GENERATING FACES WITH GENERATIVE ADVERSARIAL NETWORKS

GENEROVÁNÍ OBLIČEJŮ S POMOCÍ GENERATIVNÍCH NEURONOVÝCH SÍTÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR | DANIEL KONEČNÝ
AUTOR PRÁCE

SUPERVISOR | MARTIN KOLÁŘ, M.Sc.
VEDOUCÍ PRÁCE

BRNO 2020

Department of Computer Graphics and Multimedia (DCGM) Academic year 2019/2020

Bachelor's Thesis Specification

22319

Student:	**Konečný Daniel**
Programme:	Information Technology
Title:	**Generating Faces with Generative Adversarial Networks**
Category:	Artificial Intelligence

Assignment:

1. Seznamte se s problematikou GANů
2. Získejte vhodnou datovou sadu
3. Proveďte implementaci
4. Natrénujte několik variant modelů
5. Proveďte analýzu rozložení výsledků podmíněnou latentním rozložením

Recommended literature:

- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).
- Karras, T., Aila, T., Laine, S. and Lehtinen, J., 2017. Progressive growing of GANs for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

Requirements for the first semester:

- Body 1 až 3

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Kolář Martin, M.Sc.**
Head of Department:	Černocký Jan, doc. Dr. Ing.
Beginning of work:	November 1, 2019
Submission deadline:	May 28, 2020
Approval date:	May 21, 2020

Abstract

The goal of this thesis is generating color images of faces from randomly chosen high-dimensional vectors with Generative Adversarial Networks. The next task is to analyze input vectors based on the features of faces generated from those vectors. Three different models of Generative Adversarial Network are implemented, one for generating images of handwritten digits and other two for generating images of faces. Generated images show credible-looking faces, but recognizable from real ones with a human eye. Single dimensions of input vectors are analyzed with Student's t-test. Linear Discriminant Analysis is then used to project input vectors into subspaces where the classes of features are separable. Analysis of generated data proves that the input vector can be specifically chosen to generate an image of a face with requested features with probability up to 80 %. The main result of this thesis is a model of Generative Adversarial Network for generating images of faces. A tool for generating images of faces with chosen features is implemented too.

Abstrakt

Cílem této práce je generování barevných obrázků obličejů z náhodně určených vysokodimenzionálních vektorů pomocí generativních neuronových sítí. Dále se zabývá analýzou vstupních vektorů na základě příznaků obličejů z nich vygenerovaných. Je provedena implementace generativní neuronové sítě pro generování obrázků ručně psaných číslic, poté dalších dvou sítí pro generování obrázků obličejů. Vygenerované obrázky zobrazují věrohodně vypadající obličeje, lidské oko je však dokáže odlišit od fotek reálných osob. Analýza jednotlivých dimenzí vektorů je provedena pomocí Studentova t-testu. Dále jsou vstupní vektory promítnuty do podprostorů pomocí lineární diskriminační analýzy a jsou nalezeny rozdělovací hranice mezi třídami příznaků. Analýza generovaných dat dokazuje, že ovlivněním vstupního vektoru je možné docílit generování obrázku obličeje s požadovanými příznaky s pravděpodobností až 80 %. Hlavním výsledkem této práce je model generativní neuronové sítě určené pro generování obrázků obličejů. Dalším přínosem je nástroj pro generování obrázků obličejů na základě vybraných příznaků.

Keywords

machine learning, deep learning, neural network, generative model, generating, generative adversarial network, GAN, image, face

Klíčová slova

strojové učení, hluboké učení, neuronová síť, generativní model, generování, generativní neuronová síť, GAN, obraz, obličej

Reference

KONEČNÝ, Daniel. *Generating Faces With Generative Adversarial Networks.* Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Martin Kolář, M.Sc.

Generating Faces With Generative Adversarial Networks

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Martin Kolář, M.Sc. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Daniel Konečný
June 11, 2020

Acknowledgements

I would like to express my sincere gratitude to my supervisor Martin Kolář, M.Sc. for his valuable advice, professional guidance, and important feedback. I would also like to thank Mgr. Lada Konečná for helpful consultations of mathematical topics.

Contents

Chapter 1

Introduction

Generating artificial data has always been a very important part of the field of Artificial Intelligence. With the increase in computational power of computers and more data available for learning, the development of Neural Networks overall went through dramatic growth and it affected data generation as well. Although generative models have made a lot of advances in the past 10 years, there are still many challenges that are to be completed. The goal of this thesis is to implement several Generative Adversarial Networks for generating image data and use these working models to further explore how generating images from random noise works.

This thesis in chapter 2 first introduces the main principles behind Machine Learning and Neural Networks, which are the cornerstones for many of generative models. Then it enlightens the basics of image processing with Convolutional Neural Networks. After that, Generative Adversarial Networks (GANs), which are the generative models used in this thesis, are introduced and explained. A summary of image databases which can be very useful for training of Generative Adversarial Network is given at the end of the chapter. Their advantages and disadvantages are reviewed and three of them are chosen for implementation.

In chapter 3 are described the fundamentals of developing a GAN in `Python 3` with the use of the `TensorFlow` library. Multiple GANs were developed for this thesis. At first, their common parts are explained and then their specific features are tackled one by one. Implementation is described from the most basic GAN for generating images of handwritten digits through a simple GAN for generating images of faces to an advanced one for images of faces with higher resolution and more variability. Faces are very complex, they all have some similarities but many differences too. Face should also be vertically symmetrical. All of these aspects propose very difficult challenges for generative models but GANs are capable of completing them.

The following chapter 4 dives deeply into the analysis of latent space. In other words, random noise that is used as an input for a generator model in GAN is analyzed and used for generating data with specific features. As a first task, a small dataset is manually constructed from generated data. Then the standalone dimensions of the latent space are analyzed with Student's t-test. After that, the analysis of possible projections of the latent space into a subspace with Linear Discriminant Analysis is done. Next, a model for classifying face features is constructed. The final task of generating images of faces with specific features is done with the help of this classifying model. Lastly, a simple tool that allows a user to choose features and generate a face with them is introduced.

Chapter 2

Neural Networks in image processing

This chapter gives an introduction to Generative Adversarial Networks and it is based on the information from Ian Goodfellow's book Deep Learning [4]. At first, section 2.1 briefly informs about the beginnings of Artificial Intelligence and why Machine Learning became a very important part of it. Then, section 2.2 describes why Deep Learning is a successful approach to solving real-life problems. Neural Network as a typical example of Deep Learning model is introduced. When it comes to image data processing, special kind of models called Convolutional Neural Networks are used. Their basic functionality is explained together with convolution, essential operation of these networks.

After all elementary concepts that are standing behind Neural Networks are explained, Generative Adversarial Networks can be introduced in section 2.3. The architecture of these networks, training procedures, and a special case of generating images is described. At the end of the chapter in section 2.4, available image databases that could be used for training of the network are discussed.

2.1 Machine Learning

Since the computers were created, people wondered if they can think on their own, whether some sort of Artificial Intelligence (AI) can be created. The first tries in completing this task were focusing mainly on the knowledge base approach. Pieces of information were described with a formal language and logical inference rules were used to process them into an output. Describing the information was an unwieldy process that had to been done by human staff which was the main reason why this approach did not lead to any significant success.

There was a need to process the raw data with a machine automatically. This is where the term Machine Learning comes from. AI systems can acquire knowledge from raw data. An example of a simple machine learning algorithm is linear regression. Such an algorithm is provided with information about the data known as features and finds a correlation between them and various outcomes. However, features have to be interpreted and structured intelligently and this is not always the case when raw data are processed.

For many tasks, it is too difficult to determine the features that have to be extracted, for example, face detection in an image. A face can vary in many ways such as skin color, hair, or orientation in the image. These variations cannot be described with pixel values, it is necessary to use a more complex solution. Representation learning is an approach of using a machine-learning algorithm to determine the features from data automatically. Problems that often require a team of researches and months of work can be solved with a machine learning algorithm within a couple of hours to days.

But in many cases, it is necessary to extract features that can be very abstract. This can be done easily by a human brain but it is harder for a machine. When a person sees an image of another person's head, it can face sideways and still be recognized as a face. But when a machine sees it, the pixel representation of such face is very different. Therefore, the task of representation learning is becoming as difficult as the original problem. In the example mentioned, representation learning of facial features can be as hard as the face recognition itself.

2.2 Deep Learning

Deep Learning solves the problem of feature learning differently. At first, it learns the simple concepts in data and after that, it forms them into a more complex information. More complicated features are described using the simple information. Comparison of deep learning models with other machine learning approaches can be seen in Figure 2.1. Therefore, a nested hierarchy of concepts is created. If a graph of these concepts built upon each other is drawn, it will be deep. From this abstraction comes the name of this AI approach – deep learning.

The typical example of deep learning model is a deep feedforward network. One of the most simple cases is a multilayer perceptron. It consists of a visible layer that is used as an input, several hidden layers that extract the features from data and combine them and output layer that maps the features to specified results. Example of a simple deep multilayer perceptron in shown in Figure 2.2. Input layer is called visible because it contains information that are directly observable in the data. Hidden layers contain information that is not explicitly shown in the data and they have to determine which concepts are best to describe the relationships between observed samples. Another very important advantage of deep learning models is their ability to improve with increasing number of training iterations and amount of data it is provided with.

Neural Networks

Deep feedforward networks are models that let the information flow through forward to transform it into an output. This is the reason why they are called feedforward. They contain no feedback connections that would form a cyclic dependency. They are called networks because they represent several functions composed into an acyclic graph. Deep feedforward networks are also loosely inspired by neuroscience and that is where the word neural comes from. The term Neural Network (NN) stabilized as a different name for deep feedforward networks and it is used in the following sections and chapters.

Goal of Neural Network is to approximate some function f^*. In case of using a NN as a classifier, $y = f(\boldsymbol{x}; \boldsymbol{\Theta})$ maps a vector input $\boldsymbol{x}$ to a category y while $\boldsymbol{\Theta}$ are the parameters that result in a best approximation of f^*. Neural Networks consist of multiple layers connected into a chain and each of these layers can be represented by a function $f^{(i)}$ where i is the index of layer. NN with 3 layers can be then mathematically expressed as $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$.

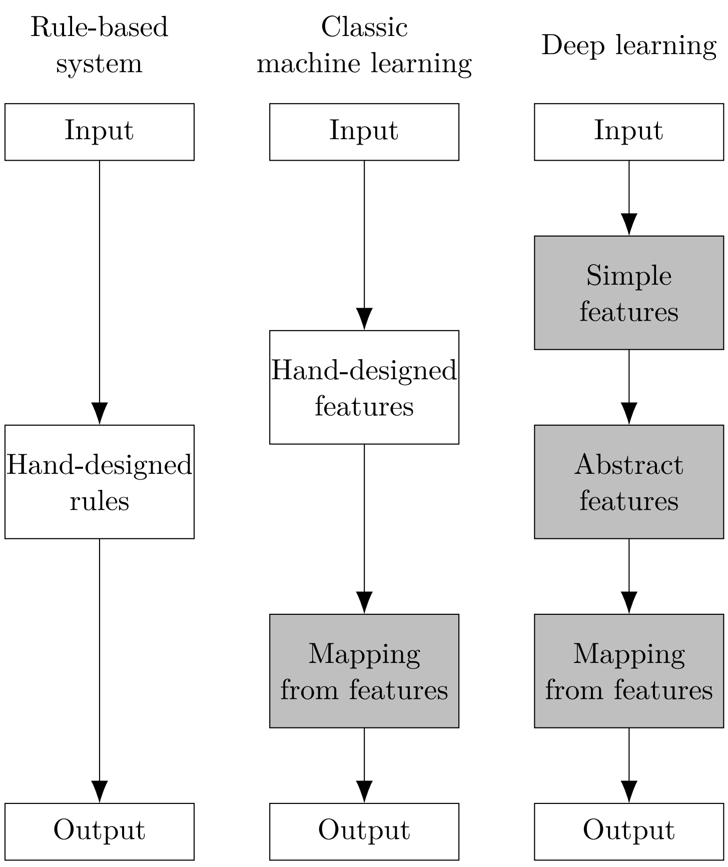


Figure 2.1: Comparison of different Artificial Intelligence approaches. Shaded boxes represent components that can learn from data.

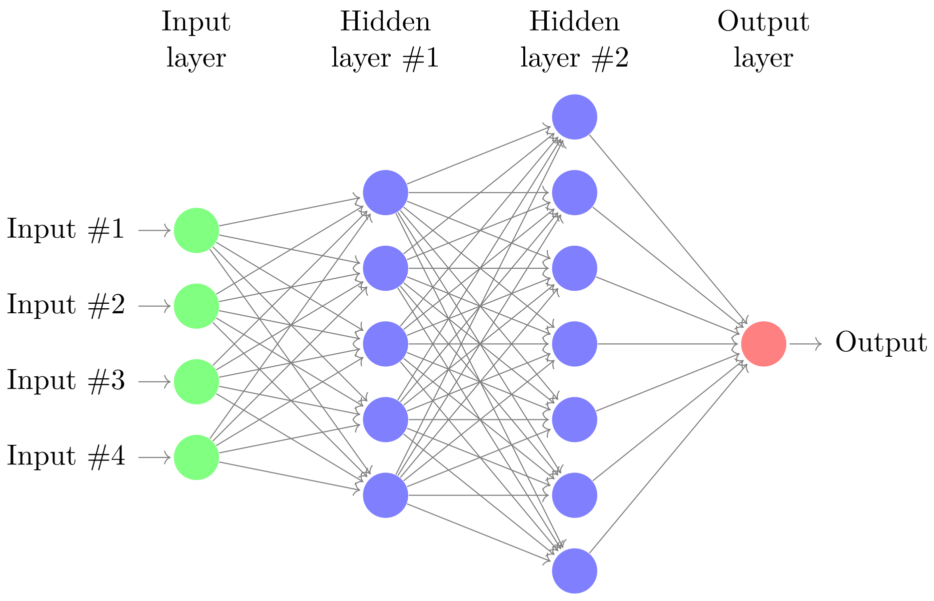


Figure 2.2: Architecture of a simple multilayer perceptron with an input layer, two hidden fully connected layers and an output layer.

Building blocks of layers are units. Each unit has a weight for each input, a bias and an activation function at the output. Example of a simple unit can be seen in Figure 2.3. During the feedforward process, features of data are set as inputs and fed forward through the network. Every unit from the first hidden layer receives a value from each unit of input layer that it is connected to. It multiplies these values by their respective weights and adds them together. After that, the bias is added too. The result is then passed through the activation function and fed forward to units of the following layer. Activation functions often serve as some non-linear transformation. The feedforward computation for a single unit is then $h = g(\boldsymbol{w}^\top \boldsymbol{x} + b)$, where g is an activation function, $\boldsymbol{w}$ is a vector of weights, $\boldsymbol{x}$ is a vector of input values and b is a bias. All vectors in formulas in this thesis are written in bold.

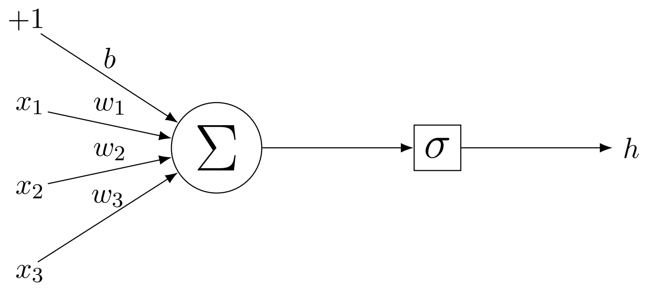


Figure 2.3: Simple unit (also called neuron) of a layer of Neural Network. It has 3 inputs, bias and a sigmoid function as an activation function.

