HOCHSCHULE HANNOVER
UNIVERSITY OF APPLIED SCIENCES AND ARTS
–
Fakultät IV Wirtschaft und Informatik

Universidad Rey Juan Carlos
Escuela de Másteres Oficiales

A comparative Study of Deep Generative Models for Image Generation

Albert Szeliga

Master's Thesis for the degree in Computer Engineering

25. September 2023

This document is licensed under the Creative Commons Attribution 4.0 International (CC BY 4.0) license, except for the images used in the work, which are taken from other works and for which individual image rights and licensing conditions apply. The respective sources of these figures are indicated in the captions.

Author Albert Szeliga
Mat. Number: 1661504
albert.szeliga33@gmail.com

First Examiner: Prof. Dr. Adrian Pigors
Abteilung Informatik, Fakultät IV
Hochschule Hannover
adrian.pigors@hs-hannover.de

Second Examiner: Prof. Dr. Holger Billhardt
Universidad Rey Juan Carlos (Madrid)
holger.billhardt@urjc.es

Declaration of Authorship

I hereby declare that I have written the submitted Master's thesis independently and without external help, that I have not used any sources and tools other than those indicated by me, and that I have identified the passages taken from the works used as such, either verbatim or in content.

Hannover, the 25. September 2023 Signature

Abstract

In the last years generative models have gained large public attention due to their high level of quality in generated images. In short, generative models learn a distribution from a finite number of samples and are able then to generate infinite other samples. This can be applied to image data. In the past generative models have not been able to generate realistic images, but nowadays the results are almost indistinguishable from real images.

This work provides a comparative study of three generative models: Variational Autoencoder (VAE), Generative Adversarial Network (GAN) and Diffusion Models (DM). The goal is not to provide a definitive ranking indicating which one of them is the best, but to qualitatively and where possible quantitively decide which model is good with respect to a given criterion. Such criteria include realism, generalization and diversity, sampling, training difficulty, parameter efficiency, interpolating and inpainting capabilities, semantic editing as well as implementation difficulty. After a brief introduction of how each model works on the inside, they are compared against each other. The provided images help to see the differences among the models with respect to each criterion.

To give a short outlook on the results of the comparison of the three models, DMs generate most realistic images. They seem to generalize best and have a high variation among the generated images. However, they are based on an iterative process, which makes them the slowest of the three models in terms of sample generation time. On the other hand, GANs and VAEs generate their samples using one single forward-pass. The images generated by GANs are comparable to the DM and the images from VAEs are blurry, which makes them less desirable in comparison to GANs or DMs. However, both the VAE and the GAN, stand out from the DMs with respect to the interpolations and semantic editing, as they have a latent space, which makes space-walks possible and the changes are not as chaotic as in the case of DMs. Furthermore, concept-vectors can be found, which transform a given image along a given feature while leaving other features and structures mostly unchanged, which is difficult to archive with DMs.

Table of Contents

1 Introduction

In recent years generative AI models have gained large popularity for generating artificial, but realistic images or text not only in the scientific community, but in the public as well. In general, generative AI refers to artificial intelligence, that can generate new content based on a textual description or even without it [GBGM23]. Within this category there are deep generative models, where a deep neural network is responsible for the generation of new content [RH21]. Various models have been proposed, which can generate not only images, but text or audio as well. The use of deep generative models goes beyond generation of random images and includes so-called deep fakes, neural style transfer or combining of styles from different images. Deep fakes allow the user to replace a given face in an image with another one, while the facial expression is being maintained. This way one can in theory impersonate other people. Style transfer on the other hand allows to draw an image with the style of another one, e.g. apply the impressionist style onto other images. Because of high resolution of generated images and very good quality, they can be indistinguishable from real images, which makes headlines and surfaces to the public. Although these models have high scientific value, they may be misused to fabricate believable fake information, which may pose considerable societal and legal challenges [RH21].

The aim of this work is to provide a comparative study of three generative models: Variational Autoencoder (VAE), Generative Adversarial Network (GAN) and Diffusion Models (DM). There are other models such as flow-based [ZCYS21] or energy-based models [DM20], however these will not be considered. Each of the considered model has many variants and other improvements, however for each model a unimodal variant is chosen (e.g. models with a text prompt are not considered). This work points out the individual strengths and weaknesses of each model and compares them to the other ones. The goal is not to provide a definitive ranking indicating which one of them is the best, but to qualitatively and where possible quantitively decide which models is good with respect to a given criterion, which will be described later on.

One of the first generative models introduced were the variational autoencoders. Figure 1.1 shows the results of a VAE trained on a dataset with handwritten digits. Although the results are blurry and have a low resolution, it was at that time a novel technique to generate image data [KW22].

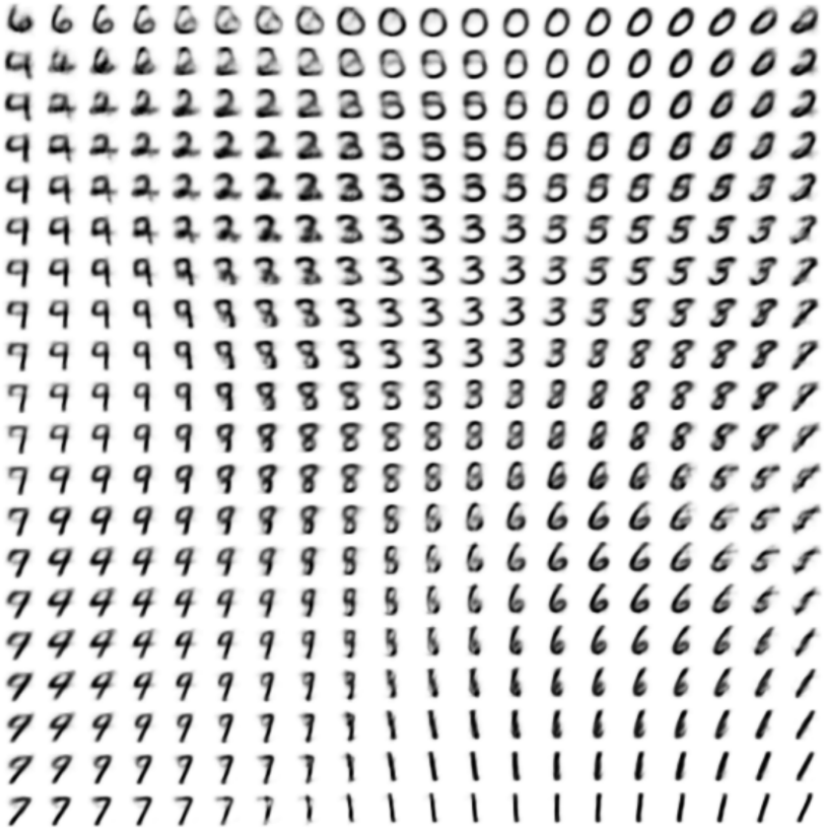


Figure 1.1: Manifold of handwritten digits generated by a VAE [KW22]

Generative models have gained large popularity when GANs [GPAM+14] were first introduced in 2014. Because of their promising results, they gained large popularity in the research community. Since then, they have been improved leading to the generation of realistic high-resolution images. Figure 1.2 shows how GANs have improved over time, generating more realistic images of better resolutions.

Todays GANs are able to perform more tasks than just the generation of images. The StyleGAN [KLA19] introduced by Nvidia is able to mix the features or styles of different images together. Figure 1.3 shows how the style from source B can be applied to an image from Source A.

However, recently a different kind of generative model was introduced, namely the Diffusion Models [HJA20]. Just like the GAN, the best DMs are able to generate 1080×1080 pictures. Additionally, DMs do not suffer from inherent problems (mainly Mode Collapse, i.e. the generation of one or a few images very similar to each other and not covering other possible images) resulting from the architecture as the GANs do, which makes them superior in this regard. Furthermore, DMs such as Dall-E [RDN+22] or Midjour-

Figure 1.2: History of GAN generated images [HHAC+21]

Figure 1.3: Mixing of styles using StyleGAN [KLA19]

ney [Mid22] can generate their images from a textual description. Surprisingly, these models will generate realistic images adhering to the description, even if the provided text is absurd and the described person or object does not exist. A viral example for this is the image of the Pope wearing a white puffer jacket [Hua23]. Even though the pope was probably never seen wearing this kind of jacket the model was able to understand the given description and synthesize a realistic image of the pope which was never taken. This shows how far the model can generate meaningful and realistic, but artificial images.

Figure 1.4: Pope in a white puffer Jacket [Hua23]

1.1 Applications of Generative Models

Generative models have other applications other than simply generating random images. This section covers a few of them. For example, Photoshop started including in recent years neural filters and other tools for editing the image not on the basis of pixels or splines, but on the basis of semantic information, such as age, facial expression or hair colour [Pho].

Figure 1.5 shows such smart portrait. Photoshop allows the user to change features on the image such as happiness, surprise, anger, facial age, gaze, hair thickness, head direction or light direction. Having these filters, the user is able to change the appearance of the person on the image with ease without having to do it by hand. Photoshop includes other neural filters such as: skin smoothing, super zoom, colorization or style transfer [Pho].

Another powerful tool offered by photoshop is the generative fill. This allows the user to fill in holes in the image or even extend them to a larger format. This tool allows the user to easily remove unwanted objects from the image [Pho]. Figure 1.6 shows an example, where unwanted persons have been removed from the image. Traditionally the user would have to remove the object by hand and later paint by hand a part of the

Figure 1.5: Photoshops smart portrait before and after applying filter [Pho]

image. This can be done by the generative fill tool in a matter of seconds. Furthermore, the generative fill tool has a text prompt, where the user can type in what should be generated, which on the other hand allows object generation [Pho].

Figure 1.6: Generative Fill tool removing unwanted objects from the image [Pho]

Not only Photoshop is employing generative modelling in order to edit images in an easier fashion, but TikTok uses it among other tools, too. One prominent example for it is the "Bold Glamour" filter, which makes a given face look very attractive while preserving the realism and similarity to the source image. Figure 1.7 shows an example of how this filter performs on a given face. What's remarkable about this is, that all this

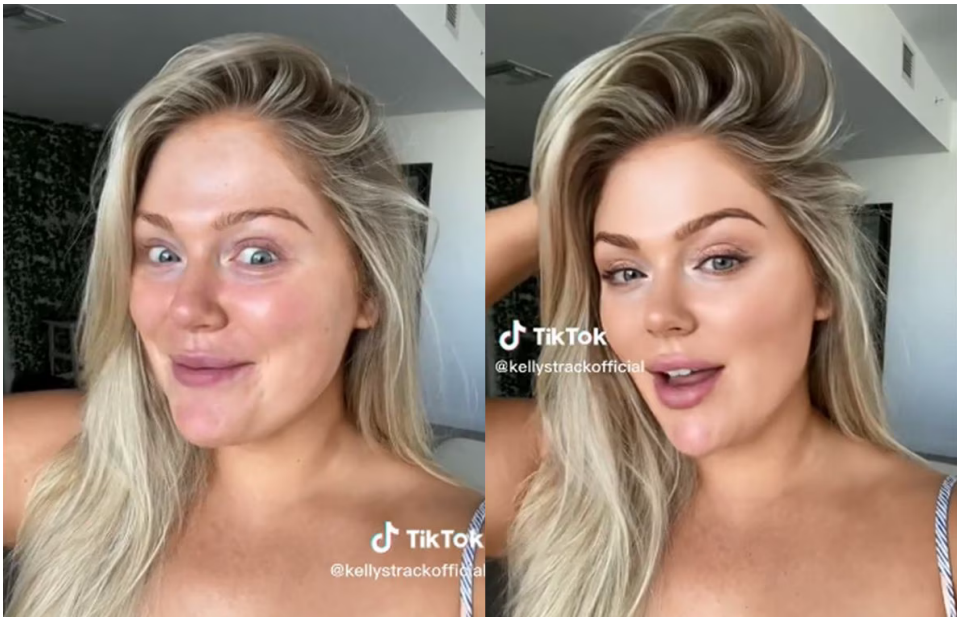


Figure 1.7: TikToks bold glamour filter with before and after [Zur]

is being done in real-time, so videos may be recorded using this filter [Zur]. Some argue, that such filters may be dangerous and harmful for the users. They may make the users feel worse about their real appearance and the usage may be linked to a desire to have plastic surgery [Bui].

Recently a generative AI model has been used by Marvel to create their first AI-generated intro for the series "Secret Invasion". Although it's not specified what kind of model it has been used, it is confirmed, that a "custom AI tool" has been utilized for this project. Figure 1.8 shows a frame from the intro. Marvel stated, that no artist jobs were replaced by this AI model, but it indicates, that they might be replaced in the future if the models get even better [Mil].

Although no company openly states which model, they are using, there are parallels between their results and the results in this work, which allows to make educated guesses, which model has been used. Most probably no company uses a plain textbook-model, but one that has been adapted to fit their concrete scenario. Nonetheless, it seems, that Photoshops smart portraits use some kind of a GAN as the results are comparable with the ones from section 6.8. Similarly, the generative fill might be done using a kind of DM, as the results and procedure is similar to the results presented in section 6.7, especially because photoshop gives different plausible results just as pointed out in section 6.7. Furthermore, the newest DMs are known for accepting a text prompt [RDN+22] [Mid22], which is another hint for them being used in the generative fill tool (Although it is possible for the GAN to have a text prompt, too [KZZ+23]). With regards to Marvels

Figure 1.8: A frame from the intro of Marvels "Secret Invasion" [Mil]

intro for "secret invasion" it has been stated, that text prompts have been used as well [Mil]. Furthermore, when watching the intro, it can be noted, that consecutive frames do not fit together well, which suggest a kind of DM. This can be somewhat seen in the results of interpolating with the DM presented in section 6.6. Lastly, with regards to the bold glamour filter it is hard to pin point a specific model, and TikTok does not provide any additional information on the used models.

1.2 Structure of this Work

The aim of this work is to provide a comparative study of three Deep Generative models, namely the Variational Autoencoder, Generative Adversarial Network and Diffusion Models. This work is structured in the following way: after this introduction, neural networks are explained briefly, which includes an introduction to unsupervised learning and generative modelling. Here the basic structure of neural networks and their training procedure is explained. After this general introduction the three models that are being taken into consideration in this work are explained in detail and how they work internally. Then, the models are compared with respect to different criteria: realism, generalization and diversity, sampling, training difficulty, parameter efficiency, interpolating and inpainting capabilities, semantic editing as well as implementation difficulty. Finally, this work ends with a conclusion.

2 Neural Networks

A neural network is basically a function with parameters θ. Depending on the parameters the shape of the function is different. The task that neural networks are solving is then to find a set of parameters θ that approximate an unknown function best, given only a finite number of points, that lie on the function or are close to it. This is accomplished through a training or learning process.

This kind of function can be for example used to classify images. The input for the neural network would be an image and the output a vector containing a probability distribution which indicates for each class the probability of the image belonging to that class. The training process tries then to find a set of parameters, such that the neural network classifies the given dataset best and generalizes beyond the given dataset. This means that the network should be able to classify images, that semantically belong to the dataset, but the network has not actually been trained on them.

This kind of tasks are called classification problems. Besides classification tasks there are regression tasks, where the network does not predict a probability distribution, but a value, such as price of a house or the outside temperature. These kinds of problems belong to the field of supervised learning, because the training data includes the input and the output for a given row of data and the network is being "supervised" by the given labels and trained to mimic them. On the other hand, unsupervised learning contains only data without any labels. The problems that belong to the category of unsupervised learning are for example clustering, dimensionality reduction or generation of similar datapoints. Lastly, there is reinforcement learning, where an agent is placed into a world and can take actions, for which it receives rewards. The task of the agent is then to find for each state of the world the action, that leads to the highest reward. One of such algorithms is Q-Learning.

Although neural networks by design need an input and a desired output (supervised learning) in order to be able to be trained, there are ways to have them solve unsupervised or reinforcement learning problems. Different ways to perform generative modelling being an example of unsupervised learning will be presented in the following chapters. This chapter briefly introduces neural networks and describes how they can be trained. If not otherwise specified, it is mainly based on different books on Deep Learning [Gé19] [Elg19] [GBC16] [Cho18] [Mur22] [Mur23].

2.1 Artificial Neurons

The smallest unit of an artificial neural network is an artificial neuron. It consists of multiple inputs and has one output. For each input value the neuron has a weight, which together form a part of the set of learnable parameters θ. For an input $\mathbf{x}$ the neuron calculates a weighted sum of the input and additionally adds a bias value b. The result is passed to a so-called activation function, which adds non-linearity into the neural network.

$$y = \sigma(\mathbf{w}\mathbf{x} + b) \tag{2.1}$$

Equation (2.1) shows the calculation performed by a single neuron. The weighted sum is written here as the dot-product between the input vector $\mathbf{x}$ and the weight vector $\mathbf{w}$. There are different activation functions $\sigma(x)$, which can be applied. Common activation functions include: Rectified Linear Unit (ReLU), the sigmoid function or the hyperbolic tangent.

Neurons are grouped into layers, where each neuron receives the whole input vector. Each neuron calculates one value, which are then put together into an output vector.

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.2}$$

Such layer of neurons can be expressed in terms of a matrix multiplication as shown in Equation (2.2). The weights of the individual neurons are stored in a weight matrix $\mathbf{W}$, where each row corresponds to a neuron. Consequently, the bias value $\mathbf{b}$ is here a vector which contains the biases for each neuron. The activation function is then applied element-wise. This type of layer is called a Fully-Connected layer (or Dense in TensorFlow or Linear in PyTorch), because there is a connection from every input to every neuron. Consequently, the number of learnable parameters in such a layer is the number of input dimensions multiplied by the number of output dimensions (plus the number of biases). Multiple of such layers can be concatenated together to form a deep neural network. Because the input vector is being passed and transformed from layer to layer, this procedure is being called the forward-pass.

Figure 2.1 shows an example of an neural network with three layers. The first input layer does not actually perform any calculations, but is necessary to specify in order to know how many connections have to be made (i.e. number of columns in the weight matrix of the next layer). The second layer and any other layer, that is not the first and last, is called the hidden layer. Here the calculation in equation (2.2) is performed. The last layer - the output layer - does the same calculation and outputs the calculated value. The number of layers and neurons in each layer as well as the activation functions can be varied as needed depending on the problem, that is being solved by the neural network.

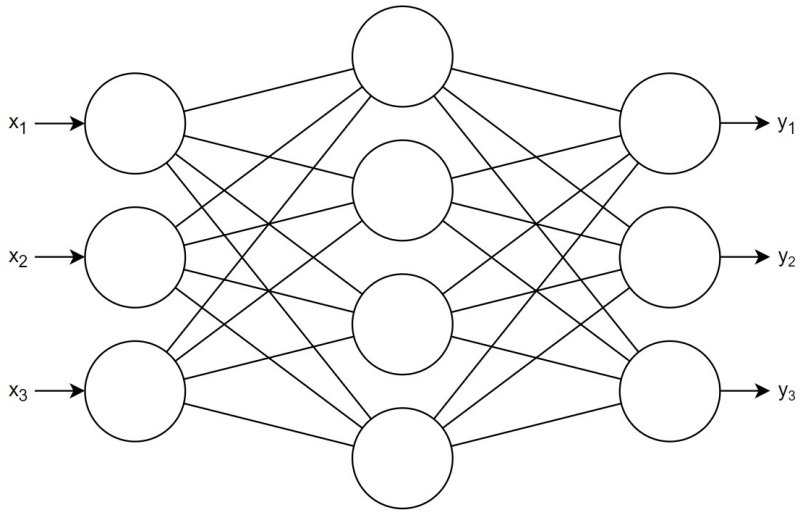


Figure 2.1: Example of a neural network with three layers

2.2 Loss Function

In order for the network to be able to learn, a loss function has to be defined. It serves as a performance measure, such that the learnable parameters of the network can be adapted to minimize the loss function. A common loss function for classification tasks is the cross-entropy:

$$E(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{C} y_i^{(n)} \log(\hat{y}_i^{(n)}) \tag{2.3}$$

Cross-entropy compares two probability distributions of true classes $\mathbf{y}$ and predicted classes $\hat{\mathbf{y}}$ (C refers to the number of classes and $y_i^{(n)}$ is the value from i-th dimension from the n-th sample). It returns a small value, if the distributions are close to each other and a large value if the distributions differ from each other. Additionally, the loss function is typically calculated for a minibatch of N samples, which means that for each sample the cross-entropy is calculated and the final result is the mean value of all cross-entropies in the minibatch.

For regression tasks a common loss function is the mean squared error:

$$E(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{n=1}^{N} \left(\mathbf{y}^{(n)} - \hat{\mathbf{y}}^{(n)}\right)^2 \tag{2.4}$$

Here the function compares the predicted values with the true values as well (Here the square refers to the dot product). These functions show why neural networks by default learn supervised problems. There has to be a label, such that the output of the network is compared against the label. Without knowing the label for a given input, a loss function could not be calculated and, consequently, the network parameters, e.g., the weights, could not be optimized.

