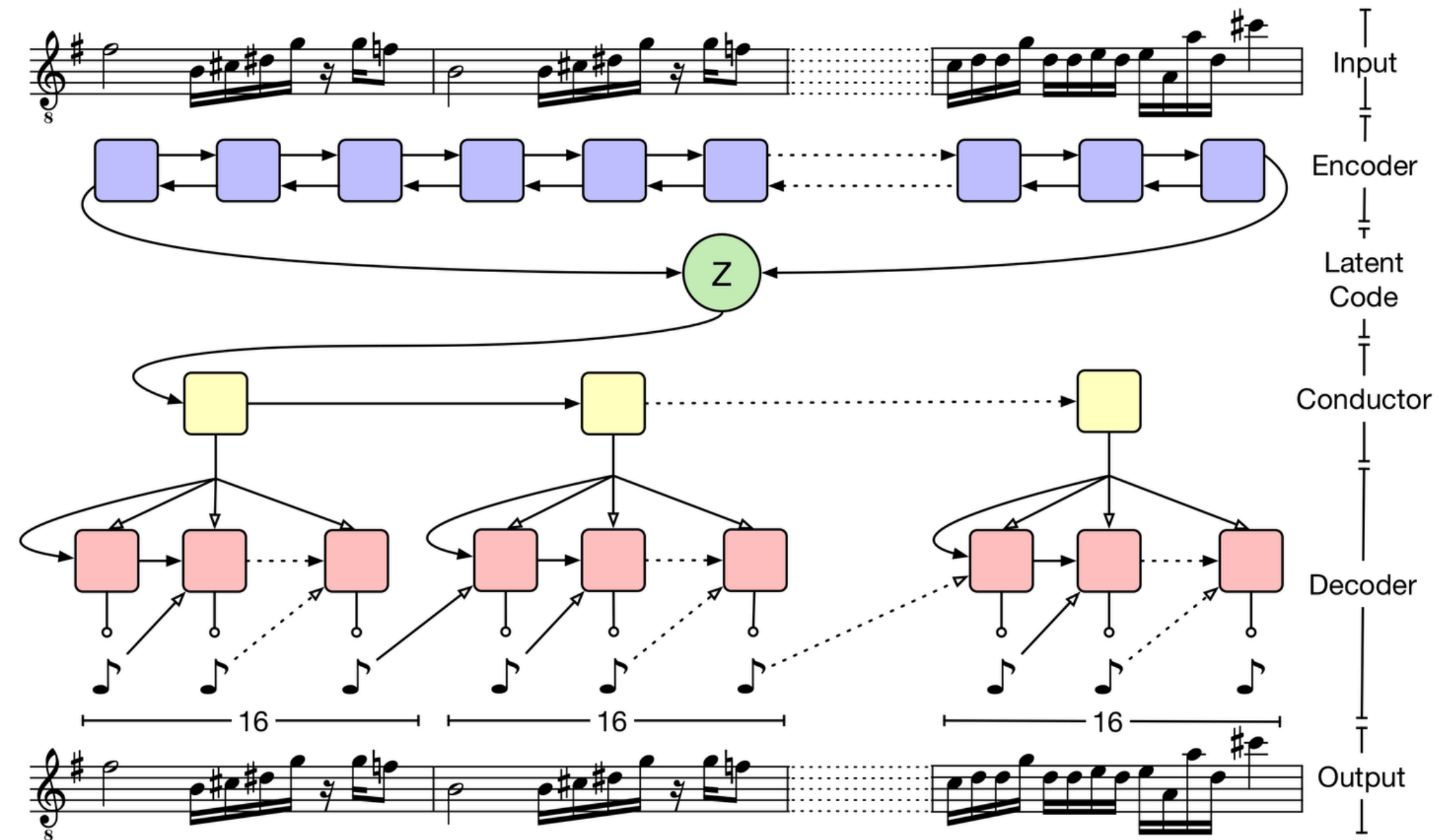
Below we show an interpolation between pairs of images. This is obtained by moving along the line that connect the two images in the latent space and generate a smooth transition between the two.



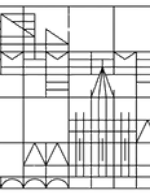
D. Foster, Generative Deep Learning – Teaching Machines to Paint, Write, Compose and Play. O'Reilly Media, Inc., Jun. 2019.



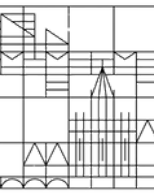
MusicVAE



<https://magenta.tensorflow.org/music-vae>



Generative Adversarial Networks



Limits of VAEs

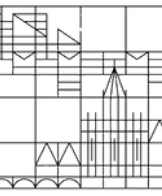
VAE are a very powerful tool

- they are most of the time just better autoencoders
- their latent space has all the good properties we would like it to have
- they are incredible in dimensionality reduction and denoising

However they have strong limits in data generations, especially when it comes to images

- the quality is not bad
- however images are blurred and details are missing
- this derives from the presence of the gaussian regularization

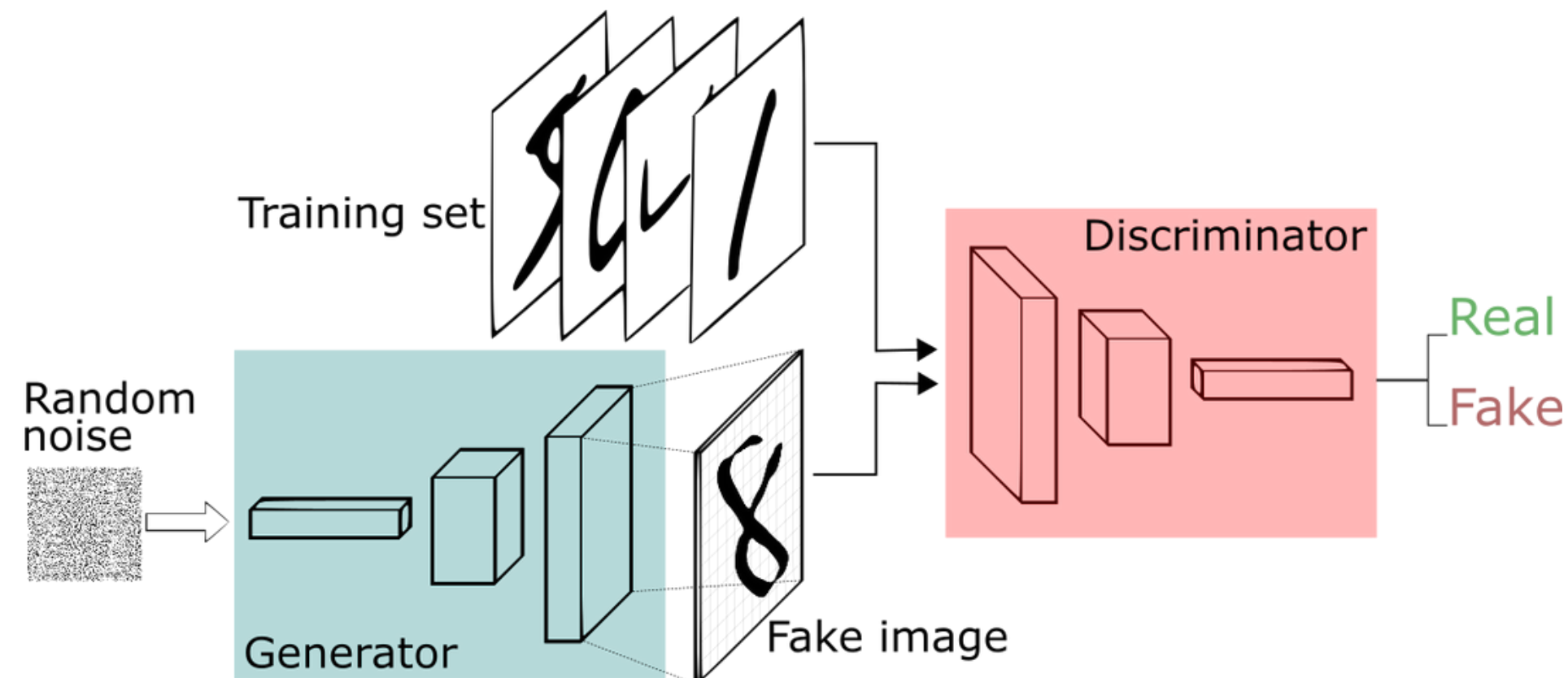


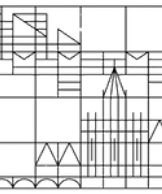


Generative Adversarial Networks

Generative Adversarial Networks (GANs) are much more powerful when it comes to generate realistic images. They are conceptually very similar and are composed of two models