It is important to have a function that can evaluate how good the approximation of f^* by f is. This function is called loss or cost function and it measures the performance of NN by computing the error between the predicted values and expected values. Neural Networks often use a measure called Cross-Entropy which computes the difference between two probability distributions. Binary Cross-Entropy calculates the average number of bits required to represent an event from a distribution Q by a distribution P, where Q is the predicted distribution (approximated) and P is the expected distribution (original) [14].

$$H(P, Q) = -\sum_{x \in X} P(x) \log_2 Q(x)$$

The approximation of f^* is called the training of the Neural Network. Parameters of NN that provide the best approximation cannot be calculated analytically, an optimization algorithm has to be used instead. Gradient descent is an example of iterative, gradient-based optimizer that is trying to minimize a loss function. Probably the most used optimization algorithm in Machine Learning, in general, is the Stochastic Gradient Descent (SGD). It uses a small number of samples called a minibatch to compute the gradient and it applies it to improve the performance of the NN. Gradient itself is just the direction of the step towards an optimum, an important part of SGD is also the size of the step. This size is called a learning rate and it can be fixed during the training. For a loss function L, gradient descent requires computing:

$$\nabla_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\Theta}} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\Theta}), \boldsymbol{y}^{(i)}).$$

Improved versions of SGD are used as optimization algorithms. One of the disadvantages of SGD is the fixed learning rate and the difficulty of setting it correctly. It has a significant impact on the training process. One way to improve the SGD is to use different learning rates for every parameter. Another way to improve it is by changing the learning rate throughout the training process. Both of these approaches are used in the Adam optimization algorithm that is used in this thesis. Adam algorithm uses the mean and uncentered variance of parameters to adapt the learning rates. It is computed as follows.

$$\begin{gathered}
\boldsymbol{s} = \rho_1 \boldsymbol{s} + (1-\rho_1)\boldsymbol{g} \\
\boldsymbol{r} = \rho_2 \boldsymbol{r} + (1-\rho_2)\boldsymbol{g} \odot \boldsymbol{g} \\
\hat{\boldsymbol{s}} = \frac{\boldsymbol{s}}{1-\rho_1} \\
\hat{\boldsymbol{r}} = \frac{\boldsymbol{r}}{1-\rho_2} \\
\Delta\boldsymbol{\Theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta} \\
\boldsymbol{\Theta} = \boldsymbol{\Theta} + \Delta\boldsymbol{\Theta}
\end{gathered}$$

Where:

$\boldsymbol{g}$ is the computed gradient,
ρ_1, ρ_2 are exponential decay rates for moment estimates (mean and variance),
$\odot$ is an element-wise product,
$\boldsymbol{s}$ is an updated biased first moment estimate,
$\boldsymbol{r}$ is an updated biased second moment estimate,
$\hat{\boldsymbol{s}}$ is a correct bias in the first moment,
$\hat{\boldsymbol{r}}$ is a correct bias in the second moment,
ϵ is a step size and
δ is a small constant used for numerical stabilization.

The whole set of training data called a dataset is divided into minibatches to best suit an SGD-based optimization algorithm. The size of such minibatch can be from lower tens to higher hundreds. Minibatches should be randomly chosen from the dataset to prevent overfitting of the model. The optimization algorithm computes the gradient from the whole minibatch, then it computes the update of the parameters and applies it. Other options are to use a whole dataset as a training batch but that might result in a premature convergence and requires a lot of memory for large datasets. Another approach is to update the parameters for every single sample but that is computationally less efficient and it might be more difficult for the model to reach the optimum.

The computing of gradients is done by the backpropagation algorithm. The algorithm computes the gradient of the loss function with respect to the parameters of NN. The gradient is calculated as a derivative of a function and since layers of the networks are composed functions, the chain rule of calculus can be used. It states that:

$$\frac{\mathrm{d}x}{\mathrm{d}z} = \frac{\mathrm{d}x}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}z}.$$

Which can be generalized beyond the scalar case with partial derivatives. Suppose that $\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{R}^n$, $g : \mathbb{R}^m \to \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$. If $\boldsymbol{y} = g(\boldsymbol{x})$ and $z = f(\boldsymbol{y})$, then:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}.$$

Or, written in vector notation as:

$$\nabla \boldsymbol{x}^{z} = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla \boldsymbol{y}^{z},$$

where $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ is the $n \times m$ Jacobian matrix of $\boldsymbol{g}$. This is how the gradient is computed for every layer of NN with the backpropagation algorithm.

Convolutional Neural Networks

Convolutional Neural Network (CNN) is a special kind of Neural Network that includes at least one layer that performs a mathematical operation called convolution instead of matrix multiplication. Specialization of these networks is data with grid-like topology, for example, time-series data that can be thought of as 1-dimensional grid or image data that are 2-dimensional grids of pixels. Such image data are used in this thesis as well.

In CNNs, convolution calculates a weighted sum, which means it takes a fixed number of values, multiples them with some constant and sums them together. This operation is done for the whole input. If the input is a 1-dimensional array, it starts at the beginning and computes the convolution for a fixed number of values. After that, it moves further in the 1-dimensional by some step and computes the convolution again. The result of this operation is the weighted sums arranged in a 1-dimensional array. Figure 2.4 displays an example of a 1-dimensional convolution.

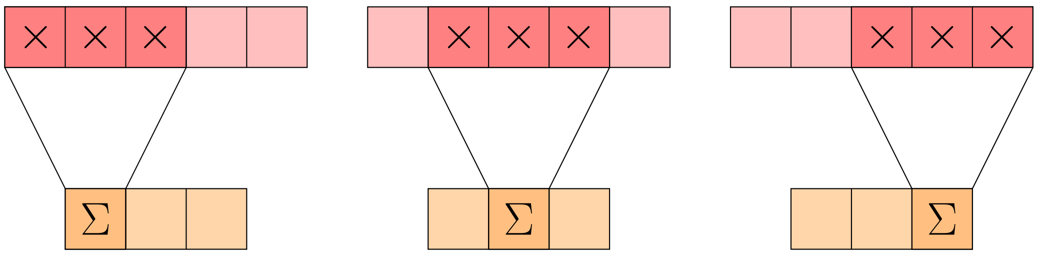


Figure 2.4: Example of 1-dimensional convolution with input size 5, kernel size 3 and step size 1.

The input of the convolution can be multidimensional. Weights used for the weighted sum are then arranged in a multidimensional array which is called a kernel. The output of the convolution is referred to as a feature map and it is multidimensional as well. The operation of convolution is often denoted with an asterisk. For a 2-dimensional case with image I of size $m \times n$ and kernel K, the convolution is calculated as:

$$S(i, j) = (I * K)(i, j) = \sum_{m} \sum_{n} I(m, n) K(i - m, j - n).$$

Convolutional layers in comparison to normal layers in NN do not use matrix multiplication with weights that are trained. They perform convolution and instead of training weights, the neural network is trying to improve kernels to produce better results. Feature maps are then produced as an output.

The convolutional layer can be also used to perform downsampling. This can be achieved by setting the stride size to more than 1. The layer produces fewer values for the same input, approximately half of the values for stride size 2, third for stride of 3, and so on. Downsampling is often accompanied by an increasing number of kernels applied in one layer.

Each convolution performs a different filtering operation, therefore, kernels are referred to as filters. For an example of image processing, the image's resolution is lower but the number of channels it is represented with is higher. Channels of the image are in this example the feature maps produced by the convolutional layer.

An important part of the convolutional layer is how it deals with the edges of input. When the input is not padded, the feature map on the output will have a different size than the input (as can be seen in Figure 2.4) but it uses only valid values. Therefore, this option is often referred to as `valid`. To keep the size the same, input has to be padded with zeros (as can be seen in Figure 2.5). This option of padding is called `same`.

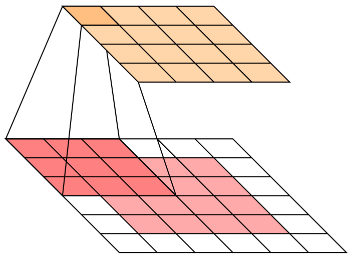


Figure 2.5: Example of 2-dimensional convolution with input size 4, kernel size 3, step size 1, and padding size 1. Feature map has the same size as input because of the padding.

The main advantage of CNN is that it can capture the context of pixel values, it does not only process independent values without any connection between them. Another big advantage is its low memory requirements in comparison to how advanced results it produces. In normal NN, weight is only applied to a single value and then never used again in the feedforward process. But the CNN uses weights in a kernel multiple times when applying it to different values of the layer input, for example to pixels of an image. This reduces the number of parameters needed for training while not affecting the runtime. Another important advantage is equivariance. When explained on an example for image processing, it means that when the convolutional layer detects edge in some part of an image, the same detection can be done in other parts of the image and both results will be the same.

2.3 Generative Adversarial Networks

Generative Adversarial Network (GAN) is a deep generative model. Its main goal is not classifying samples, as it was with previously mentioned models, but generating new samples. It is constructed from 2 specific models, one is used for generating new samples, therefore, it is called a generator. The other one is called discriminator and its task is evaluating the probability of the provided sample not being generated from the generator. Architecture of a GAN can be seen in Figure 2.6. This creates a scenario where the two models are competing against each other and improving while doing so. The generator is trying to produce samples that are as close to real data as possible, so the discriminator cannot distinguish them.

In this thesis, both of the models that are representing GAN are Neural Networks. There are multiple reasons to use those. As it is mentioned in section 2.2, Neural Networks perform very well when it comes to samples whose features cannot be easily described. The feedforward process allows for a simple generating process and the backpropagation algorithm lets the network trained very efficiently. GANs in this thesis are using random

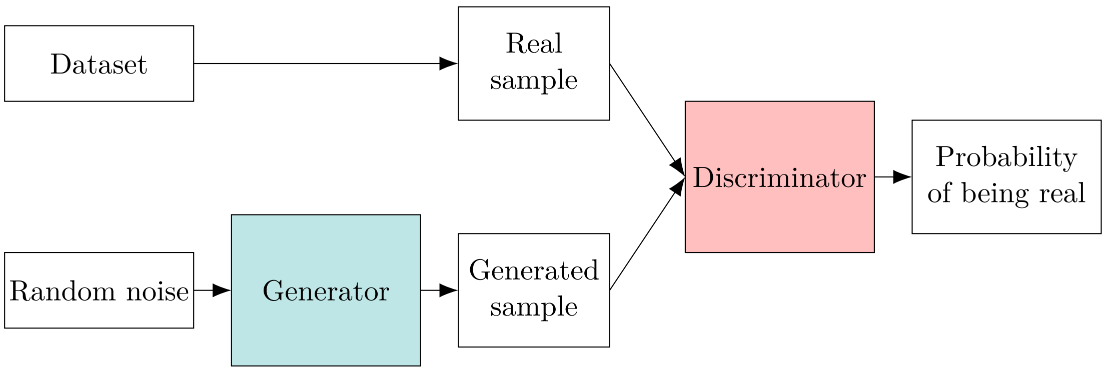


Figure 2.6: The architecture of Generative Adversarial Network where the generator produces images and the discriminator classifies them.

noise as input for the generator. The goal of the generator is then to learn how to map the latent space from which the random noise comes to real data.

The generator can be described as function $G(\boldsymbol{z}; \boldsymbol{\Theta}_G)$, where $\boldsymbol{z}$ is the random noise on the input and $\boldsymbol{\Theta}_G$ are parameters of the NN. The output of the generator is sample $\boldsymbol{x}$. The discriminator is represented by function $D(\boldsymbol{x}; \boldsymbol{\Theta}_D)$, where $\boldsymbol{x}$ is a sample on the input and $\boldsymbol{\Theta}_D$ are parameters of the NN [5]. The output of the discriminator is a single scalar, probability of the sample being real and not generated by the generator. The discriminator is trained to assign a correct label to both real and generated samples, that means maximizing of $\log D(\boldsymbol{x})$. The generator is simultaneously trained to minimize $\log(1 - D(G(\boldsymbol{z}))$. The training can be described as two-player minimax game with value function $V(D, G)$, where $\mathbb{E}_{x \sim P}[f(x)]$ is the expectation of $f(x)$ with respect to $P(x)$.

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

It is very important to carefully select the architecture of both models and their hyperparameters, otherwise the training of GAN can be unstable. It is essential to use dropout in the network architectures. Units should be stochastically dropped while computing the gradient for the generator to produce good results. GANs do not memorize the training dataset, they produce samples that were not part of the dataset neither they were close to training samples [5].

One option to train a GAN is described in this paragraph. The procedure can be divided into two parts, training of discriminator and training of generator. The discriminator is trained individually on a batch of samples, half of the batch is real training samples and the other half is generated samples. Samples are labeled to distinguish real from generated ones. The generator is trained in the whole GAN model but the discriminator's weights are not updated during this part of the training. On the input of the generator is a batch of random values. Samples are generated from the random values in the batch and evaluated by the discriminator. Error then backpropagates through the discriminator and updates the weights of the generator.

Generative Adversarial Networks for generating images

Generating of image data is a task that can be very useful in real-life applications, therefore, it is a field worth exploring with Generative Adversarial Networks. Convolutional Neural

Networks are used in most of the image classifying tasks and they produce good results when used in GANs too. A specific approach for designing of discriminator and generator model was used in this thesis and it is described in the following paragraphs.

Discriminator consists of multiple 2-dimensional convolutional layers that always reduce the resolution of the image to half while increasing the number of channels image is represented with. The generator is a mirrored version of the discriminator. It uses 2-dimensional transpose convolutional layers that double the resolution of an image while lowering the number of channels, as is shown in Figure 2.7. This architecture of both networks ensures that main features are learned as well as features representing small details [8].

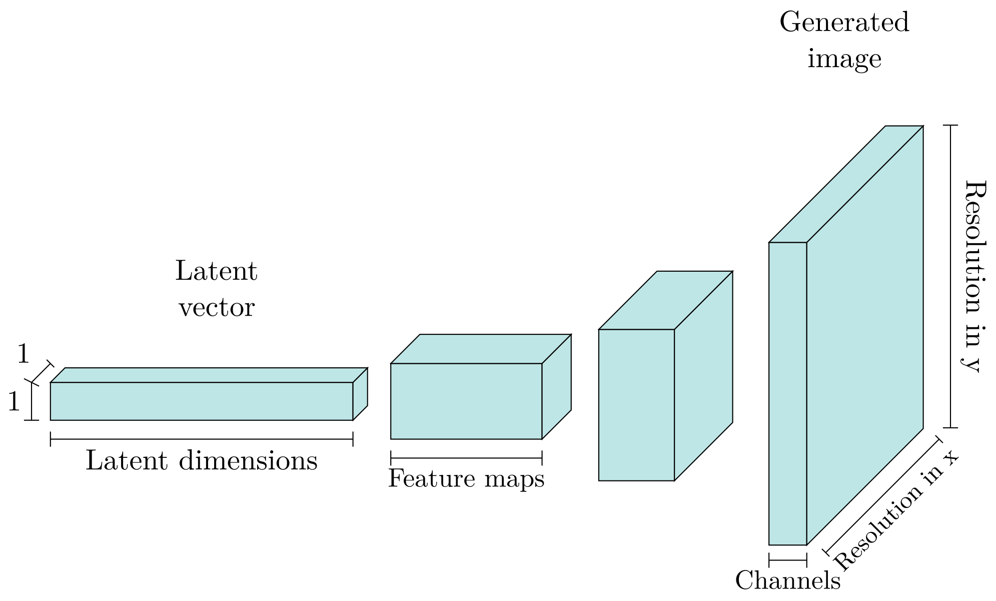


Figure 2.7: Feedforward process with data in generator with transpose convolution that gradually increases the image's resolution while lowering the number of image's representations.