2.3 Gradient Descent

With the loss function from the previous section, the learnable parameters of the network can be optimized, to minimize the loss. The process of finding a set values for each learnable parameter such that the loss is minimal, refers to the learning process. In single variable calculus, local minima can be found by setting the first derivative to 0 and determining the exact solution, however in the case of a neural network this is not possible due to the complexity of the neural network. That's why the optimization is being done iteratively with gradient descent. The idea is to initialize each parameter (e.g., a weight) to a random value, calculate the derivative of the loss function with respect to a given parameter at its position and move the value of the parameter in the direction of the steepest descent. This is done for every learnable parameter.

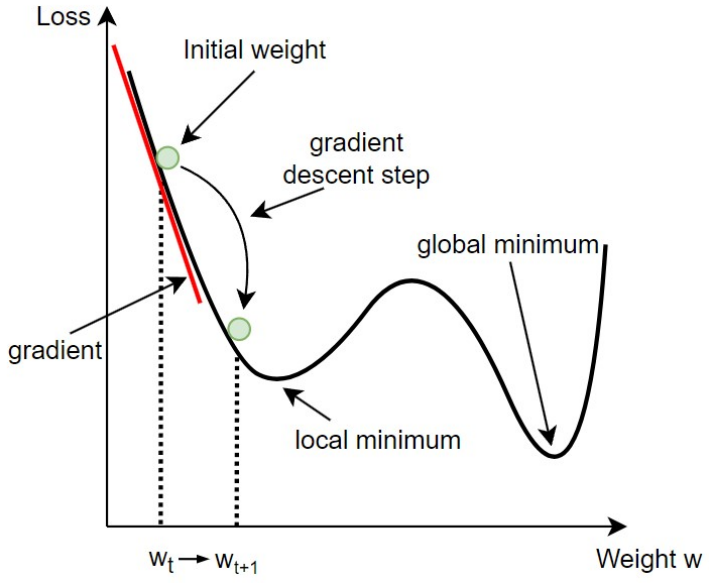


Figure 2.2: Gradient descent on a one dimensional function (inspired by [Elg19])

Figure 2.2 shows how gradient descent works in one dimension. The w-value at timestep t is being moved according to the gradient at its position in the direction such that the

loss is being minimized.

This works in the same way for neural networks. On each gradient descent step the derivative of the loss function with regards to each learnable parameter is calculated and all the parameters are moved in the direction, where the loss becomes smaller. The calculation of the gradients is being done by an algorithm called backpropagation, which applies repeatedly the chain rule of derivation to find each gradient. Because calculating the gradients using the chain rule starts at the back of the network and moves backward, this procedure together with gradient descent is called the backward-pass. As it can be seen in figure 2.2 in general, gradient descent cannot guarantee finding the global minimum. However, in practice the obtained results represent usually good suboptimal solutions.

As already hinted in section 2.2 the backward pass is not done for each row of data individually, but in a minibatch-manner. This means a batch (i.e. a set of samples) is classified by the network, then the average loss is calculated and the weights are changed such that the average loss is minimized. In this way the training procedure is more stable, because if the network would have been optimized without minibatches, the weights would move in the direction that is best for each particular row of data. This could lead to oscillations in the change of the parameters and thus, to an unstable training process. Such behaviour can be reduced when using minibatches.

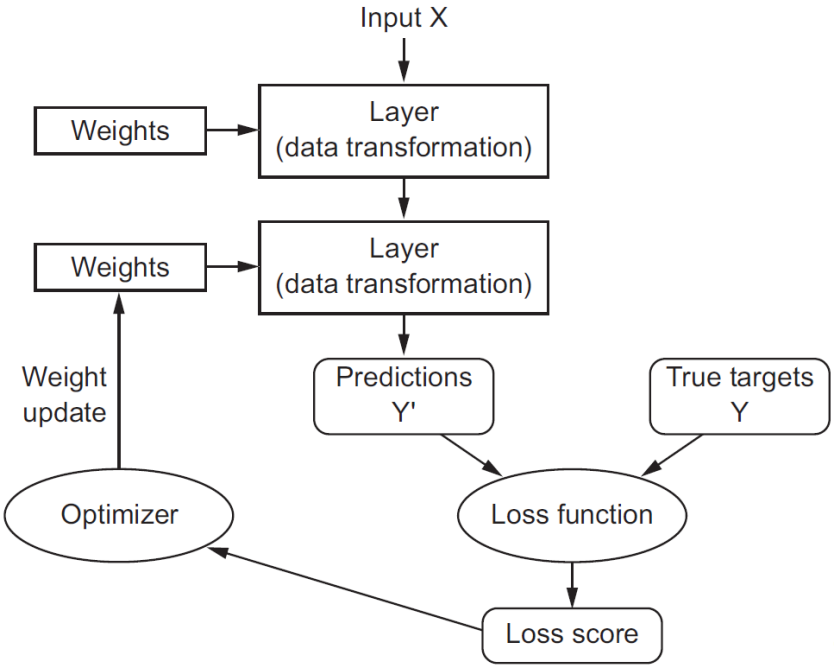


Figure 2.3: Diagram of the whole learning process [Cho18]

Figure 2.3 visualizes the whole learning process. The goal is to find right values for the

weights in order to minimize the loss function. For this, an input X is first transformed to a prediction Y' using a neural network (Here visualized as two layers). The prediction is then compared to the true target Y using a loss function. Based on the loss score the optimizer adjusts the weights (which have been initialized randomly in the beginning), such that it will lower the loss score. This is then repeated many times until the loss score does not decrease anymore.

2.4 Convolutional Layer

Especially for neural networks that deal with image data, there is another kind of layer, which is more efficient with regards to the number of learnable parameters. The idea is to not connect every input to every neuron, but to have sliding filters, that capture specific features from the image. A filter is a tensor of values whose width and height are set as hyperparameters and the depth depends on the number of channels (e.g. 3 as in the case of an RGB image) from the previous layer. If the filter size is set to 5×5 and the input to this convolutional layer has 3 channels, then the filter-kernel has the size of $5 \times 5 \times 3$. This filter slides then across the whole image and at each step the pixel value is multiplied with the value from the filter at the corresponding spot. These products are then added together. For each filter position a value is calculated and together they form a feature map which serves as the output of this convolutional layer. Such feature maps contain information about the presence of a particular feature (or pattern) captured by this filter at any given position in the image or among other feature maps. A convolutional layer may have multiple filters, which can capture various features. Each feature map is saved in a channel of the result such that, the number of output-channels is equal to the number of filters in this layer. Depending on if the input is being padded and if stride is used, the width and height of the output may stay the same or be smaller. Additionally, to each weighted sum a bias value may be added and an activation function may be applied.

Figure 2.4 shows a convolutional layer with two filters. Because the input has four channels (four feature maps) the depth of the filter kernel is four as well and because there are two filters, two feature maps are being outputted. The filter slides along the width and height of the input and for every position calculates a weighted sum as described before. This value is then written onto the corresponding position in the output feature map.

The values in each filter are learnable parameters which are being optimized during the training process using the same procedure as before: backpropagation and gradient descent. This way the network learns on its own which particular features are important to capture. Multiple convolutional layers may be stacked on top of each other. This way features are first extracted from the input image resulting in multiple feature maps from

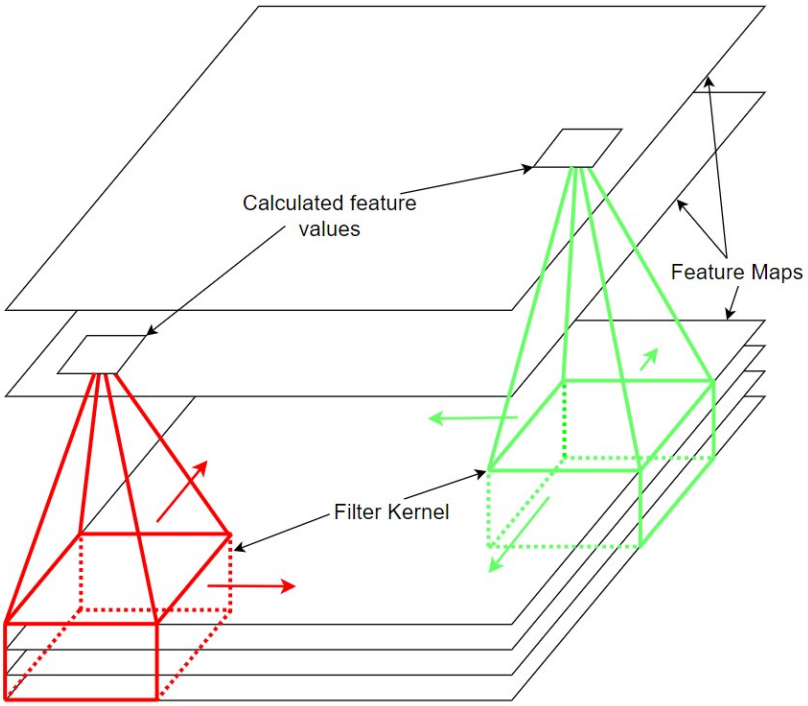


Figure 2.4: A convolutional layer with two filters and four input feature maps (feature maps at the bottom are the inputs while feature maps at the top represent the output of this convolutional layer)

which the next layer extracts more abstract features, which are then passed to the next layer. For example, the first layer might capture lines or curves, the next one the heads or legs and the final layer cats or dogs.

This way the convolutional layers are able to extract semantic information from pictures without the need for fully connected layers. Usually, in image classification tasks, features are extracted using multiple convolutional layers. Fully connected layers are then put after the convolutional layers. The task of these layers is to finally classify the provided input data.

2.5 Residual Connections

A neural network may include so-called “skip”-connections [HZRS15]. This means, that the result of a layer is directed not only to the following layer, but to a layer further in the network as well. These are formally known as residual connections, hence the name Residual Network (ResNet). Residual connections were introduced to facilitate the training of deep models, as deeper models showed to be more difficult to optimize

and produced worse results than their shallow counterparts. In theory deeper models should not produce higher error, as the additional layers could be replaced by an identity mapping, what would make the deeper model equivalent to the shallower one. This indicates, that deeper models should be at least as good as the shallow ones. Because the experiments show otherwise, it leads to the conclusion, that deeper models are more challenging to optimize. On the other hand, layers with residual connections have more flexibility and can be optimized to take a combination of values, from skip connections and from previous layer, into account [HZRS15]. Furthermore, ResNet-Architectures can be hundreds or even thousands of layers deep, as the network no longer suffers from the vanishing gradient problem, because the gradient can backpropagate freely using the skip connections. Vanishing gradient is a problem, where the gradient values arriving at early layers are very small leading to slower training [Fos19].

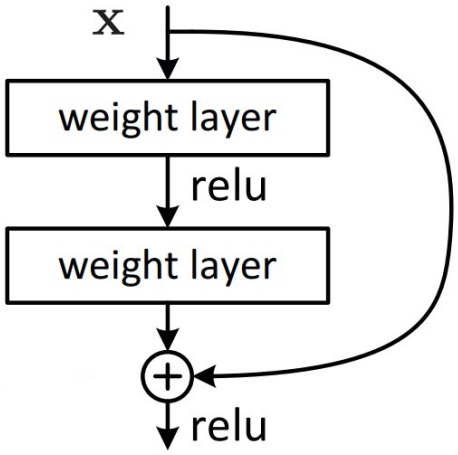


Figure 2.5: Structure of a ResNet-Block used in this work [HZRS15]

Figure 2.5 shows how a residual connection might be realized in an example network ("weight layer" refers here to a fully connected layer). The $\mathbf{x}$ is not only the input to the upper layer, but is added to the output of the layers as well.

2.6 Generative Modelling

As mentioned before the introduced neural networks are primarily used in supervised learning and data classification (discriminative modeling), where the neural network maps a data point to a label. Generative modeling does the reverse: for a given label (e.g. text or random noise) it generates a data point (e.g. an image). The label and the output image form different vector spaces: The random noise that serves as the input for a generative model is called the latent space and samples from this space are called latent

vectors (not to confuse with input space which includes all values a given model accepts). Such latent vectors are intended to be compact, which means that they have a smaller dimensionality than the high-dimensional samples, where semantic information about the sample is being encoded. A generative model is then used to map latent vectors to valid samples. Ideally latent vectors should be continuous, in a sense, that latent vectors close to each other should result in similar images. Another desired property of latent spaces is disentanglement, which means that individual dimensions of the latent vector correspond to a semantic feature (e.g. ideally one dimension should be responsible for the amount of smile in an image). Images on the other hand are defined on a so-called pixel space, which is a high-dimensional space, that includes all possible images in a given resolution. Besides that, there are feature spaces, which contain feature vectors (or maps) of an image that are calculated using a given feature extractor (typically a convolutional neural network without the fully connected layers). In order for a neural network to produce an image, it has to be structured accordingly. Training neural networks to perform generative tasks requires then a more sophisticated arrangement of one or more networks and a different training procedure. The following chapters will introduce three such generative models and explain how they work on the inside.

2.7 Metrics for Generative Models

In order to be able to compare generative models with each other a set of metrics is necessary. While it is straight forward to define a set of metrics for classification problems, the case of generative models and especially when dealing with images is more difficult. This is the case, because the output of the models is not a probability distribution as in the case of classification models, but an image with semantic information. The best way of deciding which image has better quality is to have a human decide it. However, this cannot be automated and may yield different results when different persons are asked to evaluate the same set of images. For this reason, the metrics are based on a neural network called Inception [SLJ+14]. The task of the network is then to transform information contained in images from pixel-space into feature-space or into semantic information which can be further analysed. Having a reliable set of metrics for evaluation is crucial for further advancements in the field. If a metric falsely decides that a model is better that the other, than a better model would be being discarded.

This section presents the two commonly used metrics in the literature to evaluate generative models for images and explains how they work. These metrics treat the underlying model as a black box and compare them based on the generated images.

2.7.1 Inception Score

Having the inception network, which can extract semantic information from the image, the inception score (IS) can be defined. The goal of this metric is to capture the following aspects: [SGZ+16]

- Quality: The object presented in the image should be recognizable
- Diversity: The model should be able to generate different images

To satisfy these both of these aspects two distributions are introduced: $p(y|x)$ and $p(y)$. $p(y|x)$ describes the conditional distribution of classifications y from the Inception network given an image x. This distribution is expected to have low entropy, because low entropy here correlates to good quality of the images (Entropy describes here a measure for uncertainty). If an image is clearly recognizable, then the Inception network should not have a hard time recognizing it and the predicted label should be close to a vector with the probability for one class being high and all the other probabilities being low [SGZ+16].

$p(y)$ on the other hand describes the overall distribution of the labels y given by the inception network and is expected to have high entropy, as it correlates to high diversity among generated images. If each class from ImageNet is being represented by generated images, then the distribution of the labels is close to being uniform which results in high entropy [SGZ+16].

Because these two distributions are opposite to each other with regards to the entropy the inception score is defined by the KL-Divergence between them:

$$\mathrm{IS}(G) = \exp(\mathbb{E}_x \left[D_{KL}(p(y|x) \parallel p(y)) \right]) \tag{2.5}$$

If both $p(y|x)$ and $p(y)$ satisfy their requirements, then it is expected, that the divergence is high, which makes the IS high. This means, that the higher the IS, the better is the model in terms of realism and diversity. The exponentiation is added in order to make the values easier to compare [SGZ+16].

2.7.2 Fréchet Inception Distance

The main drawback of the Inception Score is that it doesn't compare the generated images to the training data. The fréchet inception distance (FID) [HRU+18] improves the metric with this regard. The idea behind FID is to take two sets of images: one consisting of the training data and the other consisting of generated images. The Inception network is used again, but not to fully classify the images, but to extract features.

These features come from a hidden layer inside the network, where the image is not fully classified, but where the vector contains features, that are used by the following layers for the classification. From these sets of features the mean vector and covariance matrix is calculated. The FID is then given by the Fréchet distance between these two distributions:

$$\mathrm{FID}(\mathbf{m}_d, \mathbf{C}_d, \mathbf{m}_s, \mathbf{C}_s) = ||\mathbf{m}_d - \mathbf{m}_s||_2^2 + Tr\left(\mathbf{C}_d + \mathbf{C}_s - 2\sqrt{\mathbf{C}_d\mathbf{C}_s}\right) \tag{2.6}$$

The subscripts indicate the set from which the mean $\mathbf{m}$ and covariance $\mathbf{C}$ come from, with d indicating the training dataset and s the set of synthetic images. The square root in the equation refers to a matrix square root and not element-wise square root. The $Tr(\mathbf{M})$ function refers to the trace of a matrix and calculates the sum of the elements on the main diagonal of the matrix. This way the FID-score measures the distance between the distributions for training images and synthetic images in feature space. Consequently, the better is the model [HRU+18].

3 Variational Autoencoder

Variational Autoencoders were first introduced in 2013 by Diederik P. Kingma and Max Welling [KW22]. The architecture and implementation presented in this work is based on their paper [KW22]. Because other more advanced architectures deviate significantly from this architecture, they are not considered in this work.

A VAE is a model, that is capable of generating data given a training dataset which represents the distribution, that should be learnt. In order to be able to generate images the idea is to have a neural network which transforms random noise to valid images. However, there is no mapping from random noise to image space so a neural network cannot be trained directly. Variational Autoencoder [KW22] tackles this problem and solves it in a clever way.

3.1 Autoencoder

Autoencoder [Cho18] is a type of neural network, which consists of an encoder and decoder network. Together they form an autoencoder, which has a bottleneck in the middle and is trained such that it reconstructs its input.

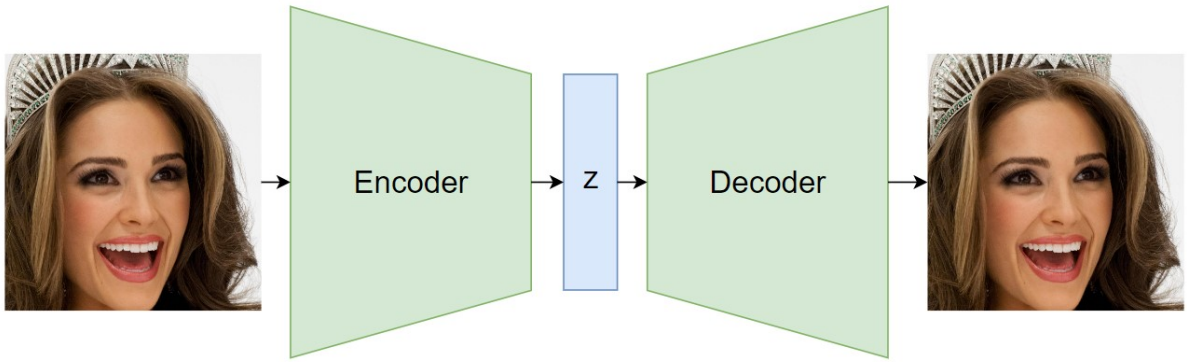


Figure 3.1: Autoencoder Network Architecture

The idea behind an autoencoder is, that the hidden state **z** in the middle of the network serves as a smaller representation of the training data. Because it is necessary smaller

than the input, and the decoder is trained to be able to reconstruct the original data, the hidden state represents a compressed version of the original image. This compressed version of the image may contain features, that correspond to semantic features of the image and represents the latent space of the autoencoder. The intuition for that is, that it is more efficient to encode the information, that someone is smiling, than encoding each pixel, that corresponds to a smile. To generate new images, one could sample a random latent vector $\mathbf{z}$ and pass it to the decoder, such that it produces a new image [Cho18].

However, this does not work well, as the hidden state $\mathbf{z}$ in the autoencoder is not regularized, which means that the values for the hidden state in the bottleneck do not have any bounds and may have values reaching from $-\infty$ to ∞. Furthermore, there might be gaps in the latent space which do not correlate to valid images. Because of this, one cannot know how to sample a latent vector, that leads to a valid image. This is why this kind of autoencoders is not suitable for generation. Instead, a probabilistic variant of the autoencoder, namely the variational autoencoder (VAE), can be used for generation. VAEs basically regularize the hidden state [Cho18].

3.2 Regularizing the Hidden State

In order to be able to regularize the hidden state, the encoder no longer directly maps the input image to the latent vector $\mathbf{z}$, but to a normal distribution, which is parametrized with mean and standard deviation values. Having these, a latent vector can be sampled from the distribution and decoded by the decoder network. This way the network can still be trained to reconstruct the input image and additionally the encoder can be trained in a way such that the predicted normal distribution is close to the standard normal distribution with $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\sigma} = \mathbf{I}$ ($\mathbf{I}$ represents the identity matrix). This way the hidden state is regularized and new latent vectors can be sampled from the standard normal distribution in order to generate valid images [KW22].