- a generator that is trained to generate artificial data
- a discriminator that is trained to distinguish artificial data from real data
- note that we do not need labelled data (we automatically know what is real and what is fake)

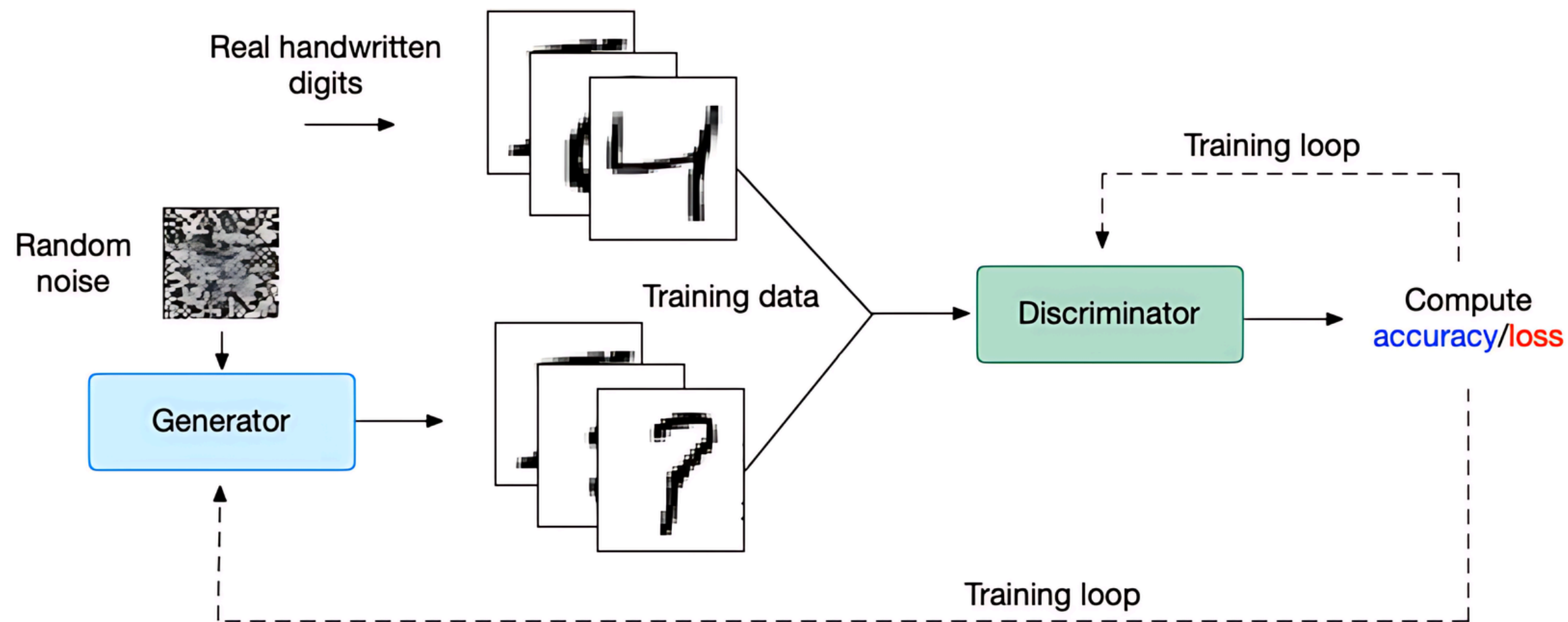




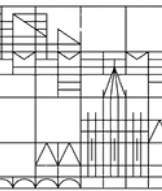
Training GANs

A GAN is trained by alternating between the Generator and the Discriminator

- the generator is used to generate fake images, that are then classified (together with real images) by the discriminator. The generator is then optimized by freezing the weights of the discriminator
- also the discriminator is then updated using the same set of images and the process is iterated



<https://poloclub.github.io/ganlab/>



MinMax Game

The loss of a GAN is not as simple as for other model

- the discriminator wants to maximize the ability to correctly classify the real and fake images

$$\max_D \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

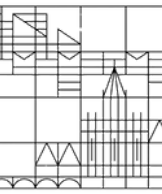
- the generator wants to fool the discriminator on fake images, but doesn't care about the real ones

$$\min_G \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Putting the two ingredient together we get the MinMax Game loss

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

It can be proven that the optimal value is achieved when the distribution produced by the generator coincides with the distribution of real data. However, there is no guarantee that by iteratively updating the two neural networks we reach this optimum.



Gradient Vanishing Problem

Generating is much more difficult than classifying

- at the beginning the discriminator will perform much better than the generator
- the generator loss is flat when the generator performs poorly
- the generator gradient vanishes and training is impossible

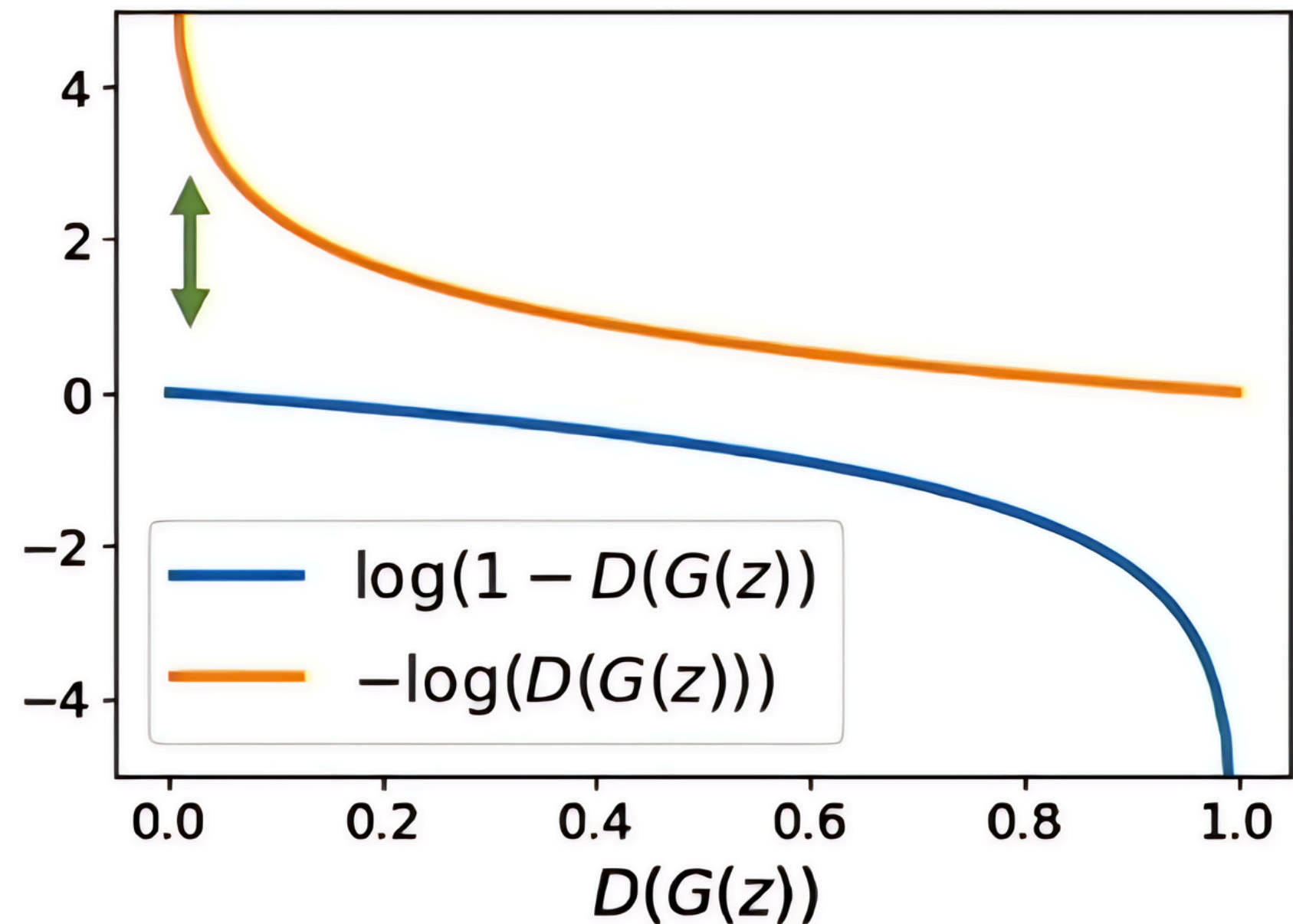
To solve this we modify the generator loss

