Johann Wolfgang Goethe University Frankfurt am Main
Institute of Computer Science Department 12 - Computer Science and Mathematics

Generate coin images from ancient numismatics using Few-Shot Learning

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science im Studiengang Informatik

eingereicht bei

Dr. Karsten Tolle

Sebastian Gampe

von

Jannik Hock

Abstract

Since the website thispersondoesntexist at the latest, generative models are known for being able to generate realistic images of a target domain. However, a strong disadvantage of these models is the required large amount of data, which is difficult to obtain in various domains. For this reason, among others, deep learning algorithms have improved over time to make them more efficient in their use of data availability and resources. The result has been Few-Shot Learning problems that can adapt from just a few examples of a target domain.
The study of ancient coins is a topic that falls exactly into this problem domain. The dataset used here only reaches the critical threshold of 40 for just under 1% of all classes, which indicates when a class can be used for the classification system.
Therefore, the focus of this work is on creating new images of ancient coins using already known GitHub repositories and their approaches of Few-Shot-Learning. Different techniques and algorithms are used to generate new images based on the StyleGAN2 architecture. Numerous experiments show the advantages and disadvantages of each model as well as potential improvements. For the analysis, individual metrics as well as visual results are used to compare the results of each experiment. The goal of the work is to increase the dataset so that each class can be classified using the VGG16 model.

Table of Contents

Table of Figure Contents

List of Tables

List of Abbreviations

GAN	Generative Adversarial Networks
StyleGAN	Style Generative Adversarial Networks
ML	Machine Learning
NN	Neural Network
CNN	Convolutional Neural Network
SVD	Singular Value Decomposition
MAML	Model-Agnostic Meta-Learning
AGE	Attribute Group Editing
cuDNN	CUDA Deep Neural Network
IS	Inception Score
FID	Frechet Inception Distance
KID	Kernel Inception Distance
LPIPS	Learned Perceptual Image Patch Similarity
MoCo	Momentum Contrast
QC	Quantum Computer

1 Introduction

This chapter provides an explanation of the work's essential beginnings. I then briefly go through the goal and structure of this thesis. This chapter gives the reader a clear and succinct understanding of the thesis's subject.

1.1 Problem Proposition

Drawing a dog is harder to recognize than looking at a picture of one. Because of this, generative models are regarded as being substantially more challenging than discriminative models. Another issue is that machine learning frequently lacks the data necessary to accomplish a task, especially with the data-intensive tasks connected to a GAN. Few-Shot Learning is a popular technique for enhancing performance using a dataset with few samples. Few-Shot Learning was initially used with discriminative models, but as time has gone on, it has increasingly been used with generative models. The dataset's domain is another challenge. The projects on GitHub reflect the fact that GANs are most frequently applied to portraiture.

1.2 Project's goal

The project's objective is to create new, high-quality coin images while retaining only a small number of sample images. The generated updated dataset should allow the VGG16 classification model to consider any class. In specifically, two routes are investigated for producing new images. In the first step, the latent code is used to alter the StyleGAN2 encoder to eliminate both category-relevant and irrelevant features. After then, new images can be created by manipulating the category-relevant attributes of a class with category-irrelevant attributes. The StyleGAN architecture serves as the fundamental basis for the second path, just like it did for the first. Only the singular values of the decomposition are trained during the training to approximate the target domain once the Singular Value Decomposition is applied.

1.3 Structure of project

The fundamental theoretical ideas are introduced at the outset, which are important to the work. The most significant earlier works that are crucial to this thesis are then looked at and analyzed. The core of the thesis is elaborated in chapter 4, where it will go into greater depth about the work described in chapter 3 and reveal the precise structure of this work. The class diagrams for each of the different repositories are displayed in the next section, along with a minor implementation choice. Evaluations of numerous significant experiments involving the individual models come next. The work's conclusion highlights the issues that were discovered during the studies and provides a look ahead.

2 Basic Concepts

The foundational elements of the research' created structure are discussed in this chapter. The most significant technologies and algorithms are described, along with their interactions.

2.1 Numismatic

Numismatics is a broad academic discipline that includes the study of history, archaeology, art history, economics, and politics. The phrase derives from the Greek word nómisma, which also means "the coin" or "the valid" [1]. Coins, paper money, medals, and other related items are the focus of numismatics in general [2]. It discusses how coins have been made, distributed, and used in commerce throughout history. Exploring the historical environment, including political and social influences, with the use of coinage is one of numismatics' primary objectives [3]. Numismatists typically do this to learn more about the culture by analyzing the writing and symbols on the coin or note. Numismatics also deals with the choice of materials or production methods, as well as the respective manufacturing processes for the various types of payment. The next section is intended for coinage because the majority of this work deals with coins. Coins have historically been made from expensive materials like gold or silver. Emperor Augustus, who already collected "old royal and foreign coins," was one of the first proponents of coinage. Roman coins can be broken down into various parts, as seen in Figure 1 [3].

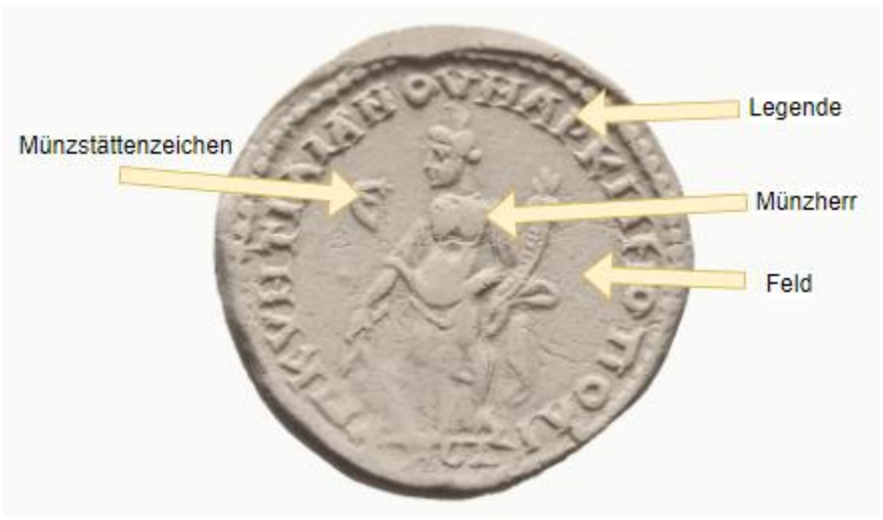

Fig. 1. Image of a coin from the dataset [54]

The legend is arguably the most significant part of a coin, but it can be challenging to read it on ancient coins because it was inscribed on the very outside of the coin and quickly becomes unreadable due to corrosion and wear. The inscription of a coin, which includes the circumscription and inscription, is referred to as the legend in numismatics. The text on the coin's circumference, which can be found on both the obverse and reverse, is referred to as the circumscription. On the other hand, the wording in the coin's center is described as the inscription. The goal of the field, often referred to as the mint face, is to make the coin themes stand out from the circumscription. The field's surface is typically smoothed and flattened to let the mintmark stand out properly. The mint marks, which are found adjacent to the images of the mint lord, tell us which mint or which mint master struck the coin. The portrait of the mint lord, which may show a sovereign or a saint, is found in the center of each coin [4].

Up to the 16th century, coins were minted manually using a hammer and an upper and lower die. In this procedure, the upper die with the coin's reverse side is freely guided and struck onto the lower die while the lower die bearing the coin's obverse side is attached to a wooden anvil. Because of the free guidance, hammer stamping is erratic, making it a distinctive feature.

Because coins are consumable products subject to the whims of nature, coin wear is a significant issue in numismatics. For instance, the metal in the coin corrodes and loses structure if it is dropped in the ground. But even with regular use, a coin loses its shape and color over years [3].

2.2 Machine Learning

A branch of artificial intelligence called ML concentrates on using a lot of data and algorithms to mimic human behavior. A computer can examine complete datasets using statistical learning and optimization techniques, which is the fundamental idea of machine learning [5]. ML can be broken down into three repeated parts, according to UC Berkeley [6].

Decision process: Predictions and classifications are common uses for ML algorithms. The input for the method is anticipated to be data, either labeled or unlabeled, which is then analyzed. A prediction is formed based on the analysis [6].

Error function: The function evaluates the prediction and indicates how strong the error was.

Optimization process: The algorithm examines the previously discovered error in this step and adjusts the model to make the error better in the following phase.

The four different learning models that typically present in ML leverage these iterative procedures [6].

Supervised Learning. The data needed for supervised learning must have class label. Based on classification, the ML algorithm, such as decision trees and Naive Bayes, discovers connections and relationships in the data. In the testing step, the model is examined once it has acquired sufficient knowledge of the data distribution. During this stage, the model must correctly classify any data that lacks labels. The success

rate of the model is then determined by comparing this prediction to the accurate labels. The idea can be applied repeatedly until the desired outcome is reached [6].
Unsupervised Learning. Unsupervised learning differs from supervised learning in that the class memberships of the data are unknown to an ML system. Algorithms based on this learning characteristic aim to identify and learn data patterns. This expertise can be used to classify the data into various groupings. It is crucial to remember that the clusters must be homogeneous within and heterogeneous between each other. The k-means clustering technique is a prominent illustration of this [7].
Semi-Supervised Learning. These two main approaches are combined in this learning behavior. In semi-supervised learning, there are two kinds of data: unlabeled data and data that has already been assigned to a label. This approach aims to group the unlabeled data points into clusters. The classes of the labeled data determine the name of all clusters [8]. There are typically significantly more unlabeled data points in a dataset used for semi-supervised learning and very few data that have class membership.
Reinforcement Learning. The algorithm receives immediate input during reinforcement learning in order to learn from its previous experiences [9]. Robotics is a well-known application field. In this scenario, agents interact with their surroundings, which offer incentives or even penalties in order to boost the algorithm.
It can be seen in figure 2 that ML is a branch of artificial intelligence and that several deep learning techniques are included in it. In the course of this chapter, there are again extensive descriptions of some Deep Learning methods.