Figure 3.2 shows the architecture of a VAE. The encoder network predicts the distribution parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ corresponding to a particular image. Using the reparameterization trick a latent vector z can be sampled from this distribution and decoded by the decoder network. The reparameterization trick means a random $\boldsymbol{\epsilon}$ is sampled from the standard normal distribution and then reparametrized, such that it follows the distribution given by the encoder: [KW22]

$$\mathbf{z}(\mathbf{x}) = \boldsymbol{\mu}(\mathbf{x}) + \boldsymbol{\sigma}(\mathbf{x}) \odot \boldsymbol{\epsilon} \quad \text{with} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{3.1}$$

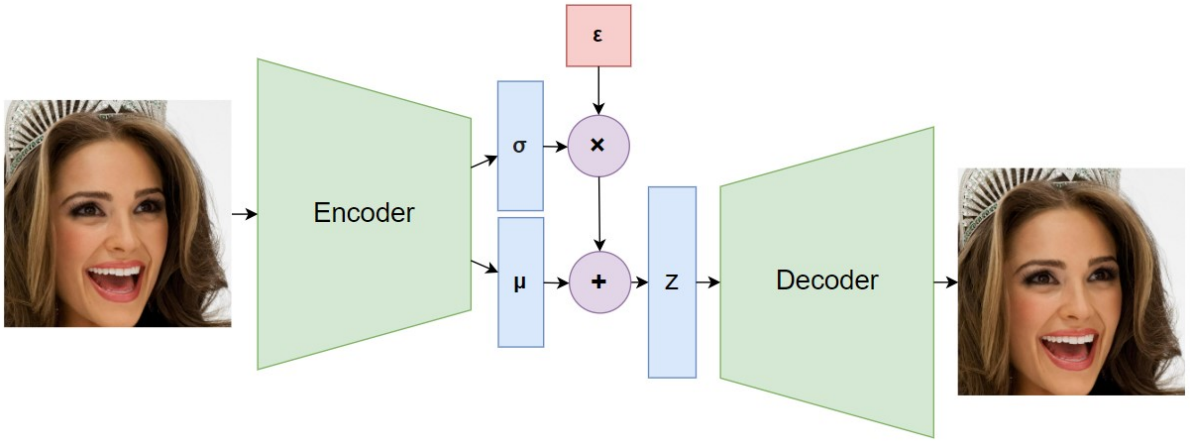


Figure 3.2: Variational Autoencoder Network Architecture

$\mathbf{z}$ represents the latent vector and $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the parameters given by the encoder. All of them depend on the input image $\mathbf{x}$. The symbol $\odot$ refers to the element-wise product [KW22].

3.3 Probabilistic Interpretation

The functionality and the loss function of the variational autoencoder can be derived using probability theory. The problem, that the Variational Autoencoder is trying to solve is the following: Given are N images from a dataset X and it is assumed, that they were generated using a distribution $p_{\theta^*}(\mathbf{x}|\mathbf{z})$. The generation process was to first sample a latent vector $\mathbf{z}$ from the distribution $p_{\theta^*}(\mathbf{z})$ and then generate the image using the distribution $p_{\theta^*}(\mathbf{x}|\mathbf{z})$. However, the true parameters θ^* and the latent variables are unknown. The problem is then to approximate the true parameters θ^* and to find a model, that can generate images given a latent vector $\mathbf{z}$ [KW22].

In order to be able to solve this problem besides the model for $p_{\theta^*}(\mathbf{x}|\mathbf{z})$ another model is introduced: a recognition model $q_\phi(\mathbf{z}|\mathbf{x})$ which serves the purpose to approximate the true $p_{\theta^*}(\mathbf{z}|\mathbf{x})$ and is refered to as a probabilistic encoder. Consequently the model for $p_{\theta^*}(\mathbf{x}|\mathbf{z})$ is being called the decoder [KW22].

3.4 Loss Function

The overall objective is to minimize the negative log likelihood:

$$-\log p_\theta(\mathbf{x}) \tag{3.2}$$

Likelihood describes a measure for $\mathbf{x}$ being sampled from the distribution p_θ. The higher the likelihood, the better p_θ represents the data distribution, hence the likelihood should be maximized. Because in machine learning the loss functions are typically minimized, the negative likelihood is minimized, such that the positive likelihood is maximized. Finally, the logarithm is added to the term for numerical stability and convenience. When using logarithms, products become sums and the individual values are not close to 0. Because the logarithm function is monotonically increasing, optimizing the log likelihood optimizes likelihood as well [KW22].

However, since optimizing the negative log likelihood on its own is intractable a tractable surrogate problem has to be found, such that when optimizing it the negative log likelihood is optimized as well. For this the likelihood in (3.2) can be rewritten as: [KW22]

$$\log p_\theta(\mathbf{x}) = D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}|\mathbf{x})) + L(\theta, \phi, \mathbf{x}) \tag{3.3}$$

In the equation (3.3) $D_{KL}(q \parallel p)$ in the first term describes the Kullback–Leibler divergence between two probability distributions p and q, which is a measure describing a distance between them. The second term $L(\theta, \phi, \mathbf{x})$ is called the variational lower bound. Because the KL-Divergence in equation (3.3) is always nonnegative and contains intractable terms, it can be dropped from the equation turning the equality into an inequality and the lower bound can be written as: [KW22]

$$\log p_\theta(\mathbf{x}) \geq L(\theta, \phi, \mathbf{x}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] \tag{3.4}$$

Maximizing the variational lower bound will maximize the likelihood, since the bound is always smaller than the likelihood. The term $p_\theta(\mathbf{z})$ describes the probability distribution of the latent space which is assumed to be normally distributed: [KW22]

$$p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{3.5}$$

This makes the calculation of the KL-Divergence easier, as the parameters for the distributions can be compared directly. The KL-Divergence in (3.4) can be consequently simplified to: [KW22]

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) = \frac{1}{2}\sum_{i=1}^{D}\left(1 + \log((\sigma_i)^2) - (\mu_i)^2 - (\sigma_i)^2\right) \tag{3.6}$$

The second term of the variational lower bound (3.4) is an expected negative reconstruction error and can be maximized when the probability of producing the same image, that was encoded by $q_\phi(\mathbf{z}|\mathbf{x})$ and decoded by $p_\theta(\mathbf{x}|\mathbf{z})$ is high. Having this, the second term in equation (3.4) can be written as cross-entropy, between the real image and the reconstructed image: [KW22]

$$\log p_\theta(\mathbf{x}|\mathbf{z}) = \sum_{i=1}^{D} (x_i \log y_i + (1 - x_i) \log(1 - y_i)) \tag{3.7}$$

Here y is the reconstruction by the decoder network given the latent vector $\mathbf{z}$ and x is the original image (here D refers to the number of all pixels). The sum of this term (3.7) and the KL-Divergence in (3.6) form together the loss function for the variational autoencoder [KW22].

3.5 Network Architecture

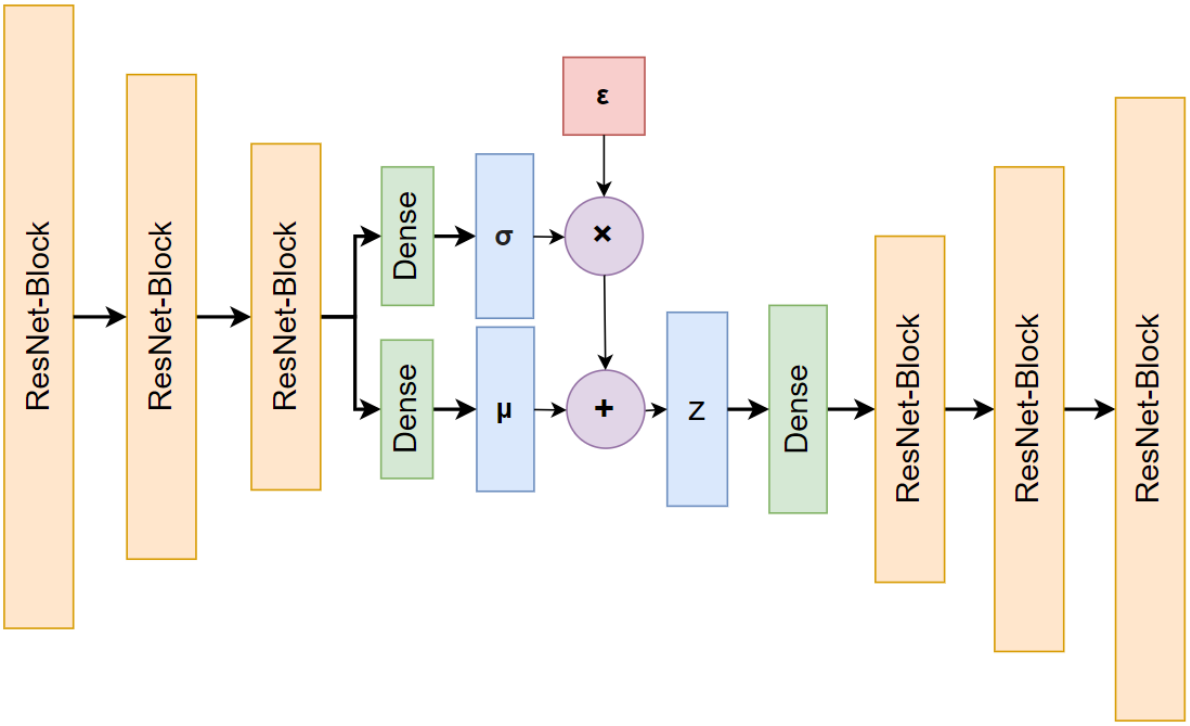


Figure 3.3: Variational Autoencoder Network Architecture (Some layers omitted, number of ResNet-blocks variable, refer to section 5.3.2 for grater detail on ResNet-blocks, ResNet-blocks for this VAE don't have additional inputs)

As the architecture of the VAE can be chosen freely the implemented and used architecture in the experiments of this work is inspired by the U-Net [RFB15], which includes elements that have probably not been used in [KW22], which might improve the performance. The used architecture can be seen in figure 3.3. The network is structured in ResNet-blocks, which contain two convolutional layers, with SiLU ($\text{SiLU}(x) = x\sigma(x)$ [EUD17]) as activation function and group normalization [WH18] together with a residual connection within each block (Structure similar to the block presented in figure 5.5). In order to convert the feature maps from the last encoder layer into $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ vectors, fully connected layers are being used. Similarly, to convert the latent vector back to the format of a feature map, a fully connected layer is used as well. This particular implementation uses six ResNet-blocks for the encoder and decoder each. After each block the resolution is halved/doubled in the encoder/decoder network respectively using interpolation for up-sampling and average pooling for down-sampling. The number of channels for each block grows with the depth and for the encoder is: 128, 128, 128, 256, 256, 512 (the decoder has the same number of channels but in reverse). Additionally, self-attention layers (refer to [VSP+17] for more details on self-attention) are used at resolutions 16 and 32. Finally images are being encoded into $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ vectors which have a dimensionality of 512.

3.6 Improved Sampling

New images in VAE are sampled (i.e. generated) by sampling first a latent vector from the standard normal distribution and then transforming it to an image through the decoder. This works fine, but upon inspecting random generated images and reconstructed training images, a significant difference with regards to e.g. the diversity, position of the head, hair colour or the background can be noted, which indicates, that generated the VAE is able to generate better images. Figure 3.4 shows a few examples of reconstructed and randomly generated images.

First it has to be noted, that the VAE aside from the reconstruction error was trained by the KL-Divergence in such a way that the encoded distribution follows the standard normal distribution. However, upon inspecting the parameters of distributions belonging to encoded images, it becomes clear, that they are only close to the standard normal distribution, but have a significant distance from it as well. Figure 3.5 shows for instance distribution of values taken from one dimension of the mean and standard deviation vectors, resulting by encoding multiple training images. This shows, that the encoder does not encode the images into the standard normal distribution. This makes sense, because if the VAE needs to reconstruct the image successfully, some information has to pass the bottleneck. If every image would be encoded into the standard normal distribution, all information about the image would be lost and the VAE would not be able

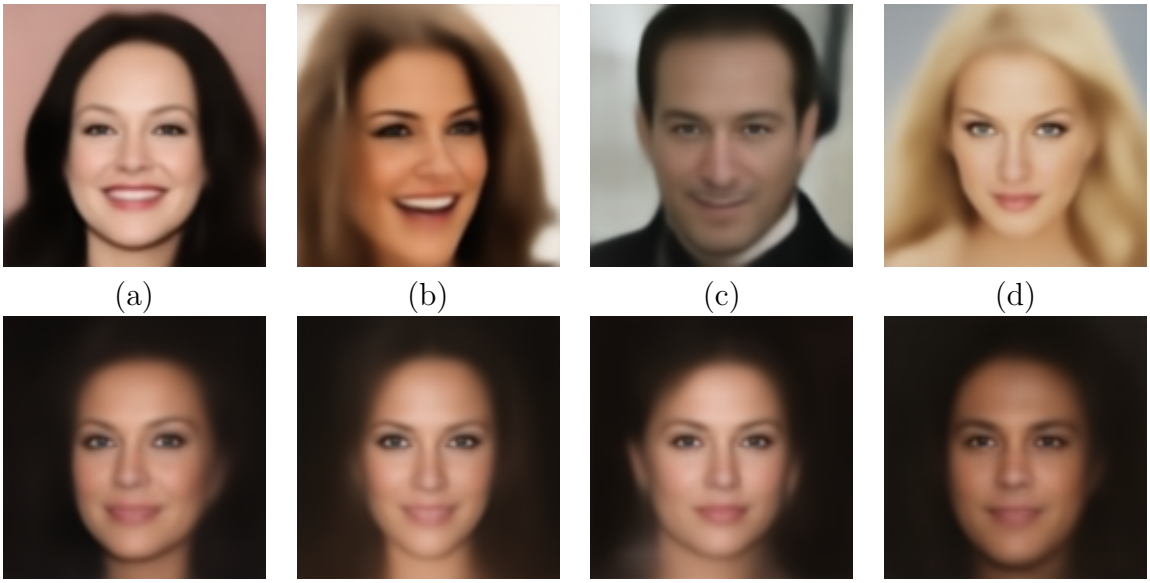


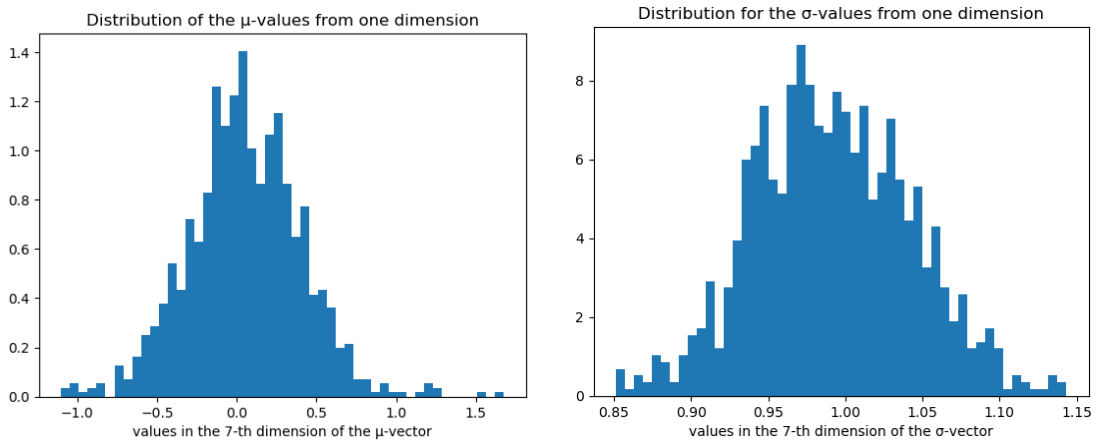
Figure 3.4: (a)-(d) show encoded and decoded training images. (e)-(h) show randomly sampled images from the standard normal distribution

to reconstruct any image.

Because there is a clear difference in randomly generated images and reconstructed images, that are not being encoded into the standard normal distribution, the quality of the generated images might improve, if the latent vectors are sampled from a different distribution.

In order to find such a distribution all training images are being encoded into mean and standard deviation values. Next, all the mean and standard deviation values are being modelled using two multivariate normal distributions: one for the mean values and one for the standard deviations. The improved sampling process is then to first sample a mean and a standard deviation from the newly created multivariate normal distributions and using those a latent vector can be sampled. Such sampled latent vectors do not form the standard normal distribution, but it seems, that they resemble better the domain of the decoder, which leads to improved samples (a few examples shown in 6.2).

It has to be noted, that by doing so the generated images might skew a little into the direction of the training images, because the latent vectors are sampled using information taken out of encoded images from the dataset. However, one might simply sample latent vectors from a uniform distribution that encompasses more values than the standard normal distribution or set the standard deviation of the standard normal distribution



(a) (b)

Figure 3.5: Distributions of values taken from the 7-th (any other dimension shows similar results) dimension of the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ vectors that result by encoding training images (a) Distribution of μ_7-values (b) Distribution of σ_7-values

higher so a larger area is covered by the sampled latent vectors. Both of these approaches are decoupled from the dataset, but include new hyperparameters, that have to be chosen. On the other hand, the procedure described above does not include any additional hyperparameters and because of the information gained from the encoding of images from the dataset, the sampled latent vectors might resemble better the domain of the decoder, which itself was trained on the dataset, too.

4 Generative Adversarial Networks

Generative Adversarial Networks were first introduced by Ian J. Goodfellow et al. in 2014 [GPAM+14] and sparked a large interest in the research community. Since then, there have been many improvements, some of which will be presented in this chapter. Specifically, this implementation is based on the version of GAN with progressive growing presented in [KALL18], because it contains enough improvements in order to sufficiently mitigate inherent problems resulting from the GAN architecture.

The idea behind GANs is to have two networks, that are competing with each other, hence the name "Adversarial" (see Figure 4.1). One network, the generator, generates images from random noise $\mathbf{z}$, that are labelled as fake, and the other, the discriminator, is trained to distinguish fake-images from real ones. The "competition" consists of the generator trying to fool the discriminator by generating realistic images, and the discriminator wanting to distinguish real images from fake images as good as possible. In this way, after the training has been completed, the generator can generate realistic images from random noise [GPAM+14].

4.1 Loss-Function and training

The generator network transforms latent vectors (vector $\mathbf{z}$ in Figure 4.1) to images and the discriminator network classifies the images as real or fake, where 0 means the image is fake and 1 means the image is real. The "competition" is implemented using the following value function:

$$V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{4.1}$$

The $\mathbf{x}$ represents the real samples and the $\mathbf{z}$ is the latent vector. $\mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})]$ is the expected value of $\log D(\mathbf{x})$ given that $\mathbf{x}$ is sampled from the training data distribution p_{data}. The generator $G(\mathbf{z})$ is trained to minimize the value function, while the discriminator $D(\mathbf{x})$ is trained to maximize it. The value function is set up in such a way, that it is minimized, when generated images are predicted to be real by the discriminator and maximized when the discriminator correctly classifies the images. The networks are not trained at once, but in an alternating manner. While the generator is being trained,

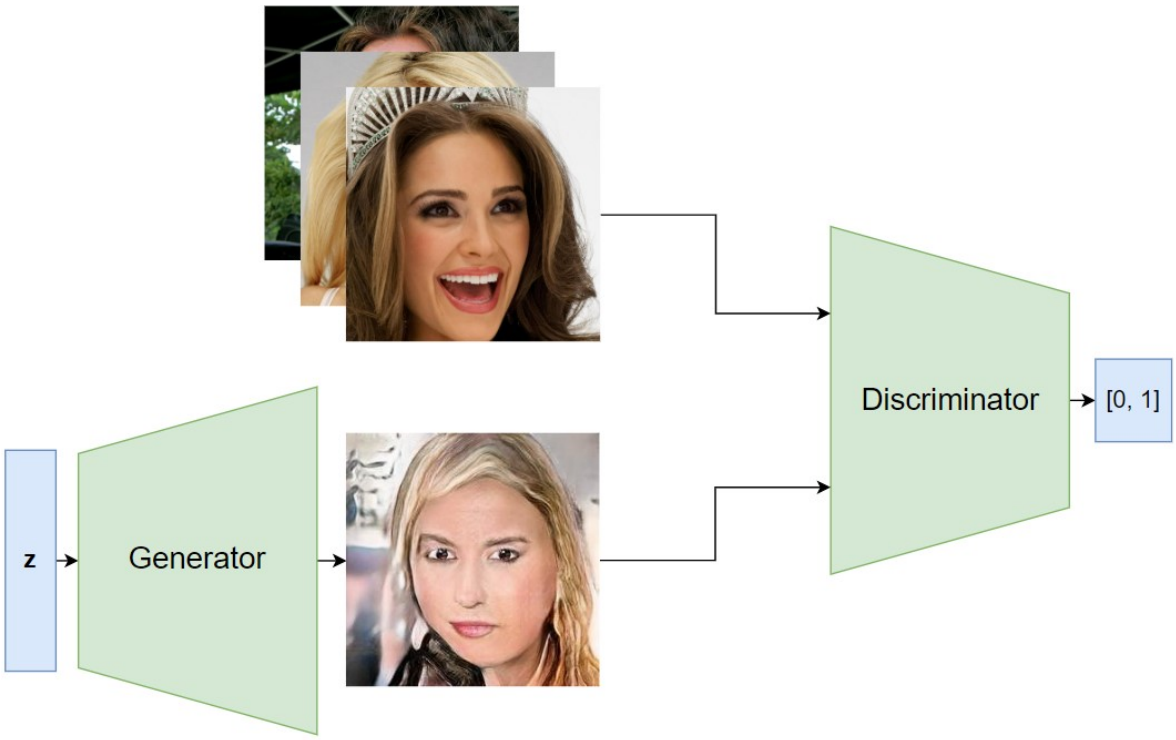


Figure 4.1: GAN Network Architecture

the parameters of the discriminator are frozen, and vice versa. When the discriminator is being trained it receives one batch of real images and one batch of fake images generated by the current version of the generator. Based on the classification the weights are optimized such that the value function is maximized, i.e. in a way, that images are being classified correctly. When the generator is being trained, first a batch of latent vectors is sampled and transformed into a batch of fake images. These images are then classified by the discriminator. In order to be able to perform gradient descent on the generator, backpropagation is executed starting with the discriminator and continues calculating the gradients for the generator, based on the incoming gradients from the discriminator. The weights of the generator are then optimized in a way, that minimizes the value function, i.e. in a way, that fake images are being classified as real by the discriminator. After both networks have been trained sufficiently the generator network should be able to generate random images from any latent vector [GPAM+14].