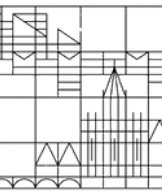
- original loss

$$\min_G \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

- modified (heuristic) loss

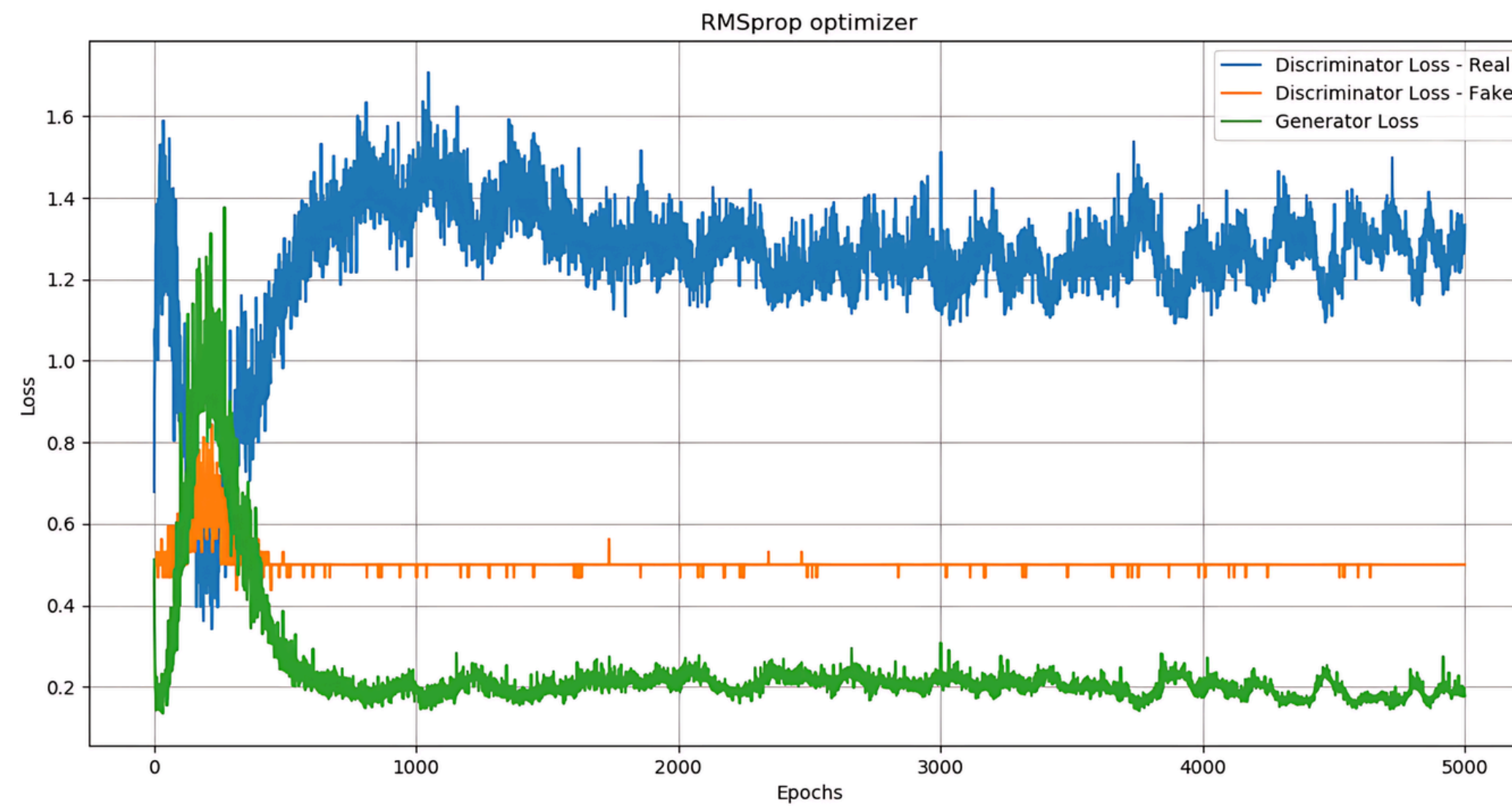
$$\min_G \mathbb{E}_{z \sim p_z(z)} [-\log D(G(z))]$$