Transpose convolution is computed almost the same way as a normal convolution. The difference is that it first adds a zero padding to the image [3]. In case of doubling the resolution by using the stride size 2, zeros are added to the edges as well as between all the pixels. Edges may not be padded evenly if an image of exactly twice as high resolution should be produced. Then a normal convolution with stride 1 and given kernel is performed and a feature map with higher resolution is produced. Figure 2.8 provides an example of a transposed convolution.

2.4 Image databases

Image classification is an often implemented task in practice but also for educational purposes. For that reason, there are many databases of labeled images available online for anyone to use. Generative Adversarial Networks can be trained on any images and do not even require the images to be labeled. The only requirement is an equal resolution of all images and preferably also some common features between all images. A database can contain photographs of animals, or photographs of vehicles for example, but not both of these categories.

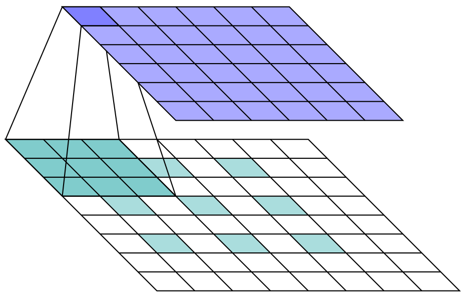


Figure 2.8: Transposed convolution used to double the size of image input. Input of size 3 is padded with zeros on the edges as well as between the pixels. The edges have to be padded unevenly. Input is then convolved with a kernel of size 3 with stride size 1, and the size of the feature map will be 6. Notice that the stride size of this convolution is 1 but in the settings of the transposed convolution would be the stride size 2.

Handwritten digits

Modified National Institute of Standards and Technology (MNIST) database is one of the most commonly used databases for educational purposes in image processing [11]. It contains images of handwritten digits from 0 to 9 that are scaled and cropped to have a similar size and be positioned in the middle of the image as can be seen in Figure 2.9. No preprocessing of images has to be done beforehand. Images have a resolution of 28×28 pixels and are grayscale, which means only one value represents every pixel. In most cases, this value is an unsigned integer in the range between 0 and 255. Each image is labeled with the value of its digits, which gives 10 classes in total. The whole database contains 70000 samples from which 60000 are meant for training and 10000 for testing of the trained model.

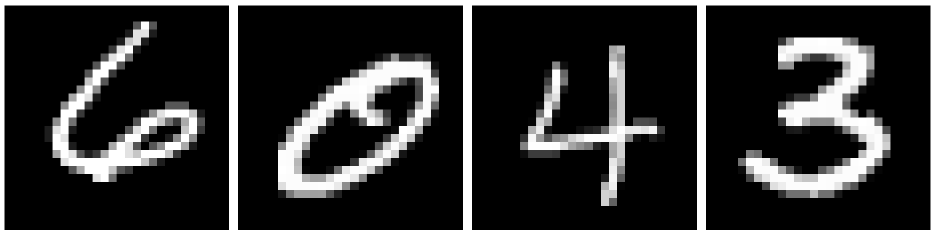


Figure 2.9: Examples of different digits from the MNIST digit database.

MNIST database is an ideal choice when starting the implementation of image processing neural network. Low resolution and only one channel of each image make the task as easy as possible. The training time is short and the neural network can be constructed from a relatively small number of layers (under ten). Therefore, it is not necessary to use a lot of computational power. Another advantage is the small size of images, they can be loaded into memory and used at the same time without reloading. Images of the same digit can be different because they are all handwritten. This is shown in Figure 2.10. Such variety presents a difficult enough challenge for the neural network.

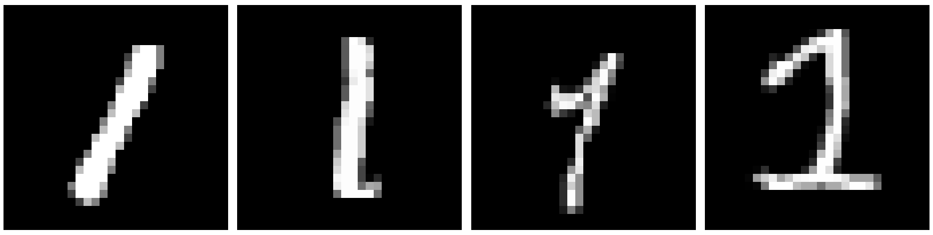


Figure 2.10: Examples of identical digits from the MNIST digit database. Even the same digits can be written in different ways.

Labeled Faces in the Wild

Labeled Faces in the Wild (LFW) is a public database of color images of faces collected from the web [7]. The database contains 13233 photos of 5749 people with 1680 out of them on two or more photos. Each image is labeled with the name of the person on it. The resolution of original images is 250×250 pixels and each pixel is represented by 3 channels (red, green, and blue). Examples from the LFW database can be seen in Figure 2.11.

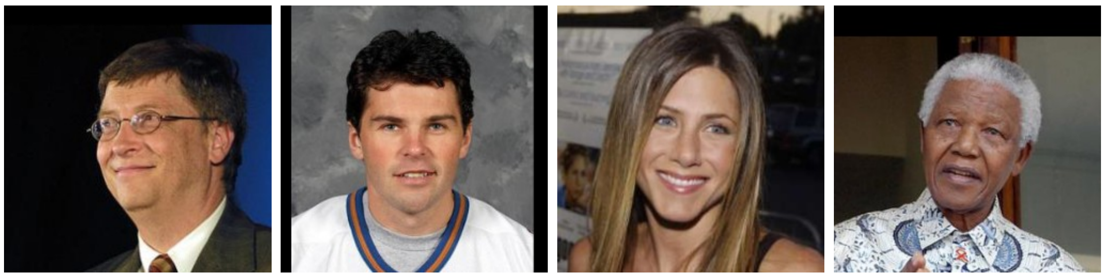


Figure 2.11: Examples from LFW database.

Faces are located in the center of the picture and can point also partially sideways, not only to the front. Peoples' faces can be covered with headwear or glasses. As the database's name suggests, images have a large variety of backgrounds behind the photographed people. All of these properties make the task of image processing more challenging but also closer to real-life problems.

LFW database is good for research and education but should not be used for commercial purposes. The reason is not so high variability of the people in the images. There are very few children, no babies, almost no people over 80 years old, and a relatively small amount of women. Also, many ethnicities are represented in a small portion or not at all. The size of the database is also not big enough for strong statistical conclusions about subgroups.

Faces from the LFW database can be used as a next step in developing an image processing neural network. However, they should not be used as a training database for the finalized network because of the reasons mentioned.

Flickr-Faces-HQ

Flickr-Faces-HQ (FFHQ) is a database of high-quality images of faces created as a benchmark for Generative Adversarial Networks [9]. Images were extracted from photographs uploaded to service Flickr and then aligned, cropped and saved as `png` files. Only images with licenses for free use for non-commercial purposes were used for this database. The size of the database is 70000 images with a resolution of 1024×1024 pixels. Creators of this database offer also thumbnails of each image with a resolution of 128×128 pixels and original images from which the faces where extracted.

FFHQ database contains photographs of people with a wide variety of ages, ethnicity, and image background. Also, people wearing different kinds of headwear and glasses are included as shown in Figure 2.12. Faces are photographed from different angles and in different light conditions. These properties make the database ideal for the training of image processing neural networks.

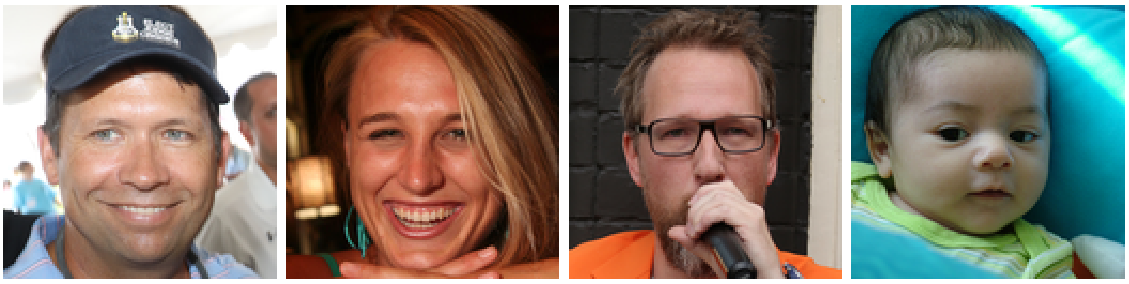


Figure 2.12: Examples from FFHQ database.

Other databases of faces

CelebFaces Attributes (CelebA) database contains 202599 photographs of celebrities [12]. Each image has 40 binary attribute annotations, such as the person's facial expression and shape of the face, whether the person is wearing glasses, hat or other accessories, and other features of a face. It contains a large variety of poses and backgrounds of images.

There is also a large number of databases that have a wide variety of images sorted into classes. For the training of GANs, only one class can be used at a time. Some of these databases are described in the following paragraphs.

ImageNet is an image database that is organized into classes according to the WordNet[1] hierarchy [1]. It contains about 14 million images from 100000 classes that are supposed to serve for research and educational purposes.

LSUN database contains around one million images of all kinds [17]. They are labeled for each of 10 scene categories and 20 object categories. It uses a crowdsourcing platform for labeling the images and therefore can grow in size.

The CIFAR-10 and CIFAR-100 are databases with a large variety of images sorted into 10 and 100 classes respectively [10]. They both contain 60000 color images with a resolution of 32×32 pixels. Images in this database are subsets of the 80 million tiny images database [16].

[1]WordNet®is a large lexical database of English, more information at https://wordnet.princeton.edu/

Chapter 3

Implementation of Generative Adversarial Network

All the Generative Adversarial Networks (GANs) mentioned in this thesis are implemented in `Python 3` with `TensorFlow 2` library through `Keras` API, `NumPy` library for advanced mathematical operations, and other libraries for specific smaller tasks. `TensorFlow` is a deep learning `Python` library and it provides users with an easy way to design, train, and evaluate neural networks. Very important is also an option to predict with the neural network, which is also the case for generating data with GAN. Using `Keras` API on top of the `TensorFlow` library makes the building of neural networks more intuitive.

The code for every GAN usually consists of several similar parts. Common properties of each part are described in section 3.1. In the following sections are mentioned differences for each specific one. The first implemented GAN in this thesis generates images of handwritten digits and it is described in section 3.2. In the next section 3.3, are explained the differences in implementation of a GAN for images of faces from the LFW database. Finally, section 3.4 informs about the details of implementing GAN for high-quality images of faces from the FFHQ database.

3.1 Common segments of every Generative Adversarial Network

Training of every GAN has to start with the loading of the training data. In most cases, the data has to be also preprocessed in some way. Generator in GAN uses a random noise as an input, therefore, it is necessary to have a function that generates the random noise in a format that is accepted by the generator. The architecture of both discriminator and generator has also some common properties which are described in this section. They are always somehow connected into the whole GAN, in this thesis, it is done with another model. After the models are defined, they can be trained. It is necessary to monitor the training and evaluate the results to achieve the best possible generated images. When the training is finished, the generator can be used for generating new data.

Loading of training images

At first, images have to be loaded into memory and in many cases also preprocessed. The preparation of images can consist of changing the resolution and cropping the image that leads to the resolution being easily factorized to small prime numbers, which is very useful for GANs. The number of channels of an image can also be altered (for example converting a color image to grayscale). Training data, in this case, an image database, are usually referred to as a dataset.

Neural networks process inputs with weights with small values [4], therefore, it is a good practice to represent channel of each pixel by a value between 0 and 1, otherwise the weights can increase to large numbers and make the network unstable and unable to generalize well. In many cases, conversion of input values must be done because of pixels being represented by unsigned integer values from 0 to 255.

Generating points in latent space

Points from latent space are necessary for GANs because they are used as an input for the generator. The dimensionality of latent space can vary from lower tens to higher hundreds. A certain power of number 2 is usually used as the number of dimensions. This point stores all the information about the generated result. The generator from trained GAN is used as a tool that deterministically generates the result from this point.

In this thesis, the latent vectors correspond to a random point on an n-dimensional hypersphere, where n is the dimensionality of the latent space. That means that squared values in all dimensions of one latent vector have to sum up to a chosen constant which is the hypersphere's radius squared. The radius of the hypersphere is set to 1 in this thesis. Muller's algorithm is used to produce uniform random values with this property [13]. The random point on a hypersphere is computed with the following formulas, where d is a dimension index, n is the dimensionality of the latent space, and Y_d is the computed coordinate in dimension d.

$$S = \sum_{d=0}^{n} X_d^2, \quad \text{where} \quad X_d \sim \mathcal{N}(0, 1)$$

$$Y_d = \frac{X_d}{\sqrt{S}}$$

Building the model of discriminator

The discriminator is a sequential model, which means its layers are saved in a linear stack. Input is an image and output is a value between 0 and 1 representing the probability that the image is real.

Very important layers in the model are 2-dimensional convolutional layers (Conv2D). The operation of convolution applies a filter to an image to obtain its feature map. Each feature map stores some information about the image. Filter used for obtaining each of all feature maps is the trained parameter of the convolutional layer. After the first convolution image is no longer represented with channels of image's resolution but with feature maps. These feature maps are then handled in the following layers.

Convolutional layers in discriminator decrease image's (and feature maps') resolution to half by using the stride of 2 and "same" padding. Kernel size can vary in different cases but the size of 3 is used in many cases [8]. The number of filters is usually a power of 2 in the range between 32 and 512. The more convolutions are applied the higher number of filters is used. That way more feature maps are obtained to represent the image.

The convolutional layer is followed by a layer with a leaky version of a Rectified Linear Unit (LeakyReLU) as an activation function.

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

This function is easy to compute because it consists of two linear parts but it can still learn complex structures in data because of its nonlinearity. These two properties make LeakyReLU a good activation function for deep neural networks. The steepness of the function for $x < 0$ can be set. An example of LeakyReLU can be seen in Figure 3.1.

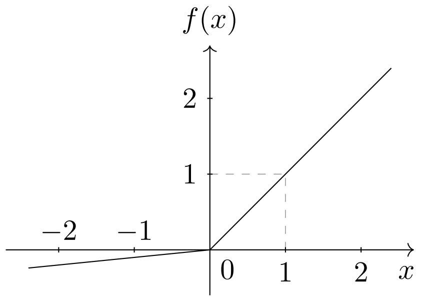


Figure 3.1: An example of Leaky Rectified Linear Unit.

After that is applied a layer that drops out a certain number of nodes. This layer is called Dropout and its drop out rate are set by a value between 0 and 1. It prevents the network from overfitting to the training samples.

The combination of the Conv2D layer, LeakyReLU activation function, and Dropout layer repeats multiple times. In the end, the image is transformed into a 1-dimensional list using the Flatten layer. This layer has no trainable weights, it is only used as a preparation for the following Dense layer. Dense means that it uses connections of all nodes between layers, in other words, it is fully connected. In this case, it connects all nodes from the Flatten layer to one node representing the probability of the input image being real. This value is normalized by the sigmoid activation function to values between 0 and 1. The shape of the sigmoid function is shown in Figure 3.2.

$$f(x) = \frac{1}{1 + e^{-x}}$$

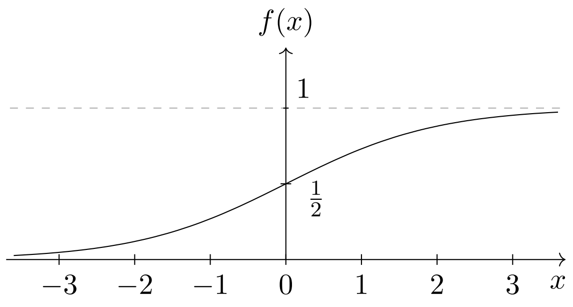


Figure 3.2: Sigmoid function.