4.2 Mode Collapse and Training Instability

The main drawbacks of GANs are mode collapse and training instability. Mode Collapse refers to the problem of the generator generating in the worst case the same single image for all points in the latent space (i.e. for different vectors $\mathbf{z}$). Less severe mode collapse appears, when very similar images or only few different images are being generated. This issue occurs, because there is no term in the value function, that penalizes low variety among generated images. Furthermore, the generator optimizes in a way to generate one image, that fools the discriminator best and minimizes the value function. Even when the discriminator updates and becomes sensitive to this one image, the generator can adjust and change it accordingly, without the necessity to have a variety among the generated images [SC23].

The second problem is the training instability. In the case of classification tasks, the neural network has the objective to correctly classify given data. This data and the classifier itself do not change during the training. In the case of GANs, because the generator and discriminator are being trained in an alternating manner and each training step depends on the other network, the training objective changes with each iteration. In one iteration the generator might optimize in one direction and in the next iteration the direction might be different because the discriminator has changed. This applies to the discriminator equivalently. Furthermore, the loss value from the generator is uninformative, as it is calculated based on the current state of the discriminator. Because the discriminator is changing over time, the individual loss values from the generator in different timesteps cannot be compared to each other, which makes the training process hard to monitor. On top of that, the loss value doesn't correlate to the quality of the generated images [SC23].

4.3 Improvements

Because of the problems mentioned above various techniques have been introduced, to stabilize the training process and to mitigate the problem of mode collapse. In this section some of them are presented.

4.3.1 Wasserstein Loss

One way to improve the training stability is to replace the value function by the Wasserstein-Loss:

$$L(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[D(G(\mathbf{z}))] \tag{4.2}$$

With this loss function the discriminator no longer outputs a probability value between 0 and 1, but its output is unbounded. Some papers no longer call it a discriminator, but a critic, since it doesn't decide whether a given image is real or fake, but grades an image based on its authenticity. That is: the higher the value, the more real looks a given image. As with the previous loss function the discriminator is trained to maximize and the generator is trained to minimize the Wasserstein-Loss. With the new loss function the discriminator is encouraged to output high values for real images and low values for generated images. A GAN with this loss function is called the Wasserstein GAN (WGAN) [ACB17].

The advantage of using Wasserstein-loss over the traditional loss function is that it makes the training procedure more stable and provides reliable gradient information for the generator [ACB17].

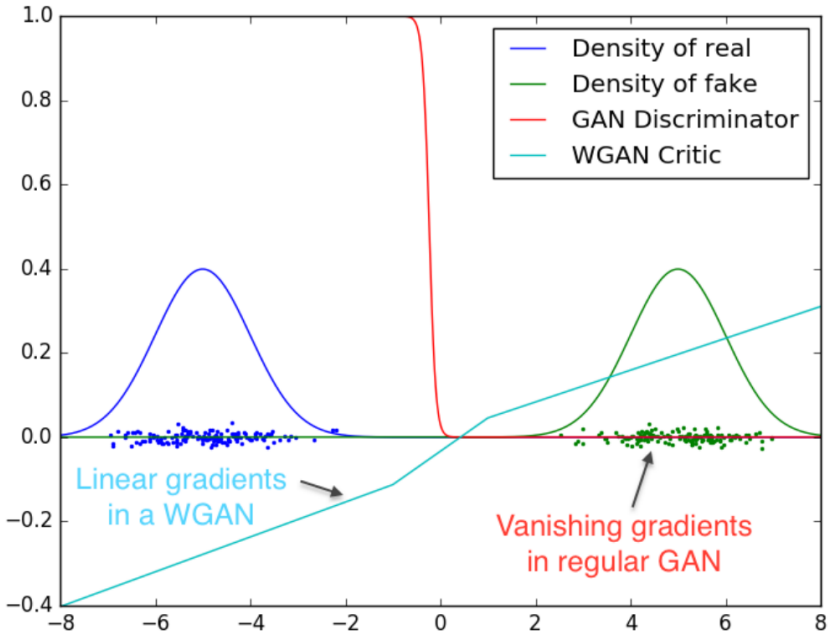


Figure 4.2: Comparison of gradients using a traditional GAN and a WGAN [ACB17]

Figure 4.2 visualizes how both loss functions behave in a case where a trained discriminator differentiates between two normal distributions. While the GAN discriminator correctly classifies samples from both distributions, the gradient is close to zero, which makes it difficult for the generator to learn from. On the other hand, the WGAN discriminator correctly grades the samples and provides linear gradients, which makes gradient descent easier for the generator. This way the training of the generator and the discriminator does not have to be perfectly balanced, because the gradients from the discriminator are always informative for the generator [ACB17].

However, for all this to work the discriminator needs to be a 1-Lipschitz continuous function. A function f is K-Lipschitz continuous, when the following is satisfied:

$$\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n : \frac{\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\|}{\|\mathbf{x}_1 - \mathbf{x}_2\|} \leq K \quad f : \mathbb{R}^n \to \mathbb{R}^m \tag{4.3}$$

It can be shown, that the discriminator satisfies this property, when its weights are clipped after each gradient descent step to a small range for example $[-0.01, 0.01]$ [ACB17].

4.3.2 Gradient Penalty

Weight clipping comes with major problems such as capacity underuse or exploding/ vanishing gradients. Because the weights are clipped to a small range the network cannot make full use of the weights, which in turn limits its capacity to learn complex functions. Furthermore, it can be observed, that the gradients explode or vanish depending on the choice of the clipping threshold c [GAA+17].

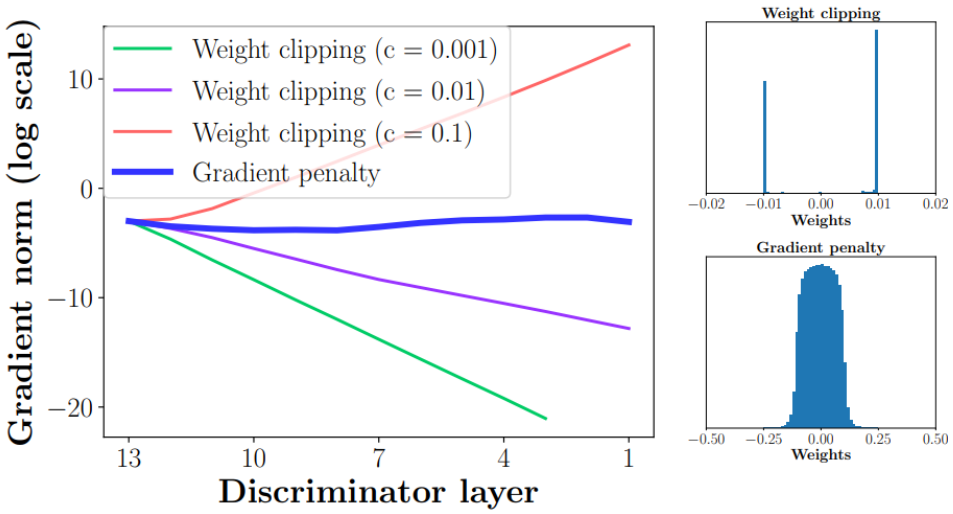


Figure 4.3: Gradient norms and weights with weight clipping and gradient penalty [GAA+17]

Figure 4.3 shows how the gradient norm (Euclidean norm of calculated gradients) behaves depending on the clipping threshold depending on the layer in the discriminator. Exploding or vanishing gradient is a phenomenon, that occurs in deep neural networks while executing the backpropagation algorithm. Because calculating the gradient with respect to each weight includes matrix multiplications for each layer, with many layers and certain numeric values for the weights the gradient can either rise rapidly or vanish

close to zero. This happens due to the fact, that multiplying multiple numbers with values between 0 and 1 makes the product small and multiplying values greater than 1 makes the product large. Besides having vanishing or exploding gradient with weight clipping, the individual weights do not distribute among the possible values, but converge towards the clipping threshold, which can be seen on the righthand side of figure 4.3 [GAA+17].

This is why "weight clipping is a clearly terrible way to enforce a Lipschitz constraint" [GAA+17] and why a gradient penalty term was added to the Wasserstein loss:

$$L(D,G) = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_z}[D(G(\mathbf{z}))] + \lambda \mathbb{E}_{\hat{\mathbf{x}}\sim p(\hat{\mathbf{x}})}[(\parallel \nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}}) \parallel_2 -1)^2] \tag{4.4}$$

The first two terms here represent the Wasserstein Loss and the third one calculates additionally the gradient penalty. The λ represents the penalty coefficient and is basically a scaling factor (authors use $\lambda = 10$[GAA+17]). The main idea behind gradient penalty is to penalize the network for having a gradient whose norm is not equal to 1. This is done by taking the mean squared error between the gradient from the discriminator and 1. This quantity is then added to the loss function such that it will be minimized in each run. This way the gradient norm will approach 1 which enforces the discriminator to be a 1-Lipschitz continuous function. The gradient is taken with respect to $\hat{\mathbf{x}}$ which is a randomly interpolated image on a line between a real image and a generated image in pixel-space: [GAA+17]

$$\hat{\mathbf{x}} = \epsilon \mathbf{x} + (1-\epsilon) G(\mathbf{z}) \quad \text{with} \quad \epsilon \sim \mathcal{U}(0,1) \tag{4.5}$$

[GAA+17] explains why the gradient is taken with respect to the combination of real and generated images. Figure 4.3 shows the improvement of gradient penalty over weight clipping with respect to the gradient norm and weight distribution. [GAA+17]

4.3.3 Progressive Growing

Another improvement to GANs is progressive growing [KALL18]. The idea here is to train the networks in a step-by-step manner instead of training the whole networks at once. Here the networks layers are structured into blocks. Initially, only the first block of the generator and discriminator networks are trained on 4×4 images. Then the next block is introduced and both networks are trained on 8×8 images. This procedure is continued until the desired resolution is reached. Figure 4.4 visualizes the training process [KALL18].

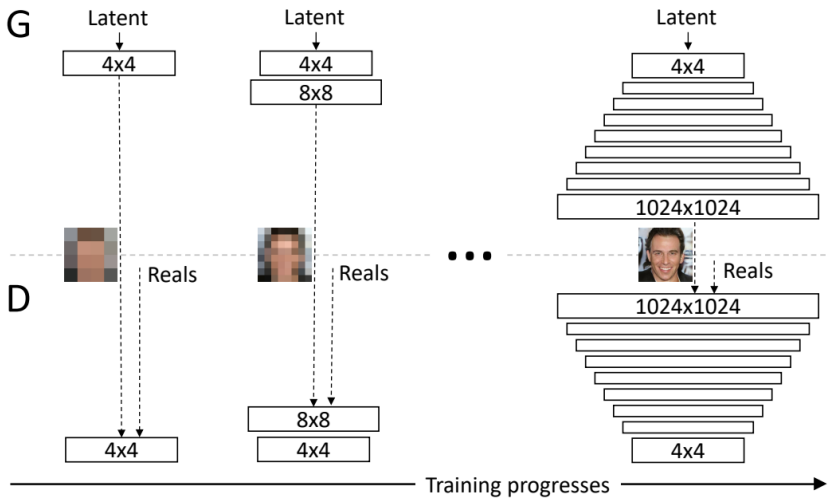


Figure 4.4: progressive growing of the generator and discriminator during the training

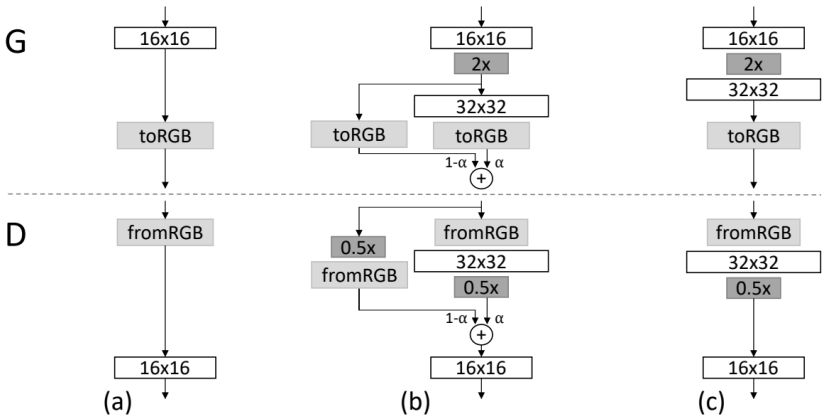


Figure 4.5: Fading in of new blocks into the generator and discriminator

Each new layer is not placed right into the network, but smoothly faded in. Figure 4.5 shows how each new layer is being faded in [KALL18].

On the side of the generator the idea is to combine the output from old resolution and the new resolution, such that in the beginning the final output consists only of the old resolution and with time the output shifts to the new resolution, what is controlled by α. As α grows with each gradient descent step the output shifts to the image that comes from the newly inserted block with its "toRGB"-layer. This procedure is the same for the discriminator but flipped. For each stage the training is done in two phases: first a new block is faded in during a number of epochs and then the network is being stabilized during the next epochs after which the next stage is introduced. While a stage is being introduced the previous layers of the network are not being frozen, but continue being

trained [KALL18].

This way the network can first focus on the large-scale structure of the data distribution and then try to reproduce finer details in the dataset. Furthermore, the network does not have to learn to produce high resolution images from nothing, but is gradually trained and asked simpler questions, what makes the training more stable. Another benefit when using progressive growing is the reduced training time. Because training on smaller image resolutions and fewer layers is much faster, the early stages can be pretrained and later on when the network is asked to produce high-resolution images, it does not have to start from the beginning, but has the most of the network pretrained and does not require as many time-consuming iterations as it would be the case without progressive growing [KALL18].

4.3.4 Minibatch Standard Deviation

Minibatch Standard Deviation [KALL18] is a way to force the generator to produce various images and not to fall into mode collapse. The idea is to calculate a statistic across the minibatch at one point in the network and to append it to each element in the minibatch. Concretely, for each element in a minibatch the standard deviation over its features in any spatial location is calculated. Then a mean value is calculated from these standard deviations. The result is then replicated to form an additional feature-map, which is then added to each element from the minibatch [KALL18] Figure 4.6 visualizes the performed calculation.

This way the discriminator can use this information to decide if a batch of images is real or generated. Because this statistic is calculated over real images as well, the generator is forced to generate images, which show similar statistics as real images, which in turn makes the generator produce various images [KALL18].

4.4 Network Architecture

The generator and discriminator networks are structured into blocks, that are being introduced at each stage of the training procedure according to the progressive growth. The network architecture used in the experiments for this work is inspired by the official implementation found in [KALL18]. Here each block consists of two convolutional layers each with leaky ReLU as activation function and with pixel normalization afterwards. Leaky ReLU works the same way as the usual ReLU, but has a slight slope on the negative side:

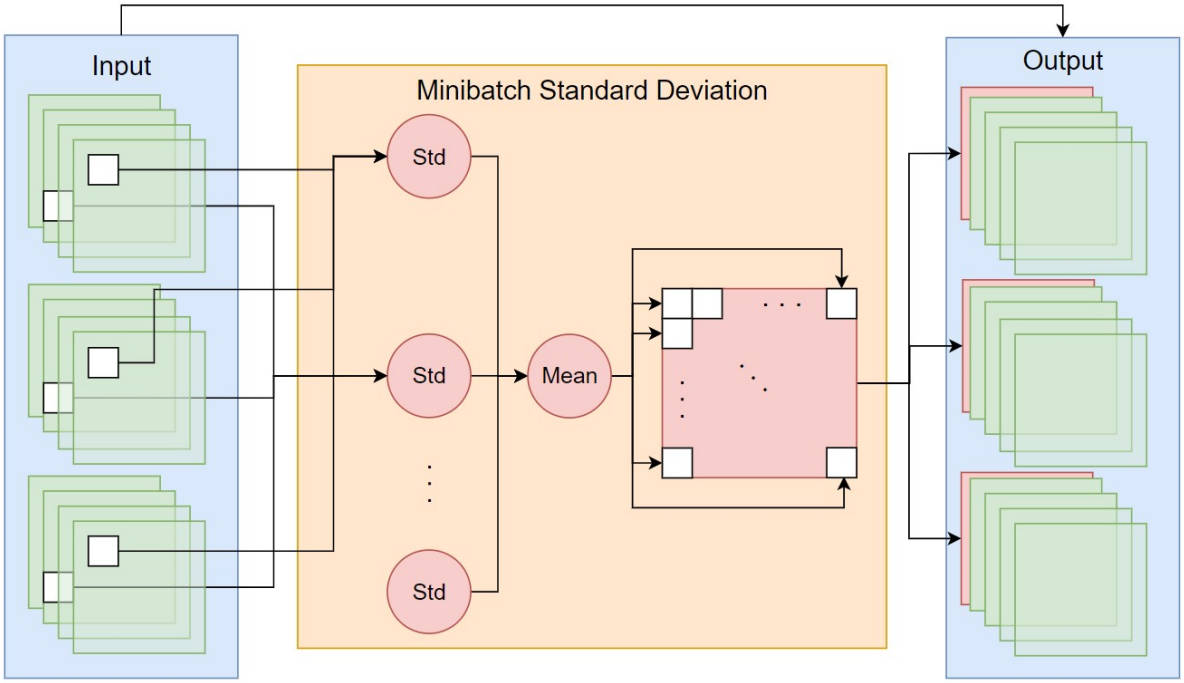


Figure 4.6: Minibatch Standard Deviation

$$LReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{4.6}$$

Where α is a small value. In this case $\alpha = 0.2$ is used. With this activation function the gradient is not set to zero, as in the case of the ReLU function, which has a zero derivative on the negative side, but has a small value. Pixel normalization [KALL18] is a layer which normalizes each feature vector across all feature maps:

$$\mathbf{x}'_{a,b} = \frac{\mathbf{x}_{a,b}}{\sqrt{\frac{1}{N}\sum_{j=0}^{N-1}\left(\mathbf{x}^{j}_{a,b}\right)^2 + \epsilon}} \quad \text{with} \quad \epsilon = 10^{-8} \tag{4.7}$$

The values a, b denote the pixel position in a given feature map and j denotes the index of the feature map. The idea behind pixel normalization is to prevent the magnitudes of the feature vectors (all feature-values corresponding to one pixel-position in given feature-maps) from spiralling out of control due to the competition between the generator and the discriminator [KALL18].

After each block the resolution of the image is being doubled using bilinear interpolation in the case of the generator and halved using average pooling in the case of the discriminator. The number of channels in each block should decrease with each block, such

that the last layer has only three channels. The minibatch standard deviation block is placed in the discriminator before the final block which converts the lowest resolution feature maps to a single value [KALL18].

In this particular implementation used in the experiments for this work the generator and discriminator network have six blocks each with the following number of channels: 512, 512, 512, 512, 256, 256 (the discriminator has the same number of channels but in reverse) and the dimensionality of the latent space is 512. More details can be found in the accompanying code.