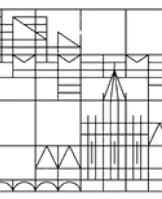




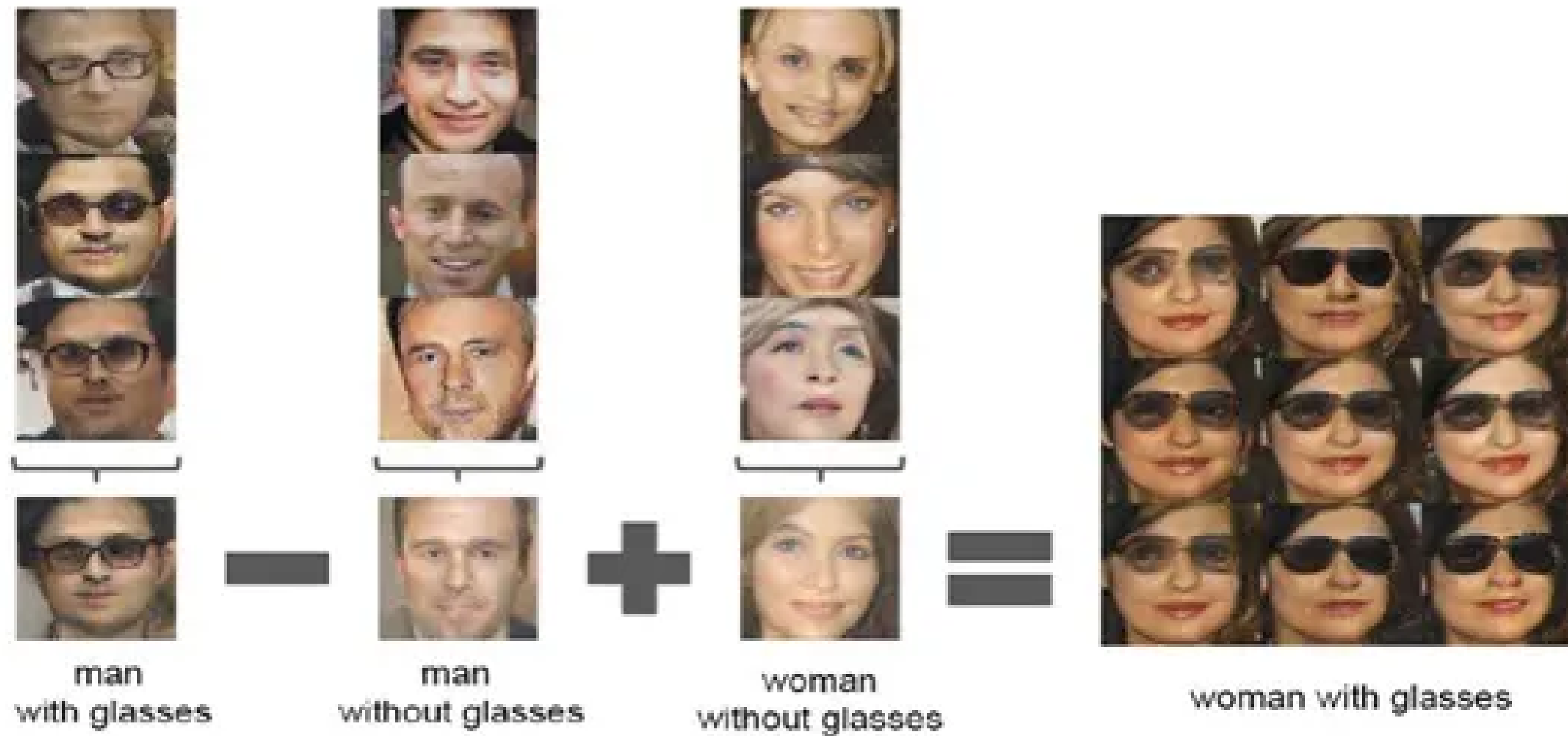
Visualizing the Loss in Training

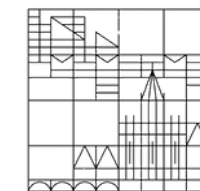
The loss of GANs is much more hard to interpret than the loss of other models. It will often show erratic fluctuations and it is hard to understand whether or not the model is learning.



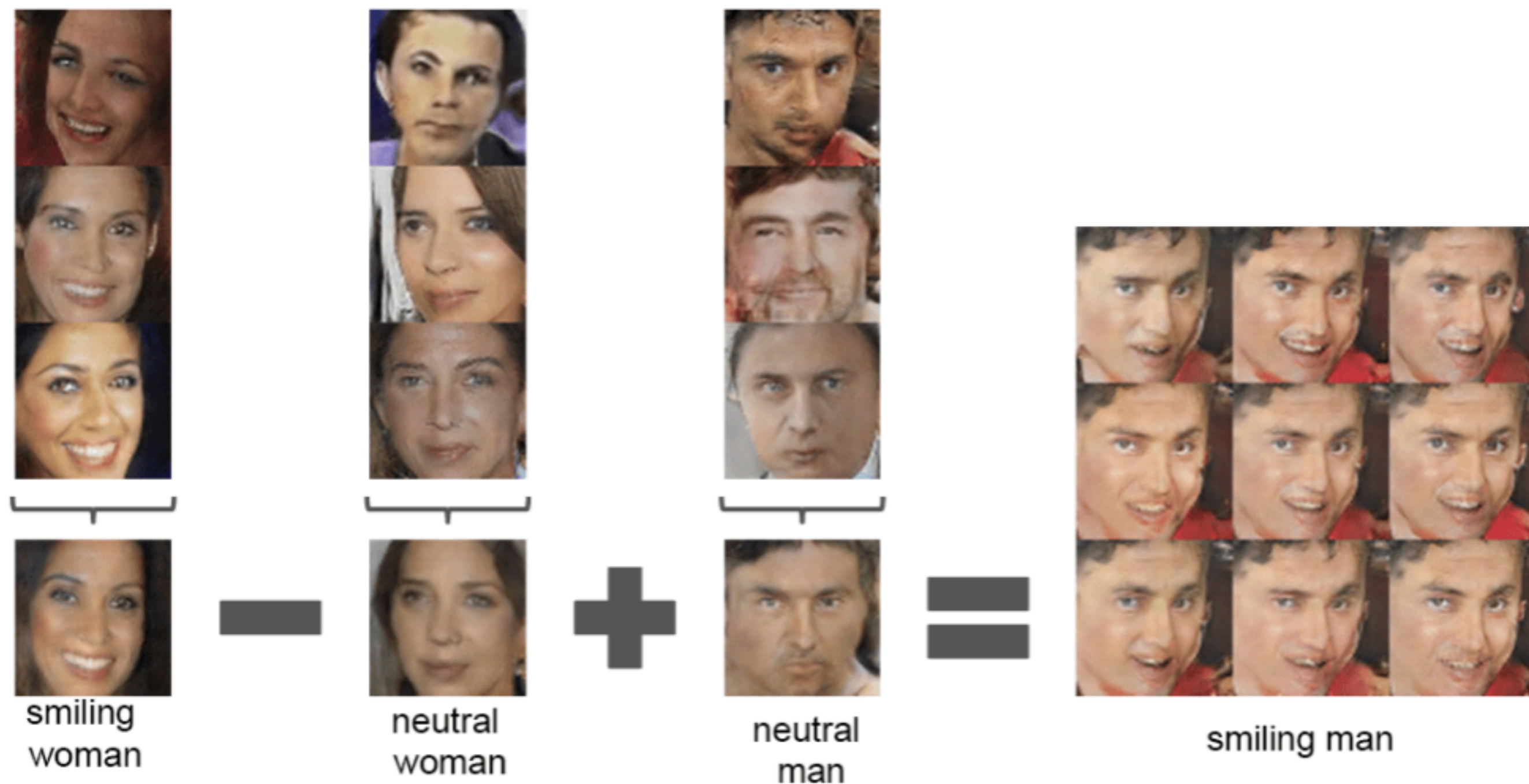


Vectors in GANs



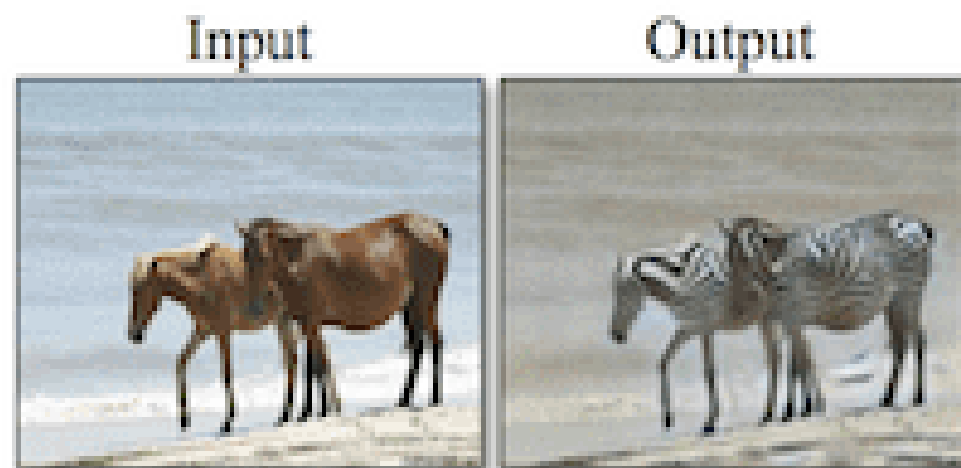
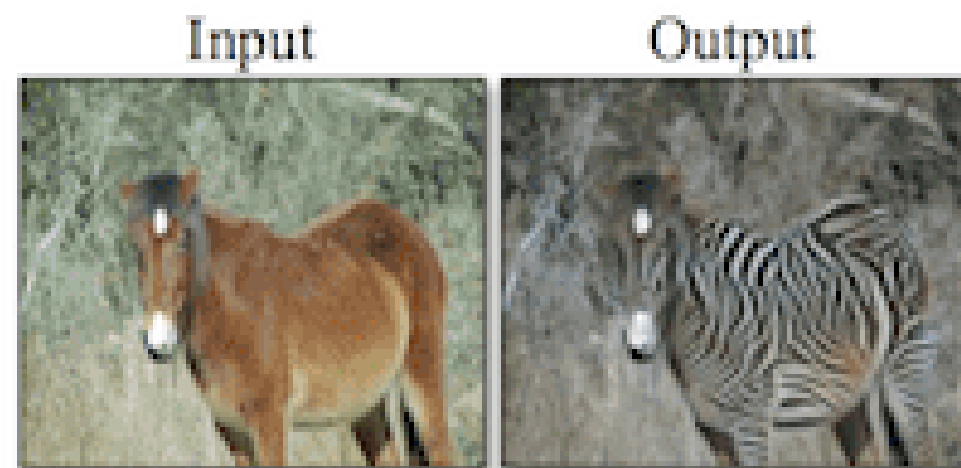