The optimization algorithm used when training discriminator is Adam. This algorithm updates every weight with its own learning rate. Learning rates also adapt during the training by using the first and second moments of the gradient. Thanks to these advantages it is widely recommended to be used in Deep Neural Networks. For GANs in this thesis, the learning rate is set to 0.0002 and exponential decay rate for the first moment (mean) estimates to 0.5.

The loss function that is being minimized is the Cross-Entropy. Likelihood of image being real (belonging to class 1) or not (belonging to class 0) is a binary classification problem. Cross-Entropy is advised to be used for this type of problem.

Building the model of generator

The generator is also a sequential model and in most cases, it consists of multiple transpose convolution layers that are used to increase the resolution of the image. It is basically a mirrored version of the discriminator. Input is a point from latent space and output is a generated image.

Generator first makes many low-resolution representations of the desired image. The dense layer can be used to connect the input array of random numbers with the low-resolution representations of that image. LeakyReLU is applied as the following layer to improve the generalization of the model. After that, it is necessary to reshape the array to three dimensions – two for the image itself and one for all of the image representations. Reshape layers can be used for this purpose.

Two-dimensional transposed convolutional layers (Conv2DTranspose) are then applied to increase the resolution of the image. The transpose convolution is described in section 2.3. The input image can be thought of as a feature map and trained weights are actually filters of the transpose convolution. Doubling of resolution of an image is achieved by setting the stride to 2 and padding to “same”. The more transpose convolutions are used, the fewer filters are usually used, which means the number of image's representations is lower. The Conv2DTranspose layer is often followed by a LeakyReLU layer for generalization improvement.

When the desired resolution of the image is obtained, one convolutional layer (Conv2D) can be used to convert all the representations of an image to channels (one channel for grayscale images or three channels for color images). The number of filters is the number of channels in the final image, the stride is set to 1 and padding to “same” to keep the resolution of the image unchanged. The size of the kernel can vary in different situations. The sigmoid activation function is then applied to keep the pixel values between 0 and 1.

Connecting discriminator and generator together into one model

Generative Adversarial Network itself is represented by a sequential model as well. It consists of discriminator and generator by themselves. The discriminator is not trained in this model, only the generator's weights are updated according to the discriminator's performance on generated images. That means that after discriminator evaluates the probability of provided images being real, backpropagation goes through the discriminator and then generator but only updates the weights in the generator. Training optimization is done with the Adam algorithm, the learning rate is set to 0.0002 and exponential decay rate for the first moment (mean) estimates to 0.5. Binary Cross-Entropy is used as a loss function.

Training of Generative Adversarial Network

The training consists of a certain number of epochs (hundreds or thousands). Epoch is an iteration over the whole dataset of images. In each epoch, there is training done with images grouped into batches. The size of one batch can be in the lower hundreds. It is usually some power of two because these sizes can use hardware resources more efficiently [4]. The number of these batches used to train the GAN in one epoch is the number of available images for training divided by the size of the batch.

First comes the training of standalone discriminator. It is provided with a batch consisting of real and generated images and its task is to determine which one is real and which one is generated. This training of discriminator is done using the `train_on_batch` function from `Keras` API that updates the weights of the discriminator.

After that, the generator is trained in the whole GAN model. It provides the discriminator with a new batch of generated images. The discriminator determines whether the images are real or generated and then the backpropagation is used to update the generator weights. Once again, the `train_on_batch` function from `Keras` API is used for this task.

Both discriminator and generator are evaluated after each epoch. The discriminator is evaluated by recognizing real images, the generator is evaluated by discriminator's ability to recognize its generated images. Information about training loss of both models is printed after training on each batch.

Evaluation of results and monitoring of training

There is no objective way to automatically evaluate generated images, the human eye has to decide about their quality. There is still the option of reviewing generated images with the discriminator but it depends on how good the discriminator is in recognizing real and generated images from each other. Discriminator can be tested the same way by providing it with real images and expecting to evaluate them all as real. This evaluation can be done with `evaluate` function from `Keras` API.

It is very important to monitor the process of training to know if the model is making progress. This can be done by following the scalar training loss that is returned by `train_on_batch` function from `Keras` API during the training. After a few epochs of training, these values should be balanced. If, for example, training loss of discriminator approaches 0 and training loss of generator increases above 10 and further, the model is probably not improving. Its design or training process should be changed to improve performance.

Generating new images from Generative Adversarial Network

New images are generated from the trained generator model (not the whole GAN). For this reason, it is necessary to save the whole generator model or its weights after the training. If only weights are saved, the exact same generator model has to be defined before loading the weights and generating new images. When the model is loaded, it has to be provided with a point from latent space and then a new image can be generated with the `predict` function from `Keras` API.

3.2 Generating images of handwritten digits

It is very useful to start implementing GAN for simpler images than photographs of faces. One of the very basic image databases is the Modified National Institute of Standard and Technology (MNIST) handwritten digit database. The resolution of these images is 28×28 pixels. Every image is also labeled with the number that the handwritten digit represents. Further information about the MNIST database can be found in section 2.4.

Loading and preparing images

MNIST digit database can be obtained from the `TensorFlow` library via `Keras` API using the `load_data` function. The dataset comes in two lists (train and test images) with shape (*number of images*, 28, 28) together with the labels in two additional lists. There is no need for labels when training a GAN, the learning process is unsupervised. Only images for training are needed, testing and training is done simultaneously using newly generated images and the training images. Therefore the sets of 60000 training images and 10000 images can be merged into one and used.

Each pixel is represented with an unsigned integer value from 0 to 255. As mentioned in section 3.1, it is better practice to convert the values to floats and normalize them to range between 0 and 1. It is also necessary to update the number of dimensions to have the grayscale channel as an additional dimension. The shape of the training dataset is now (70000, 28, 28, 1) – the grayscale value of each pixel is in its list.

The size of the digits from the MNIST database is 28×28 pixels. Prime factors of number 28 are 2 and 7. And if the image is grown gradually, that means that at a certain point there has to a big step to make the image 7 times larger in pixel size. This big step is represented by many trainable weights in a layer of the generator. If a neural network has too many weights in one layer, the training can be time consuming and unstable.

To optimize this process, every image can be changed to a size of 32×32 pixels by just adding 2 pixels with 0 values on each side. This will not affect any of the results because each digit image already has 0 values on the sides. Resizing of images just by adding zeros is a very fast process that will not affect the training speed almost at all. Number 32 has only one prime factor and that is number 2. Therefore the image can be grown from 4×4 pixels up to 32×32 pixels just by small steps of making it 2 times larger. The generator neural network will have more layers but it will be able to generate more precise images and the training will be faster. The shape of the final training dataset will be (70000, 32, 32, 1).

Architecture of Generative Adversarial Network

This GAN uses latent space with 64 dimensions. The architecture of the discriminator model can be seen in Table 3.1 and the generator model in Table 3.2.

Layer	Settings	Shape	Parameters
Input		(32, 32, 1)	0
Conv2D	Filter count: 32	(32, 32, 32)	64
	Kernel size: (1, 1)		
	Stride size: (1, 1)		
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 32	(16, 16, 32)	9 248
	Kernel size: (3, 3)		
	Stride size: (2, 2)		
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 64	(8, 8, 64)	18 496
	Kernel size: (3, 3)		
	Stride size: (2, 2)		
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 64	(4, 4, 64)	36 928
	Kernel size: (3, 3)		
	Stride size: (2, 2)		
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Flatten		(1024)	0
Dense	Units: 1	(1)	1 025
	Activation: Sigmoid		
Total trainable parameters			65 761

Table 3.1: GAN for handwritten digits – layers of discriminator model.

Layer	Settings	Shape	Parameters
Input		(64)	0
Dense	Units: 1024	(1024)	66 560
LeakyReLU	Alpha: 0.2		0
Reshape		(4, 4, 64)	0
Conv2DTranspose	Filter count: 64 Kernel size: (3, 3) Stride size: (2, 2)	(8, 8, 64)	36 928
LeakyReLU	Alpha: 0.2		0
Conv2DTranspose	Filter count: 32 Kernel size: (3, 3) Stride size: (2, 2)	(16, 16, 32)	18 464
LeakyReLU	Alpha: 0.2		0
Conv2DTranspose	Filter count: 32 Kernel size: (3, 3) Stride size: (2, 2)	(32, 32, 32)	9 248
LeakyReLU	Alpha: 0.2		0
Conv2D	Filter count: 1 Kernel size: (1, 1) Stride size: (1, 1) Activation: Sigmoid	(32, 32, 1)	33
Total trainable parameters			131 233

Table 3.2: GAN for handwritten digits – layers of generator model.

Training of Generative Adversarial Network

The batch size chosen for training is 256. When training the discriminator, half of the batch are real images and the other half are generated images. The real images are chosen randomly from the dataset instead of dividing the dataset into batches prior to training. Because of using the stochastic gradient descent for training, it is better to shuffle the dataset before training, and by choosing random images for each batch, shuffling is done automatically. Both approaches are correct but the random selection is a simpler solution. When training the generator in the GAN model, the whole batch consists of generated images. The model of the generator is saved after each epoch with method `save` from the `TensorFlow` library. The whole model with weights can be then loaded and used for generating images.

Results of training

The GAN learns basic patterns of digits after few epochs but it takes about 100 epochs to regularly produce digits from random inputs. Figure 3.3 presents the training progress of GAN for the same input. After 1000 epochs most of the generated images are almost unrecognizable from real images as shown in Figure 3.4. See Figure A.1 for a larger set of results.

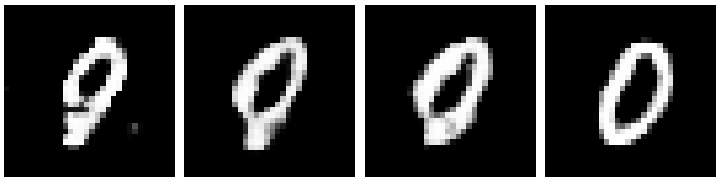


Figure 3.3: Handwritten digits generated from the same point in latent space and by the same generator. The only thing that changes is the number of epochs the generator was trained for. Indices of generator's epochs are 10, 50, 100, and 1000.

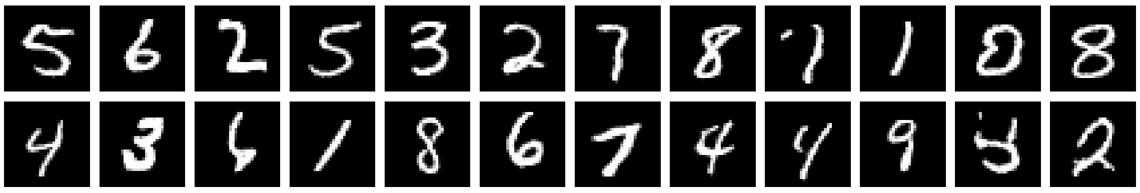


Figure 3.4: Handwritten digits from generator trained for 1000 epochs.

3.3 Generating faces using Labeled Faces in the Wild database

Labeled Faces in the Wild (LFW) database is a collection of photographs of famous actors, politicians, sportsmen and other people. Each photograph is cropped and scaled down to a resolution of 62×47 pixels (as seen in Figure 3.5) and labeled with an identifier of the person in it. Samples are available in grayscale and RGB color format. Further information about LFW database can be found in section 2.4.

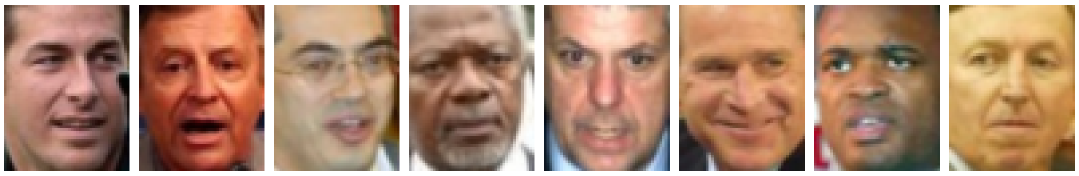


Figure 3.5: Examples of LFW database images obtained from `scikit-learn` library. They are cropped and scaled down to 62×47 pixels.

Transition from handwritten digits to faces

GAN described in the previous section can also be used to generate faces. The only thing that needs to be done is cropping and scaling down of loaded grayscale LFW images. The first task is to crop the images to 47×47 pixels and then resize them to 32×32 pixels. Then the images are ready to be used with GAN used previously to generate handwritten digits. After 1000 epochs of training, it seems that the GAN is working well and the transition to new GAN can continue (examples in Figure 3.6). A larger set of results can be found in Figure A.2.

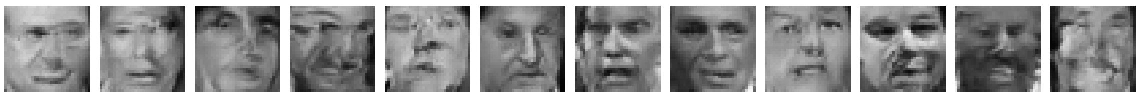


Figure 3.6: Generated images from GAN trained on grayscale cropped LFW database.

GAN can now be adjusted to the real resolution of images in the LFW dataset. The only problem is that resolution 62×47 pixels cannot be factored to small primes, therefore, it is better to change the resolution to 64×48 pixels. After adjusting the discriminator and generator to this resolution, GAN can be tested again. As it is shown in Figure 3.7, 1000 epochs are enough to see that it can generate credible images of faces. More images can be seen in Figure A.3.

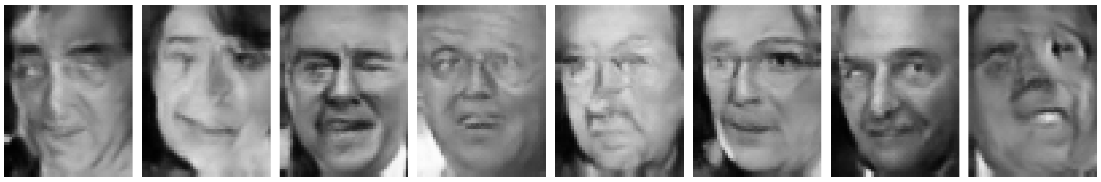


Figure 3.7: Generated images from GAN trained on grayscale LFW database.

The last step in this transition is replacing the grayscale channel with RGB channels and generating color images. Loading of grayscale images has to be changed to loading color images and discriminator and generator also have to be changed. The input of the discriminator is no longer set to 1 channel but 3 channels and output of generator are 3 channels instead of 1. These are the only changes that are needed. Details of implementation are described in the following paragraphs and results are shown at the end of the section.

Loading and preparing images

Images from the LFW dataset can be obtained with `fetch_lfw_people` function from `sklearn.datasets` module of `scikit-learn` library. By default grayscale images are fetched but by setting the `color` argument to `True` color images are fetched. After that, it is useful to resize the images to the resolution of 64×48 pixels. This can be done by duplicating the first and last row and first column of all images. It will barely affect the quality of the images. Pixel value can also be normalized from unsigned integer values between 0 and 255 to float values between 0 and 1. The shape of the training dataset will be (13233, 64, 48, 3).

These operations take some time (approximately 85 s) and to have them done whenever training has to be done can be too time-consuming. A better practice is to save the already prepared images and load them before training. Images loaded and prepared to `ndarray` from `NumPy` library. The library offers the option to save the `ndarray` to a file in non-compressed (`npy` file) or compressed (`npz` file) form. The a non-compressed form is loaded faster (0.5 s) but takes up more space (930 MB). The compressed one loads a few seconds slower (3.8 s) but is significantly smaller (156 MB). Therefore it is better to use the compressed form in this case. After images are once loaded, prepared, and saved they can be loaded and used again within a few seconds.

Architecture of Generative Adversarial Network

The latent space in this GAN has 128 dimensions. The architecture of the discriminator model can be seen in Table 3.3 and the generator model in Table 3.4.