5 Diffusion Models

Diffusion models are the newest of the three as they were introduced in 2020 by Jonathan Ho, et al. [HJA20]. Due to their great results already in the first paper, DMs quickly gained popularity and have been improved upon. The in this chapter presented model architecture and its implementation is based on [HJA20]. Because this model on its own offers satisfactory results, it is sufficient for this comparative study. More sophisticated models offer faster sampling times or generation of images based on a textual description, which is not considered in this work.

DMs work differently compared to VAEs and GANs. The latter models generate samples in one forward pass in the neural network. DMs on the other hand are based on an iterative process, where multiple forward passes in the neural network are needed. The core idea behind DMs is to take pictures from the training dataset and gradually add noise to them until the image contains only gaussian noise. The task of the DM is then to learn to iteratively denoise the noised images. Having this, an image containing only gaussian noise can be gradually denoised until a valid image is generated.

5.1 Forward Diffusion

As mentioned before the first step is to gradually add noise to an image from the dataset. This procedure is controlled by a so called β-schedule and defined as a sampling operation from the following normal distribution q:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{5.1}$$

The index t describes a timestep in the noising sequence, where $t = 0$ means there is no noise added and $t = T$ means there is only gaussian noise in the image left. Given this distribution each consecutive image can be sampled given the previous image with less noise. The β-schedule can be linear where the values of β_t are uniformly spaced (e.g. between 0.0001 and 0.02). The $\mathbf{I}$ indicates an identity matrix. For this work $T = 1000$ noising/denoising steps are performed, as the authors suggest.

The noising procedure can be rewritten to directly sample a noised version of the original image at any timestep t without the need to literately sample from the distribution in equation (5.1) to arrive at timestep t:

$$\alpha_t := 1 - \beta_t \tag{5.2}$$

$$\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s \tag{5.3}$$

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I}) \tag{5.4}$$

Having the $q(\mathbf{x}_t|\mathbf{x}_0)$ distribution images at any timestep t can be directly calculated using the reparameterization trick:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon} \quad \text{with} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{5.5}$$

The $\mathbf{x}_0$ indicates the starting image from which the noised image $\mathbf{x}_t$ can be calculated at any timestep t using the formula above. Figure 5.1 shows the forward diffusion process.

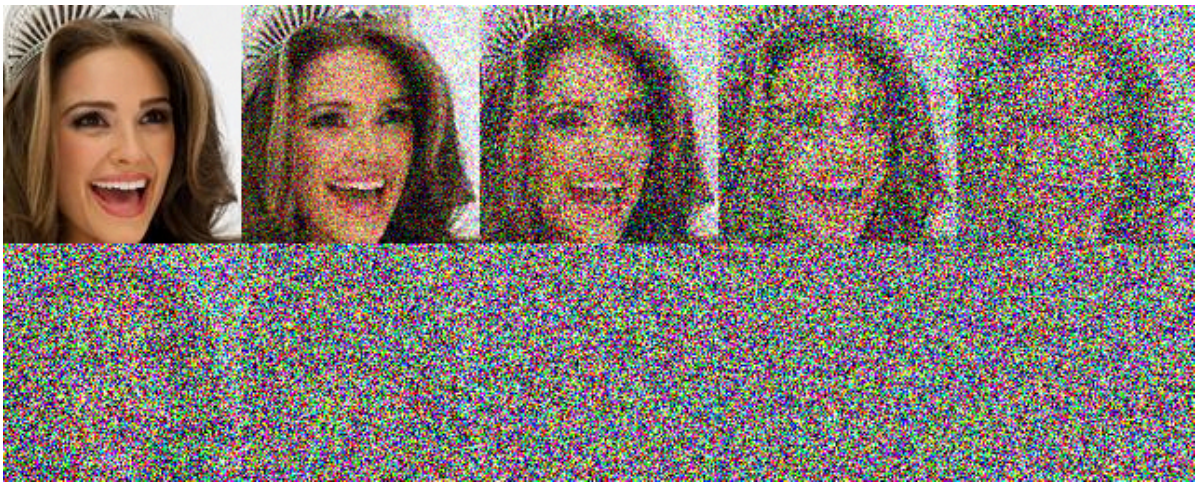


Figure 5.1: Forward Diffusion (clear image corresponds to timestep 0 and fully noised image to the last timestep, timesteps for images in between are spaced uniformly)

5.2 Reverse Diffusion

Now the forward diffusion process needs to be reversed in order to produce valid images from gaussian noise. Precisely, the distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ has to be found. One way to determine this distribution would be to use Bayes' theorem:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t) = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1}) \cdot q(\mathbf{x}_{t-1})}{q(\mathbf{x}_t)} \tag{5.6}$$

While $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is given by the forward process, the terms $q(\mathbf{x}_{t-1})$ and $q(\mathbf{x}_t)$ represent the probability distributions to observe $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ respectively which both depend on the probability distribution of the data $\mathbf{x}_0$ which is unknown. Furthermore, if one could approximate $\mathbf{x}_0$ the calculation of $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ would involve nested integrals which are untractable.

That's why $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is approximated by another parametrized distribution p_θ, which is defined by a normal distribution:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_t; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \beta_t \mathbf{I}) \tag{5.7}$$

In this definition $\boldsymbol{\mu}_\theta$ will be approximated by a neural network and the variance is fixed to a time dependent constant according to the β-schedule [HJA20].

In order to train the model $\boldsymbol{\mu}_\theta$ a loss function has to be defined. What should be minimized is the negative log likelihood:

$$\mathbb{E}_{q(\mathbf{x}_0)}[-\log p_\theta(\mathbf{x}_0)] \tag{5.8}$$

Likelihood describes a measure for $\mathbf{x}_0$ being sampled from the distribution p_θ. The higher the likelihood, the better p_θ represents the data distribution, hence the likelihood should be maximized. However, as in the case of the VAE the likelihood is intractable and cannot be computed. That's why the negative log likelihood is not optimized directly but indirectly via the variational upper bound:

$$\mathbb{E}_{q(\mathbf{x}_0)}[-\log p_\theta(\mathbf{x}_0)] \leq \mathbb{E}_{q(\mathbf{x}_0)}\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] := L \tag{5.9}$$

Because the upper bound is always larger that the negative log likelihood, minimizing the variational upper bound minimizes the negative log likelihood. The upper bound can be further rewritten to:

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[\underbrace{D_{KL}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t>1}\underbrace{D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}}\underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\right] \tag{5.10}$$

$D_{KL}(q \parallel p)$ describes the Kullback–Leibler divergence between probability distributions p and q, which is a measure describing a distance between them (it is not a metric for distance as it is not symmetric and does not satisfy the triangle inequality).

The first term L_T in (5.10) can be ignored as it does not depend on any parameters of the model. $q(\mathbf{x}_T|\mathbf{x}_0)$ describes the forward diffusion process which does not have any learnable parameters and $p_\theta(\mathbf{x}_T)$ describes the reverse process at timestep T which is gaussian noise.

The second term L_{t-1} in (5.10) contains the intractable part $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$, but when conditioned on $\mathbf{x}_0$ it has a closed form:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1};\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0),\tilde{\beta}_t\mathbf{I}) \tag{5.11}$$

$$\text{with}\quad \tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_{t-1}}\mathbf{x}_t \quad\text{and}\quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{5.12}$$

Further, the definition of $\mathbf{x}_t$ in (5.5) from the forward diffusion process can be rearranged after $\mathbf{x}_0$, plugged into the equation for $\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0)$ and together rewritten to:

$$\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}\right) \quad\text{with}\quad \boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}) \tag{5.13}$$

Because the variance is held constant, the KL-Divergence in the L_{t-1} term in (5.10) can be simplified to:

$$L_{t-1} = \mathbb{E}_{q(\mathbf{x}_0)}\left[\frac{1}{2\sigma_t^2} \parallel \tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) - \mu_\theta(\mathbf{x}_t,t) \parallel^2\right] + C \tag{5.14}$$

where C is a constant, which can be ignored, since it does not depend on any learnable parameters. With this loss a neural network could be trained to predict the mean directly, however $\mu_\theta(\mathbf{x}_t,t)$ can be can be defined the same way as $\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0)$ in (5.13), where only the $\boldsymbol{\epsilon}$ is being predicted by the network:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) \tag{5.15}$$

Now $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ represents the output of the neural network from which the mean $\mu_\theta(\mathbf{x}_t, t)$ can be calculated. Having the definitions of both means, they can be plugged into the L_{t-1} term (5.14) and simplified to:

$$L_{t-1} = \mathbb{E}_{q(\mathbf{x}_0)}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)} \parallel \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \parallel^2\right] \tag{5.16}$$

Regarding the last term L_0 in (5.10), it can be shown, that it is minimized when the network directly predicts the natural image from $\mathbf{x}_1$. Since $\tilde{\mu}_1(\mathbf{x}_1, \mathbf{x}_0)$ is by definition of the reverse process close to the real image, L_0 can be minimized when the network approximates $\tilde{\mu}_1(\mathbf{x}_1, \mathbf{x}_0)$. For this L_{t-1} can be reused. Consequntly, during the last step in the reverse diffusion process $\mu_\theta(\mathbf{x}_t, t)$ is taken as the final image.

The loss function in (5.16) can be further simplified, by setting the large factor in front to 1 as it only scales the gradients. It has been found, that setting the factor to 1 is beneficial to sample quality as well. The resulting simplified Loss is:

$$L_{simple}(\theta) = \mathbb{E}_{q(\mathbf{x}_0)}\left[\parallel \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \parallel^2\right] \tag{5.17}$$

Having the loss function, which is tractable, a neural network can be trained to minimize it on a given dataset.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure 5.2: Training and sampling algorithm for a DM [HJA20]

Figure 5.2 shows the respective algorithm for training a neural network and sampling images from the DM based on the derivations above. The training algorithm consists of sampling a batch of training images, timesteps, and gaussian noise. The images are then noised to a timestep t according to the forward diffusion process and the β-schedule specified before. Next, the neural network predicts the noise that was applied to the

image given the noised image and the timestep t. This is then compared against the true noise that was applied and a gradient is calculated, which is used to optimize the neural network.

The algorithm for image generation with the trained neural network consists of first sampling an image, that consists of gaussian noise. The network predicts then from this image and the timestep t (which starts with the largest value T) the noise from which the mean of the $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ distribution can be calculated. Having the mean and the variance given by the fixed β-schedule, next less noised image can be sampled using the reparameterization trick. This procedure is repeated until timestep 1, where the last image is just the mean value of the $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ distribution.

5.3 Network Architecture

Although in theory any network could be used to model $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$, a U-Net [RFB15] shaped network with residual connections and self-attention blocks is proposed by [HJA20] and implemented in this work. As explained before this network predicts the noise or $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ from a previously noised image and a timestep t. In the following the network architecture used in the experiments for this work and its main components will be explained in detail.

5.3.1 Sinusoidal Positional Embedding

The time t is not simply fed into the network as a single value, but is pre-processed first using the sinusoidal positional encoding. The idea for this kind of encoding comes from the Transformer-Model [VSP+17] and makes sure, that no large numbers are being passed into the network, in order to ensure numerical stability and that the steps are identifiable by the network.

The encoding for a timestep t is a vector where each element i is calculated using the following formula:

$$SPE(t)_i = \begin{cases} \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right) & \text{if} \quad i \bmod 2 = 0 \\ \cos\left(\frac{t}{10000^{\frac{2i}{d}}}\right) & \text{if} \quad i \bmod 2 \neq 0 \end{cases} \tag{5.18}$$

The parameter d specifies into how many dimensions the timestep t should be encoded, and consequently indicates the dimensionality of the output vector. The basic idea for this formula is to have each dimension of the positional encoding correspond to a sine wave of different frequency. This way the encoding does not include large values and is

continuous [VSP+17]. The Matrix in Figure 5.3 shows the mapping from each timestep t to its encoded version.

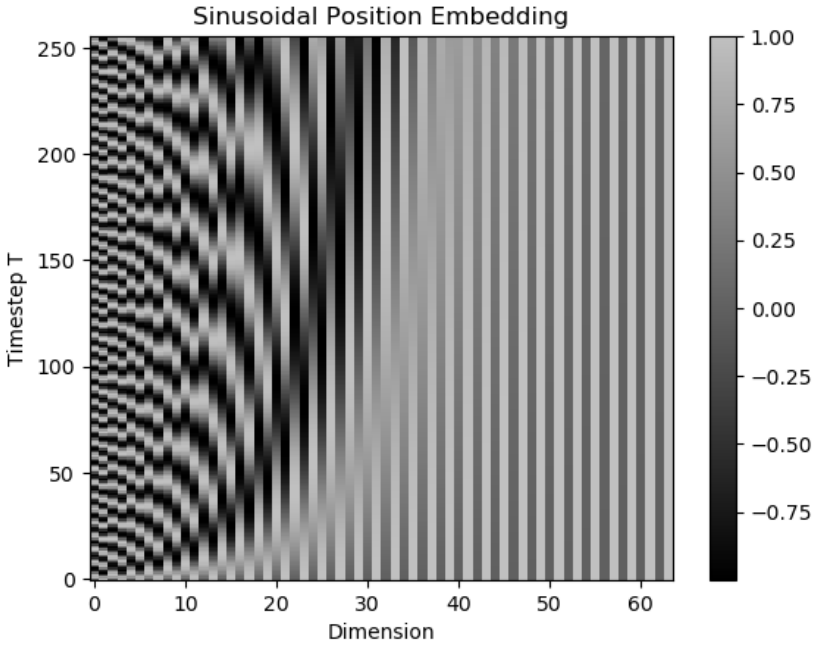


Figure 5.3: Sinusoidal Positional Embedding - Matrix (inspired by [WSLY22])

For this visualisation 256 timesteps were embedded into vectors of size 64. Each row of this matrix corresponds to the encoding for a given timestep t.

5.3.2 U-Net

Figure 5.4 shows the architecture of the U-Net network. The network is structured in ResNet-blocks and is shaped with a bottleneck in the middle. Figure 5.5 shows the structure of a ResNet block used in the network for the DM implemented for this work. In addition to the residual connection within the block the block itself, it has the timestep t as an input, which is first passed through a dense layer before being reshaped and added to the feature maps. Additionally, there are group normalization layers [WH18] and SiLU (SiLU$(x) = x\sigma(x)$ [EUD17]) as the activation function [HJA20].

In addition to the residual connections within each block, there are residual connections outside the blocks, that forward the output of an early block towards the end of the network skipping a number of blocks, where it is concatenated with the input to the given block, as shown in figure 5.4. Additionally, self-attention blocks (refer to [VSP+17] for more details on attention) have been introduced in between the ResNet-blocks (not

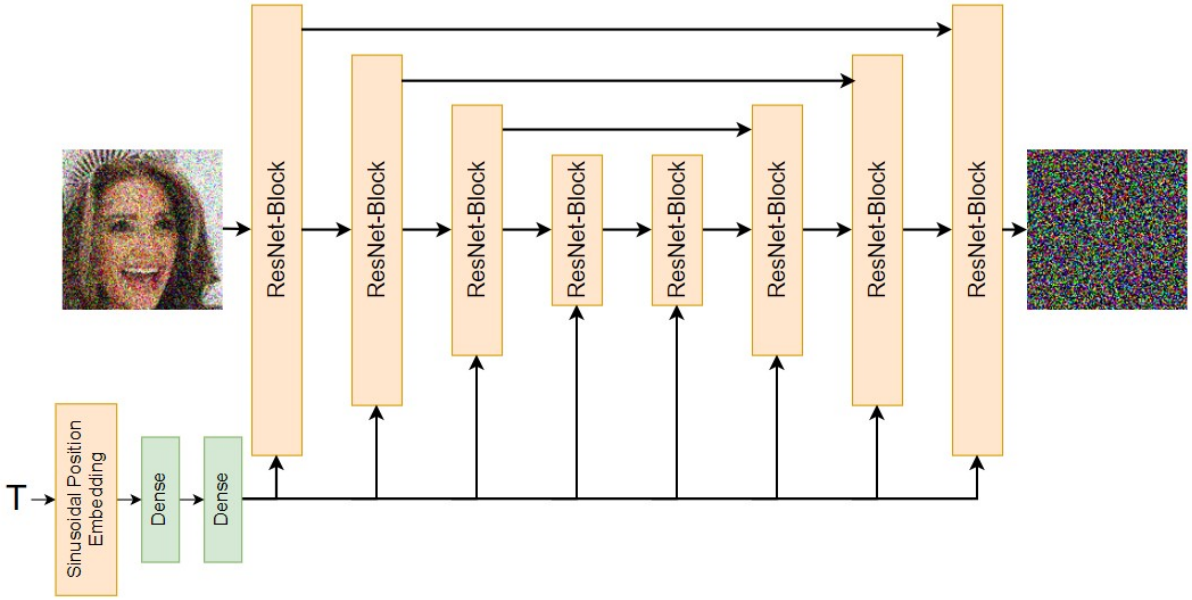


Figure 5.4: U-Net Architecture used in DMs (some layers omitted, number of blocks can vary) [HJA20]

shown in figure 5.4) [HJA20]. The timestep t is firstly embedded using sinusoidal positional embedding and then fed to a network consisting of two dense layers. The result is then fed into each block [HJA20].

In this implementation the number of channels is: 128, 128, 256, 256, 512, 512 in the first half of the network, what is then mirrored to form the other half of the network as shown in figure 5.4. The resolution of the feature-maps is being halved in the first half of the network between the ResNet-blocks and doubled in the second half using interpolation for upsampling and average pooling for downsampling respectively. The two dense layers after the sinusoidal position embedding have a dimensionality of 512. More details can be found in the accompanying code.

5.3.3 Exponential Moving Average

When training a neural network Exponential Moving Average (EMA) can improve its performance [CRM+21]. In general EMA is performed for smoothing out sequences of data points. For each data point the following is calculated:

$$s_0 = x_0 \tag{5.19}$$

$$s_i = \alpha x_t + (1 - \alpha) s_{t-1} \quad t > 0 \tag{5.20}$$

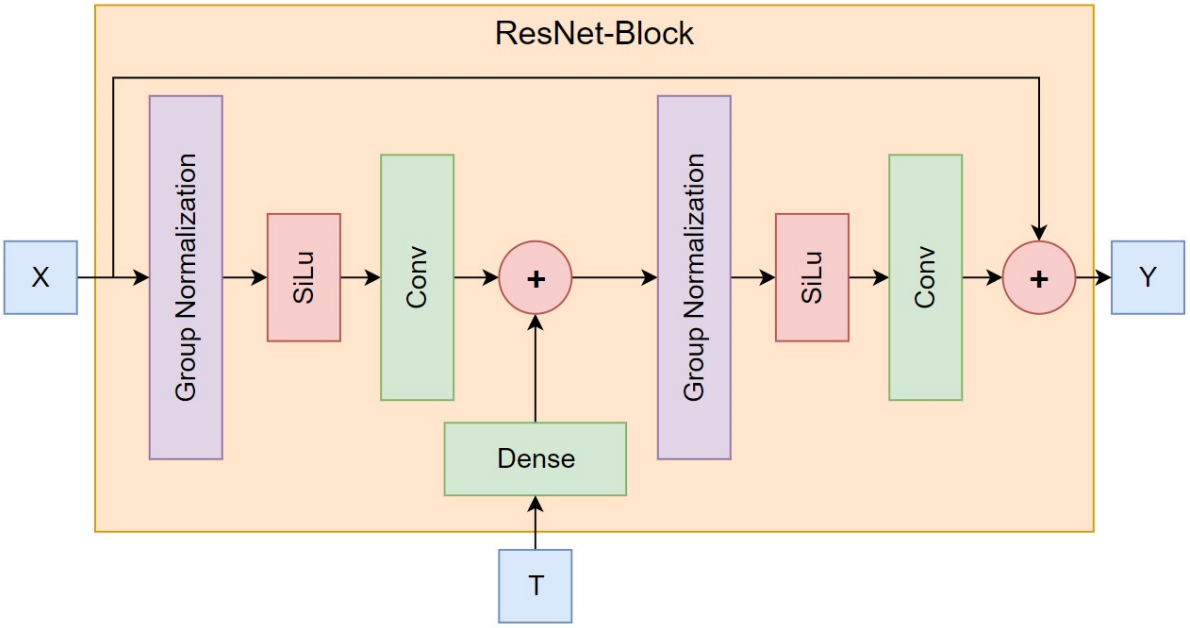


Figure 5.5: Structure of a ResNet-Block used in this work

where α serves as the smoothing factor $0 \leq \alpha \leq 1$. The larger the α the smaller the smoothing effect. Consequently, the less the α the smoothing effect increases, but the smoothed curve is less responsive to changes. Having this, the sequence of s_i represent the smoothed sequence of x_i [CRM+21].