Vectors in GANs

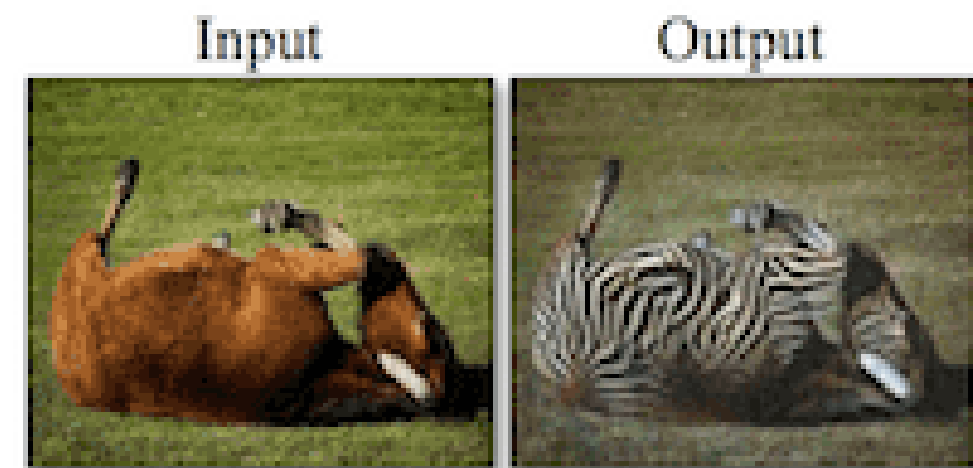


CycleGAN

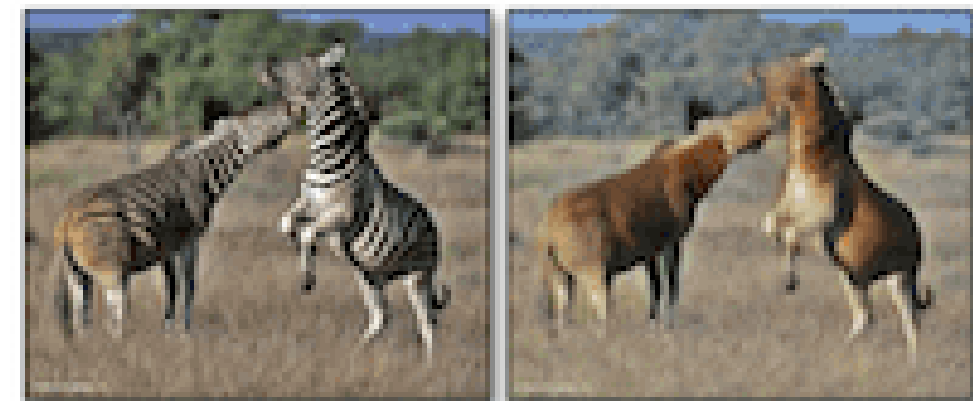
By training two coupled GANs in the so called CycleGAN (2017) it is possible to perform domain transformation. In other words this corresponds to a sort of image-to-image unsupervised translation process, since we only need images from two domains, but no existing pairs.

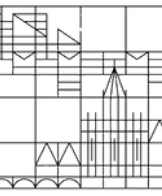


horse → zebra



zebra → horse

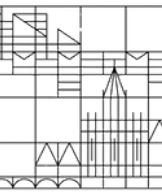




StyleGAN

StyleGAN (Nvidia, 2019) is one of the most powerful GAN model based on convolutional layers. It can generate faces and objects with different resolutions.





Progress of GANs

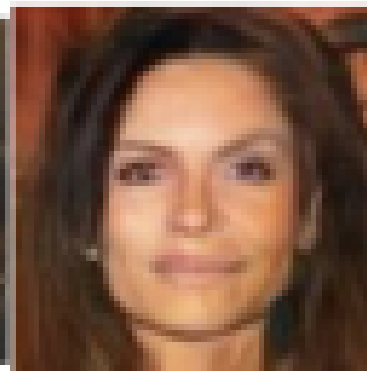
GANs have reached incredible level of detail and resolution. Nowadays identifying AI generated GAN images of human faces is almost impossible, but there are some tricks.



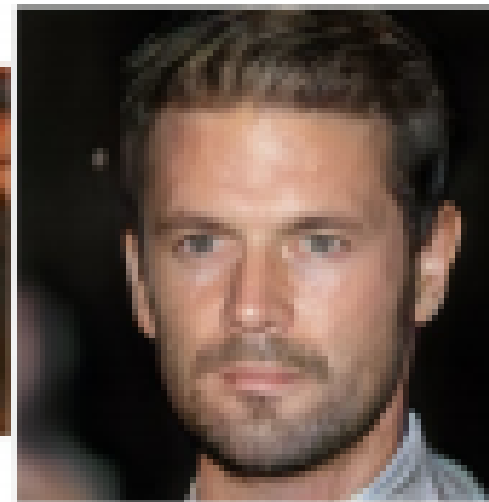
2014



2015



2016



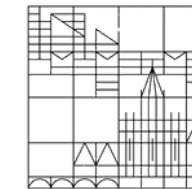
2017



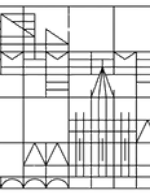
2018



2020



Diffusion Models



Challenges of GANs

GANs are very elegant and conceptually simple, but training them is a nightmare:

- typically generating is much more hard than discriminating, balancing discriminator and generator is a very hard task
- GANs involve two models and training two coupled models at the same time is not simple. The process is often unstable and convergence is never granted
- there is no universally accepted universal metric for evaluating GANs
- the generator may produce limited varieties of outputs, ignoring parts of the data distribution, still resulting in a good overall loss

These limitations are the main reasons why GANs are no more state of the art models, especially for image generation. In this field, diffusion based models such as Stable Diffusion or Midjourney, are now the new standard.

From Noise to Images

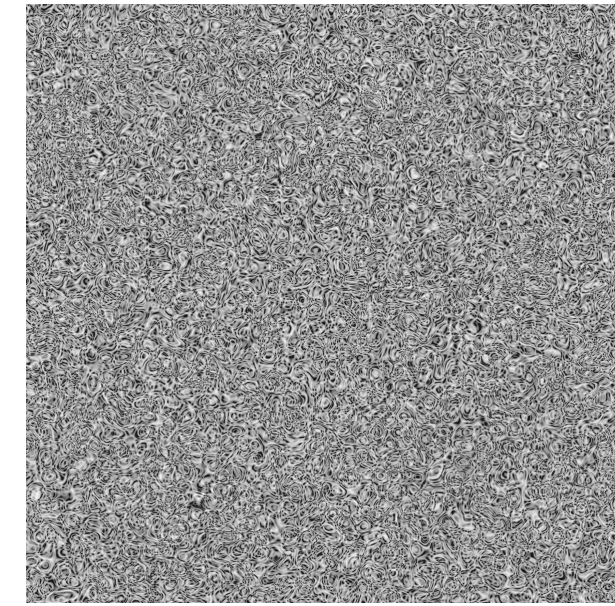
In both VAEs and GANs, the process of generating images begins with random noise.