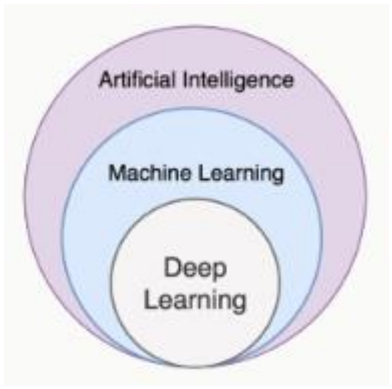

Fig. 2. Intersection of AI

2.3 Numismatic in ML

Numismatics has already seen some interesting work with resulting models in the ML [10,11,12,13]. The state of coins, particularly their corrosion and wear over time, is a major topic in these articles. This issue is precisely one of the many problematics associated with numismatics in ML models, which will be explain in the following.

Condition of the coin

There are numerous studies that discuss the condition of ancient coins, and it is evident that many of the coins that need to be evaluated are in bad condition and hence challenging to analyze [10,11,12]. In general, it can be considered that the toughest challenge in numismatics is undoubtedly the condition of a coin. The coin's condition reveals how much wear and tear it has endured since it was made. Three separate classes are offered as part of the data science training. The most significant characteristics of a coin are represented by a coin in good condition. More fine details can be noticed on the coin when it is in a very fine form. When a coin is nearly in the same condition as when it was made, it is said to be in extremely fine condition [12].

Additionally, if the coin is not centered properly, some details, like the legend, might not be present. Nature is another opponent of the coin's state. Chemical reactions that affect the coin's color or even shape can happen when it comes into contact with the environment [12]. Additionally, one piece specifically addressed the deterioration and wear of coinage [14]. Thus, a coin's corrosion can be determined by holes, incrustations, and exposure to the elements. Another issue is wear, which is represented by the abrasion value of the coin's relief.

Similarities and differences

The overwhelming variety of coins, with often negligible variations between coin classes, is another issue in numismatics. Frequently, the only difference between two sorts of coins is their legend [15]. Additionally, there may occasionally be significant variances within a coins class as a result of the manufacture of the coins. Both of these arguments relate to the fact that ML algorithms are less effective on coins then on other certain datasets.

Another challenge are the high number of different classes in the domain of numismatics. Roman Republican coins are classified into more than 1900 classes and subclasses in the classic reference work by M. H. Crawford [76]. When compared to other datasets, this data collection has a big number of different classes.

Data availability

Despite the large number of classes, the major issue with numismatics in the Data Science discipline is data availability. Rarely do datasets exist that are qualitatively comparable to other datasets from different fields. The issues above mentioned are thus also to blame for the possibility that some data instances won't be added to the dataset due to poor quality. The unbalanced distribution of data in a dataset can also be attributed to the data availability of individual coins, as some coins have a lot of images while others only contain few to none examples. However, several specialized fields also use few-shot learning because their datasets are limited. For this reason, the thesis will also specialize in the method in the following work.

2.4 Neural Networks

Deep learning methods are built on NN, a subset of machine learning. A neural network is organized like the human brain and has three interconnected layers: an input layer, at least one hidden layer, and an output layer. The input layer is in charge of taking in data and converting it into values that the network can read. While the data still undergoing additional processing in the hidden layer, the data will be the output of the output layer. Each of these layers has a threshold value and a weight. Data is sent to the next layer as soon as the output of the layer exceeds the threshold value. No data is sent if the threshold is not reached [16].

Types of NN
The perceptron is the oldest type of neural network. It is referred to as a linear binary classifier since it is typically used to split data into two pieces. In a perceptron, each input is multiplied by its corresponding weight before being added to the other inputs. The strength of each node is reflected in the weights when they are combined together. When it reaches the output layer, it examines the buzz of all earlier input layers and categorizes as a result. The feed forward neural network is another form. The unusual feature of this network is that its information only flows in one direction. Even though the data is sometimes transmitted across numerous secret layers, it always travels in the same direction and never goes backward. These networks are made up of sigmoid neurons rather than perceptrons because a lot of real-world situations are non-linear [17].

Convolutional Neural Network
Another type of neural network is a CNN, although unlike a perceptron, a CNN has a multi-layer Deep Learning design. Here, the CNN is intended to resemble human visual perception and, like a NN, is made up of neurons with trainable weights. CNNs are capable of processing and converting unstructured input, such as images or movies, into numerical values. With Deep Learning, CNN has advanced to the state of the art in computer vision. When given an image as input, the Deep Learning algorithm assigns weights and biases to certain parts of the image in order to identify them from other parts. The fundamental job of CNN is to change the image into a format that makes it simpler to process without losing crucial details [18]. In this scenario, the feature extraction, which is in charge of finding the same features on its own, is what makes CNNs unique. The unique hidden layers enable feature extraction. It is feasible to use the data with preprocessing and handle incredibly big datasets by detecting the features on their own, like the color of the coin.

Convolutional Layer. A CNN includes a convolutional layer in addition to the classification layer, unlike perceptrons, which only have the classification layer. Important elements of an image are extracted at the convolutional layer by compressing them into a matrix form, so lowering their original shape but retaining crucial information. A kernel runs over the input values of the image and extracts the most crucial information to produce such a compressed version. A convolutional operation is dis-

played in Figure 3. Initialized at the top left of the image matrix, a fixed-size filter, or kernel, travels progressively to the right until it reaches the bottom of the matrix. The above-described convolution is computed at each step. The convolution in this case is accomplished via the scalar product between the corresponding marked fields, which decreases the input image's dimension [18]. The range of potential values in the filter is another feature of a convolutional process. A filter's values differ, forcing the process to produce distinct outcomes.
For instance, the operation with the filter (1,0,-1; 1,0,-1; 1,0,-1) of size 3x3 must be applied to highlight the vertical edges in an image. The horizontal borders can be highlighted using the filters (1,1,1; 0,0,0; -1,-1,-1).

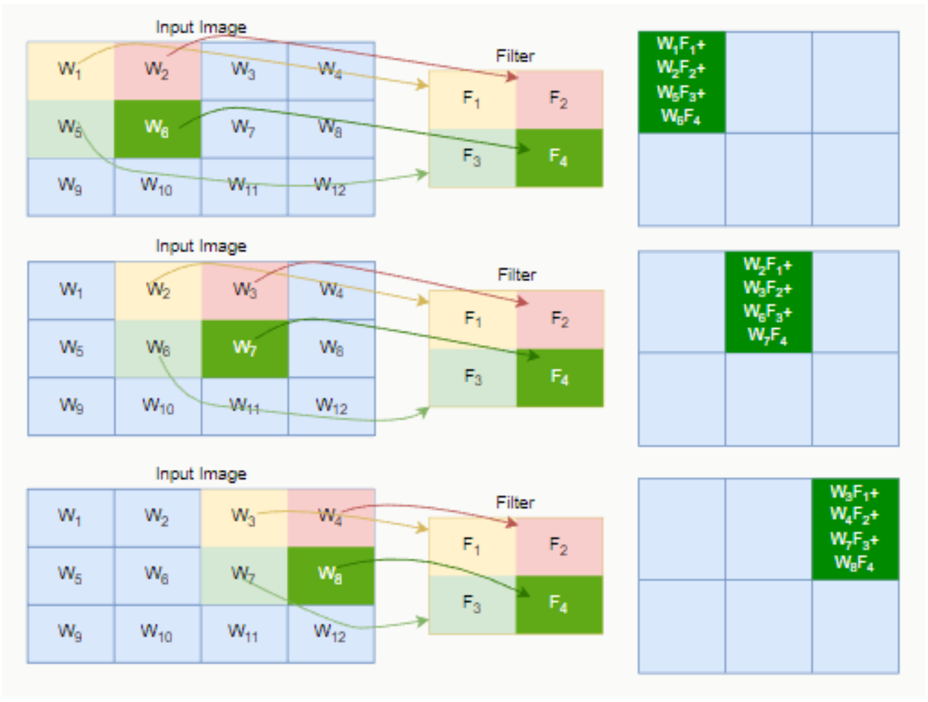

Fig. 3. Convolutional Operation on a 3x4 Matrix with stride 1

Two separate factors that can be changed can affect how the convolutional procedures turn out.

Padding. It is possible that at the conclusion of the iteration there is almost no information left about the image because at each convolutional operation the size is lowered and we can have an unknown number of iterations. Additionally, in accordance with the principle of operations, pixels in the image's corners receive less attention than those in its center. Some number of pixels that store the value 0 can be added to the edge of the image to get around this issue. This stops the image from getting too small [19].

Stride. The step size with which a filter moves over the image can be changed by changing this parameter. This method allows for less convolutional processes, which result in the creation of a matrix with a bigger final size. The Stride parameter is set to one in figure 3. If the figure had a size of two, the third step would actually be the second in the example. A reduction in the output matrix is the result of increasing the parameter [20].