Having the exponential moving average, the idea in relation to neural networks is to have two models: a first model which is trained as usual (called the student model) and a second one which serves as a model for the exponential averaging (called the teacher model). The student model is trained using the usual training procedure. After each optimization step the weights from the student model are copied to the teacher model, where the EMA is calculated from the current teacher model weights and the incoming weights. These averaged weights are saved in the teacher model. In other words, the weights in the teacher model can be interpreted as the smoothed ensamble of the temporal sequence of weights resulting from the training of the student model [CRM+21]. After the training, samples can be taken from both models and in the case of the DMs the teacher model seems to perform better than the student model. For such purposes the α is chosen to a value close to 1 e.g. 0.9999 [CRM+21].

6 Comparison

Having established the three generative models and the common metrics for evaluating generative models they can be compared against each other. This chapter covers the comparison between the Autoencoder, Generative Adversarial Network and the Diffusion model. These models will be compared with regards to realism, generalization and diversity, sampling, training difficulty, parameter efficiency, interpolating and inpainting capabilities, semantic editing as well as implementation difficulty. The goal here is not to find the best model, but to give various use-cases and see which model performs best in a particular scenario.

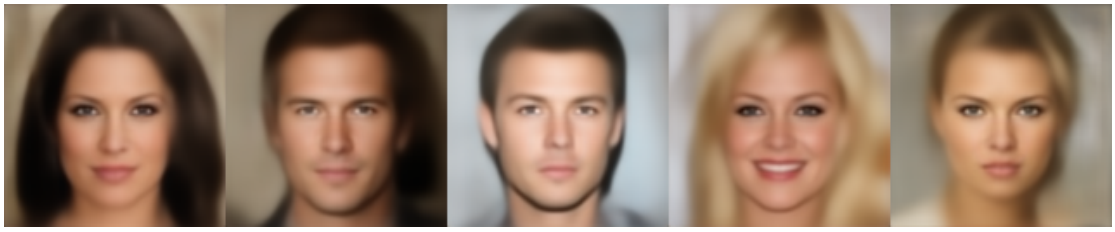
All the models have been trained on a server with a Nvidia Ampere A100 card with 40GB of graphical memory over multiple days using the "CelebA-HQ"-dataset [KALL18]. The "CelebA-HQ"-dataset consists of 30.000 images of faces of celebrities. The images come in the resolution of 1024×1024, but have been resized to a smaller resolution of 128×128 in order to be able to train the models on the available hardware. The dataset comes with each image transformed in a way to have the same structure. Figure 6.1 shows each transformation step that has been performed on raw images.

Figure 6.1: Individual transformation steps that have been performed on every image in the CelebA-HQ dataset [KALL18]

(a) shows the raw original image found on the internet. The next steps shown in (b) improve visual quality, remove JPEG artifacts and do 4x super-resolution. Next, the image is extended using mirror padding (c) and the padding is blurred (d). Finally, the image is rotated and scaled based on the position of the eyes and the mouth (e) such that the head is always in the same position. (f) shows then the final image used in training

of the models. Because of these transformations each image has the same structure, that is having the eyes and mouth always in the same position, which makes the training easier [KALL18]. Figure 6.2 shows a few images from the dataset and samples from the trained models. However, it is important to mention, that no model produces realistic images every time. Among generated images are not only ones with high quality, but ones with poor quality or images where one cannot even recognize a face.

(a) Samples from the CelebA-HQ dataset [KALL18]



(b) Samples generated by the VAE

(c) Samples generated by the GAN

(d) Samples generated by the DM

Figure 6.2: Samples from the dataset and the models

6.1 Realism

The first comparison criterion is the realism of the generated images. To measure it the earlier introduced FID and IS scores are used. Because the inception network accepts images with the size of 299×299 the images produced by the network have to be rescaled accordingly. For the FID, the training images are first scaled to the size the network was trained with, and then resized again to 299×299. This way details are obviously lost, but the generated images are compared to the images the network was trained on, instead of comparing low resolution generated images to original 1024×1024 images.

Model	IS	FID
VAE	1.72	96.30
GAN	2.36	15.77
Diffusion	2.72	9.02

Table 6.1: IS (higher is better) and FID (lower is better) scored of the three models

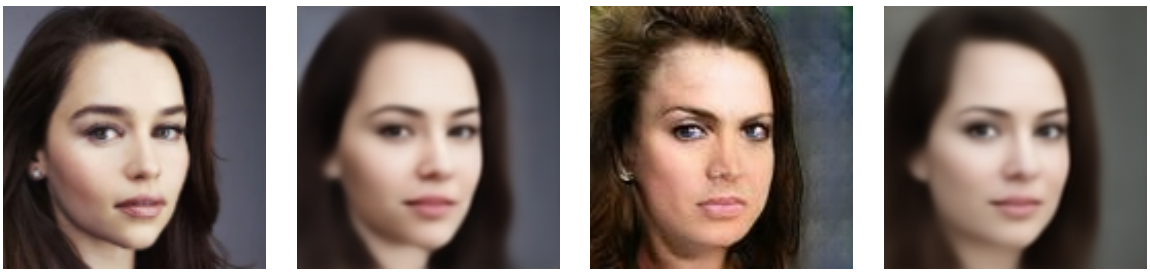
Table 6.1 shows that the DM performs the best of the three models with regards to both metrics, while the score for the VAE is significantly worse, than the score for the DM or GAN. This might be because of the blurry images that the VAE is producing, which in turn might result in features from the inception network, that are significantly different from the ones from the training set, as they are sharp. The DM and the GAN have scored similar results, as the images from both models are good, however the GAN seems to generate bad pictures more often, than the DM. After the inspection of a grater number of generated images, the scores seem to be consistent with their quality (some samples presented in figure 6.2). From both the visual inspection and the archived scores the results, especially from the GAN and the DM, are comparable with the literature [KALL18] [HJA20]. Although in this case the scores are consistent with each other, in further analysis only the FID score will be used, as it is effectively replacing the IC-score in the literature.

6.2 Generalization and Diversity

It is difficult to quantitatively access how well a model generalizes and how diverse the results are (previous metrics are based on these criteria and provide an overall score, but based on those one cannot decide for example which model has more diverse images: images from the VAE seem to be more diverse than the images from the GAN 6.7, but the GAN has a better FID-score). In classical classification problems generalization can be accessed by holding out a part of the dataset. On this validation dataset the model

is not trained, but tested if it can correctly classify samples, it has never "seen". This way validation accuracy can be calculated and compared against other models.

In the case of generative modelling this cannot be done. A similar test which can be performed on some models is to take a picture which belongs to the distribution of the dataset but is not contained in it, and then to check if this picture can be generated by the model. In the case of the GAN, one would try to optimize a randomly initialized latent vector such that the generated image resembles the target image as closely as possible. For this the mean absolute error (MAE) in pixel-space between the generated image and target image is used as the loss function. This optimization is being done with gradient descent and backpropagation over the whole network, but the gradients are taken with respect to the latent vector. While performing this optimization the weights of the network are frozen and not changed. This procedure is called "latent vector recovery" [WBQFM21]. The same procedure can be performed by the decoder part of the VAE or the target image can be encoded by the encoder and then decoded. Because the VAE was trained to perform reconstruction ideally the same image should be reconstructed. The DMs are quite different, as they do not have a latent space as the VAE or GAN and the generation process is being done iteratively in many steps (here: 1000). For this reason, it is infeasible to access by the same procedure as in the VAE or GAN if a given image can be generated by a DM.



(a) (b) (c) (d)

Figure 6.3: Latent vector recovery of an Image with VAE and GAN: (a) Original [emi] (b) VAE Recovered (c) GAN Recovered (d) VAE encoded and decoded

Figure 6.3 shows how an image is recovered using the above method by the VAE and GAN. While the image recovered by the VAE is very similar to the original one, the image recovered by the GAN is not as good. Although it shows some similarity with regards to the hair style and colour or the position of the head in the original image, it cannot be said, that the latent vector was recovered successfully, as the image is clearly different and does not represent what one would expect. In the case of VAE the image can be encoded and then decoded, which is shown in figure 6.3 in image (d). The result here is visually quite good but not as good as the recovered image from the same VAE.

As mentioned before in the case of the DMs the image cannot be searched in the latent

space because they don't have one (although the set of all random variables could be seen as a kind of latent space, the optimization would be infeasible due to the iterative process. [HJA20] describes individual images $\mathbf{x}_t$ in the denoising process as latent vectors, as same noised image, as far as $t = 750$, results in images with similar visual features). However, random noise can be added to the given image and the image can be denoised by the DM. This doesn't prove that the image can be produced by the DM, but shows if the model is able to work with random images, which would be a sign for good generalization.

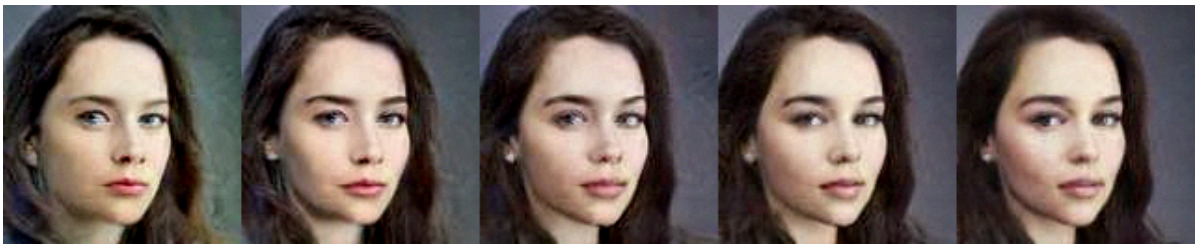


Figure 6.4: Noising and denoising of an image [emi] using 300, 200, 100, 50, 25 steps

Figure 6.4 shows the denoised images where the original was noised various times. Obviously, the more the image is noised the more information is lost and the image diverges from the original. However, the DM is able to adapt and work with images that it has never "seen" during the training process, which is a strong sign for good generalization. Furthermore, when confronted with constraints, in this case the constraint being the noised original image, the DM seems to adapt well and removes the noise, despite the image not being a sample from the training set nor the model itself.

Another method for qualitative assessment of the generalization is to look for nearest neighbours of a generated image among the training images. If the generated image is very similar to the training images, then this could be a sign for poor generalization, as the model would just recreate the images from the dataset. The model should be able to recreate training images, as they belong to the distribution the model has been trained with, but more importantly it should generate new unseen images as well. In this section for each model the nearest (or furthest) neighbours are considered in pixel- and feature space (features extracted using the Inception network) using the Euclidian distance.

In figure 6.5 it can be seen that no model simply reproduces the training data, which shows that the models do generalize in this regard. However, based on this it is difficult to decide, which one does generalize best.

What might be interesting with regards to the generalization as well, are the furthest neighbours (shown in figure 6.6) of a sample among other random samples. Together with the nearest neighbours 6.7 one might get an idea of the span of images that are being generated.

(a) VAE

(b) GAN

(c) DM

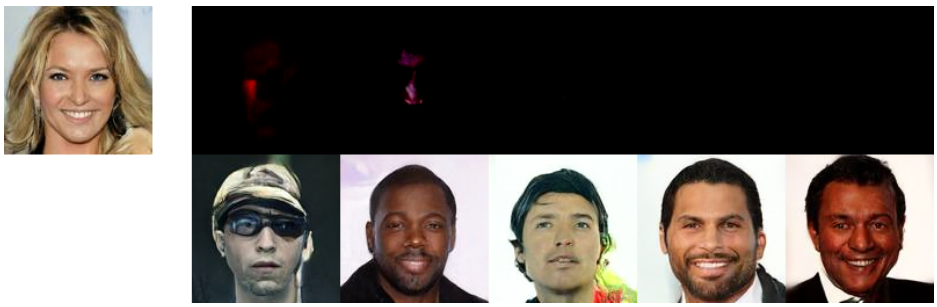
Figure 6.5: Nearest Neighbours of a sample among the training data. Nearest neighbours are searched for in pixel space (first row each) and feature space (second row each).

Interestingly, the furthest pixel neighbours of a valid image seem to belong to the worst images generated by a given model. Although the DM generates the best images overall its worst images are worse than the bad images of the other models. On the other hand, the bad images of the VAE are more acceptable.

As described in previous chapters, the GANs suffer the most from mode collapse. Various techniques have been applied to combat mode collapse, and the results are better when compared with previous versions. However, it cannot be said, that mode collapse has been solved completely. Measuring mode collapse is difficult if even possible, but in order to get

(a) VAE

(b) GAN



(c) DM

Figure 6.6: Furthest Neighbours of a sample among other 8096 randomly sampled images. Furthest neighbours are searched for in pixel space (first row each) and feature space (second row each).

an idea of how much a model collapses into one or few modes, the nearest neighbours of a sample can be searched among other samples. This way all three models can be accessed for their mode collapse and the mode collapse of the GAN can be compared to the other models. Additionally, this gives an idea how divers are the generated images in each model. In this case for each model one sample has been chosen and nearest neighbours have been searched among other 8096 random samples both in pixel and feature space. Figure 6.7 shows the found nearest neighbours for a sample for each model.

In the case of the VAE and the DM the faces can be visually assigned to different

(a) VAE

(b) GAN

(c) DM

Figure 6.7: Nearest Neighbours of a sample among other 8096 randomly sampled images. Nearest neighbours are searched for in pixel space (first row each) and feature space (second row each).

persons. This is not the case for the GAN. Although the images are not completely the same there are pairs or triples, which could visually belong to the same person (e.g. 1st row 2nd image, 1st row 2nd image and 2nd row 1st image are very similar or 2nd row 4th image and 2nd row 5th image could be the same person, as well) This shows, that mode collapse is to a degree still present in this model. Other models that do not suffer from this problem don't show this kind of similarity among the same number of random samples. With regards to the diversity among the other two models it seems, that the DM offers a wider diversity of generated images, as the nearest neighbours of a sample

among other samples differ from each other more than the nearest neighbours in the case of the VAE. While the nearest neighbours are expected to lie semantically close to each other, the DM still generates images with varying hairstyle or hair colour. In the case of the VAE the images are blurred which makes it more difficult to access how different are the images from each other. However, from what can be seen on the generated images, the variety seems not to be as high as in the case of the DM.

All in all, from the visual inspection of the results it seems, that the DM generalizes the best, as the model is able to work with random images and have a high variety among the generated samples. On the other hand, the GAN is not able to reconstruct a random image and has signs of the mode collapse, what makes it the worst of the three in this regard. Although the DM seems to be the best here, it is not possible to formally prove that DM generalize best.

6.3 Sampling

With regards to the sampling the most interesting metric is the sampling time. Sampling time is especially important, when the application requires real-time, where sampling has to be done as fast as possible. In this case the time to generate a batch of 32 images was measured. The experiment was conducted on a server with a Nvidia Ampere A100 with 40GB of graphical memory. Table 6.2 shows the measured times.

Model	Time in s
VAE	1.04
GAN	0.12
Diffusion	546.08

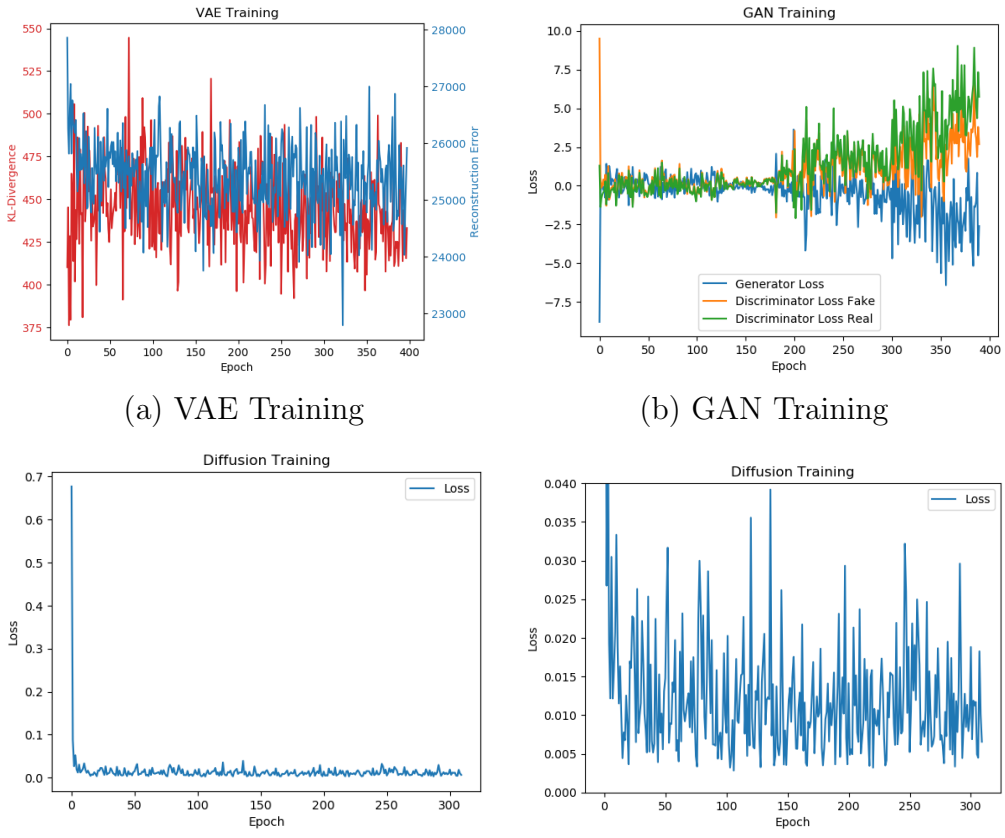
Table 6.2: Time taken for generation of one batch of 32 images

The sampling times for the DM is obviously the worst due to its iterative nature. Not only the forward pass has to be done many times, but in between the steps new random vectors have to be sampled and the result from the previous step is manipulated with additional calculations. In contrast, VAEs and GANs produce their images from one single forward pass and a denormalization step in order to bring the images back to the required range between 0 and 1. The higher times from the VAE in comparison to the GAN may stem from the fact that the VAE has fully connected layers in the bottleneck and more sampling operations than the GAN especially because of the improved sampling technique described in 3.6.

6.4 Training Difficulty and Stability

Training difficulty and stability of the different models can only be compared qualitatively. The criteria here are the convergence of the models and interpretability of the loss graphs resulting from the training of each model. Naturally high interpretability of the training graphs is desired in order to be able to debug the model and to optimize the hyperparameters more effectively. Having a model that converges is desirable as well, as it ensures, that the model will not eventually spiral out of control.

With regards to the convergence the GANs are known to be very unstable and may even diverge if trained too long, which is why the training process has to be closely monitored. Even with the addition of the Wasserstein-Loss and the batch discrimination the network may still diverge. This is not the case for DMs and VAEs.



(a) VAE Training

(b) GAN Training

(c) Diffusion Training

(d) Diffusion Training zoomed in

Figure 6.8: Training Graphs of different models

Figure 6.8 shows the training histories of the three models. Unlike in a classical classification problem all of the graphs are more unstable and mostly uninformative. The

instability might come from the inclusion of random noise in all of the models. Only the graph for the VAE contains interpretable information as the measured metrics are the reconstruction error and the KL-Divergence. However, these values hardly correlate to the quality of the generated images. While the quality of the images is steadily rising especially in the first epochs of the training process, no clear tendency of falling or rising can be seen in the graph. This graph however does illustrate the relation between the KL-Divergence and the reconstruction error. For example, around epoch 50 or 260, it can be seen, that while the KL-Divergence is decreasing the reconstruction error is rising (the reverse case can be somewhat seen at around epoch 90), which is another indicator for this VAE not being able to optimize perfectly both the reconstruction error and the KL-Divergence.

With regards to the GAN-Model the overall loss can be broken down into its three components: discriminator loss on real images, discriminator loss on generated images and the generator loss, which is the discriminator value on generated images when training the generator. In spite of this breakdown of the loss function the interpretability remains low. Every 60th epoch the resolution is being increased and a new layer is being introduced. This can be seen in the changing behaviour of the graph at epochs that are a multiple of 60. Later on, in the graph the loss values start to behave chaotically and possibly to diverge in spite of the increasing quality of the images. Other problems such as mode collapse or the quality of the images cannot be identified from the graph which makes the debugging and the hyperparameter optimization more difficult. The diverging loss values are especially problematic with regards to image quality, because from the graph it is not possible to see when to stop the training process before the quality of generated images starts to diverge. One possibility to circumvent this would be to measure the FID-Score every epoch and stop the training, when it starts to deteriorate. However, this would significantly increase the training time as measuring the FID-Score includes sampling at least 8000 samples and extracting the features from each of them.

The DM seems to be the most stable one. However, the graph does not contain any valuable information either. While the observed quality of the generated images increases over the whole training, no equivalent decrease in the loss can be observed. While the loss values at epoch 20 and 300 are almost the same, the quality of the generated images differs significantly at these points in time. One could argue, that the training graph for the DM contains even less information than the one for the GAN. While both graphs are uninformative the loss values for the GAN change significantly over time, while the loss for the DM does not change its behaviour over the whole training process with the exception of the first few epochs.