Training of Generative Adversarial Network

Training, monitoring, and evaluation of results is done the same way as with the GAN for handwritten digits in the previous section.

Results of training

Basic patterns of a face can be seen in generated images after just a few epochs of training, but it takes a few hundred epochs to generate faces of better quality. Learned patterns can change during the training (for example skin color but even gender) as seen in Figure 3.8. After 5000 epochs of training, generated faces look credible (examples shown in Figure 3.9). See Figure A.4 for a larger set of results.

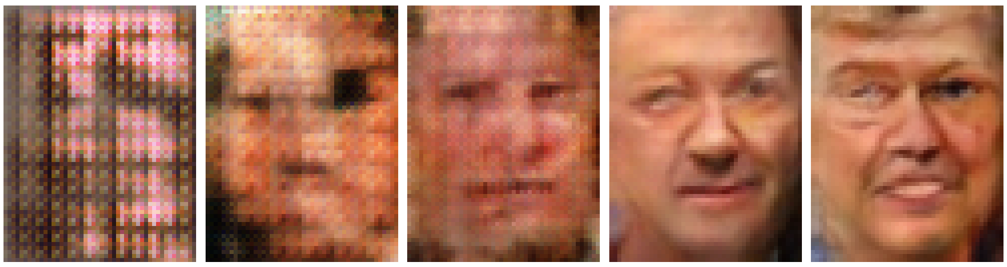


Figure 3.8: Faces generated from the same point in latent space and by the same generator for the LFW dataset. The only thing that changes is the number of epochs the generator was trained for. Indices of generator's epochs are 10, 50, 100, 1000, and 5000.

Layer	Settings	Shape	Parameters
Input		(64, 48, 3)	0
Conv2D	Filter count: 32 Kernel size: (3, 3) Stride size: (1, 1)	(64, 48, 32)	128
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 64 Kernel size: (3, 3) Stride size: (2, 2)	(32, 24, 64)	18 496
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 64 Kernel size: (3, 3) Stride size: (2, 2)	(16, 12, 64)	36 928
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 128 Kernel size: (3, 3) Stride size: (2, 2)	(8, 6, 128)	73 856
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 128 Kernel size: (3, 3) Stride size: (2, 2)	(4, 3, 128)	147 584
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Flatten		(1536)	0
Dropout	Rate: 0.4		0
Dense	Units: 1 Activation: Sigmoid	(1)	1 537
Total trainable parameters			278 529

Table 3.3: GAN trained on the LFW dataset – layers of discriminator model.

Layer	Settings	Shape	Parameters
Input		(100)	0
Dense	Units: 1536	(1536)	198 144
LeakyReLU	Alpha: 0.2		0
Reshape		(4, 3, 128)	0
Conv2DTranspose	Filter count: 128 Kernel size: (3, 3) Stride size: (2, 2)	(8, 6, 128)	147 584
LeakyReLU	Alpha: 0.2		0
Conv2DTranspose	Filter count: 64 Kernel size: (3, 3) Stride size: (2, 2)	(16, 12, 64)	73 792
LeakyReLU	Alpha: 0.2		0
Conv2DTranspose	Filter count: 64 Kernel size: (3, 3) Stride size: (2, 2)	(32, 24, 64)	36 928
LeakyReLU	Alpha: 0.2		0
Conv2DTranspose	Filter count: 32 Kernel size: (3, 3) Stride size: (2, 2)	(64, 48, 32)	18 464
LeakyReLU	Alpha: 0.2		0
Conv2D	Filter count: 3 Kernel size: (1, 1) Stride size: (1, 1) Activation: Sigmoid	(64, 48, 3)	99
Total trainable parameters			475 011

Table 3.4: GAN trained on the LFW dataset – layers of generator model.

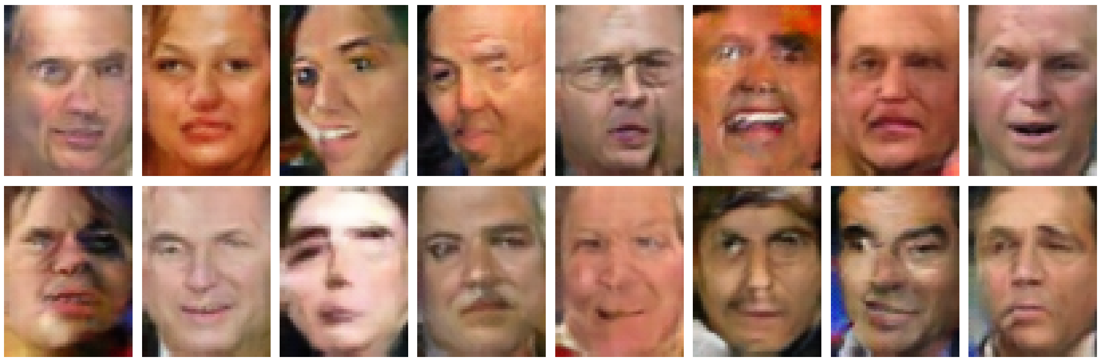


Figure 3.9: Random generated faces from the generator trained on faces from the LFW dataset.

3.4 Generating faces using Flickr-Faces-HQ database

Flickr-Faces-HQ (FFHQ) is a high-quality image dataset of photographs of faces created as a benchmark for Generative Adversarial Networks [9]. Images were extracted from photographs uploaded to service Flickr and then aligned, cropped and saved as `png` files. Dataset consists of 70000 color images with a resolution of 1024×1024 pixels with a large variety of age, ethnicity, background, and accessories as glasses and hats. Images with a lower resolution of 128×128 pixels were used in this thesis. More information about the FFHQ database is provided in section 2.4.

Loading and preparing images

Images can be downloaded either manually or with a script prepared by the dataset creators[1]. The whole dataset is too large to be held in the memory during the training (it takes up over 60 GB of space), it has to be divided into parts and each part has to be loaded during every epoch. Therefore, it is necessary to guarantee the shortest possible time for loading by preprocessing of the images. They have to be loaded from `png` files to `NumPy ndarray` and then saved to `npy` files. The non-compressed version of saved `NumPy ndarray` loads faster, the only downside is that it requires more disk space but that is not very important in this case. Each `npy` file contains 1400 images, which means the dataset is saved into 50 of these files.

Architecture of Generative Adversarial Network

The dimensionality of the latent space used in this GAN is 256. The architecture of the discriminator model can be seen in Table 3.5 and the generator model in Table 3.6.

Training of Generative Adversarial Network

The size of the training batch is set to 256. At the beginning of each epoch, the first part of the dataset is loaded. For the training of the discriminator, 128 samples from the dataset are used together with 128 generated samples to form a batch. Then another batch is formed the same way and this continues until the part of the dataset has been completely used. If there is only a lower number of samples available in the dataset, more generated

[1]FFHQ dataset is available at https://github.com/NVlabs/ffhq-dataset

Layer	Settings	Shape	Parameters
Input		(128, 128, 3)	0
Conv2D	Filter count: 32 Kernel size: (1, 1) Stride size: (1, 1)	(128, 128, 32)	128
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 64 Kernel size: (3, 3) Stride size: (2, 2)	(64, 64, 64)	18 496
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 128 Kernel size: (3, 3) Stride size: (2, 2)	(32, 32, 128)	73 856
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 128 Kernel size: (3, 3) Stride size: (2, 2)	(16, 16, 128)	147 584
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 256 Kernel size: (3, 3) Stride size: (2, 2)	(8, 8, 256)	295 168
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Conv2D	Filter count: 256 Kernel size: (3, 3) Stride size: (2, 2)	(4, 4, 256)	590 080
LeakyReLU	Alpha: 0.2		0
Dropout	Rate: 0.4		0
Flatten		(4096)	0
Dropout	Rate: 0.4		0
Dense	Units: 1 Activation: Sigmoid	(1)	4 097
Total trainable parameters			1 129 409

Table 3.5: GAN trained on the FFHQ dataset – layers of discriminator model.

Layer	Settings	Shape	Parameters
Input		(256)	0
Dense	Units: 4096	(4096)	1 052 672
LeakyReLU	Alpha: 0.2		0
Reshape		(4, 4, 256)	0
Conv2DTranspose	Filter count: 256 Kernel size: (3, 3) Stride size: (2, 2)	(8, 8, 256)	590 080
LeakyReLU	Alpha: 0.2		0
Conv2DTranspose	Filter count: 128 Kernel size: (3, 3) Stride size: (2, 2)	(16, 16, 128)	295 040
LeakyReLU	Alpha: 0.2		0
Conv2DTranspose	Filter count: 128 Kernel size: (3, 3) Stride size: (2, 2)	(32, 32, 128)	147 584
LeakyReLU	Alpha: 0.2		0
Conv2DTranspose	Filter count: 64 Kernel size: (3, 3) Stride size: (2, 2)	(64, 64, 64)	73 792
LeakyReLU	Alpha: 0.2		0
Conv2DTranspose	Filter count: 32 Kernel size: (3, 3) Stride size: (2, 2)	(128, 128, 32)	18 464
LeakyReLU	Alpha: 0.2		0
Conv2D	Filter count: 3 Kernel size: (1, 1) Stride size: (1, 1) Activation: Sigmoid	(128, 128, 3)	99
Total trainable parameters			2 177 731

Table 3.6: GAN trained on the FFHQ dataset – layers of generator model.

images are used to have the batch size fixed. A new part of the dataset is loaded instead of the previous one afterward and the training continues. This procedure repeats for each epoch. The generator model is trained on a batch consisting of 256 generated images.

Training of this GAN takes a long time and therefore it is necessary to somehow checkpoint the training. One option of checkpointing is saving weights of models and loading them to continue with training. Saving of states of models can also be used for comparison of models' improvements during the training. This GAN saves weights of the discriminator, the generator, and also the whole GAN model after each epoch. When the training is launched again, the weights are loaded and it can continue from where it last stopped.

Results of training

The first patterns of faces can be seen after just a few epochs but it takes a few hundreds of epochs to generate a good image of faces. The progress of training can be seen in Figure 3.10. After 700 epochs of training, generated faces are starting to look credible (examples shown in Figure 3.11). See Figure A.5 for a larger set of results.

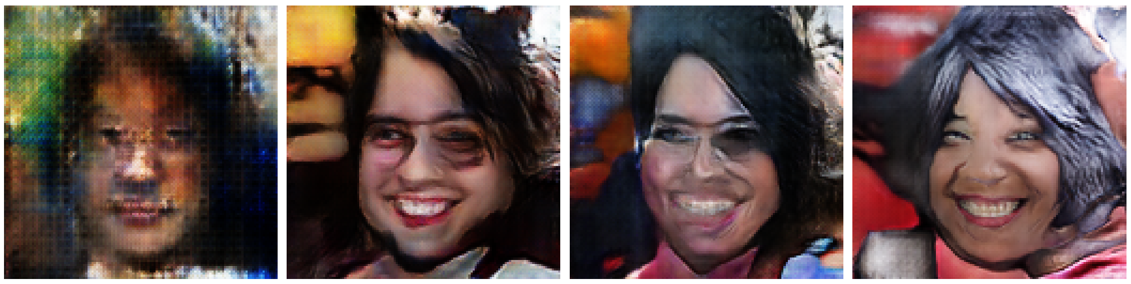


Figure 3.10: Faces generated from the same point in latent space and by the same generator trained on the FFHQ dataset. The only thing that changes is the number of epochs the generator was trained for. Indices of generator's epochs are 10, 50, 100, and 700.

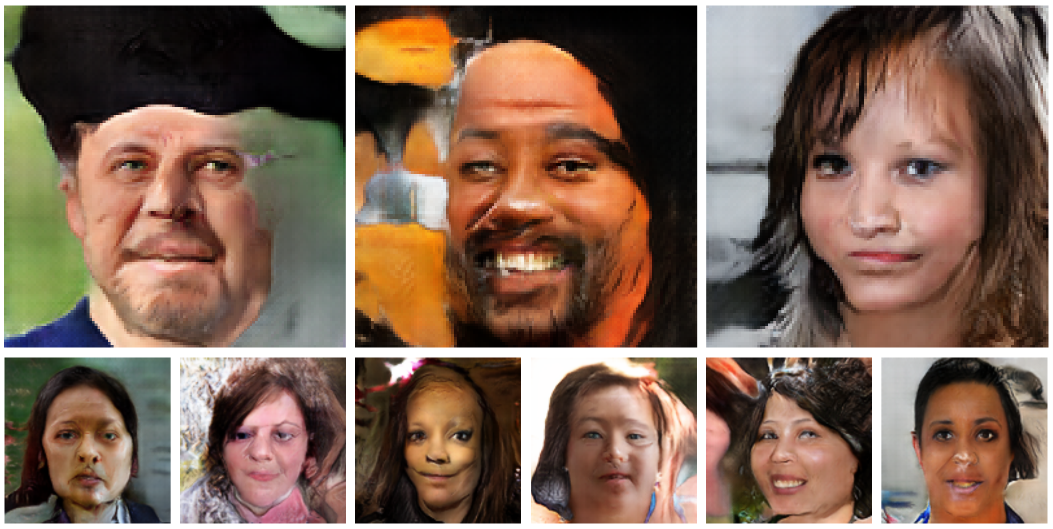


Figure 3.11: Random generated faces with GAN trained for 700 epochs on the FFHQ dataset.

Possibilities for improvement

The achieved results are not the best possible ones that the GANs can produce. Several practices can provide better results but they were not implemented because they are beyond the scope of this thesis. This subsection is a summary of these practices and methods.

One of the most important factors in deep learning overall is the computational power and time available for the training. With more resources, better results can be always achieved. In practice, that means that images with higher resolution could be used for training and generated images would then also have higher resolution. If the GAN is trained for a longer time, produced images are also more trustworthy.

GANs can be also trained with a different methodology that focuses on a progressive growing of both discriminator and generator simultaneously [8]. GAN is first trained on images with a small resolution of 4×4, then another layer is added to process 8×8 images and the training continues. This approach is repeated until the final resolution of images is achieved. All of the parameters of both models are trained during the process. The training process is more stable because the GAN learns the large-scale structure of the image distribution and then gradually learns the finer details. It also reduces the training time because the training iterations at lower resolutions are faster.

Another way to improve the GAN is by adding a minibatch standard deviation layer to the end of a discriminator [8]. This approach increases variation in the results. Improvements can be also achieved by using He's initializer for normalizing weights during the training. It replaces an adaptive Stochastic Gradient Descent method such as Adam. The dynamic range of all weights is the same and therefore, their learning speed is equal.

Chapter 4

Latent space analysis for generating faces with specific features

The process of generating an image from a latent point is deterministic, the generator model always generates the same image when provided with the same latent point. From this fact arises a question of whether a specific feature of a face is located in some part of latent space. If this is true, a position of a latent point implies specific features of a face generated from this latent point. Therefore, the following hypothesis is formed. "It is possible to generate images with specific characteristics."

Before the analysis can be done, it is necessary to obtain features of a face and a latent point used to generate that face. These two pieces of information then can be used to analyze the latent space. This process is discussed in section 4.1. The analysis is divided into two parts, first the dimensions of latent space are analyzed separately in section 4.2, then in section 4.3, the whole latent space is analyzed by projecting it into a subspace with the Linear Discriminant Analysis. Finally, a tool for generating images of faces with requested features is introduced in section 4.4.