Pooling Layer. This layer is in charge of lowering an image's dimensions and corresponding computing expense. The convolutional operation's output serves as this layer's input. The input matrix's dimension is reduced using an aggregation function. A filter applies to the various input values of the image and reduces the image's dimension, much as the convolutional operation. For compression, there are three alternative aggregation functions that can be utilized. The average pooling operation, which determines the average of all pixel values within the kernel for a kernel size, is one option. The maximum and minimum operations, on the other hand, take the highest and lowest values in the range, respectively. The resulting value is written to the feature map [21].

Fully connected Layer. The CNN's Fully Connected Layer, which is at the very end, is connected to every output. The layer takes the vector from the preceding layer as input and sequentially runs a linear function and an activation function to best classify the image. A vector with a dimension equal to the number of classes is the outcome of the fully connected layer, and each item in the vector denotes the likelihood that a picture belongs to a particular class [22].

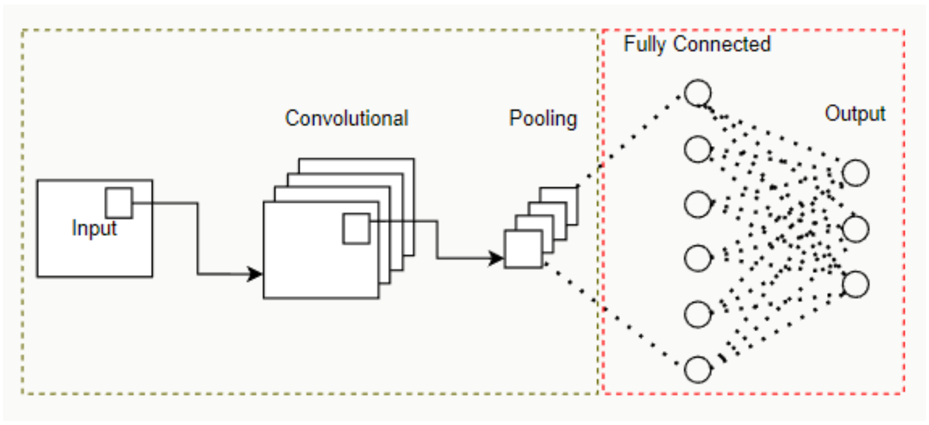

Fig. 4. Architecture of CNN, in the style of [21], extraction of features in green and classification part in red

2.5 Generative Adversarial Networks

In recent years, the machine learning technique known as GAN has gained a lot of popularity. The GAN is an example of unsupervised learning and allows two AI models to compete against one another, hence the term "adversarial." The model is made up of two smaller sub-models that keep an eye on one another. The discriminator is a discriminative network, whereas the generator is a generative network. The generator's task is to create fake instances for a target domain, and the discriminator's task is to determine if a given example is real or fake by using the generator's fake or a real example from the target domain. In other words, by using fake examples, the generator seeks to deceive the discriminator [23].

Discriminator. A CNN-based image classifier called the discriminator makes an effort to separate fake images produced by the generator from real data from the target domain. A discriminator requires data from two independent sources in order to be trained. There are two types of examples: first, actual data instances from the relevant target domain that are used as positive examples by it, and second, fake examples produced by the generator that are used as negative examples. The discriminator provides connections between two loss functions, which updated the networks weights based on the outcome of the loss functions using backpropagation. The discriminator loss is directly used to improve the discriminator by penalizing it when a real data instance is classified as false or a fake instance is classified as real [24]. The discriminator is a straightforward CNN made up of numerous convolutional layers, which causes the images to get smaller with each additional layer while gaining depth. The leaky ReLU activation function, which offers outstanding training stability, is employed after each convolutional layer. A sigmoid activation function is utilized after the final activation function to get the classification's deciding probability [25].

Generator. A Generator is an autoencoder, or CNN, which consists of an encoder and a decoder. It gains knowledge from the discriminator's feedback and develops fake data that the discriminator should be label as real. Convolutional operations used by the encoder to break down the input data are then reconstructed by the decoder using internal representations and learning weights. The most crucial meta-information of the image is retrieved during decomposition, enabling the creation of comparable yet diverse images [26]. Given that it is initially unsure of what potential data instances might look like, the generator in this process uses a random input, much of which is random noise. A uniform distribution, for instance, can be utilized as the input as the noise's distribution is not important. The GAN can produce a variety of data that may also be located at various distances due to the noise that is converted into a data instance by a network [23]. The weights of a neural network are modified during training in order to lower the loss function. However, one is not connected to the loss that is supposed to be adjusted in the generator. The discriminator network is required for this since it produces the desired outcome. As a result, the effects of a generator's weights heavily depend on the discriminator's weights. So the backpropagation begins at the discriminator's output rather than at the generator itself. The discriminator transmits his output of the classification to the generator [27].

How does a GAN work? In a two-player, zero-sum game, the discriminator and the generator of both networks compete to outperform the other. With a lot of data from the target domain, the discriminator is initially trained alone. The discriminator gains knowledge of the essential traits and characteristics of the instances from the target domain during training. When the discriminator is training, the generator is frozen and doesn't turn on until the discriminator has finished training. Following the completion of the discriminator's training, the generator constructs a random input vector, also known as a noise vector, to produce its own fake samples from the target domain. The discriminator then receives the created images and must determine if it is a real or fake example. The right result is subsequently communicated to both models, who are then modified in accordance with the outcome. The winner of the game doesn't change, but the loser has to adjust his model's weights. The objective is for the discriminator to lose its ability to discriminate between false and real after a significant number of iterations. A GAN's main parts and structure is seen in Figure 5. Both sub models are usually implemented as a CNN or a Transformer, since both are excellent at identifying patterns in images [28].

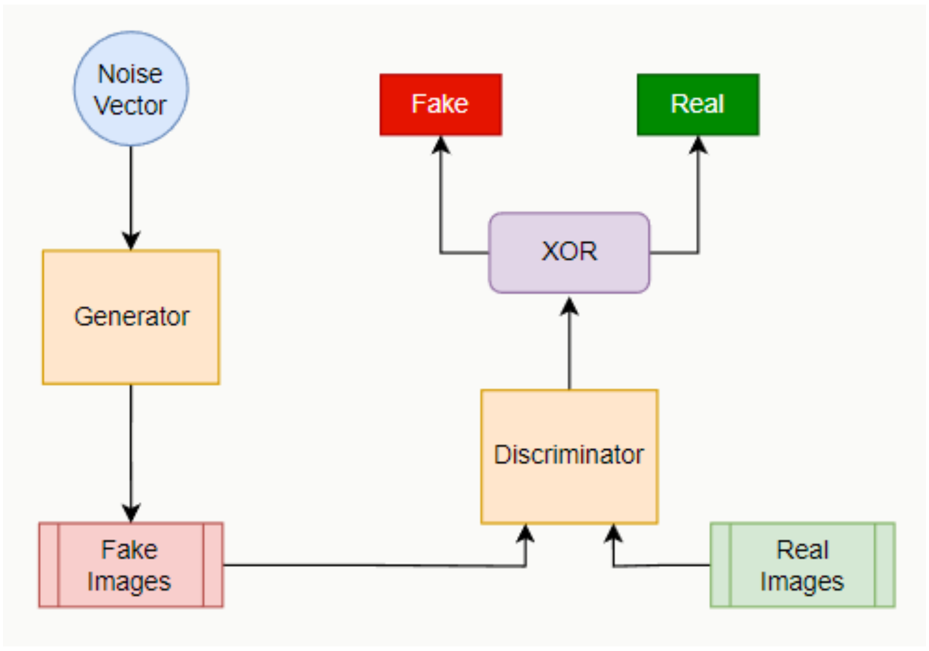

Fig. 5. Architecture of GANs, based on [23]

GAN loss-functions. The standard loss function, also called min-max-loss, was first introduced by Ian Goodfellow in 2014 [23].

$$E_x[\log(D(x))] + E_z[\log(1 - D(G(z)))] \quad (1)$$

The min-max loss is so named because the generator seeks to minimize the function while the discriminator seeks to increase it. The discriminator's assessment that x is a real data instance is expressed as D(x). The output of the generator with a random vector input z is G(z), and E_x is the expected value across all data instances. D(G(z)) is the likelihood that the fake instance z is real. Finally, E_z is the expected value across all of the generator's random inputs. There are two additional parts to the conventional GAN loss function [23].

Loss of Discriminator. The discriminator distinguishes between actual instances and fake examples passed by the generator while it is being taught. By doing this, it punishes itself for making wrong classifications, such as judging real photos to be fake or evaluating fakes to be real.

Loss of Generator. The generator searches for a random input vector and sends it to the discriminator while it is being trained. The discriminator's categorization is then used to compute the generator loss. If the discriminator is fooled, the generator is rewarded; if not, he is penalized.

2.6 Singular Value Decomposition

With the help of their eigenvalues and eigenvectors, quadratic matrices may be completely defined, and their effects on vectors can be demonstrated. There is a SVD that can be used to attain this attribute also for asymmetric matrices. Here, the SVD represents unsymmetric matrices in their canonical form and contains singular values that can be easily interpreted as eigenvalues [31].
It is possible to read off the y-axis intercept in a function's canonical form, also known as normal form.
A vector with the same size as the original vector is produced by multiplying a square matrix by a vector. By rearranging the equation, the eigenvalue issue shown in formula 2 can be solved to determine the eigenvalues (lambda) and the eigenvector (v). The problem is thus solved by looking for a vector that, when multiplied by the symmetric matrix, produces the identical vector as a result, just multiplied by a prefactor [31].

$$A*v = \lambda*v \tag{2}$$

Singular Value Decomposition. An asymmetric matrix is factorized by SVD into three unique matrices. The three distinct matrices that emerge from the factorization of A are displayed in Figure 6. The orthonormal eigenvectors of the matrix $A*A^T$ are stored in the matrix U. The singular values of the matrix A are contained in the matrix Σ, while the orthonormal eigenvectors of the matrix A^T*A are contained in the matrix V. The purpose of the SVD is to extract the most crucial data from the matrix by making the singular values understandable, much like the eigenvalues in a symmetric matrix [31].