All in all, with regards to the training difficulty and stability the VAE seems to be the best, as it does not diverge and has interpretable training curves. The worst model in this category would then be the GAN, as it may diverge and has not interpretable training graphs. In general, it seems, that there is a negative correlation between the

interpretability and the quality of generated images [GBY+19]. The more quality the generated images have, the harder it is to explain why this particular model is better than the model with lower quality images.

6.5 Parameter Efficiency

When resources are scarce and the highest quality possible is to be archived, the parameter efficiency is a good metric for deciding which model to use. Parameter efficiency refers to a measure of how much FID-points per parameter can the model archive. The higher the archived FID-score with less parameters the better the model for such use case. Because the FID-score is better when the FID-value is lower, first the reciprocal value is being formed. The result is then divided by the number of parameters, to get the parameter efficiency. The following formula shows how the parameter efficiency has been calculated:

$$\text{Efficiency} = \frac{\frac{1}{\text{FID}}}{\#\text{Parameters}} = \frac{1}{\text{FID} \cdot \#\text{Parameters}} \tag{6.1}$$

Model	#Parameter	FID	Efficiency in 10^{-9}
VAE	69.846.659	96.30	0.149
GAN	49.107.456	15.77	1.291
Diffusion	109.929.987	9.02	1.009

Table 6.3: Parameter efficiency of the three models

In this particular case each model has a different number of parameters. This results from them occupying a different amount of space during the training process. Each of them is build in a way that maximizes the number of parameters while being able to be trained on the given hardware (Nvidia Ampere A100 with 40GB of graphical memory). Table 6.3 shows the individual efficiency values for each model. The GAN has reached the highest efficiency, which together with its high sampling speed suggest, that it might be used in scenarios, where the resources are limited. The VAE on the other hand not only has more parameters than the GAN, but performs significantly worse that the GAN, which results in a much lower parameter efficiency. The DM has the best FID-score, but has more than double the number of parameters, which makes it slightly less efficient than the GAN.

These results come from the training of one model each, which makes the results less meaningful. In order to be able to make a better decision, which model is most parameter efficient, one would need to train multiple models with various number of parameters,

in order to see how the FID-score changes depending on the number of parameters. However, this would need to be done correctly, as every model can be built in a way which does not produce good results, but has a large number of parameters (e.g. a model with one layer with 50.000.000 parameters). Because there is no "correct way" to structure a neural network, the parameter efficiency can only be compared on the basis of single instances and hardly in general.

6.6 Interpolations and Continuity

All models have the capability to perform interpolations between images and can be compared against each other on their ability to do so. Here a good interpolation is judged based on the similarity of the consecutive images, which correlates to the continuity of the latent space, and the ability of the model to interpolate directly from the starting image to the end image without including images, that differ significantly from both the start and end image (e.g. when interpolating between two persons with dark hair the should not be a person with blond hair in the interpolation).

In the case of the VAE and GAN, interpolation is performed by first interpolating the respective latent vectors and then decode all resulting vectors to an image. For latent vectors close to each other the resulting images should be "visually close" to each other. This way each consecutive image is more and more similar to the end image, which results in an interpolation from image to image.

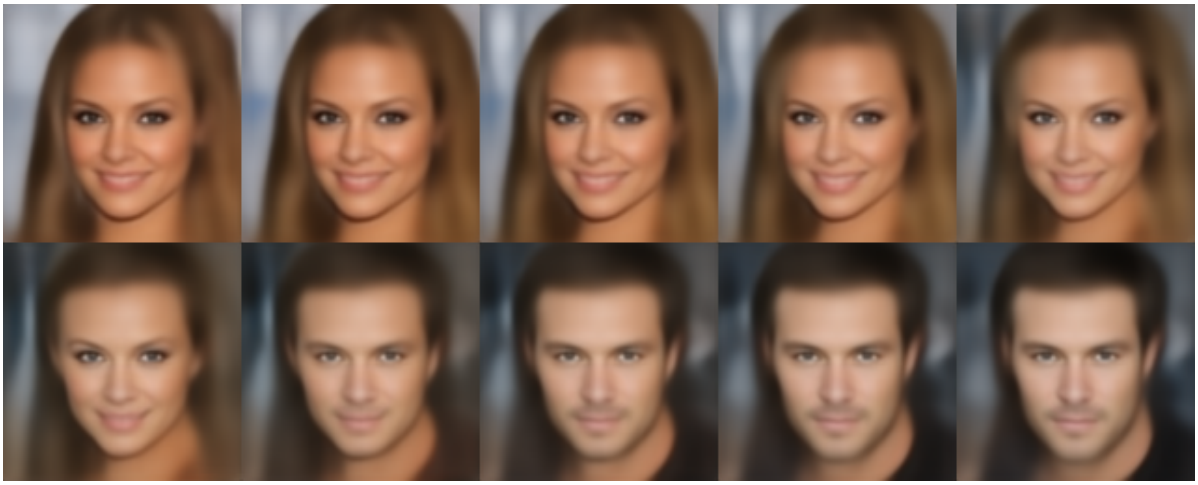


Figure 6.9: Spherical interpolation unsing the VAE

Figure 6.9 shows the interpolation between two samples with the VAE. Here the vectors have been interpolated spherically. Because of the high dimensionality of the latent

space, the length of all random latent vectors is approximately constant, which makes them form a n-dimensional hypersphere. When interpolating linearly between two latent vectors, the intermediate vectors will have a shorter length, depending on the position of the two source vectors (in the worst case one interpolated vector may be the zero-vector if the source vectors are exactly opposite to each other). Because the network was trained on random latent vectors, which all lie approximately on a n-dimensional sphere, vectors that lie outside of it may produce unexpected results. Figure 6.10 shows the result when interpolating from a random latent vector $\mathbf{x}$ to $-\mathbf{x}$. Although a face is still recognizable in the middle of the interpolation the quality is much worse. To avoid this latent vectors can be interpolated spherically, which preserves the length of each interpolated vector (small differences in length of the source vectors are being interpolated linearly). This way better results can be archived.

Figure 6.10: Linear interpolation unsing the VAE starting from a latent vector $\mathbf{x}$ and ending at $-\mathbf{x}$

For the GAN the interpolation is being done the same way. Figure 6.11 shows an interpolation between two samples generated by the GAN. Unlike the VAE in the case of the GAN latent vectors can be interpolated linearly without a decrease in quality. This consequently leads to a break in continuity at the zero-point in the latent space, because if one would interpolate between a value $\mathbf{x}$ and the value $-\mathbf{x}$, exactly at the zero-point the vector changes its direction which produces a different image. This sudden change of the image indicates a discontinuity in the latent space. The cause for this may be the pixel normalization layer being the first layer in the generator network, where the latent vector is normalized. Because of the division by $x \cdot x$, which is in case of $x = 0$ a division by 0, there is a discontinuity in the generated image. No error is thrown at this point, because a small ϵ is being added for numerical stability. Although mathematically speaking the whole network is continuous, sudden changes in the image space are not desirable, as they make the interpolations and movement in the latent space in general

Figure 6.11: Linear interpolation using the GAN

unpredictable.

In the case of the DMs the interpolation does not work this way, because of their different latent space. Interpolations are performed by first noising both start and end image to a given number of steps and then interpolating between them linearly in pixel space (interpolating the source images first and noising after works as well, but the former seems to give better results). The resulting images can be then denoised resulting in an interpolation of the source images. The result naturally depends on the number of noising steps. The more the source images are being noised, the more information is lost and the resulting images may look different from both source images. On the other hand, if the images are not noised enough the model does not have enough timesteps to remove the remnants of the linear interpolation.

Figure 6.12 shows the interpolation using the DM. As described before the images that have been noised using many steps are quite different from both source images. A good number of steps seems to be at around 50 to 200, although the results are not as convincing as the interpolation from a VAE or a GAN. This might be because of the iterative denoising procedure. Because it is long and involves multiple forward-passes consecutive images are not as close to each other as in the case of the GAN or VAE, which indicates, that the latent space of the DMs is more chaotic in comparizon to VAEs or GANs. However, it is worth noting that the DM can interpolate between any random images not necessarily ones that have been generated by the model itself. In the case of the GAN the latent vectors for the given images would need to be recovered, before they can be interpolated. In the previous section an example was shown, which gives an idea of the capability for latent vector recovery for the GAN and the results were rather poor. On the other hand, previous examples have indicated, that interpolating between random images with the VAE would be possible, which makes it superior to the GAN in this

Figure 6.12: Interpolation (horizontal) using the DM with noising steps: 0, 50, 100, 150, 200, 250, 300, 400, 500 (vertical)

regard.

6.7 Inpainting

Inpainting refers to the problem of filling the missing portions of the image by the model in a realistic way. This is very useful in scenarios where some features or artefacts have to be removed from the image. The user can then mark the area to be replaced and the model fills it with its own portion of the image according to its training and the

rest of the image stays unchanged. After explaining the inpainting procedure for each model, the results will be compared with regards to the quality of the inpainted region, how well the inpainted region fits into the original image and how many variations the models are able to produce.

Depending on the model's architecture, the inpainting problem is being solved differently. For models that have a latent space and a network that maps latent space to image space as in the case of GANs or VAEs inpainting is being done similarly to latent vector recovery: For a given image with a region that is to be replaced a latent vector is being searched, such that the generated image and the target image are as similar as possible in the pixel-space with regards to the pixels, that are not to be replaced (mean absolute error as loss function). This is being done by first choosing a random latent vector x and then optimizing it with gradient descent, such that the above described criterium is being fulfilled. Having the optimized latent vector and the corresponding generated image, the missing part of the original input image can be replaced by the corresponding part from the generated image (the one obtained from the recovered latent vector). In the ideal case, where the network generalizes perfectly, the error between the images should approach 0 and the missing part can fit perfectly.

In the case of DMs, the inpainting procedure looks differently, as there is no single latent vector or a model, that produces an image directly in one single forward-pass. Instead, the missing part is being generated iteratively, similarly to the usual generation process. Having an image with a missing part it is firstly noised to the maximum number of noising steps using the according noising schedule and the missing part is being initialized with random gaussian noise. In each denoising step the image is being denoised, but in order to prevent the model to generate a completely different image, the known part of the image is being noised again using the corresponding number of noising steps and replaced into the current image while the unknown part remains unchanged. This way the model is forced to denoise the unknown part of the image in a way such that it fits the original image. In the last step the known part of the image without any noise is being placed into the image with the fully generated missing part.

In order to access the inpainting capability of the models a random image from the internet has been chosen and one eye has been removed. Figure 6.13 shows the original image and the same image with the removed eye marked black. This region will be inpainted by all the three models.

Figures 6.14 6.15 and 6.16 show the respective results by all the models. The DM clearly produces the best results, while the GAN suffers from a problem that was pointed out in section 6.2. Because the GAN has difficulties in recovering the latent vector, the inpainted part has a different skin tone. This is not the case for the VAE, as the eye fits almost perfectly. However, because the images produced by the VAE are blurry it is to be expected, that if another portion of the image would have to be inpainted, the results might not be as convincing as in this case. With regards to the GAN, it can be

(a) Original [emi] (b) Removed eye

Figure 6.13: Original image with marked region which is to be inpainted

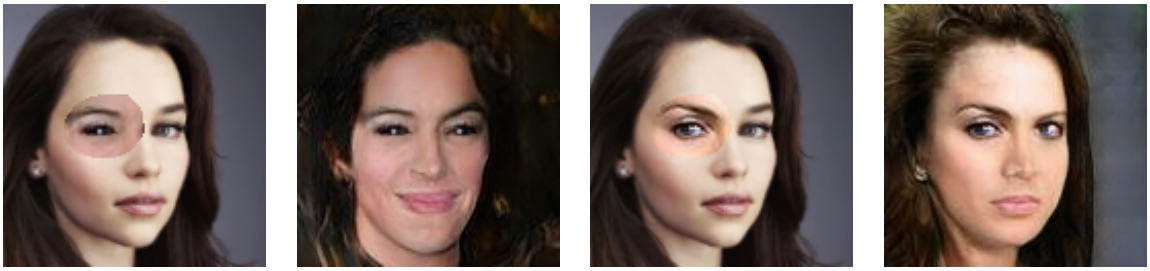
(a) (b) (c) (d)

Figure 6.14: Inpainting by VAE: (a), (c) original image with inpainted eye; (b), (d) corresponding reconstructed images

seen, that some high-level features (looking direction, hair colour, nose shape) are being correctly reconstructed, but it is not enough to produce a convincing inpainted eye.

With regards to the runtime of the inpainting process none of the presented model would be able to perform in real-time. While the DM performs almost the same algorithm as sampling new images. In the case of VAE and GAN, they have to perform latent vector recovery, that is they have to perform backpropagation through the whole generator/-decoder network and gradient descent on the latent vector, which is iterative and takes a comparable amount of time as the sampling algorithm on the DM (The usage of the encoder network in the VAE is here not possible as a portion of the image is missing. The missing area might be coloured white, black or with the colour of neighbouring pixels, such that the encoded image is somewhat valid, but as the encoder was not trained to encode such images, so the results might be poor or unexpected).

The DM is superior to the other two with regards to the number of variations that the model is able to produce as well. While every run produces a different eye that fits the



(a) (b) (c) (d)

Figure 6.15: Inpainting by GAN: (a), (c) original image with inpainted eye; (b), (d) corresponding reconstructed images

(a) (b) (c) (d)

Figure 6.16: Inpainting by the DM (All images are different inpainting runs by the same DM)

image very good, in the case of the GAN the recovered latent vector converges to one of few latent vectors, that lie in different local minima in the latent space. Additionally, there is the fact, that these local minima have to be found in the latent space, which can be done by optimizing different latent vectors, that have been initialized to different starting positions. By doing so different local minima can be found, but some latent vectors will converge to the same position in the latent space, which results in repeated latent vectors and wasted computational resources. Furthermore, the number of local minima seems to be limited. For the VAE however there seems to be only one result, that fits a given image best. This might be the case for the GAN as well, if it had generalized well enough, which further limits the options on the diversity of the inpaintings. On the other hand, when using the DM the user has a lot more flexibility, as one image may be chosen from a pool of many possible images.

6.8 Semantic Editing

Generative models can be used not only to generate images, but to change them on the semantic level as well. This ability will be considered in this section. Concretely, the comparison will take into the consideration the quality of the results, the difficulty of performing such a semantic change with a given model and the performance of such procedures.

Especially in models, that have a latent space such as the VAE and the GAN has, arithmetic calculations can be performed within the latent space, which may correlate to a semantic change in the resulting image. It has been found [Whi16], that in GANs, vectors can be found, that indicate the direction of change for a specific feature. Specifically, there can be a vector for example, that leads from black hair to blond hair. Having this vector, an image with black hair can be transformed by adding the "blond-vector" to the latent vector of the given image with black hair. Such vectors are called concept-vectors. The resulting latent vector would in the best case produce an image with the same person having blond hair with all the other features being preserved and not changed. This works the other way around, too. Having an image with blond hair, the "blond-vector" can be subtracted, which would produce an image of the same person with black hair. This can be applied to other features being present in the images. To find a concept-vector, first two sets of latent vectors have to be found. The first being the set of latent vectors that correspond to images having the specified feature and the other set corresponding to images without this feature. The concept-vector is then, the difference between the mean vectors in both sets. By adding or subtracting the calculated concept-vector to any other image a feature can be added or removed accordingly. The same technique works for the VAE [Whi16].

Figures 6.17 and 6.18 show the interpolations from a face with dark hair to the same face with blond hair after adding a concept-vector corresponding to blond hair to its latent vector. What's remarkable about this is that the addition of the "blond-vector" mostly only affects the hair colour and not the face. This does not work with every sample as good as shown in the figures. Some facial features might be changed by adding the "blond-vector". This is because concept-vectors often correlate to other features. For example, many women in the dataset have blond hair. In this case if one would subtract the "blond-vector" from a man with dark hair, the resulting image could transform into a woman. This of course depends on how much of the vector is added/subtracted from the latent vector and this effect can vary from sample to sample. Such radical change in features occurs when too much of a given concept-vector is added or subtracted, but shows that small amounts of a concept-vector may change other features as well.

Figure 6.19 shows the addition of the "blond-vector" to a recovered latent vector from the same image shown in 6.3. It can be seen, that the hair colour is successfully changed, but other face features are being changed as well, which is undesirable. This might be

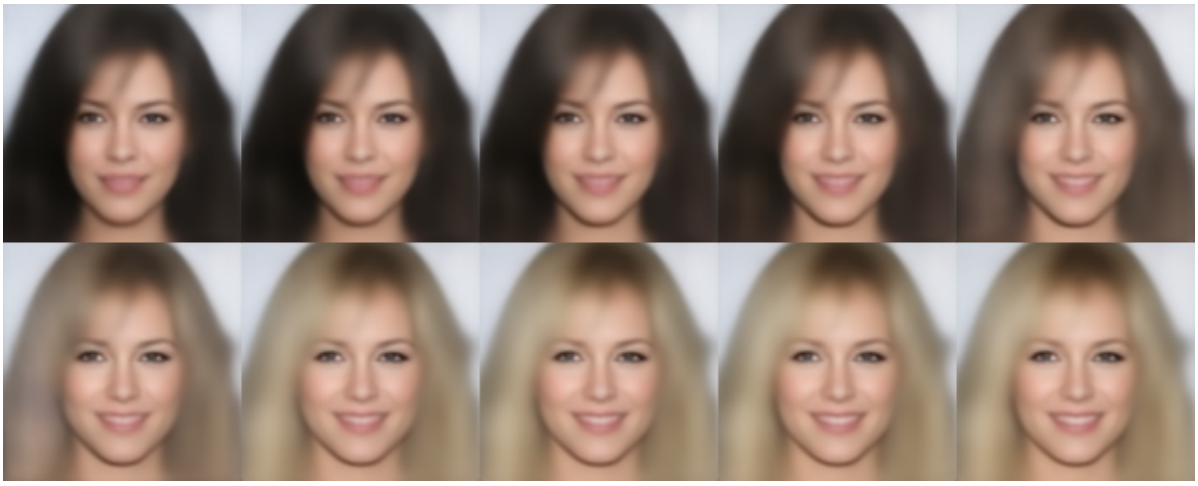


Figure 6.17: Interpolation between a sample generated by the VAE and the same sample with the "blond-vector" added to it

Figure 6.18: Interpolation between a sample generated by the GAN and the same sample with three times the "blond-vector" added to it

because of some correlations in the dataset or in the chosen images for the calculation of the mean vectors, as described above (a correlation might be for example, that blond women smile more, which would explain the change of the smile in figure 6.19).

For models that do not have a latent space, such arithmetics cannot be performed. However, there are other techniques that archive a similar result. In the case of the DMs one such technique is the so-called "Scribble-Guided Editing" [ALF22]. The idea here is to loosely draw the desired change onto the given image, perform a number of noising steps, and then let the model remove the noise. This way the model can reconstruct a realistic image from the loose drawing, with the features the editor intended the image

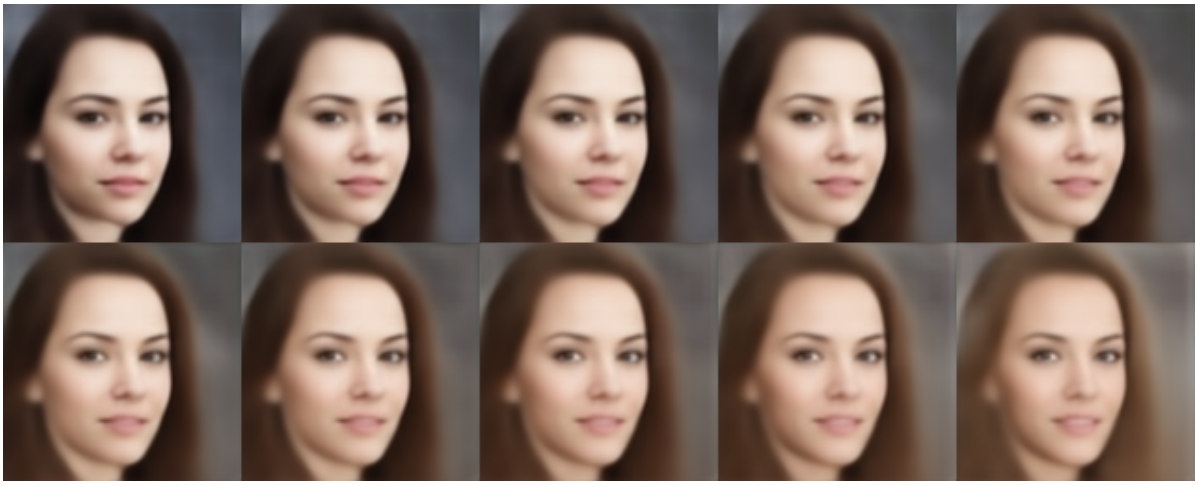


Figure 6.19: Interpolation between a reconstructed image (not generated nor contained in the dataset) by the VAE and the same reconstruction with 1.5 times the "blond-vector" added to its latent vector (Additionally the resulting vector has been normalized to the mean latent vector length and the interpolation has been performed spherically)

to have. For example, the hair colour change can be archived by taking an image and loosely drawing over the hair with a "blond" colour. The resulting image can be then noised and denoised by the model resulting in a person with blond hair. The number of noising steps is important, because too many steps may add too much noise and the added changes may get lost due to the noise and too few steps may not give the model a chance too correct the changes. Additionally, a mask might be specified, that marks the region, that should remain unchanged. Having such a mask the known region is then artificially inserted into every diffusion step, just as in the case of inpainting. This way it can be assured, that the important region is not changed by the model.