4.1 Manual labeling of generated images

Generated images sometimes do not offer good enough quality for automatic labeling of requested features, therefore, a manual approach was chosen in this thesis. In the beginning, a latent point is generated and saved in a CSV format[1]. After that, a face is generated from this latent point and displayed on the screen and also saved. The user then fills in a simple questionnaire about the face's features. After completing it, features are saved in a JSON format[2]. It contains one main JSON object with separate features as shown here.

```
{
    "glasses": "no",
    "hair_color": "brown"
}
```

[1]CSV – Comma-Separated Values, more information in RFC 4180 at https://tools.ietf.org/html/rfc4180

[2]JSON – JavaScript Object Notation, more information in STD 90 at https://tools.ietf.org/html/std90

During the labeling of images, the user can also label an image as not valid, when the observed feature is hard to recognize. Images that are not valid are not used later for the latent space analysis. The generator is not perfect, it can create faces that are far from being real. The ability to label them as not valid makes the results of latent space analysis more precise.

Two features were chosen for observing. The first one is the presence of glasses on a person's face. The main reason for choosing this feature as the first one that is observed is that it has only two classes, a person either has or does not have glasses. The second observed feature is the color of a person's hair. This feature was chosen as one that can be separated into more than 2 classes, exactly 4 were chosen: blond, brown, black, and gray. Both of these features are very easy to recognize in an image when the classification is done. Other features that could be observed are for example gender, skin color, or direction the head is facing, but all of these are harder to manually classify. Observing of the features was done separately.

For this thesis, 500 faces were added information about having or not having glasses. Another 500 faces were then labeled with the hair color. This amount of images is sufficient for the analysis with the used methods. Examples of the labeled faces are shown in Figure A.6.

Before the analysis, latent points are loaded into `NumPy ndarray` and feature classes are converted from strings to integers. The conversion is done with a Label Encoder[3] model from the `scikit-learn` library [15]. This model is also saved for later use when generating new faces with specific features.

4.2 Analysis of separate dimensions of latent points

This section deals with the following hypothesis. "It is possible to use a single dimension of a latent point to separate classes of a single feature of the generated face." For experimental purposes, the presence of glasses on a person's face was chosen as an observed feature. This feature contains two classes and therefore Student's t-test can be used to analyze dimensions in a latent space.

Student's t-test is a statistical hypothesis test that determines whether two samples are expected to come from the same population [6]. It compares the means of the samples and outputs the difference between them.

Initial assumptions about data

There are 3 initial conditions of Student's t-test that have to be met. The first one is that the samples have to be independent. Since the latent points are obtained as random values from Standard Normal Distribution without any connection to each other, this condition is fulfilled.

The second condition is that samples come from Normal Distribution. As mentioned in the previous paragraph, latent points are generated from Standard Normal Distribution but it is also possible to compare the samples with this distribution if there are any differences. For this, `stats.normaltest` function from the `SciPy` library was used. The test outputs a 2-sided chi-squared probability (p-value) that can be compared with a significance level α. If the p-value is lower than α, for that dimension the data might not be distributed

[3]More information about Label Encoder in the library's API at https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

normally. Significance level α was set to 0.01. After this test was done, 7 dimensions were indicated as not coming from Normal Distribution (dimensions number 13, 42, 206 for samples without glasses and dimensions number 58, 86, 94, 99 for samples with glasses). This violation of normality is not big enough for samples not to be considered as normal.

The third condition is equality of variance of samples, which can be calculated with Levene test from `SciPy` library[4]. This test computes the p-value for each dimension and if it is smaller than α, the variance in that dimension is not equal for the two classes. After doing this test for all dimensions, 3 of them were found as having different variances (dimensions number 164, 171, 220). This violation is considered insignificant, therefore, the variance equality condition is also fulfilled.

Calculation of Student's t-test

After all initial conditions are met, the Student's t-test can be done. Latent points and classes of all samples are loaded into two `DataFrames` structures of `pandas` library according to their class.

The first task is to form the Null and Alternate Hypothesis, in this example, the Two-Tailed test was chosen. Null Hypothesis H_0 expects the difference between the means of classes to be equal to a value c. In this case, the expected difference c is equal to 0. Alternate Hypothesis H_1 then expects the classes to have a different mean. The task of a Student's t-test is to determine whether the Null Hypothesis holds [6].

$$H_0 : \mu_1 - \mu_2 = c$$

$$H_1 : \mu_1 - \mu_2 \neq c$$

$$c = 0$$

Next step is to calculate the t-statistic for the data by the formula following this paragraph. M_1 and M_2 are means of the data, n_1 and n_2 are counts of samples of classes and S^2 is the weighted average of standard deviations S_1^2 a S_2^2.

$$t = \frac{M_1 - M_2}{S\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

$$S^2 = \frac{(n_1 - 1)S_1^2 + (n_2 - 1)S_2^2}{n_1 + n_2 - 2}$$

After that, a critical region is computed using the significance level α, which is a probability of Null Hypothesis being rejected despite it being true. The critical region is calculated using the following formula.

$$W = (-\infty; -t_{1-\alpha/2}(n_1 + n_2 - 2)\rangle \cup \langle t_{1-\alpha/2}(n_1 + n_2 - 2); \infty)$$

If test criterium t belongs to the critical region, the Null Hypothesis is rejected on the significance level α and Alternate Hypothesis holds.

Dimensions, for which the Null Hypothesis gets rejected, affect the given feature (glasses presence). After doing the calculation, dimensions for which the Null Hypothesis got rejected are: 20, 25, 30, 46, 70, 110, 143, 144, 148, 156, 160, 171, 206, 212, 216, 230, 237, 248,

[4]Details of the Levene test can be found at https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.levene.html

250. Data can be further examined using histograms of the sample distribution. At first sight, classes are not separable even in these dimensions. Representative examples are shown in Figure 4.1, others look very similar.

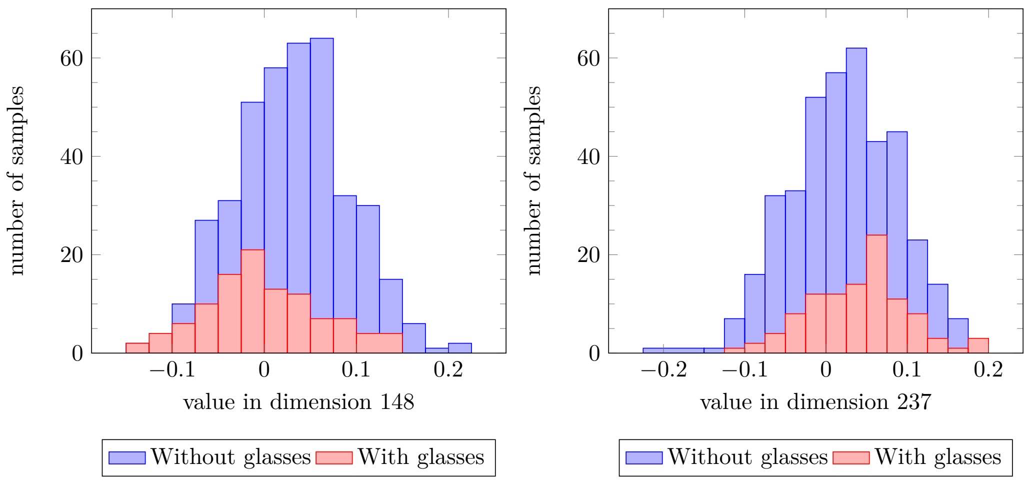


Figure 4.1: Distribution of samples with values from dimensions number 148 and 237.

Results of Student's t-test

It is clear that some dimensions of a latent space have a larger impact on the feature than the others but no single one of them can be used to determine a feature. Therefore, the proposed hypothesis is rejected. However, a combination of certain dimensions could be used for the latent space analysis. This approach is discussed in the following section.

4.3 Linear Discriminant Analysis for latent space projection

If one single dimension cannot be used to separate the features, a linear combination of multiple dimensions might be better for this task. In this thesis, a technique called Linear Discriminant Analysis (LDA) is discussed. It projects data to a lower-dimensional space where the class-separability is better [2]. The hypothesis proposed in this section is as follows. “Latent space can be projected to a subspace in a way that makes the classes of features of a face generated from a point in that latent space separable.”

Input data for this analysis are 256-dimensional vectors representing a single point in a 256-dimensional latent space. The calculation is focusing on data with the feature of glasses presence on a person's face, which means a feature with 2 classes. LDA assumes normally distributed data, statistically independent features (in this case, features are the dimensions of latent space), and identical covariance matrices for every class. The fulfillment of these assumptions is explained in section 4.2.

Calculation of Linear Discriminant Analysis

The first step of LDA is calculating of 256-dimensional mean vectors $\boldsymbol{m}_i$ for each of the classes i. This vector contains, as the name suggests, the mean value of the samples in all of their 256 dimensions. D_i is a set of all samples $\boldsymbol{x}$ in class i and its cardinality is n_i.

$$\boldsymbol{m}_i = \begin{bmatrix} \mu_{i_1} \\ \mu_{i_2} \\ \vdots \\ \mu_{i_{255}} \\ \mu_{i_{256}} \end{bmatrix}, \quad \text{with} \quad i = 0, 1 \qquad\qquad \boldsymbol{m}_i = \frac{1}{n_i} \sum_{\boldsymbol{x} \in D_i}^{n_i} \boldsymbol{x}$$

In the next step, 256×256-dimensional within-class and between-class scatter matrices are computed. These matrices are used to make estimates of covariance matrices. The within-class scatter matrix S_W is computed by the following formula, where c is the number of classes. Indexing of classes starts at 0, therefore, $c - 1$ is used in the formulas.

$$S_W = \sum_{i=0}^{c-1} S_i, \quad \text{where} \quad S_i = \sum_{\boldsymbol{x} \in D_i}^{n} (\boldsymbol{x} - \boldsymbol{m}_i)\ (\boldsymbol{x} - \boldsymbol{m}_i)^\top$$

And between-class scatter matrix S_B is computed by the following formula, where $\boldsymbol{m}$ is the overall mean of data and N_i is the cardinality of a sample set of the respective class.

$$S_B = \sum_{i=0}^{c-1} N_i (\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^\top$$

In the following step, linear discriminants are computed by solving the generalized eigenvalue problem for matrix $S_W^{-1} S_B$. The computed eigenvectors and eigenvalues are important information about the distortion of a linear transformation. Eigenvectors are basically the direction of distortion and their length is 1. Eigenvalues describe the magnitude of this distortion, they scale the eigenvectors. When performing the LDA, eigenvectors are forming the axes of a new subspace and eigenvalues are giving information about how important are respective eigenvectors for the separation of classes.

For LDA, the number of linear discriminants d is at most $c - 1$, in other words, one less than the number of classes. That means that for the glasses presence feature, latent space can be only projected to 1-dimensional space when using the LDA ($d = 1$). The eigenvalues can be sorted in decreasing order to determine the number of chosen linear discriminants d. Respective eigenvectors of chosen eigenvalues are forming the eigenvector matrix $\boldsymbol{W}$, in this case, it is 256×1-dimensional. Matrix $\boldsymbol{W}$ is used to transform $n \times 256$-dimensional data $\boldsymbol{X}$ to $n \times 1$-dimensional data $\boldsymbol{Y}$.

$$\boldsymbol{Y} = \boldsymbol{X} \times \boldsymbol{W}$$

Implementation of Linear Discriminant Analysis

Python's library `scikit-learn` already has an implementation of LDA, therefore, it is not necessary to implement it manually [15]. The implemented LDA model[5] will perform the operations faster and it is also easier to use. But the main reason for using the `scikit-learn` LDA model is that it can also serve as a classifier. The model can be saved into a file for later use with the `joblib` library. Function `dump` saves it to a file and function `load` takes care of loading the model when it is used later[6].

[5]More information about scikit-learn LDA model in library's API at https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

[6]Official website of joblib library with further details at https://joblib.readthedocs.io/en/latest/persistence.html

The main parameter when initializing the model is a number of components, this is the dimensionality of a subspace the data are projected into. Then the model is trained using the `fit` method provided with the original data (latent points) and their labels (classes). Method `transform` does the projecting of data into a subspace. In Figure 4.2 can be seen the transformed samples of latent points labeled with the feature of glasses presence. It is clear that the classes can be separated and therefore the initial hypothesis, whether the samples can be projected to a subspace where the classes are separable, is accepted for a feature with 2 classes.

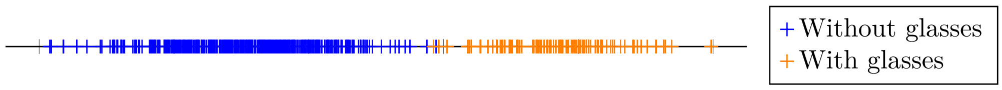


Figure 4.2: 500 samples of faces labeled with glasses presence transformed from 256-dimensional space to 1-dimensional space using LDA.

LDA model can also be used to transform and classify multi-class samples such as faces labeled with hair color. 4 different classes were chosen for this feature: blond, brown, black, and gray hair. The number of components for the model is set to a maximum value of 3, one lower than the number of classes. Figure 4.3 displays the transformed samples. LDA managed to project the samples into such subspace where the classes are separable and therefore the initial hypothesis holds for multi-class samples.

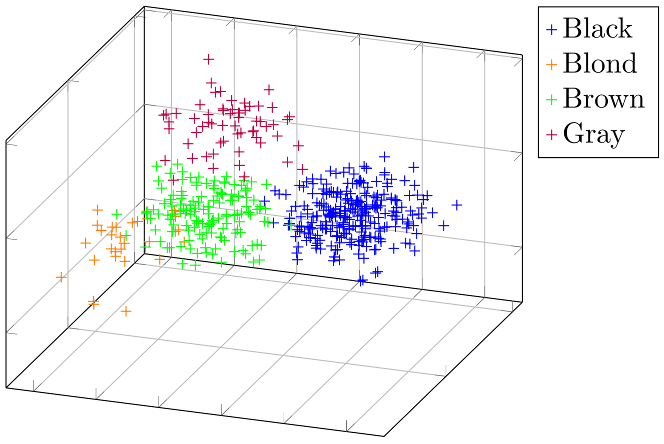


Figure 4.3: 500 samples of faces labeled with hair color projected from 256-dimensional space to 3-dimensional space using LDA.

4.4 Generating faces with specific features

The simplest approach to generating faces with specific features is to generate a large number of latent points (for example 1000) and use the classifier to determine whether the generated face will have the asked features. The first latent point that meets these requirements is used to generate a face. If 1000 generated latent points are not enough, 1000 more are generated and the classifying continues. The latent point has to satisfy all of the conditions for requested features, only then it is accepted.

Classification of features from the latent point can be done by multiple methods of the LDA model, for example, `predict`, `predict_proba`, or `decision_function`. LDA classifier for glasses presence was tested on 100 unseen images that were manually labeled and its success rate was 75 % when using the `predict` method. To ensure that generated latent point belongs to a face with requested features, a classifying method that offers more information than just hard decisions might be used.

Method `predict_proba` gives information about the probability of the sample belonging to each of the classes. The latent point can be then accepted for generating a requested face only in the case where the probability of requested features is multiple times bigger than the probability of other classes of those features. In this thesis the multiplier for latent point acceptance is 10^8. For example, the probability of a face with glasses has to be 10^8 higher than the probability of face without glasses, only then is the latent point accepted. This increases the probability of generating the requested face.

An experiment was done to test how the multiplier affects the probability of the requested feature being part of the face. The glasses presence was chosen as a requested feature. For each of the multipliers, 100 images were generated and manually labeled. Results are only approximate but as can be seen in Table 4.1 higher multiplier can positively affect the generator success rate. In the original dataset of 500 labeled faces, 21 % of them were with glasses.

Multiplier	1	100	10^4	10^6	10^8
Probability	45 %	53 %	62 %	71 %	84 %

Table 4.1: Comparison of how the multiplier affects the probability of generating face in the requested class. The multiplier is used to make the difference between the requested class and other classes bigger by multiplying the probability of other classes before comparing with the requested class probability.

The main hypothesis of this chapter is as follows. “It is possible to generate images with specific characteristics.” Face with requested features can be generated with probability approximately 80 % and therefore the hypothesis is credible. Generated images of faces with various features can be seen in Figure A.7.