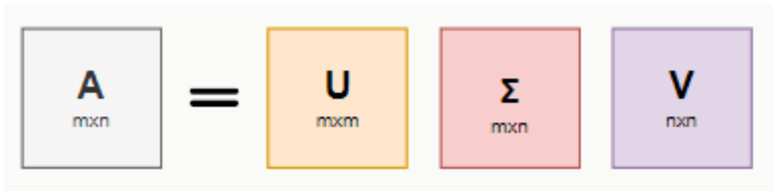

Fig. 6. Singular Value Decomposition, in the style of [31]

Singular values. The singular values define the most significant characteristics of the matrix, much like the eigenvalues in symmetric matrices. Since only real numbers are present in the matrix, they have an advantage over the eigenvalues for that they are easier to understand [31]. The square roots of the eigenvalues from A*A are what the singular values in this case are. The two Gene Golub techniques produce the singular value computation. Together with Kahan, the direct technique was created in 1965 and in 1970 the iterative method with the help from Christian Reinsch was created. Both approaches use orthogonal transformations to convert the matrix to the bidiagonal form and the QR algorithm to determine the singular values [32]. The singular values in the matrix ∑ are sorted in descending order, with the strongest singular value at the front position. A matrix's singular values have several significant characteristics and applications. The principal component analysis, a dimensionality reduction method used in machine learning and data analysis, likewise heavily relies on them. The directions of maximum variance in the data are identified using the singular values of the data matrix in the analysis. These directions can be utilized to reduce the dimensionality of the data while retaining as much information as feasible.

Singular vectors. In the figure 6 you can see the two singular vectors. The major axes of stretching or compression applied to the column space of A are represented by the columns of the matrix U, which are known as the left singular vectors. The major axes of stretching or compression applied to the row space of A are represented by the columns of the matrix V, which are known as the right singular vectors. Just like the singular values, the singular vectors are also used for the principal component analysis [32].

Singular Value Decomposition in Data Science. SVD includes a number of advantageous features for data scientists. By simply obtaining the most crucial details about the solitary values, it first aids in the reduction of datasets with multiple values. Additionally, tiny datasets can benefit greatly from the application of SVD [33]**.**

Dimensionality reduction. Here, SVD can be used to identify a dataset's primary components, which show the information that accounts for the most volatility. The SVD decomposition is used in this instance to calculate the covariance matrix. The k most crucial major features can now be selected in order to decrease the dimensions while yet retaining as much data as possible. Costs of computation are significantly reduced as a result, and machine learning methods may also benefit [46].

Data compression. Another point is the reduction to data of large datasets. By decomposing the dataset, the k-best singular values and the corresponding singular vectors can be taken to approximate the original matrix while reducing its size.

SVD in recommender systems. Products are frequently suggested to the user in online applications like Amazon or even music apps based on past searches. To suggest the appropriate products to clients, numerous strategies are employed, including SVD.

Two distinct predictions can be made by an SVD recommendation model. One is making product recommendations to users or making predictions about specific items for people [34]. Microsoft uses SVD, among other things, for the Evaluate Recommender part of Azure Machine Learning Designer.

Take Netflix's movie suggestions as an illustration of suggesting things. Here, a matrix M includes the customers' individual ratings for each movie in addition to the evaluators' ratings for each movie as a row and column, respectively. Thus, the relationship between the movie and the rater is used by SVD to extract features and correlations. The movie's genre is just one illustration. In order to do this, SVD would produce factors from the matrix factorization of, say, humor vs. action [35].

2.7 Few-Shot Learning

Few-shot learning is a branch of machine learning that focuses on model training with a small amount of data. Typically, highly accurate machine learning models need a large amount of data. Few shot learning is designed for the model to learn in the same way as the human brain. This is because machines are expected to learn in a similar way to humans on the basis of a few examples. Additionally, this implies that trustworthy models can be established for specialized domains with little data. Naturally, this can also save on data and computational resources since a good model only requires a small amount of data [36]. Robotics and other fields, such as Natural Language Processing, use few-shot learning. In general, there are two different approaches to solve few-shot learning problems.

Data-Level-Approach. The data-level technique is the less complicated of the two methods. The data-level approach focuses on making the limited training data available more meaningful and informative for the actual task. This can be achieved in particular through data augmentation or the generation of synthetic images using generative models. The goal is to make the internal representation of the data more understandable to algorithms, which use these data. [37]. In order to increase the size of the necessary dataset, the strategy that is essential for this work is to generate new images using generative models like GANs.

Parameter-Level-Approach. Another method focuses on the parameter level of each individual ML model instead of the input data. The techniques emphasize how the model's parameters are more flexible and can fast adjust to new demands since they are more general. This method aims to train a model that can adapt to just a few samples of a new target domain without requiring a complete retraining of the model. Meta-learning, with its well-known algorithm MAML, is a well-known technique [38].

MAML. The model's parameters are very slightly changed, often just by one gradient step as soon as a new task needs to be learned. By doing this, the learner's parameters are adjusted depending on the few training datasets for the novel task. The initialization of the parameters in this process by MAML is robust and dependable, enabling quick and efficient learning without running the risk of overfitting, which is immediately anticipated with sparse data [38]. First-order MAML is a unique variation of MAML that significantly minimizes the algorithm's enormous memory consumption.

Few-Shot variants. Four main forms of Few-Shot are typically distinguished.

Without any prior training, zero-shot learning creates previously undiscovered classes. It differs from the other variations in that sample data for each class is not required. This makes it possible to use fewer labeled training data. Without examples, new classes are acquired by blending knowledge from existing ones. The objective is to mirror human learning's capacity for matching unfamiliar items [39].

Machine learning models may identify and categorize new classes using just one sample in one-shot learning. Face recognition is one example of a traditional application field.

Few-Shot Learning makes it possible to form Machine Learning models more accurately with fewer training data. The other three variations are also included in the broader idea of N-Shot Learning. The purpose of all four versions has been to prioritize the minimization of the required test data.

Few-Shot classification. Few-Shot Classification uses a small sample size from a dataset to train a classifier to identify previously unidentified classes. There have been a variety of intriguing ways to implement Few-Shot Classification in previously published work [39]. Metric-based meta-learning and optimization-based meta-learning are two of the most widely used algorithms.

The most crucial concepts that are essential for the remaining work were defined in this section, and a relationship between them was demonstrated. The concepts and the described processes will be clarified once more in the following chapter by using examples from other works that are close to our issue.

3 Related Works

The topic of few-shot learning has been addressed in a number of articles [40, 41, 42, 43]. The different approaches to Few-Shot Learning in the papers are analyzed in this chapter.

The paper "A Style-Based Generator Architecture for Generative Adversarial Networks" addresses the issue that conventional GANs lack control over specific image-generation properties. Here, adjustments are performed to the generator, in contrast to other methods that alter the GAN's structure at the discriminator. As a result of the new generator architecture's separation of an image's style from its content, it is now possible to teach the generator to produce images in a specific style while maintaining the information about the image's content. A linear affine transformation is used to enter the style code into the generator [44].

The authors suggest a mapping network to enable this separation of the style from the content. The network changes this input into another latent space instead of allowing the initial random noise vector to be put straight into the generator architecture of the GAN model. The differences between these two approaches are shown in figure 7. In the StyleGAN, the random noise vector serving as the network's input is the same size as the processed output at 512x1. One can alter any style without impacting the others by separating the various styles from one another. The disentanglement of the styles that occurs means that a wide range of images can be produced without losing the image's most crucial structural elements [44].

The synthesis network creates the final image by combining numerous convolutional and upsampling layers. The synthesis network uses the output of the mapping network as its input, and the synthesis network then employs this input to produce an image in a multi-scale manner. A low-resolution (4x4) version of the image is first created, and this image is progressively improved until the final high-resolution of 1024x1024 pixel image is created [30].

The StyleGAN, which was first shown by Nvidia in 2018, is one of the most well-known GAN extensions. StyleGAN2, which debuted in 2020 and offers some improvements over StyleGAN, is an extension stage of StyleGAN [23].