- in VAEs the decoder generates images by sampling from the latent distribution.
- in GANs the generator creates images starting from random noise

In both these models images are generated in one transformative step from a noise distribution

- images are very complex objects
- a single step may not be enough for generating complex structures such as faces

Autoregressive models and diffusion models use instead a multi-step approach

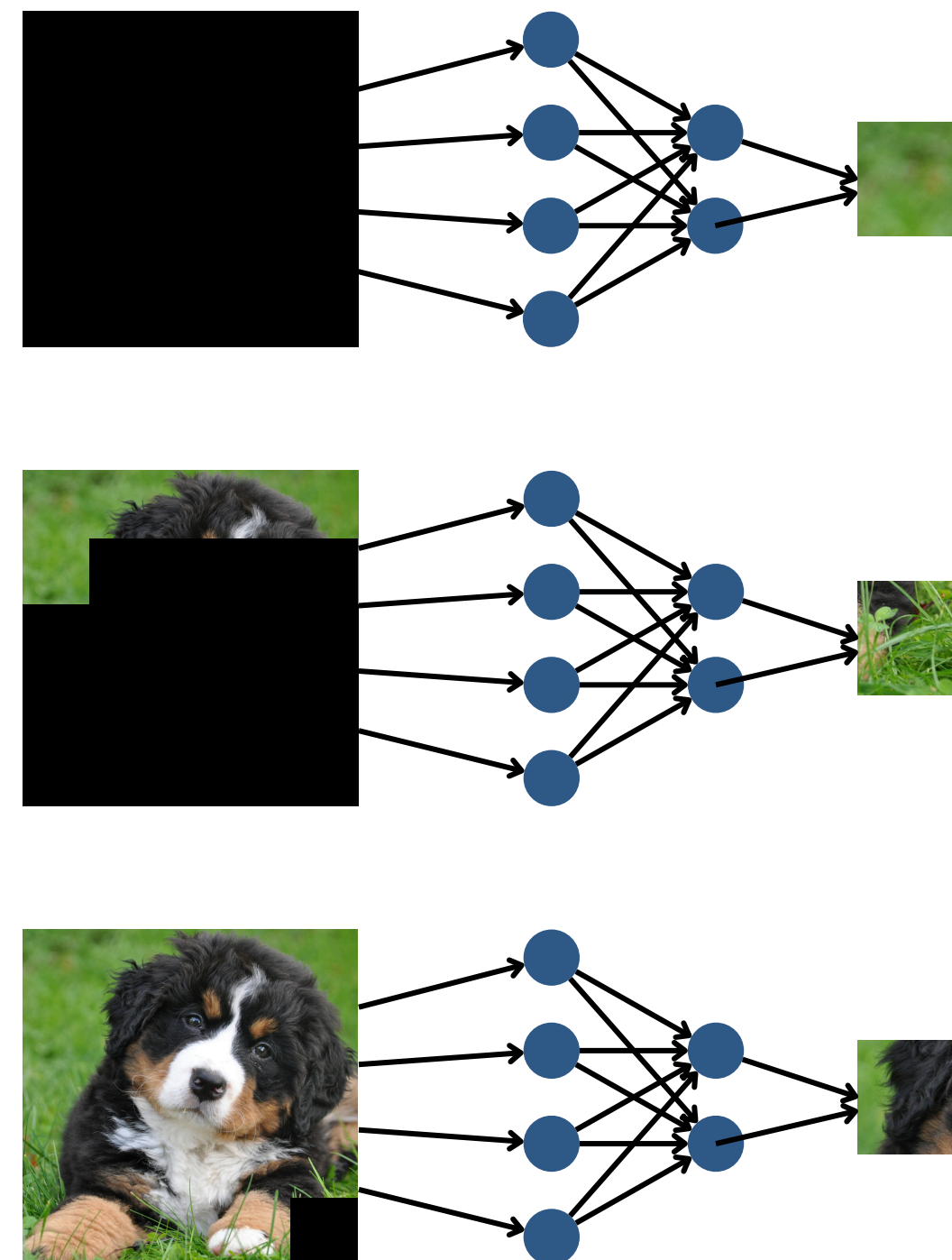


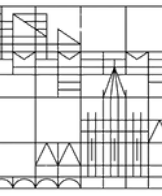
Autoregressive Generative Models

Autoregressive Generative Models generate images step-by-step. Unlike VAEs and GANs, which produce entire images in one step, these models generate one part of the image at a time.

- They begin with an empty image and sequentially generates the remaining parts.
- A neural network predicts the next part of the image based on previously generated content.
- They manage the complexity of generating detailed structures by using a multi-step approach.

This is similar to how generative LLMs work!

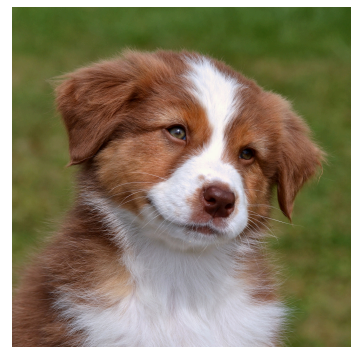
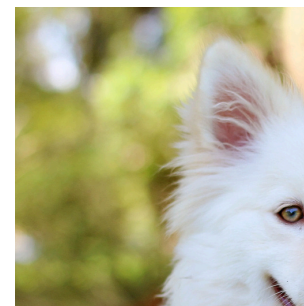
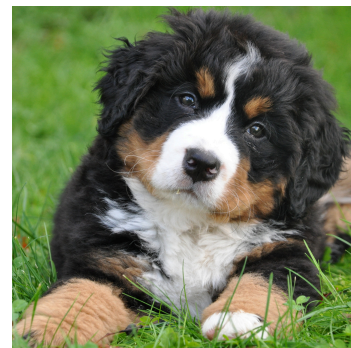


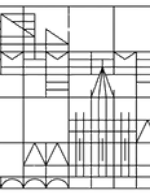


Limits of Autoregressive Generative Models

Autoregressive models face significant challenges when generating high-resolution images:

- When too many pixels are generated at once, the model tends to average over multiple possible images, resulting in a blurred and indistinct output.
- To counteract the blurring, autoregressive models require generating images in many small steps. This multi-step approach, while improving quality, significantly increases computational complexity and processing time.





Diffusion Models

Diffusion models introduce a powerful approach to generating high-quality. Unlike autoregressive models, which predict pixel-by-pixel, diffusion models work by gradually denoising an initially noisy image.

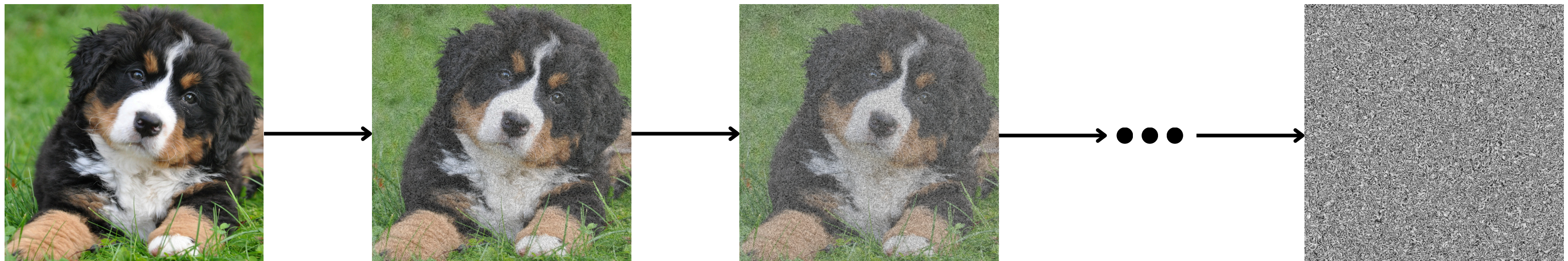
- **Multi-Step Refinement:** The process begins with pure noise, and through a series of steps, the model refines this noise into a clear and detailed image.
- **Noise Structure:** Noise has no inherent correlation or structure, so even when averaging over multiple possible noises, the result remains a valid noise. This property prevents the blurring effect seen in autoregressive models.
- **Bidirectional Process:** Diffusion models consist of a forward process, where the image is progressively noised, and a reverse process, where the noise is incrementally reduced.