Further improvement could be achieved by using the LDA only for projecting of the samples to a subspace and then using another model, such as Support Vector Machine, for classification. Another way of improving the latent space analysis would be to use Quadratic Discriminant Analysis (QDA) instead of Linear Discriminant Analysis. Classes are not always easily linearly separable and by introducing a non-linearity to the process, results could improve.

Chapter 5

Conclusion

This thesis had two main goals. The first goal was to design and develop a Generative Adversarial Network (GAN) that is able to generate credible images of faces. And the second goal was to develop a tool that provides an option to generate faces with specific features without any changes to the GAN itself. Both of these goals were completed.

In order to obtain the knowledge necessary to understand how GANs work, different fields of Machine Learning had to be studied. Basic knowledge about Neural Networks and especially Convolutional ones is fundamental to complete this task. Generative Adversarial Networks were then thoroughly studied. Available image databases were reviewed and three were chosen for the implementation.

Three models of GANs were successfully implemented. The first one can generate images of handwritten digits in grayscale. The second one was gradually implemented as a variation of the previous GAN and it is able to generate low-resolution color images of faces. Last GAN was focused on higher resolution color images with a large variance in the look of faces and backgrounds and it is capable of generating 128×128-pixel color images with real-looking faces.

Latent space that represents a source for generator inputs was analyzed with two methods. A small dataset of faces labeled with information about glasses and hair color was used for the analysis. At first, Student's t-test was used to determine that no single dimension of latent space can be used to separate classes of a feature. Next, Linear Discriminant Analysis was computed to project the latent space to a subspace where the classes of features were separable. This accomplishment leads to the possibility of providing the generator with specific input to generate face with requested features with probability approximately 80 %. A simple tool that allows a user to set features and generate a face with them is implemented.

The quality of generated images could be improved by using advanced techniques in GAN design as progressive growing or minibatch standard deviation. As it is very common for Neural Networks, better results could be also achieved with more computational power and longer training process.

Latent space analysis could be improved by an automatic feature recognition to expand the number of possible requested features and to achieve higher precision in the latent space analysis. A non-linear method as Quadratic Discriminant Analysis could be used for the projection into a subspace and an advanced classifier, such as Support Vector Machine, could be used after this projection.

Bibliography

[1] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. et al. ImageNet: A Large-Scale Hierarchical Image Database. In: *CVPR09*. 2009. Available at: http://www.image-net.org/papers/imagenet_cvpr09.pdf.

[2] Duda, R., Hart, P. and G.Stork, D. Pattern Classification. In:. 2001. ISBN 0-471-05669-3.

[3] Dumoulin, V. and Visin, F. *A guide to convolution arithmetic for deep learning*. 2016. Available at: https://arxiv.org/abs/1603.07285v2.

[4] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. MIT Press, 2016. ISBN 978-0262035613. Available at: http://www.deeplearningbook.org.

[5] Goodfellow, I., Pouget Abadie, J., Mirza, M., Xu, B., Warde Farley, D. et al. Generative Adversarial Nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D. and Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014, p. 2672–2680. Available at: http://papers.nips.cc/paper/5423-generative-adversarial-nets.

[6] Hazewinkel, M. *Encyclopedia of Mathematics*. Springer Verlag GmbH, European Mathematical Society, 1994. ISBN 978-1-55608-010-4.

[7] Huang, G. B., Ramesh, M., Berg, T. and Learned Miller, E. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. October 2007. Available at: http://vis-www.cs.umass.edu/lfw/lfw.pdf.

[8] Karras, T., Aila, T., Laine, S. and Lehtinen, J. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2017. Available at: https://arxiv.org/abs/1710.10196v3.

[9] Karras, T., Laine, S. and Aila, T. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2018. Available at: https://arxiv.org/abs/1812.04948.

[10] Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. *University of Toronto*. May 2012. Available at: https://www.cs.toronto.edu/~kriz/cifar.html.

[11] LeCun, Y. and Cortes, C. *MNIST handwritten digit database*. 2010. Available at: http://yann.lecun.com/exdb/mnist/.

[12] Liu, Z., Luo, P., Wang, X. and Tang, X. Deep Learning Face Attributes in the Wild. In: *Proceedings of International Conference on Computer Vision (ICCV)*.

December 2015. Available at:
https://liuziwei7.github.io/projects/FaceAttributes.html.

[13] Marsaglia, G. Choosing a Point from the Surface of a Sphere. *Ann. Math. Statist.* The Institute of Mathematical Statistics. April 1972, vol. 43, no. 2, p. 645–646. DOI: 10.1214/aoms/1177692644. Available at: https://doi.org/10.1214/aoms/1177692644.

[14] Murphy, K. P. *Machine learning : a probabilistic perspective.* MIT Press, 2013. ISBN 9780262018029. Available at: https://www.cs.ubc.ca/~murphyk/MLbook/.

[15] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research.* 2011, vol. 12, p. 2825–2830.

[16] Torralba, A., Fergus, R. and Freeman, W. T. 80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 2008, vol. 30, no. 11, p. 1958–1970. Available at:
https://people.csail.mit.edu/torralba/publications/80millionImages.pdf.

[17] Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T. et al. *LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop.* 2015. Available at: https://www.yf.io/p/lsun.

Appendix A

Generated images

This appendix shows images generated from Generative Adversarial Networks (GANs) described in this thesis. All examples are fair random draws, they are not cherry-picked.

Generated handwritten digits

Images generated from GAN trained on the Modified National Institute of Standards and Technology (MNIST) digit database. The resolution of generated images is 32×32 pixels.

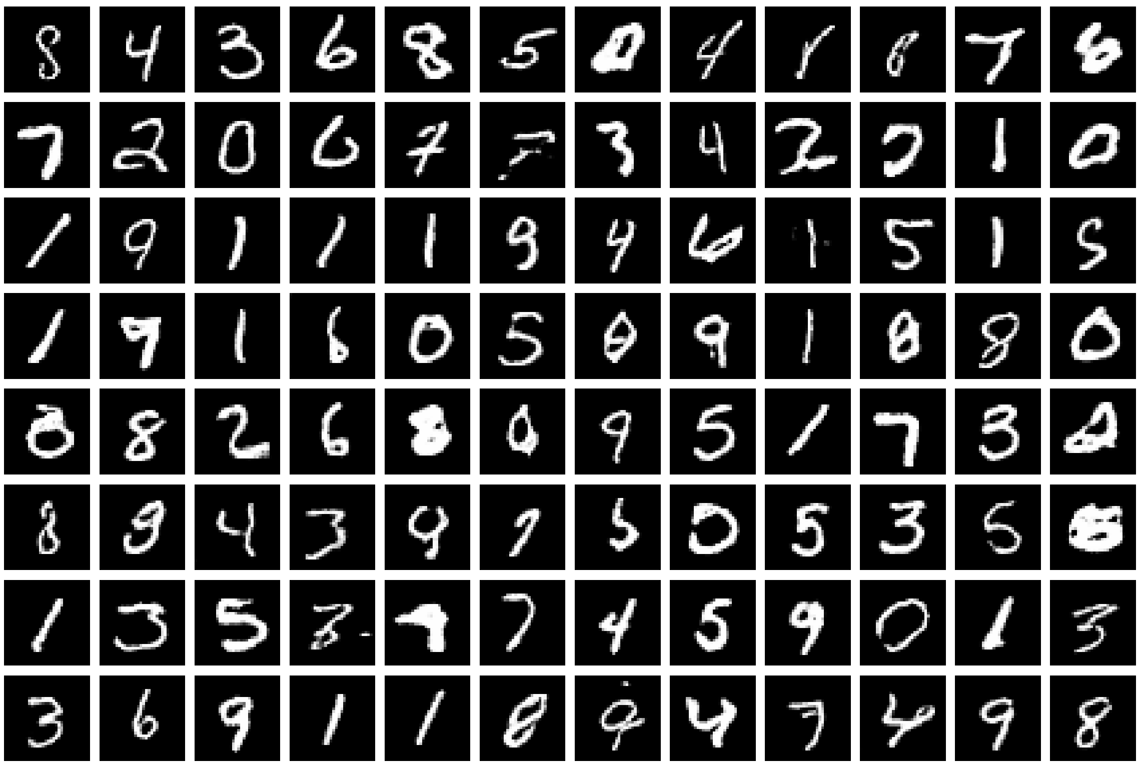


Figure A.1: Handwritten digits from the generator trained for 1000 epochs.

Generated faces from Labeled Faces in the Wild database

Images generated from GAN trained on Labeled Faces in the Wild (LFW) database. Images are separated into 3 figures that display the transition from GAN used for handwritten digits. Figure A.2 shows images generated by the same GAN, just trained on the LFW database. Figure A.3 displays images with the resolution of the LFW database but still grayscale. And finally, Figure A.4 presents images matching the original LFW database.

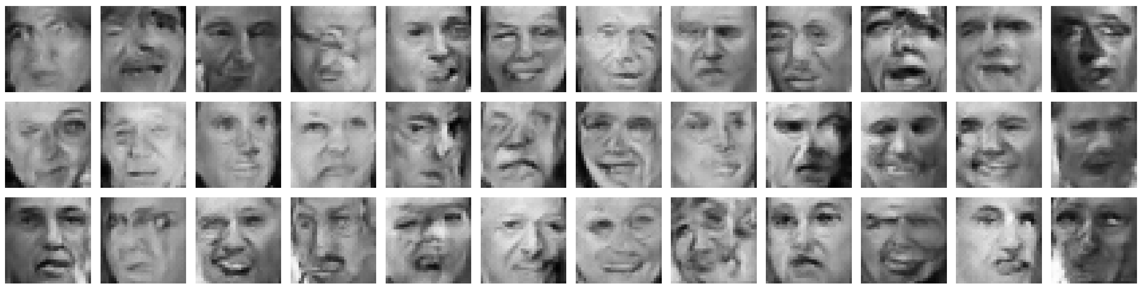


Figure A.2: Grayscale cropped faces from the generator trained for 1000 epochs on the LFW database.

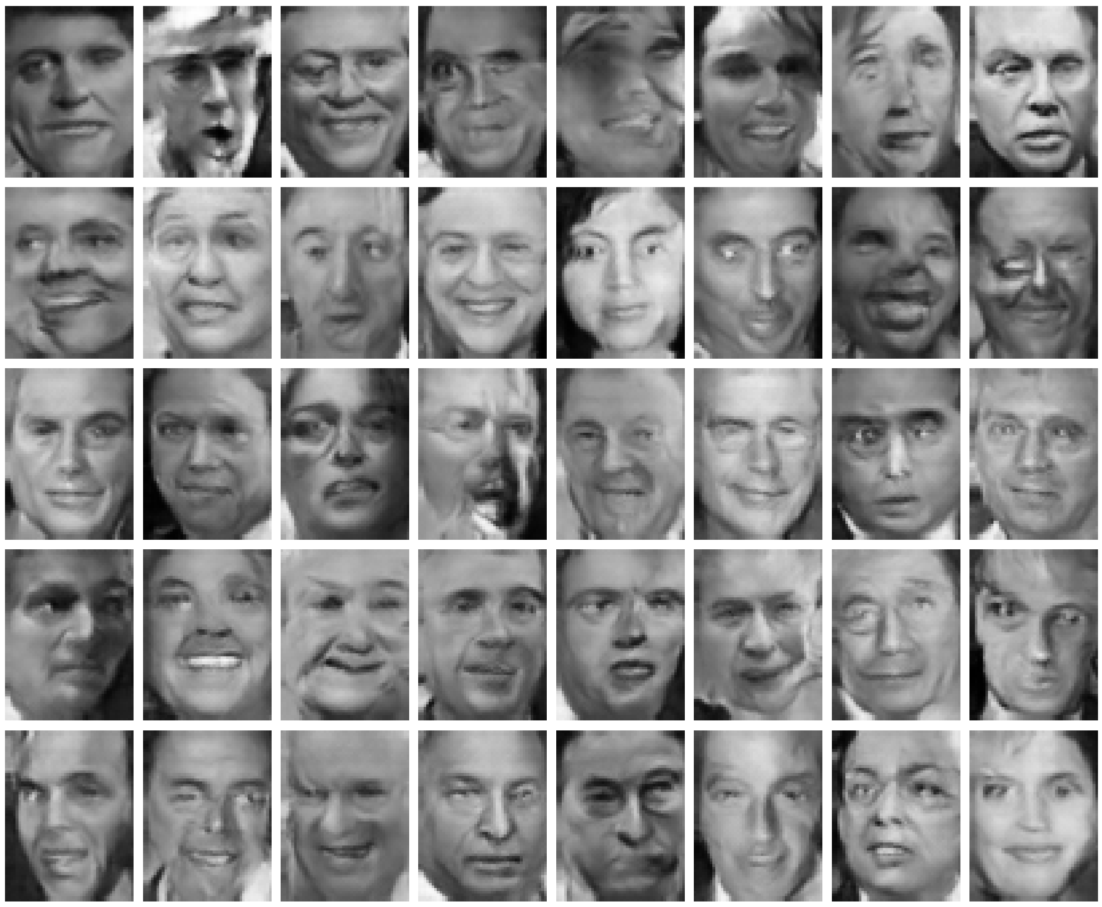


Figure A.3: Grayscale faces from the generator trained for 1000 epochs on the LFW database.

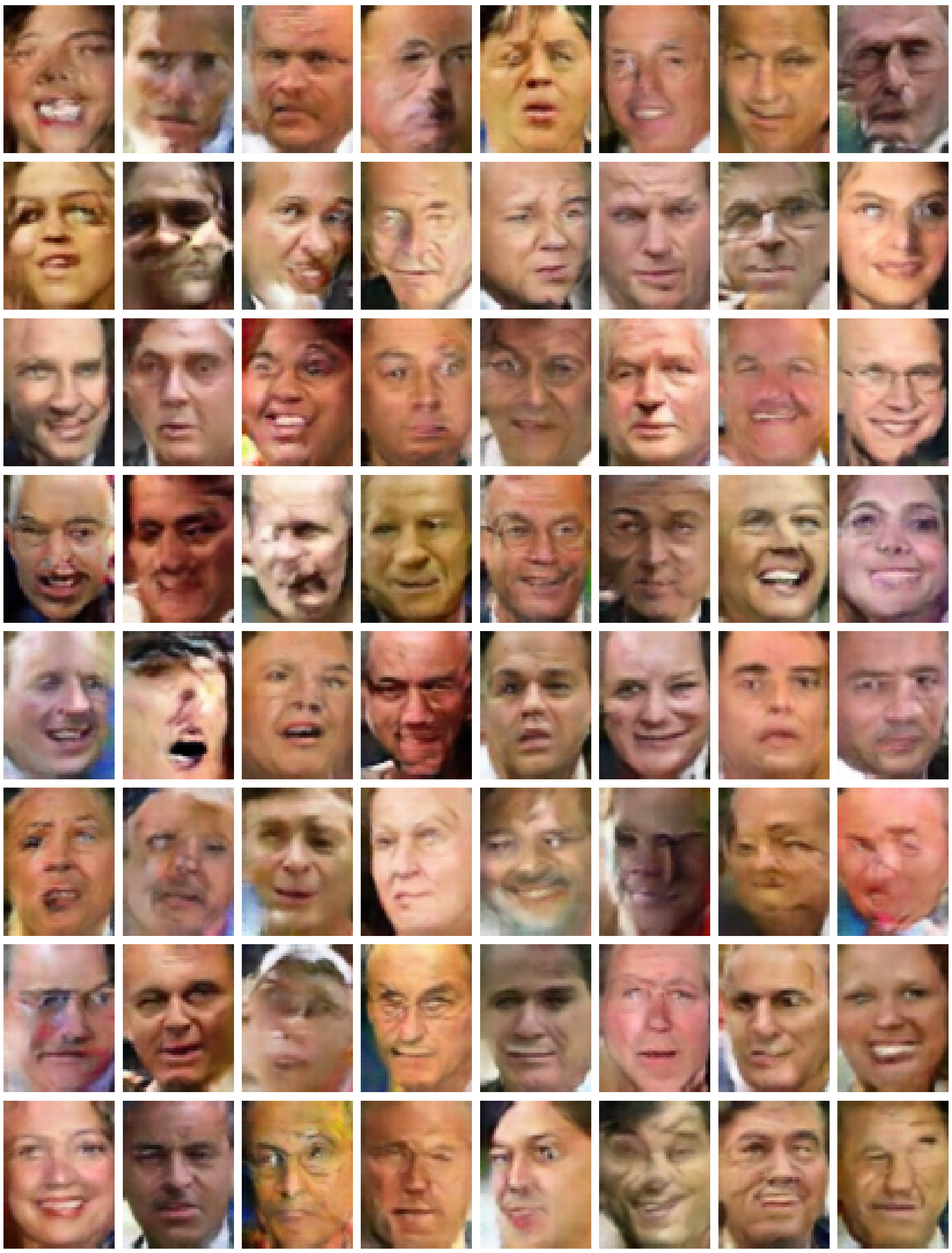


Figure A.4: Color faces from the generator trained for 5000 epochs on the LFW database.