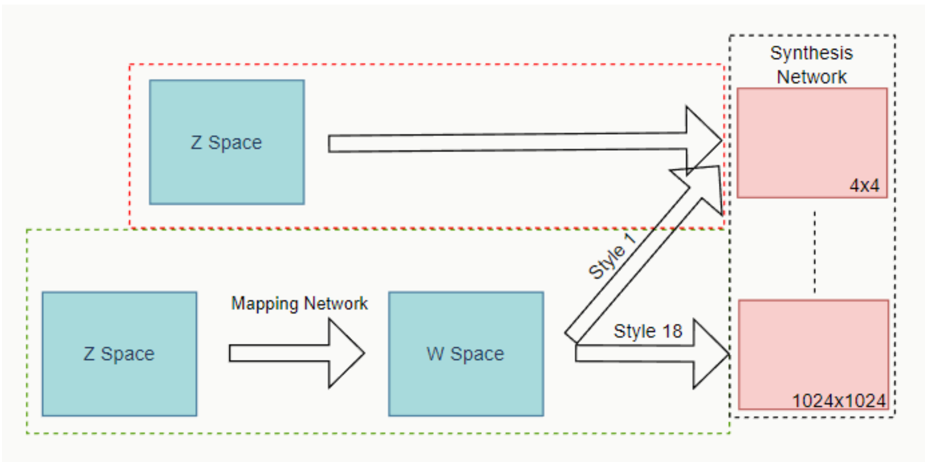

Fig. 7. Pre-Generator Architecture of traditional GANs (red) and StyleGAN (green)

The paper "META-GAN for few-shot image generation" deals with the generation of few-shot images for GANs, using the two meta-learning algorithms MAML and Reptile. In doing so, the authors attempt to generate new realistic images from only a few examples. The work shows that images with high similarity are generated, but with little diversity in the output images. The approach was tested on the MNIST dataset with both TransGAN and Res-SNGAN networks, among others. At training time, digits zero through eight were taken as the dataset and images of the previously unknown class nine were generated during testing [45].

The authors Robb et al. provide a novel method that deals with singular value decomposition in order to enhance the issues of data availability and the related generalization of GANs as well as to adapt more quickly to a new domain. In this method, the singular vectors are frozen throughout training to take use of SVD's property, and the singular values of the pre-trained GAN are only updated after SVD. A substantially smaller parameter space is produced during training by freezing the singular vectors, enabling the network to quickly adapt to new domains. For this, the authors make use of a StyleGAN2 model that has already been trained on portraits (FFHQ). With the help of two alternative methods, the algorithm is tested. First, on a target domain that is related to the source domain (FFHQ ➔ Celebrity Portraits), and secondly on a slightly different domain (FFHQ ➔ Art Portraits). Five examples from the target domain are more than enough to produce images of high quality, which is an excellent outcome for the near-domain adaptation [46].

An alternative strategy refers to an issue caused by the StyleGAN architecture. Here, the inputs have to be converted into W-space before being embedded in the StyleGAN2 generator architecture. Major challenges still exist with the Latent Vector Optimization and Encoder-Based Inversion techniques currently being used for the process of the inversion. While optimization requires too much time, inversion is insufficiently precise. The authors suggest an encoder that may be trained using a pre-trained StyleGAN to close the gap. The pre-trained StyleGAN's generator architecture receives the input code from the trained encoder, which outputs it in the W vector space. The most crucial data from an image is extracted by the device's latent code, which is the output after receiving an image as input [41]. A CNN network named style2map, created especially for StyleGAN architecture, was used for the implementation in order to output the latent code of the image. To improve the encoder's performance, the two loss functions identity loss and regularization loss are used during training. With the aid of this technology, the various StyleGAN pyramid scales will be mapped and integrated into the synthesis network.

The writers employ a face identity loss in their training since they start with portraits of real people. Comparing these results to earlier work [42,43], they are incredibly encouraging. The methodology from the pSp encoder can be easily adapted to other domains thanks to a study from 2021, in which an implemented MoCo-based ResNet was utilized for feature extraction instead of an ArcFace network [47].

The restyle encoder is an alternative strategy to the psp encoder. Once more, a trained StyleGAN is used to train an encoder to convert an image's latent code into the W vector space. This system is an iterative improvement process, unlike the technology mentioned above. The encoder's output is utilized as the input for the subsequent iteration after each step. The iteration process is running until the encoder's output resembles the original so that the latent code produces a good outcome [48].

The study by Ding et al. utilizes one of the aforementioned encoders to solve the issue of few-shot image production while utilizing a novel technique known as AGE. AGE is a method for creating images that relies on editing and has very few examples of the target area. The authors make the assumption that every image is made up of a set of characteristics and that the direction of attributes within a class is common. These internal representations are precisely what AGE analyzes to determine the semantic directions of characteristics. As a result, each class has attributes that are relevant to its category and those that are not. Using AGE, images of a class can be created by processing the category-irrelevant attributes from the irrelevant attributes of other classes while leaving the category-relevant attributes untouched [49]. Prerequisites for the work are both a well-trained StyleGAN and an encoder architecture, which transform latent code in the W-space. The methodology in the field of Few-Shot Learning is considered to be one of the transformation-based methods. A training set with a set of M classes that are already known at runtime makes up the structure of the experiments, while the test set is made up of classes that are still unknown at training time and only contains a small number of data. The model simply needs to be trained once and doesn't need additional training cycles to provide images for all classes from the test set.

The paper by Liu et al. is about generating higher resolution images for smaller datasets. For this purpose, the authors use a new approach to image synthesis, which aims to solve two difficult challenges. The first is the consistency of the training, and the second is the quality of the images that are produced. The authors provide two new methods for solving these issues. It will be simpler to alter an image with styles and content from other images because the generator's Skip-Layer Module skips layers with smaller scales. In addition the discriminator is nevertheless regularized by a self-monitored extra output. Here, a novel loss function balances the contributions of the global discriminator and a number of smaller local discriminators. In several benchmark datasets, the findings indicate that the developed method is superior to the previous state-of-the-art methods. It specifically displays how well it can create images from a limited set of examples [50].

The research by Zhao et al. addresses the scarcity of training data as well. The authors suggest Differentiable Augmentation to increase the data efficiency of GANs by preventing the discriminator from remembering the precise dataset. They do this by adding various forms of distinguishable augmentations to both actual and fake images. One of the most popular ways to use overfitting is data augmentation. Examples include cropping, rotation, and color jittering. The FFHQ, LSUN-Cat, CIFAR-10, and CIFAR-100 datasets were used to test the architecture [51].

Two papers even go one step further and train a generative model on only one example image.

The goal of Shaham et al. is to create a novel generative model that can generate several images while maintaining the original image's style and structure. The model is trained on just one example image. The so called SinGAN model does this by employing a recursive architecture made up of numerous resolutions, each of which is trained using the results of the resolution before that. The algorithm is created so that the discriminator does not see the complete image, instead the image is divided into several overlapping sections (patches), preventing the discriminator from memorizing the sample image. The discriminator then assigns a real or fake classification to these patches. The generator with the least resolution is first fed a random noise vector. Either the output from the generator or the sample image that has been downscaled serves as the input for this sized discriminator. The generator is trained until the output can fool the discriminator. If so, it is possible to upscale the generator's output so that the image has the same resolution as the generator which comes next. This generator accepts the upscaled image as input just like it would a random noise vector with the same resolution. Up until the generator with the highest resolution is found, this process is repeated. It is significant to notice that the discriminator sees a decreasing percentage of the patched image as resolution rises. As a result, the generator must only produce real patches of the low-resolution image as resolution rises. This architecture produces realistic images at all possible resolutions [52].

A generative model known as the Generative Model One-Shot GAN can be trained on a single example and can thereafter produce various new patterns belonging to the same class. [53]. They demonstrate how the generated images lack diversity since the usual GAN discriminator [23] simply memorizes the image. The objects in the sample image are distorted by the patch-based GAN discriminator, which is frequently employed to prevent overfitting. The authors present a two-branch discriminator to address the issues. Here, an ordinary pooling procedure is used to transmit the image representation to the content branch. This branch describes the content of the image but not its location. The image is condensed to a channel for the Layout Branch using a Convolutional Layer. This branch describes the locations of the objects but not which ones are mapped. To prevent potential overfitting, the Layout and Content Branches each get a scaled-down version of input from the corresponding prior levels. The loss function of each branch and the loss function following the image representation are both used to determine the discriminator loss. Both the generator and the discriminator are optimized using the loss functions [53].

The key concepts and models that are crucial to my work were defined in this chapter. The purpose of this chapter was to make the subsequent chapters simpler to comprehend and understand. Some prior research that is relevant to my implementation and theory was described for this purpose.

4 Core of work

This chapter describes the problem of my task and explains in detail how the previous works described above is used for my thesis. Here, some applied algorithms and models are shown again in more detail.

4.1 Cause of work

As of the current date, 20 October 2022, there are around 11,000 different coins in the corpus nummorum database, each with a unique number of images [54]. This database is used to construct a dataset in which each type of coin represents a class. This separates each image of a coin into an obverse and a reverse image. A VGG16 network is trained to properly classify the various coins using this dataset as its training data. The network's current accuracy figures for the Top-1 are 79% and the Top-5 are 97%. However, only classes with more than 40 images were eligible to be a part of the dataset for the VGG16 classifier.

As a result, not all classes in the dataset can be used as input for the VGG16 network due to the critical threshold. In order to exceed the barrier, new images must be created for about 99% of all classes.

4.2 Solution approach

The goal of this thesis is to expand the dataset by creating new images of the classes with fewer than 40 examples using generative models. This is done in order to use potentially all classes of the dataset for the VGG16 model.
It has been established that Few-Shot Learning can be used to build images of a class with only a few examples, as discussed in Chapter 4, and that the lack of data in this area is not a new problem in the field of data science.

In this thesis, a few distinct generating models are employed and contrasted in order to have a big variety of generative models and hence be able to guarantee many experiments and evaluations. The basic framework for this work is the StyleGAN2 generator architecture [55].

As many classes as in the comparison experiment, 164, are fed into the VGG16 network in order to examine the quality of the generated images in the classification model, and the results of the network with the created images are compared with the previous values of Accuracy.