In practice we train a neural network to denoise images by predicting what noise was added to them

Forward Process

The forward process in diffusion models involves progressively adding noise to an image over several steps until it becomes pure noise. This gradual noising process serves to map the image to a noise distribution.

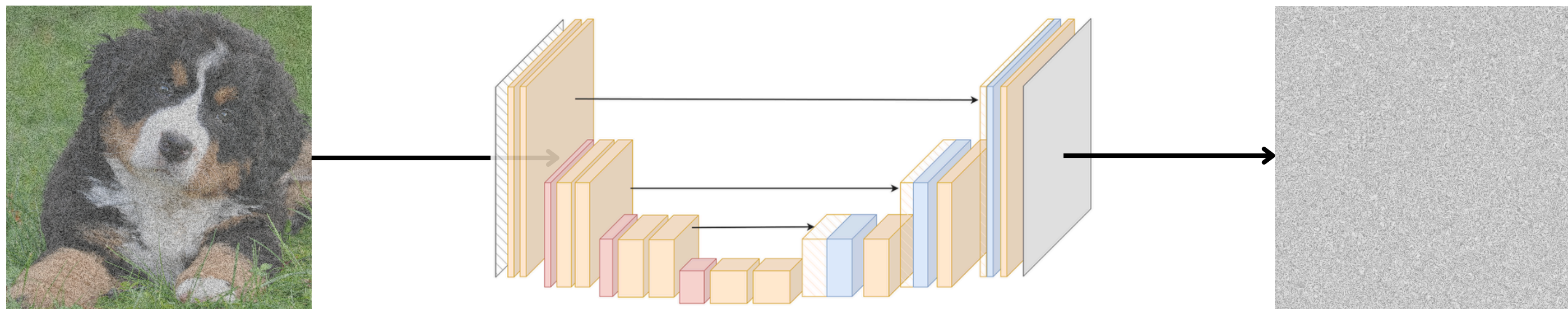
- less noise is added in the beginning, while more noise is added in the last steps (cosine schedule)
- this process produces a series of noisy images, each with a different noise level, that can be used for training the denoising network
- the process is completely unsupervised

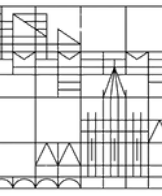


Reverse Process

The reverse process in diffusion models focuses on denoising the image step-by-step, transforming the noise back into a clear image

- The core idea is to train a neural network to predict the noise added at each step in the forward process. By accurately predicting this noise, the network can progressively reduce it.
- A specific CNN architecture called UNet is commonly used for this task. UNet consists of an encoder-decoder structure with skip connections
- The UNet model is applied iteratively, each time reducing a small amount of noise, gradually reconstructing the image from the noisy version.

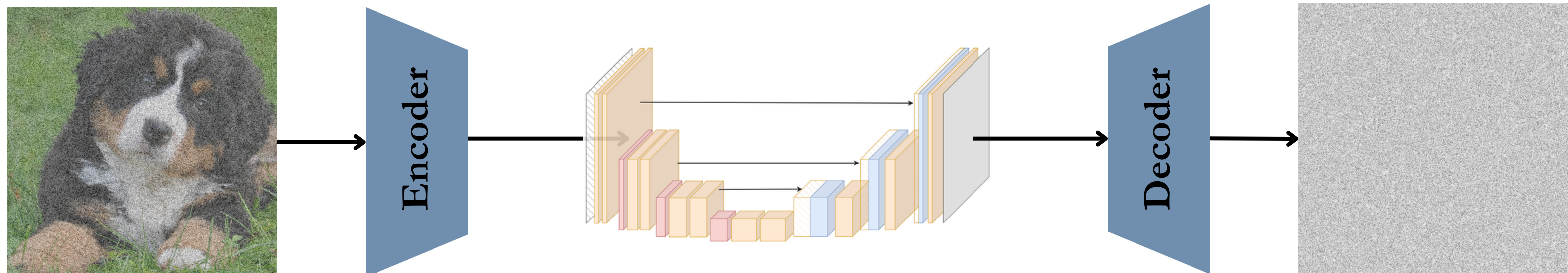




Latent Diffusion Models

Latent diffusion models improve upon traditional diffusion models by operating in a lower-dimensional latent space instead of the high-dimensional pixel space.

- Directly working with pixel data involves very high dimensionality, making the process computationally expensive and less efficient.
- To address this, latent diffusion models first use an encoder to transform the high-dimensional image data into a lower-dimensional latent space.
- Efficient Denoising: In the latent space, the model performs the denoising process, which is computationally more efficient and effective.



GANs vs Diffusion Models

GANs often suffer from a phenomenon called mode collapse, where the generator produces a limited variety of images. This results in many images looking very similar, as the generator fails to capture the full diversity of the training data. Diffusion models, instead, do not show this problem.

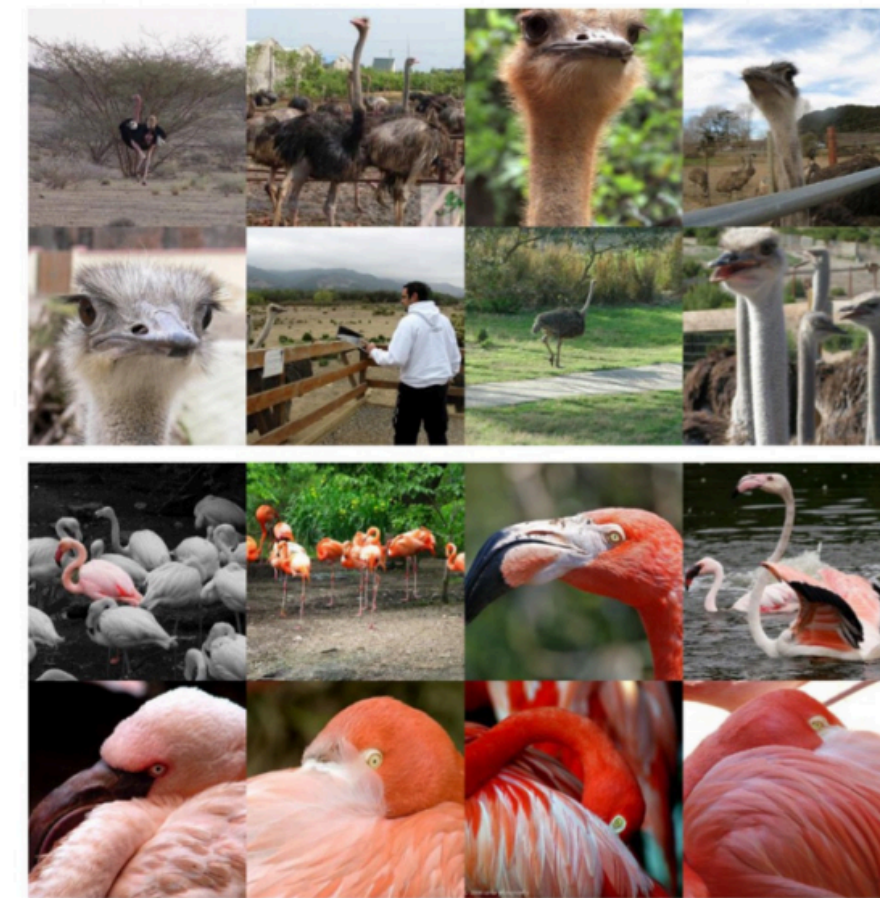
GAN

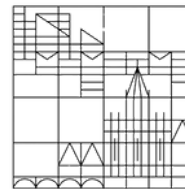


Diffusion



Training Data





Conditional Generation

Conditional generation allows models to generate images based on specific text inputs