Generated faces from Flickr-Faces-HQ database

Images generated from GAN trained on Flickr-Faces-HQ (FFHQ) database. Figure A.5 shows randomly generated faces without any labels. Manually labeled faces can be seen in Figure A.6. Faces generated with features requested by the user are in Figure A.7.

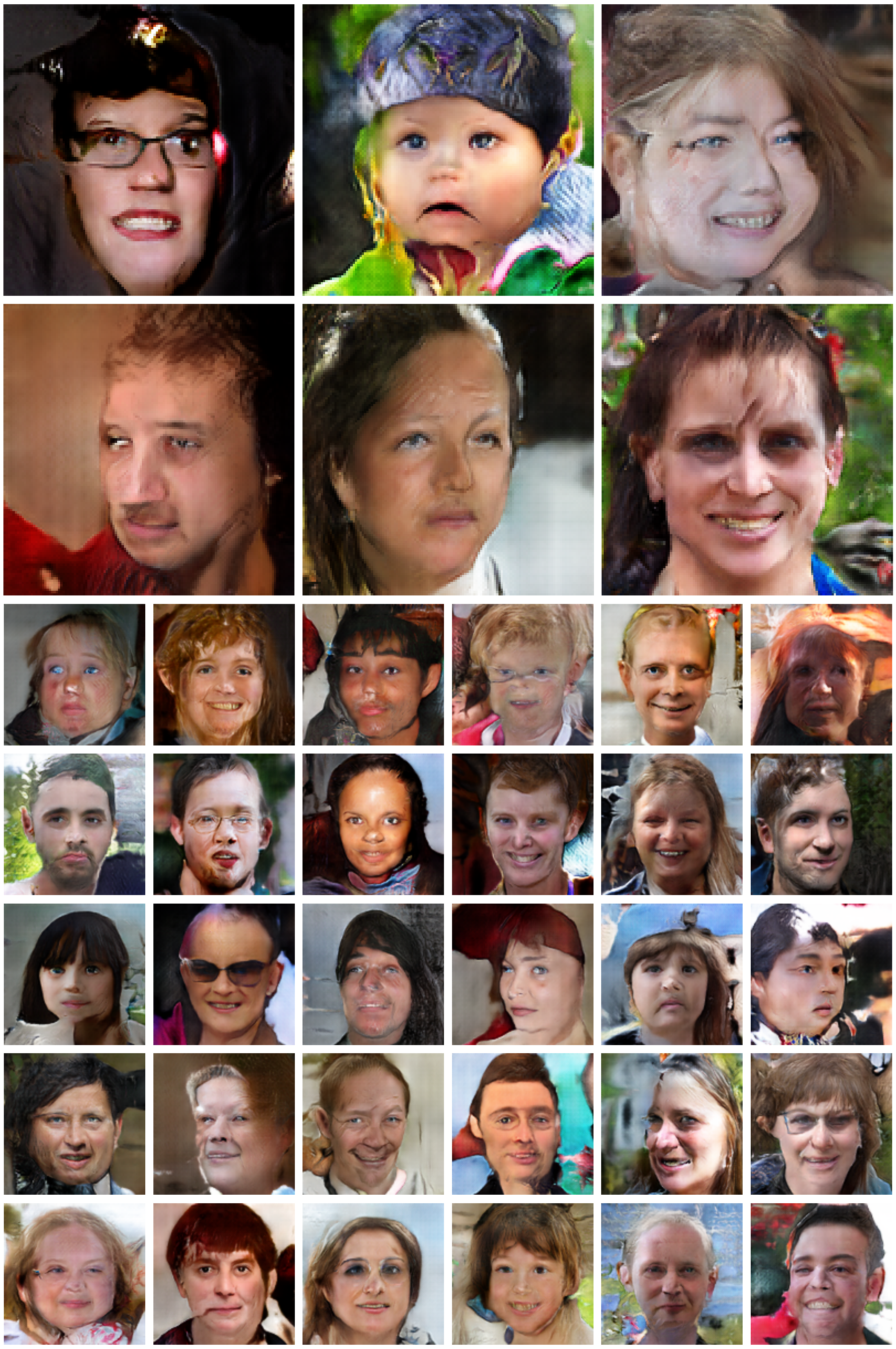
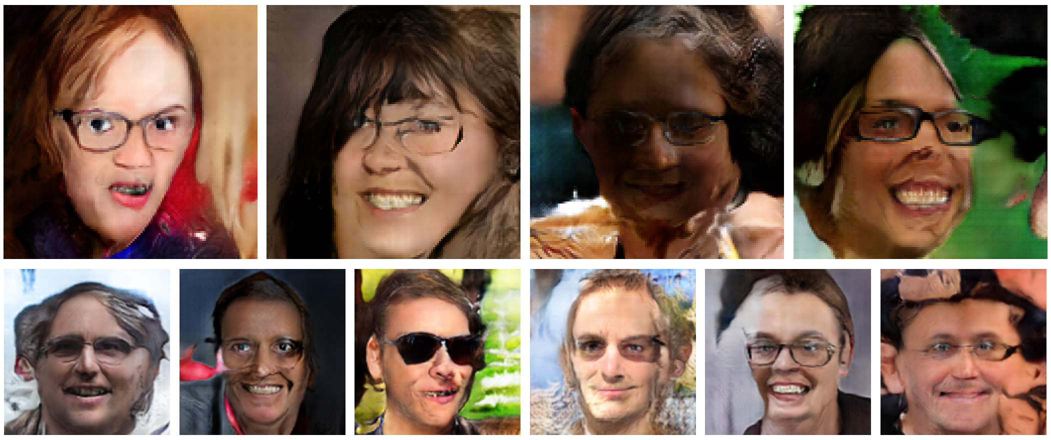
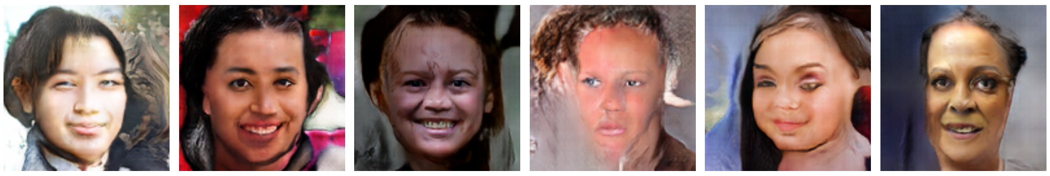
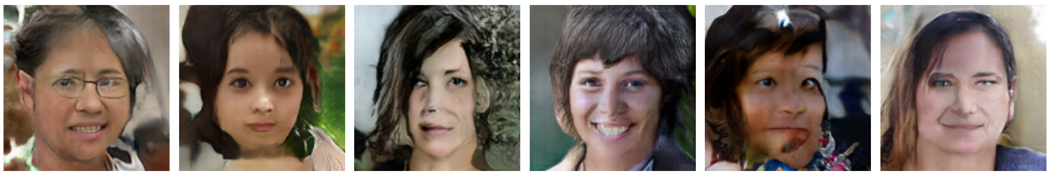
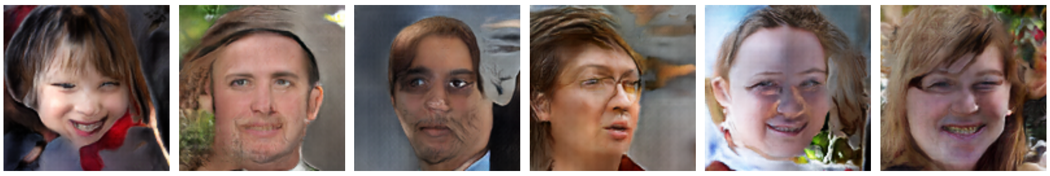


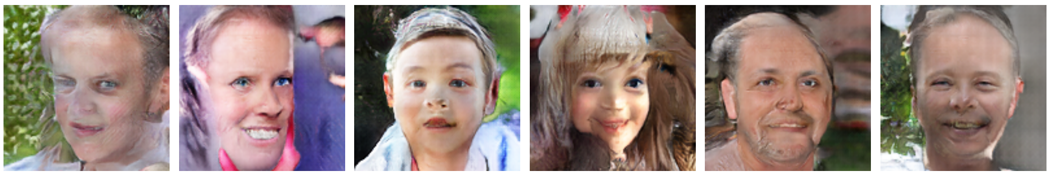
Figure A.5: Random faces from the generator trained for 700 epochs on the FFHQ database.


(a) Faces with glasses.


(b) Faces without glasses.


(c) Faces with black hair.


(d) Faces with brown hair.


(e) Faces with gray hair.

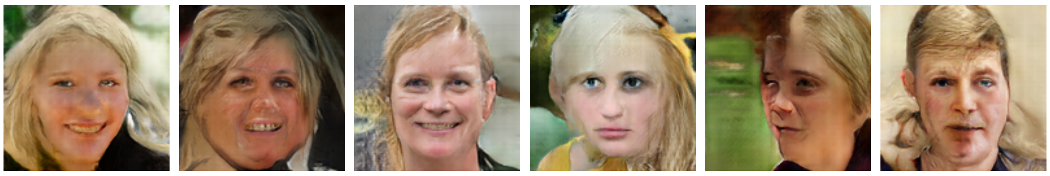
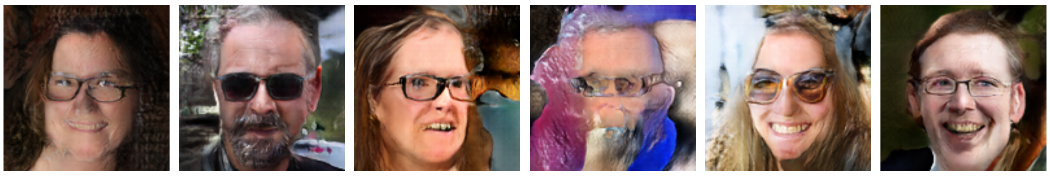
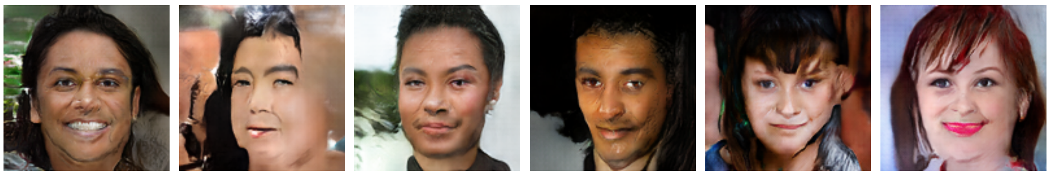
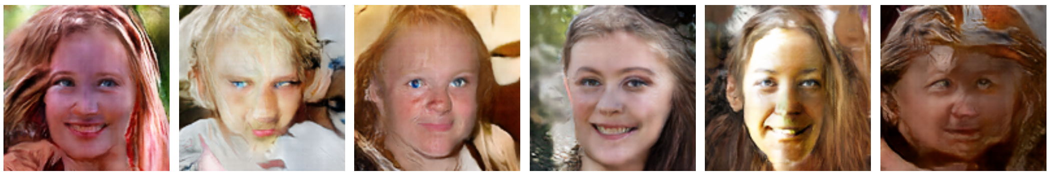
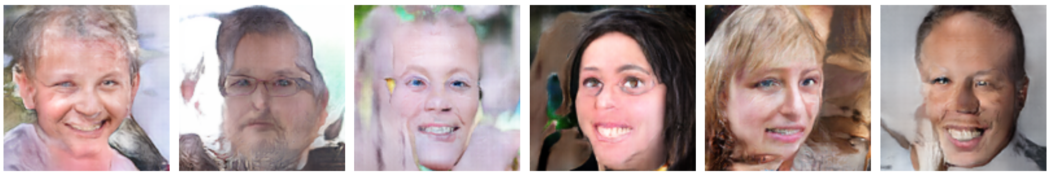

(f) Faces with blond hair.

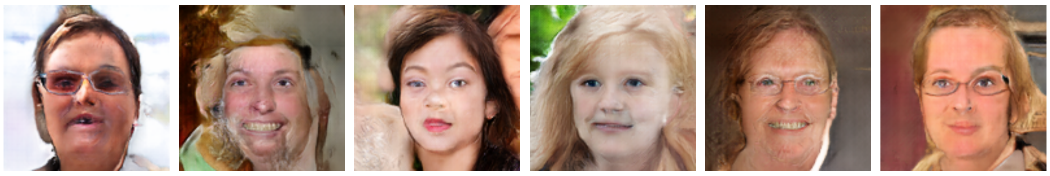
Figure A.6: Manually labeled faces from the generator trained for 700 epochs on the FFHQ database.

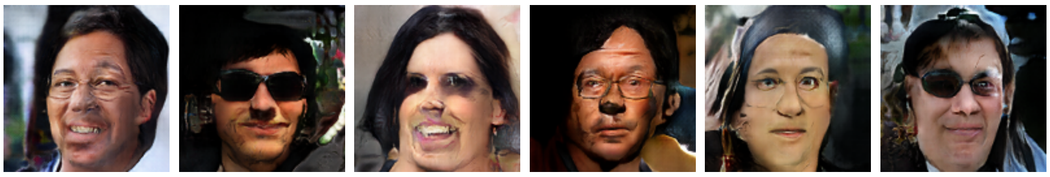

(a) Faces that should have glasses.


(b) Faces that should have black hair.


(c) Faces that should have brown hair.


(d) Faces that should have gray hair.


(e) Faces that should have blond hair.


(f) Faces that should have glasses and black hair.

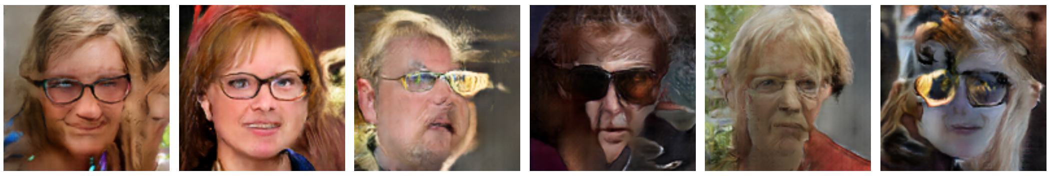

(g) Faces that should have glasses and brown hair.

Figure A.7: Faces generated with requested features from the generator trained for 700 epochs on the FFHQ database. The procedure used to generate face with a specific feature only increases the probability of the feature presence, it does not guarantee a precise result.