4.3 Dataset

Any machine learning algorithm depends heavily on data, and the quality of the dataset is a crucial component. This is so that Data Science algorithms can only produce findings that are at least as good as the training data. The information for this research was obtained from a public database [54] and consists of 69,921 total images divided into 11,105 classes. Figure 8 illustrates the dataset's organizational structure, which is comprised of subfolders that correspond to the different classes. The images for each class are organized into front (obv) and back (rev) images in the subfolders. The size of the images is not consistent, but will be limited to 256x256 pixels during this work.

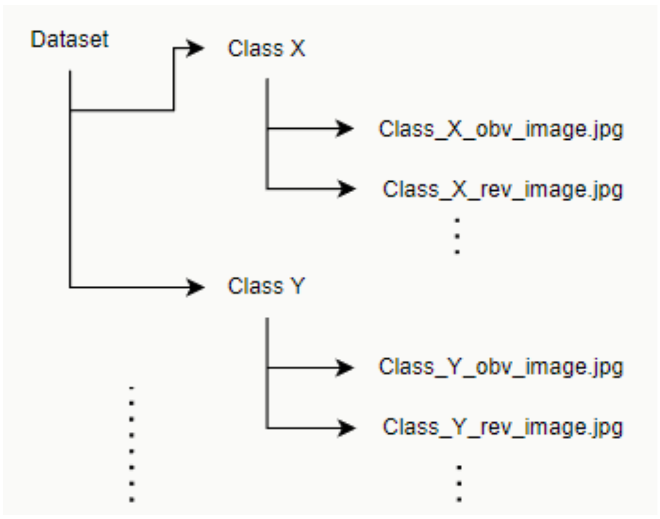

Fig. 8. Structure of our Dataset

To get a more detailed overview of the dataset used here, Figure 9 shows the distribution of images per class. It is notable that almost every class, nearly 99%, have less than 40 images stored. The graphic clear demonstrates that to guarantee the threshold for each class, new images must be generated for almost every class. Furthermore, the dataset's apparent imbalance is noteworthy. The goal of each dataset must be an even distribution of instances across all classes. There are several ways in which unbalanced datasets can be fixed. One method is resampling, which can be achieved by either over- or undersampling. Undersampling removes as many images from the larger classes until they are at the same level of the smaller classes. Oversampling, on the other hand, is not as intuitive, because here new data must be generated from the minority classes so that they end up with as many examples as the majority classes. The method most commonly used for this is the Synthetic Minority Over-sampling technique, which generates data points on the line segment connecting a randomly selected data point to one of its K-nearest neighbors.

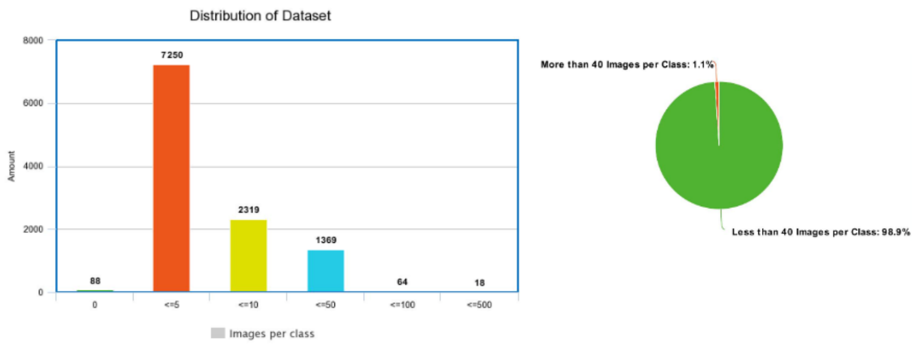

Fig. 9. Distribution of our Dataset

4.4 CUDA und cuDNN

CUDA stands for Compute Unified Device Architecture and was developed by NVIDIA and first released in 2007. The Parallel Computing Platform is a programming model that allows developers to use the power of NVIDIA GPUs for general computing tasks and only requires a suitable NVIDIA GPU. GPUs actually designed for graphics computing are processors that work in parallel, allowing them to perform multiple computations simultaneously, making them ideal for complex computational tasks such as machine learning. The platform is usable for many programming languages, including Python and C++, and provides a runtime system with a compiler and a set of qualitative libraries. CUDA is popular in both computer vision and natural language processing [56].

The NVIDIA-developed cuDNN is a library for deep learning using neural networks. It was created to work in tandem with the CUDA platform to speed up Deep Learning training and inference on NVIDIA GPUs. The library provides powerful implementations of many deep learning operations, including as pooling, normalization, and activation functions, which are crucial building blocks for deep learning architectures like a CNN. TensorFlow and Pytorch can have their performance significantly increased by an NVIDIA GPU with the help of cuDNN. An important benefit of the library is that cuDNN handles GPU performance optimization for the customer, saving them time and effort [56].

4.5 Methodology and procedures

In this chapter it will go into great depth regarding each of the used repositories. It will be demonstrate both their similarities and connections in this way. In a subsequent chapter, the work's structure is displayed in exact detail once more.
First, the algorithm's foundation, the Nvidia StyleGAN2-ada, is demonstrated. On the basis of this, the two pathways of my research are explored, together with their algorithms.

Nvidia StyleGAN2-ada

The StyleGAN2-ada repository from Nvidia aims to create a generative model that can produce excellent images of the target domain. By doing this, training can produce pleasing results with as few as a few thousand instances. Because the training time is up to 30% faster than the Tensorflow implementation, I decided to utilize the Pytorch implementation of StyleGAN2-ada. StyleGAN2 is an upgraded version of StyleGAN, particularly in terms of output performance and image quality [55].

A StyleGAN2 model's input is a structured dataset that classifies all of the images into folders. After finishing the training, you will have access to a generator that can produce high quality images for each target domain.

Functionality. You can quickly and easily produce images with high resolution using StyleGAN2. In order to continuously raise its resolution, the StyleGAN2 begins by creating images with low resolution. The StyleGAN2 scales up from a resolution of 4x4 to a dimension of 1024x1024. With the StyleGAN, the generator considers the image to be a composite of various styles, where each style influences certain aspects of the image. The authors make three distinctions between them [30]. I'll briefly take the example of a portrait to demonstrate the styles in greater depth.

Coarse Styles. The resolutions 4^2 and 8^2 both using these styles. High-level aspects of the image are analyzed in this style. Here, among other things, the hair or the face's contour are examined.
Middle Styles. The resolutions 16^2 and 32^2 assess facial features like the eye and the nose. The properties are automatically segregated from a variety of potential noises and textures.
Fine Styles. The entire color schemes are examined in resolutions ranging from 64^2 to 1024^2.

Latent Space. A latent space is a high-dimensional representation of data that is used in artificial intelligence and machine learning, where each point in the space represents a particular instance of the data. By expressing the data in a more condensed, intelligible, and approachable manner, latent spaces aim to capture the underlying structure of the data.

In our example the data instance is an image, which are pixels, who can be thought of as high-dimensional representation of the image. The size of this representation makes it challenging to process, thus it is transformed into a latent space with fewer dimensions. However, even after change, the image's fundamental structure is still there. Thus, using a latent space makes it possible to manipulate the image more easily and analyze a big volume of data more effectively.

W-Space. In conventional generators, the synthesis network receives the input, which is in latent space R, directly. In contrast to the latent space R, the space W produces a vector with a dimension of 512 × 1. This latent space is used by StyleGAN and represents the distribution of the training data more precisely than the conventional Gaussian latent space. It is possible to separate an image's styles from its content using this latent space, allowing for the modification of certain styles without affecting the image's overall structure. Due to the separation, the space also have unbundled properties that are responsible for extensive image manipulations [30].

StyleGAN2 Generator. Figure 10 depicts the two current networks of the generator layout. The initial input vector is transformed by the mapping network, preparing it for the synthesis network. Each convolutional layer needs the mapping network's output as its input. There are exactly two convolutional layers for each resolution in our total of 18 layers (4x4 - 1024x1024). The linear affine transformation turns the encoded mapping latent vector w into a style-image, which is used as input for the Weight Demodulation (Demod). Also the output of the previous convolutional layer is an input for the weight demodulation [57]. Modulation and demodulation make up the process.

Two style blocks make up a resolution block, each of which has a modulation, demodulation, and convolutional layer. The resolution is doubled as soon as a new block is reached. Until to 1024x1024 resolution is reached, this process is repeated.

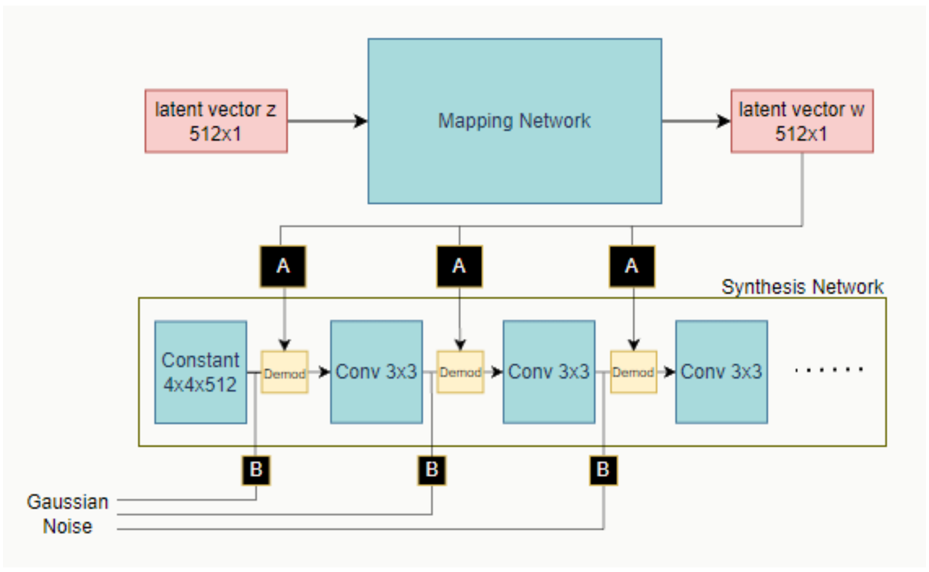

Fig. 10. Architecture of StyleGAN2, in the style of [29]

StyleGAN2 Discriminator. The discriminator reduces the resolution for every subsequent block, in contrast to the generator, where the resolution increases. The image is transformed into a feature map with the same resolution by the discriminator. Beginning with the block from the image's resolution, this feature map covers several rows of blocks. The resolution is half and the amount of features is doubled with each new block that is added. The categorization, actual or fake image, is done at the conclusion of the last block, which has a resolution of 2x2 [29].