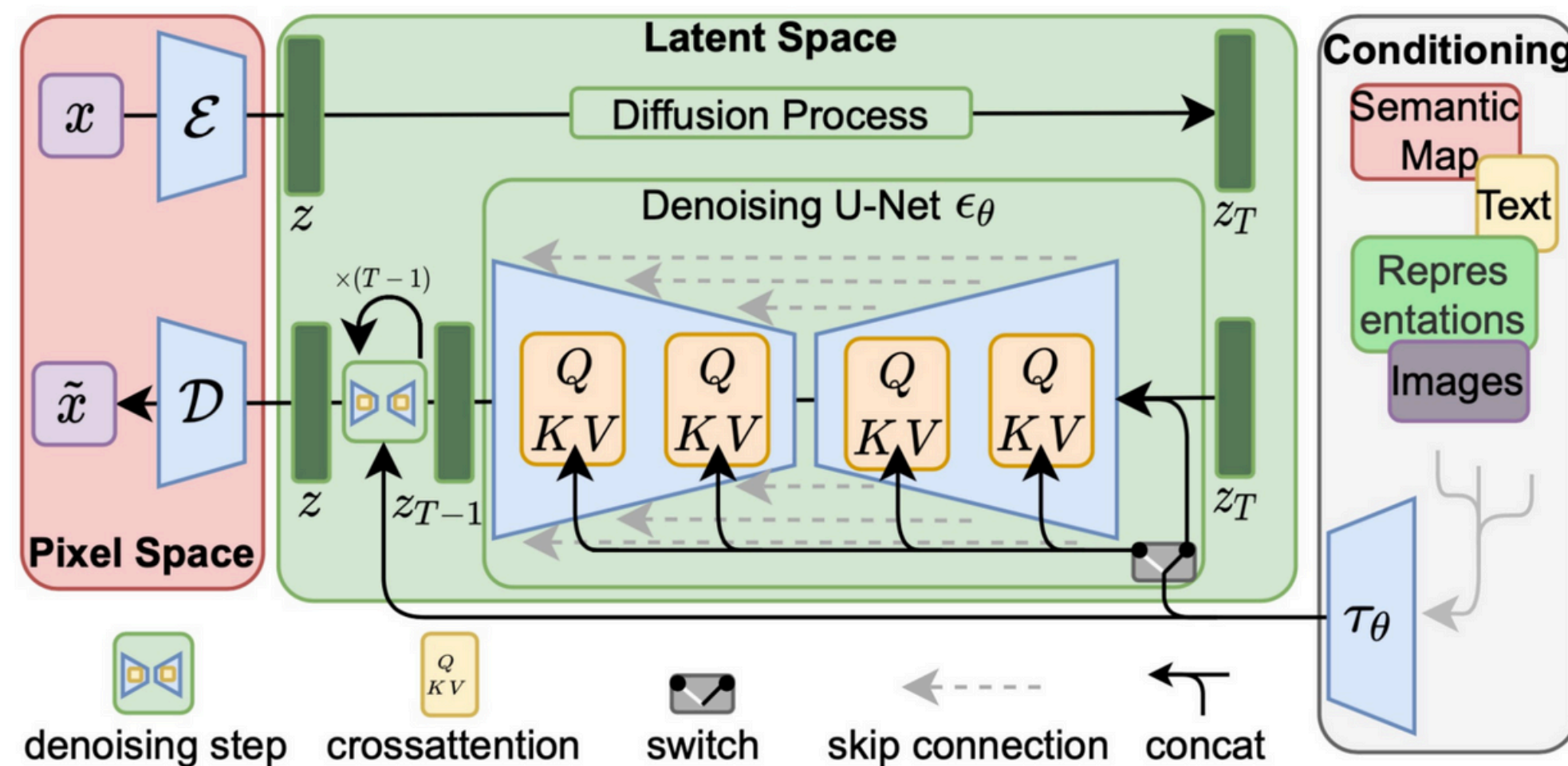
- **Explicit Conditioning:** In this approach during training, the condition is directly fed into the model alongside the noise input. This could be in the form of class labels, text descriptions, or other types of auxiliary information.
- **Classifier Guidance:** This method involves using a pre-trained classifier to guide the generation process. The classifier provides gradients that help adjust the generation towards the desired condition.
- **Classifier-Free Guidance:** In this approach, the model is trained to generate images both with and without conditions. During generation, a combination of the conditional and unconditional outputs is used to guide the final image synthesis.

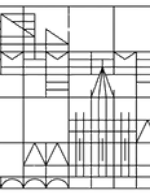
<https://poloclub.github.io/diffusion-explainer/>

Stable Diffusion

Stable Diffusion is a type of latent diffusion model that generates high-quality images efficiently.

- **Latent Space:** Reduces complexity by operating in a lower-dimensional latent space.
- **Denoising UNet:** Uses a UNet architecture to iteratively refine the image.
- **Attention:** Incorporates attention for conditioning on inputs like text or semantic maps.





State of the Art Models

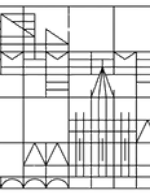
Street style photo of a young woman, red gucci jacket, blue gucci shirt, wide shot, natural lighting, soho, shot on Agfa Vista 200, 4k

Midjourney v5



Dall-E 3





State of the Art Models

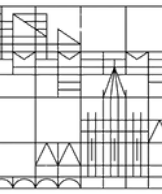
Photography shot through an outdoor window of a coffee shop with neon sign lighting, window glares and reflections, depth of field, little girl with red hair sitting at a table, portrait, kodak portra 800, 105 mm f1.8

Midjourney v5



Dall-E 3





Video Generation: Sora and Veo 3



Sora: Creating video from text

 OpenAI

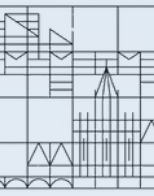


Gemini AI video generator powered by Veo 3

Turn text into videos with sound in Gemini with Veo 3 & Veo 3 Fast, our latest AI video generator from Google.

 Gemini

<https://openai.com/index/sora/>
<https://gemini.google/overview/video-generation/>



Summary

Generative Deep Learning

While discriminative deep learning reconstructs the conditional probability of the label given the data $P(y | x)$, generative deep learning gives access to the distribution of data $P(x)$

Variational Autoencoders (VAEs)

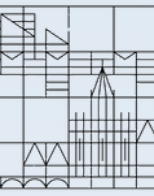
VAEs are a modification of autoencoders that include a stochastic component. Their latent space is regularized and this allows to interpolate between training data, generating new samples

Generative Adversarial Networks (GANs)

GANs consists of two models, a generator and a discriminator, that are trained together. The generator can then be used to produce artificial data.

Diffusion Models

While GANs and VAEs generate images in a single step, diffusion models exploit multiple steps of noise removal, providing better performances in image generation



Next Lectures and Events

Tomorrow Afternoon Coding Session

We will code a Variational Autoencoder to generate fashion-MNIST images and we will use a local diffusion model

Next Week

- On Wednesday 16/07 you will present your ideas on how to approach the final projects. Each group will have 5 minutes to present + 5 minutes for discussion.
- On Thursday 17/07 Simone Petruzzi (Sapienza University, Roma) will give a talk titled "Introduction to Mechanistic Interpretability of LLMs". This will be the last lecture of the course and we will then move to the biergarten.