Figure 6.20 shows the preparation that has to be done in order to perform Scribble-Guided-Editing including painting the hair and specifying the mask for the face. Figure 6.21 shows the resulting image with different numbers of diffusion steps applied. As described before the more steps are applied the more likely it is, that what has been painted is being lost. This can be seen when noising the image using 500 and 400 steps where the hair is starting to lose the intended colour. On the other hand, noising using only 100 steps, the model does not have the chance to successfully "repair" the image. Noising using 300 steps seems to give a good result in this case.

In conclusion the VAE and GAN give pretty impressive results, as (in this case) the change of the hair colour barely affects the rest of the image, but require some preparation including sorting out many images with a given feature and images without this feature. However, once the mean vectors have been found, the change can be performed in real-

(a) (b) (c)

Figure 6.20: Preparation of image for Scribble-Guided-Editing: (a) Original [emi] (b) Painted blond hair (b) Mask for the region that should remain unchanged

Figure 6.21: Noising and denoising the edited image using 500, 400, 300, 200 and 100 steps having the mask to prevent changes on the face

time. The problem here with the GAN is however, that it only works on generated samples and not necessary on random images. If it would have generalized well enough to perform latent vector recovery on every image successfully, then it would be possible to do semantic changes to any picture, however then this could not be done in real-time as the latent vector recovery is an iterative process which includes backpropagation and gradient descent. With the VAE it is possible to perform such changes in real-time as there is an encoder which encodes images to a latent vector in one forward-pass. DMs on the other hand do not require major preparation other than drawing on the picture itself and marking a region that should remain unchanged. The noising and denoising process then takes some time, but does perform well on random images. The denoising time is comparable to the time of the latent vector recovery, which make the DMs better in this regard. Furthermore, DMs are able to reconstruct the drawn feature in many ways, as each diffusion run contains different random noise. The user is then able to choose one image from many possible reconstructions, which is not the case in VAEs and GANs where the only parameter that is being changed is the amount of a given concept-vector that is being added or removed (in case of the VAE one could argue, that because an image is being encoded into a distribution, there are potentially infinite

latent vectors that correspond, to this image, however, all these latent vectors are close to each other and these small changes do not affect the image significantly. Furthermore, the best latent vector for a given image can be found using latent vector recovery, which converges to one point in the latent space).

6.9 Implementation Difficulty

Because all the models have to be implemented and their hyperparameters have to be optimized, this too can be a criterium for choosing a model. The easier the model is to implement the more appealing it may be, as the development does not require long times and the model itself might be easier to debug.

With this regard here the easiest model to implement and to find good hyperparameters is the VAE. The VAEs training procedure is stable and does not diverge. Furthermore, it uses a simple loss function that contains widely known metrics. The hyperparameter optimization is pretty straight forward and does not require many trainings runs to find a good set of hyperparameters. This may be because of the inherent poor quality of the generated images, which makes differences in various models trained on the same dataset barely visible. Because of the limited capability of generating clear and sharp images, there may be many different combinations of hyperparameters, that produce similar results, which in turn makes them easier to find.

After the VAE the next model with regards to the implementation difficulty in this work is the DM. It has a simple training procedure, just as the VAE, however the sampling algorithm is more complex to implement. Additionally, exponential moving average has to be implemented, in order to get viable samples without having to scale the model unnecessarily. Choosing a good set of hyperparameters is not difficult either and there are existing implementations with chosen hyperparameters. Those these can be used and scaled in a way that the network can be trained on a given device.

The most difficult model to implement is the GAN. A GAN on its own in the most basic form does barely work and requires additional mechanisms to be implemented in order to get viable samples. Some of them are listed in section 4.3. These mechanisms make the implementation process more complex and prone for errors, which can be cumbersome especially in context of machine learning, because in order for some of the errors to be found the whole model needs to be trained, which can take a lot of time. An example for this would be a calculation error in the loss function. Such an error would be visible in the poor results of the model. Because in the early stages of the training poor samples are expected anyways, the model has to be trained for a longer period of time in order to be able to detect that an error is present. Because the GAN has the most complex

loss function and also other components of the network are complex, such problems may occur more often than they do in other models.

In order to debug a model, a small version of the model may be trained, in order to get quick results, but the "smaller" model does have to produce images of enough resolution, such that the results can be accessed by a human. This on the other hand requires a certain number of learnable parameters, which in turn makes the training process and also the development cycle slower. Furthermore, if a set of hyperparameters is found for a small model it does not necessarily mean that scaling them up linearly will work for images of larger resolution. This problem could be observed especially in the case of the GAN and the DM.

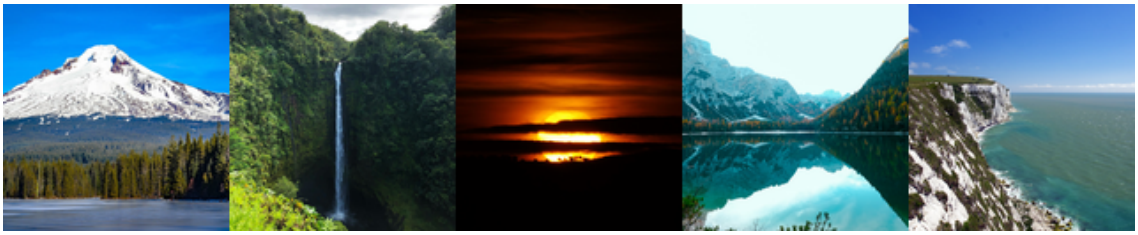
6.10 Other Dataset

In order to see how the models perform a different dataset, they have been trained on a dataset with landscape images. In these tests, the same hyperparameters have been used as for the CelebA-HQ dataset with the face images. In particular the dataset from [SSE21] has been used, which contains 90.000 images of various landscapes including sunsets, waterfalls, beaches or mountains. The key difference to the CelebA-HQ dataset is that the images don't follow a set structure as the faces do (e.g. all faces have the same size and are positioned in the same place in the image), which may make the landscape dataset more difficult to learn. Figure 6.10 shows a few generated samples, together with the FID-scores, after the models have been trained with the landscape dataset.

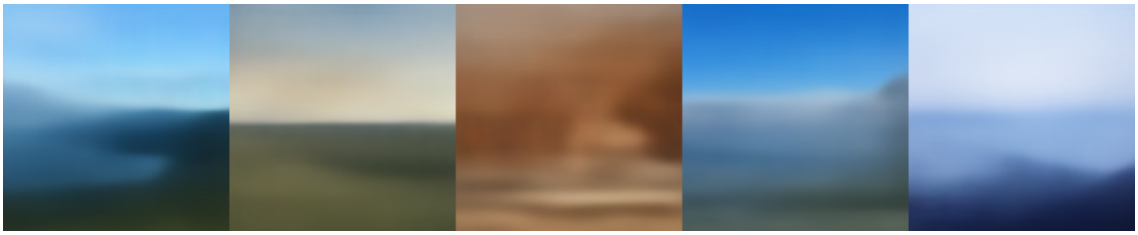
Because the hyperparameters have not been optimized for this particular dataset the FID-scores are overall higher than in the case of the CelebA-HQ dataset, but the order stays the same, with the DM having the best score and the VAE the worst. As with the faces, the VAE does generate blurry images, which seems to be more impactful than in the case of the faces. This might be, because the faces follow the same structure in every image, which the model can learn and produce sharper details in these areas. In the case of the landscapes dataset there is no such region that is being repeated, which might be the reason why the whole image is overall more blurry.

The GAN performs significantly better than the VAE and produces valid images. However, it seems, that the GAN has problems with fine details. When looking at the images from far away they seem real, but when zooming in the fine details might seem to be wrong.

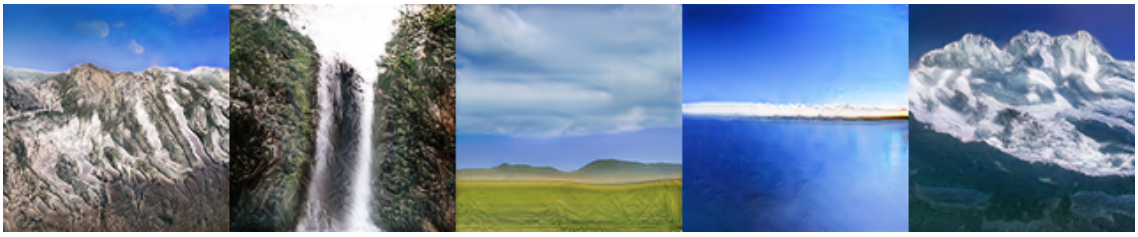
This is not the case for the DM, as the details seem to be fine. As with the CelebA-HQ dataset, out of the three models, here the results seem to be the best (most realistic) and they are convincing from a human observation perspective.



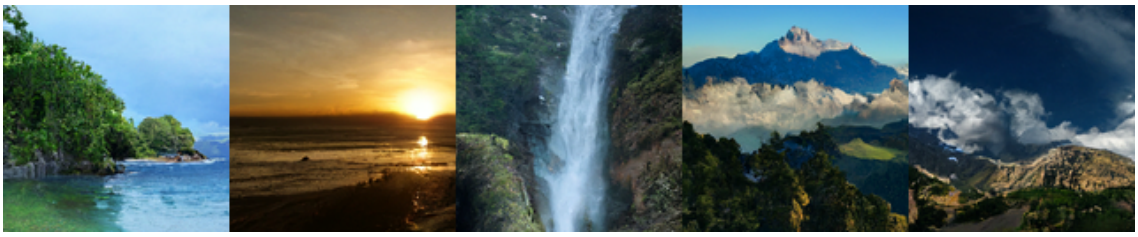
(a) Samples from the landscapes dataset [SSE21]



(b) Samples generated by the VAE (FID: 253.438)



(c) Samples generated by the GAN (FID: 30.624)



(d) Samples generated by the DM (FID: 20.548)

6.11 General Reflection about the Comparison

Although this work compares the VAE, GAN and the DM with respect to various criteria it has to be said, that in many cases only few samples have been presented to compare the models against each other. This of course is not enough to make reliable conclusions, but it gives an idea on how the models perform with respect to different tasks. Especially, where the difference in the results is large among the models it gives a stronger idea what the models are capable of and how they perform when compared to each other.

Each of the models that have been presented have been implemented and their hyperpa-

rameters have been optimized. However, it cannot be ruled out, that better combinations of hyperparameters exist, that would outperform the presented models. Nonetheless, it seems, that these models do perform sufficiently well, to be able to see differences among them, as the generated images have a high quality and the scores are comparable with the literature [GPAM+14] [HJA20].

It has to be mentioned as well, that for any specific scenario a special model can be implemented, that may be superior to any of the presented models. Such specialized models do exist and include for example the StyleGAN [KLA19] or CycleGAN [ZPIE20], which are designed with a special purpose in mind. However, in order to be able to implement such specialized models, it is good to know which models do exist such that the most adequate one can be chosen as a basis for building a model for a specific purpose.

VAEs, GANs and DMs are obviously not the only models designed for the generation of images (or other data) or for tasks similar to the ones presented in this work. There are other models and also hybrid approaches (e.g. VAE/GAN [LSLW16]), that could potentially outperform the presented models. Also, the research community continuously develops new models or improvements of the presented models (e.g. Denoising Diffusion Implicit Models [SME22]). However often the tackled problems are not totally solved, but only improved (e.g. Denoising Diffusion Implicit Models have still an iterative process, that needs longer than e.g. a GAN).

7 Conclusions

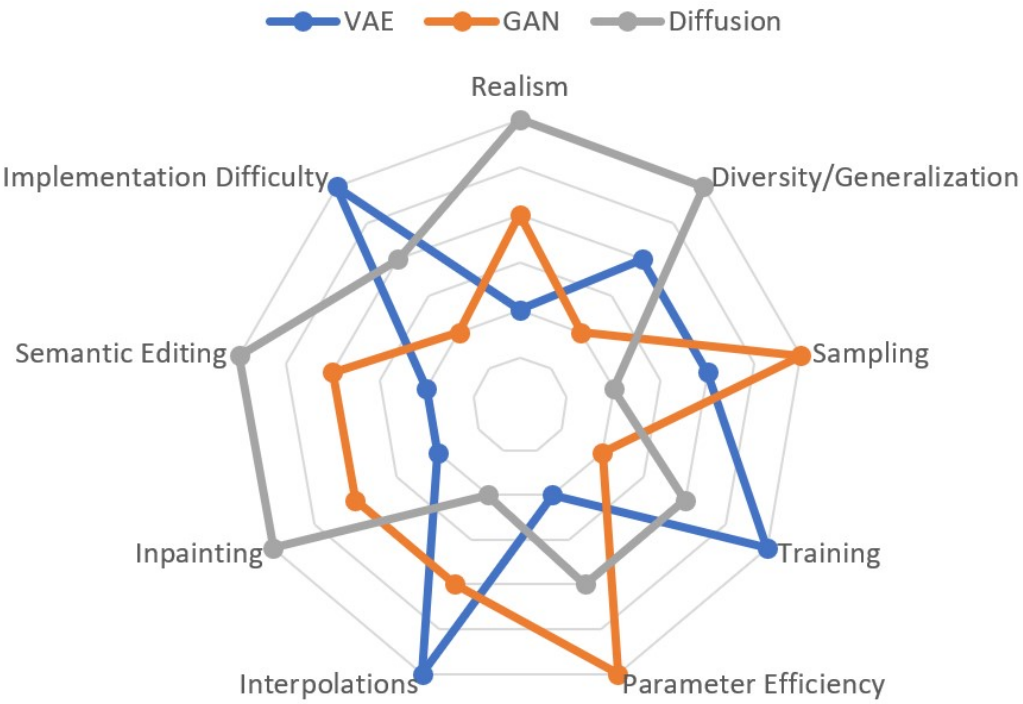


Figure 7.1: Conclusion of the comparizon of the three models (ordinal scale: the further away from the center the better)

In this work, three different generative models have been presented, analysed and compared with each other: 1. Variational Autoencoders, 2. Generative Adversarial Networks and 3. Diffusion Models. In particular, the behaviour and performance of the models have been analysed with respect to the following criteria: realism, generalization and diversity, sampling, training difficulty, parameter efficiency, interpolating and inpainting capabilities, semantic editing as well as implementation difficulty. The results of the comparison can be summarized in a diagram like the one presented in figure 7.1. Although it does not serve as a definitive reference on which model to use in which particular scenario it gives an overview on the strengths and weaknesses of the three models based on the previous findings and summarizes it in a compact manner.

Although each model has its strengths and weaknesses, what stands out is the flexibility and the quality of the DMs, which probably caused the results to be known in the public. Not only do DMs generate images with great quality, but are able to fill in pictures they have never seen before and repair images with hand drawn changes realistically and in many variations without any specialized architectures for such tasks. The main drawback is that it takes time to do so. Although there are improved DMs [SME22], that do generate images faster, than the one presented here, they are still of iterative nature which makes them slower than GANs or VAEs.

The GAN and the VAE behave almost in the same way with regards to criteria such as: interpolating, inpainting or semantic editing. They differ in that the GANs offer better quality and the VAE better mode coverage. It is especially surprizing how well a GAN or VAE is able to change the hair colour without changing the remaining structure and features of the given image, which is not the case for the DMs. Although the latter can produce many variations of the same picture with different hair colour and important regions can be fixed, it is hard to find one where the structure of the hair is preserved as good as in the case of GANs. Furthermore, GANs and VAEs are superior over the DMs with regards to interpolations, which may be because they have a latent space and generate images in one forward-pass of the respective network.

The VAE may seem as the worst model of the three, as it generates blurry images. However, it stands out with regards to the quality of the worst images. While the DM and the GAN can generate completely black images or ones with heavy distortions, it is hard to find an image among the samples of the VAE where the face is not recognizable.

All in all, it is pretty impressive, what has become possible in the recent years with regards to generative modelling and it may go even further as these models do not serve only as academic research, but have economic value as well, which can be seen on the wide range of applications of such models and their wide popularity in the general public.

Bibliography

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[ALF22] Omri Avrahami, Dani Lischinski, and Ohad Fried. Blended diffusion for text-driven editing of natural images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18208–18218, June 2022.

[Bui] CNN Buisness. 'i wish i did look like this': See user reactions to viral, new beauty filter. `https://edition.cnn.com/videos/business/2023/03/07/bold-glamour-tiktok-orig-contd-fj.cnn-business`. Accessed: 27.07.2023.

[Cho18] François Chollet. *Deep Learning with Python*. Manning Publications, 2018.

[CRM+21] Zhaowei Cai, Avinash Ravichandran, Subhransu Maji, Charless Fowlkes, Zhuowen Tu, and Stefano Soatto. Exponential moving average normalization for self-supervised and semi-supervised learning, 2021.

[DM20] Yilun Du and Igor Mordatch. Implicit generation and generalization in energy-based models, 2020.

[Elg19] Mohamed Elgendy. *Deep Learning for Vision Systems*. Manning Publications, 2019.

[emi] Image. `https://gameofthrones.fandom.com/de/wiki/Emilia_Clarke`. Accessed: 09.10.2023.

[EUD17] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning, 2017.

[Fos19] David Foster. *Generative Deep Learning*. O'Reilly Media, Inc., 2019.

[Gé19] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc., second edition edition, 2019.

[GAA+17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GBGM23] Roberto Gozalo-Brizuela and Eduardo C. Garrido-Merchán. A survey of generative ai applications, 2023.

[GBY+19] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning, 2019.

[GPAM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[HHAC+21] Gabriel Hermosilla, Diego-Ignacio Henriquez, Héctor Allende-Cid, Gonzalo Farias, and Esteban Vera. Thermal face generation using stylegan. *IEEE Access*, PP:1–1, 06 2021.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[HRU+18] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[Hua23] Kalley Huang. Why pope francis is the star of a.i.-generated photos. 04 2023.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[KALL18] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018.

[KLA19] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[KW22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[KZZ+23] Minguk Kang, Jun-Yan Zhu, Richard Zhang, Jaesik Park, Eli Shechtman, Sylvain Paris, and Taesung Park. Scaling up gans for text-to-image synthesis, 2023.

[LSLW16] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric, 2016.

[Mid22] Midjourney. https://www.midjourney.com/, 2022.

[Mil] Zosha Millman. Yes, secret invasion's opening credits scene is ai-made — here's why. https://www.polygon.com/23767640/ai-mcu-secret-invasion-opening-credits. Accessed: 01.08.2023.

[Mur22] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[Mur23] Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[Pho] Photoshop support page. https://helpx.adobe.com/support/photoshop.html. Accessed: 27.07.2023.

[RDN+22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[RH21] Lars Ruthotto and Eldad Haber. An introduction to deep generative modeling, 2021.

[SC23] Divya Saxena and Jiannong Cao. Generative adversarial networks (gans survey): Challenges, solutions, and future directions, 2023.

[SGZ+16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[SLJ+14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[SME22] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2022.

[SSE21] Ivan Skorokhodov, Grigorii Sotnikov, and Mohamed Elhoseiny. Aligning latent and image spaces to connect the unconnectable, 2021.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[WBQFM21] Jeffrey Wen, Fabian Benitez-Quiroz, Qianli Feng, and Aleix Martinez. Diamond in the rough: Improving image realism by traversing the gan latent space, 2021.

[WH18] Yuxin Wu and Kaiming He. Group normalization, 2018.

[Whi16] Tom White. Sampling generative networks, 2016.

[WSLY22] Liwei Wang, Jiankun Sun, Xiong Luo, and Xi Yang. Transferable features from 1d-convolutional network for industrial malware classification. *Computer Modeling in Engineering and Sciences*, 130:1003–1016, 01 2022.

[ZCYS21] Xingjian Zhen, Rudrasis Chakraborty, Liu Yang, and Vikas Singh. Flow-based generative models for learning manifold to manifold mappings, 2021.

[ZPIE20] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.

[Zur] José Mendiola Zuriarrain. 'bold glamour' filter ignites tiktok: 'it should be illegal'. `https://english.elpais.com/science-tech/2023-03-02/bold-glamour-filter-ignites-tiktok-it-should-be-illegal.html`. Accessed: 27.07.2023.