StyleGAN2-ada. The reason that GANs can be trained with little data is due to the extension of StyleGAN2. Data augmentation approaches are used to enable training without overfitting. The three augmentation methods that are most frequently used in generative models are color transformations, random 90 degree rotations, and isotropic image scaling. These three metrics, however, have the feature that the generator is unable to later discriminate between images that have been edited and those that are in their original state. Because of this, the authors have developed methods for the generator to recognize augmentation. Stochastic discriminator augmentation is a technique that modifies a uniform distribution of probability. An augmentation may be given a probability that represents how frequently it is activated. The greater the probability of augmentation, which is in the interval [0,1], the more augmentation was added to the image [57]. The Adaptive Control Scheme is a different way for avoiding overfitting. Here, overfitting in the discriminator is recognized using two methods. The size of the augmentation is increased once overfitting is identified [29].

Metrics for evaluation. The IS is one of the most often utilized evaluation metrics. IS provides a scalar value that measures the generated images' fidelity and diversity in relation to the training set. The loss of the dataset's class distribution is one disadvantage of the metric. That means, the generator cannot provide the same number of created images per class as is visible from the dataset. A metric known as FID was established as a result of IS's shortcomings. FID contrasts the statistics of the training data with those of the produced samples in this instance. As a result, the metric is significantly more reliable than IS on a variety of synthetic image modifications [58]. Other metrics for measuring quality are Feature Distance and Perceptual path length.

StyleGAN Encoder for Image-to-Image Translation

Before getting to the actual explanation of the paper, an image's translation will be discussed to gain a better understanding of the image2image translation. *Image-to-Image.* Image-to-image translation is a field of computer vision in which an algorithm is trained to translate an image from one domain to the respective target domain. The goal is to learn a mapping function that can convert an image from one domain to another while retaining important visual features of the original image, such as a black-and-white image to a color image or a low-resolution image to a high-resolution image (super-resolution). Another example is the translation of Sketch Faces to normal portraits. After long training, portraits of people can be rendered as Sketch Faces or the other way around. Figure 11 shows an example where a sketched portrait was transformed into a "real" one.

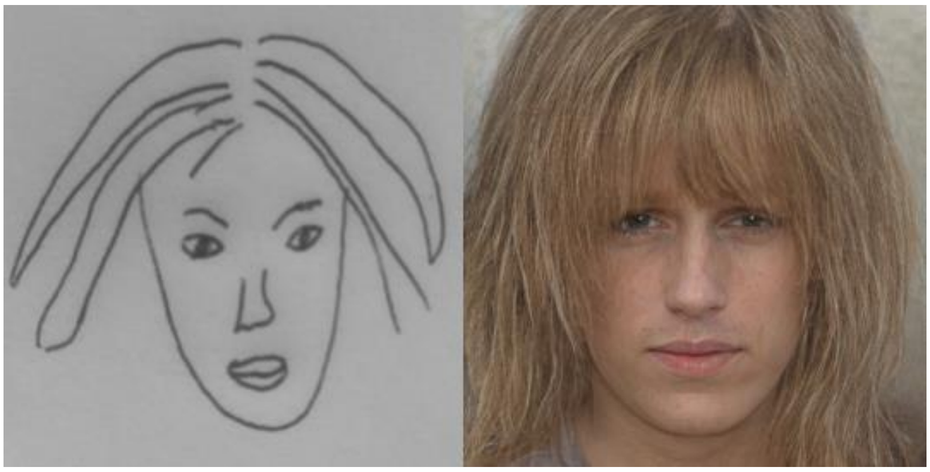

Fig. 11. Self-painted picture becomes celebrity portrait through psp-encoder

The latent space W creates an untangling of dimensions in the StyleGAN architecture, allowing for the independent editing of various properties. The input images must, however, be encoded in the W latent space in order to employ the StyleGAN architecture. There are two well-known algorithms that can be used to carry out this embedding process, which is known as GAN inversion.

Latent Vector Optimization. The latent vector, which is created as a result of training, is a lower dimensional representation of the input data. The output of the generator can be controlled by changing this code. This entails evaluating how well the generated image matches the actual target image after each iteration. The generated image is either iteratively refined once more or ends up in the StyleGAN generator architecture based on the comparison of the two images. This method's disadvantage is the duration of the training, which can last for several minutes. Figure 12 shows the general flow of this method.

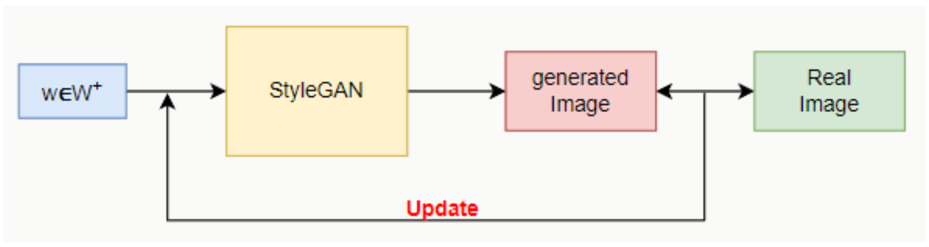

Fig. 12. Architecture of Latent Vector Optimization

Encoder-based Inversion. This is a technique from computer vision that uses a trained neural network to invert the image. It uses an encoder that extracts features from the input, which are then fed into a decoder that has been trained to reconstruct the image based on the extracted features. It is a useful technique because it can learn to reconstruct the original input image even if the input image is degraded or incomplete. This is because the encoder network can extract important features from the input image, which the decoder network can use to fill in missing information and reconstruct the original image. The major drawback to this method is that existing models do not have high accuracies.

A StyleGAN encoder for image-to-image translation is presented by the authors of the study "Encoding in Style: a StyleGAN Encoder for Image-to-Image Translation" to close the gap between the accuracy of the latent vector optimization and the training time of the encoder-based inversion. The encoder makes it easy to immediately embedd the image into the StyleGAN architecture.

The psp encoder maps the various StyleGAN pyramid scales using a feature pyramid network before encoding the image. The created image's results are compared to the original using four carefully chosen loss functions, as opposed to the optimization-based method's time-consuming operations, which updates the image's latent coding.

The newly developed psp encoder is connected in front of the trained StyleGAN generator and sends the resulting vectors as input to the architecture of the generator [41]. The input which you need to train a psp encoder is a pre-trained StyleGAN, as well as matching training data from the target domain. The final product is an encoder that can embed the latent code of the image in W-space directly into the StyleGAN generator architecture.

What happens in the encoder? Using a ResNet backbone, feature maps are retrieved from an image using a standard feature pyramid. Based on StyleGAN's idea that they can be divided into three different styles, there are three feature maps, which are called coarse, medium and fine. A tiny convolutional network is trained to extract the key characteristics from each feature map for each of the 18 styles. The tiny feature map is represented by styles 0–1, the medium by styles 3–6, and the largest by styles 7–18. The psp encoder produces a vector of size 512x1 as an input for all 18 different styles. The StyleGAN's architecture is then directly incorporated with these vectors [41].

Image2Image Framework. This framework was the main emphasis of this effort, but it is not necessary for the remaining parts of my thesis. Only the psp-encoder, which outputs an image's latent code in W-space, is required for this thesis. Image-to-image translations are possible using this framework [41]. Here, the latent code produced by the psp-encoder is processed using an image that has a latent code in charge of altering one or more image dimensions. Here, Figure 11 can be used as an illustration. The psp encoder generates a latent code for the self-painted image, which is then combined with the latent code in charge of the image's structural modification [41].

Momentum Contract Loss Function. This work primarily consisted of portraits of people, hence an identity loss was applied. The MoCo loss was utilized because this loss function is useless for solving our issue. The MoCo loss is a replacement for the ArcFace network, which is in charge of extracting features, for domains other than portraiture. It does this by using a MoCo-based ResNet. The authors of the work "Designing an Encoder for StyleGAN Image Manipulation" introduced [47] and implemented [59] this loss function.

ReStyle: A Residual-Based StyleGAN Encoder via Iterative Refinement

Like the psp encoder, the objective of this effort was to train an encoder that can convert an image into a latent code in W space. Iteratively enhancing the outcome should lead to an improvement of the latent code. Figure 13 depicts the algorithm's flow. In the initial iteration, a second image in addition to the original image is fed into the encoder. The encoder generates a latent code for the original image; however, this latent code is concatenated with the latent code of the second image rather than being directly embedded into the StyleGAN. This concatenation's output will now be generated with the StyleGAN generator. The outcome of the most recent iteration then acts as the starting point for the following one. The original image's input never changes, and the number of iterations can be increased until the outcome appears promising in order to the original image [48].

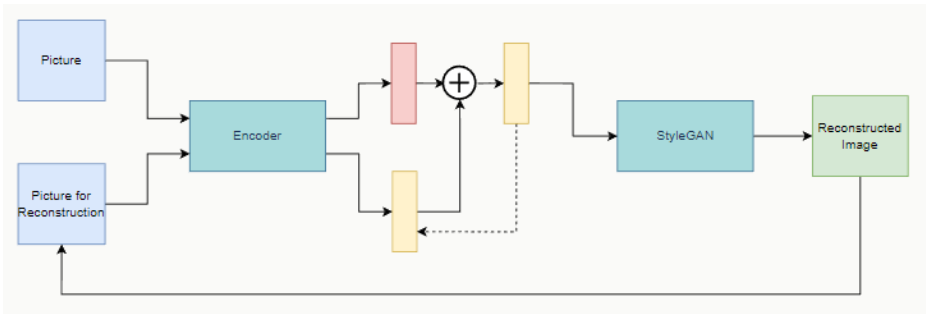

Fig. 13. Architecture of a Restyle-Encoder, in the style of [48]

Attribute Group Editing for Reliable Few-Shot Image Generation

A technique called AGE can create images of classes that have never been seen before. The model does not need to be retrained for each class from which images are to be created, which is a significant benefit for our work. The AGE learns category-relevant and category-irrelevant features of a class via the internal representation to produce new images of that class. The basic assumption behind the model is that an image is a combination of features and the editing direction of attributes is similar across all classes and new images can be created via editing some specific attributes, which are not relevant for the main content. As input for the AGE model, both a well-trained psp encoder and a StyleGAN generator with good quality training data are used. The result is a dictionary that stores the attributes that are be able to edit images. Thus, the AGE method's architecture is split into two components [49].

Image Embedding. The images to be produced must be embedded in the StyleGAN's W latent space because the AGE technique also uses the StyleGAN generator architecture. The psp encoder mentioned above is used by the authors to do this embedding. The second component of the design depends on the encoder's output, which represents the latent coding of an image [49].

Attribute Factorization. Attribute factorization attempts to identify and extract category-relevant and category-irrelevant properties from an image's latent code once it has been established in W space. Category-relevant attributes on the one hand are features that are important for the basic structure and the main content of the image. On the other hand category-irrelevant attributes storing features, which can be edited without loosing the main structure of the image. Here, for instance, a face expression or the eye direction of a creature is a possible example [49].

Category-relevant Features. An image can be classified by collecting features that are relevant to the specific class. Each class has a vector that keeps the average of all category-relevant attributes included in the class in order to correctly assign the image. The vectors for all categories are kept in the dictionary M. The image is assigned to the class that it is most similar to and has the smallest distance across. The average latent code of a class is calculated using the formula 3, where N represents the total number of images in the class and w(i) is the latent code of the current example image [49].

$$W^{Cm} = \frac{1}{N} * \sum_{i=1}^{N} w(i)^{Cm} \quad (3)$$

Category-irrelevant Features. Category-irrelevant features are essential for creating new images, by editing these specific features. By freezing the category-relevant qualities and modifying them with category-irrelevant attributes, a new representation of a class is produced. The authors make the assumption that the category-irrelevant properties are spread uniformly across all classes, which are finally stored in a dictionary. This dictionary also includes a sparse representation that uses an encoder-decoder architecture to guarantee that each element in the dictionary also has a semantic meaning. The relevant qualities of the image must remain the same in order to guarantee that only the irrelevant attributes have been altered [49].

Directions of Attributes. There are various groups that can be created from the directions of the qualities. The middle layer regulates the surface features, such as the color and expressions, while the bottom layer determines the structure and geometry of the image, such as the zoom. The background and the entire color scheme are both described on the top layer. The Singular Value Decomposition technique is applied to dictionary A in order to identify attributes that are in several layers. The following values are found in dictionary A and

$$A_f = U\Sigma V^* \quad (4)$$

saves all the common directions utilized by several layers in the final matrix U [49].

Inference. With the help of the two dictionaries mentioned above, which are created during the training, new images with a variety can now be generated quite easily. This architecture has the significant benefit of eliminating the need for additional model training for each class or target domain.

Few-Shot Adaption of Generative Adversarial Networks

The goal of Robb et al researchs was to address GANs' lack of generalization. This could make it possible for GANs to learn new domains with training datasets, which stores less images, more quickly. In their study, the authors suggest a novel approach that only needs a well-trained and effective StyleGAN generator. The methodology was built on the premise that earlier research [60, 61] had demonstrated that GANs perform better when the space of trainable parameters is constrained. The authors do this by using Singular Value Decomposition on the StyleGAN pre-trained weights. By doing this, they demonstrate that adjustments to the resulting singular values produce positive outcomes. There is a decrease in the trainable parameters and a significant improvement because just the singular values are trained rather than all network weights [46]. A combination of meta-learning and fine-tuning build the foundation of the approach. Through the use of a meta-learning network, meta-learning enables the prediction of the generator's optimal parameter settings for the new target domain. The pre-trained generator network is updated by fine tuning using a limited sample of images from the target domain. The meta-learning network's projected parameters serve as a guide for fine adjustment. The approach takes a StyleGAN2 generator that has already been trained as input and produces one that has been optimized to the target domain.

Functionality. Both the discriminator and the generator are applied to SVD. The two networks' weight matrices are directly affected by the decomposition, which is used individually on each layer of each network. The singular vectors U_0 (left singular vector) and V_0^T (right singular vector) are reflected in the SVD decomposition's outcome, which is shown in the formula 5 [46].

$$W_0^L = (U_0 * \Sigma_0 * V_0^T)^L \tag{5}$$

Following decomposition, a domain adaptation is carried out to shift the weights in the direction of the intended domain. Only the singular values $\Sigma = \lambda * \Sigma_0$ are updated during this procedure, and the two singular vectors are frozen. The training is terminated early in order to prevent overfitting.

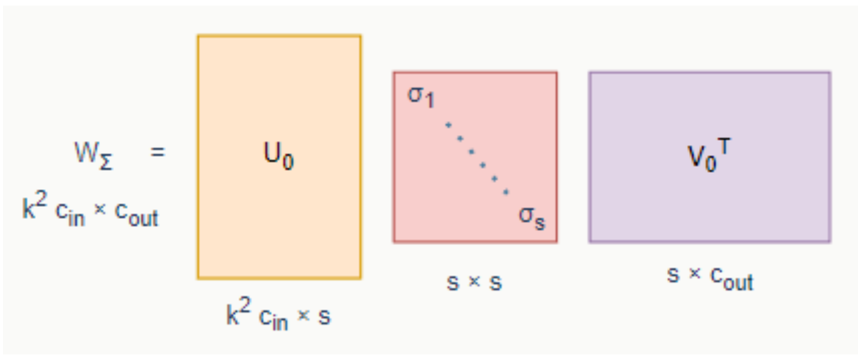

Fig. 14. Singular Value Decomposition, in the style of [46], σ_1 to σ_s are the optimized singular values

Working with various datasets allows for an analysis of the method. There are two distinct methods. The first is the near-domain technique, which transforms images from one domain into a different but related domain. Here, the authors decided to change the datasets from FFHQ to CelebA. On the other hand, the far-domain strategy involves transforming images into a domain that is located far away. Here, the authors decided to change the FFHQ images into self-painted portraits [46].

Issue. The end result of the training is a generator that can produce high-quality images and has been trained on the target domain. Many different images can be produced using this generator. The biggest disadvantage is that we have to train a model for each class to get labeled images, because the output of a model don’t have any class memberships. Because of this the strategy proves to be very expensive and sophisticated, with over 11,000 different classes. Therefore, this strategy can only be contrasted with the first path's generated images using various metrics.

VGG16 Classifier

One of the most well-liked and frequently employed CNN architectures for image classification is the VGG16 model.

There are 13 convolutional layers and 3 fully connected layers in the network's total of 16 layers. While fully connected layers are used for classification, convolutional layers are utilized to learn the key elements of an image. To learn extremely fine aspects of the image, the architecture combines small convolutional filters (3x3) and small step sizes (1 pixel) [77].

The VGG16 architecture was trained on the large ImageNet dataset, which includes millions of labeled data and 1,000 different classes, to serve as a fundamental framework for other domains. The network can be employed as a classifier for new images once it has been trained. The network processes the input image, and the output of the last fully connected layer is then used to forecast the category of the image [77].

Transfer Learning. A lengthy training period is the result of training a detailed and sophisticated model, especially when there are little or insufficient resources are available. In particular, the filters from the convolutional layers are obtained from the computed weights of the model trained on ImageNet as a result. Only the three newly added layers of the model need to be trained in order to transfer to a new domain. The knowledge gained is used to solve the previous problem. The trained model is very easy to retrieve via the Python library keras and to perform the specific changes.

Work architecture

Figure 15 displays the main structure of the practical of my work. Two paths make up the work, which will be compared at the conclusion to determine who produced the best images. The StyleGAN2-ada repository serves as the fundamental structure for both routes. The dataset must be provided as input in a folder containing all of the images. This repository has produced a generator that can generate images of coins that are of a high quality. The first route merely passes through the work of Robb et al., which calls for the input of a trained and effective StyleGAN2 generator. The singular vectors are frozen during training as the model's singular values are modified to the new target domain. The benefit of this approach is that by training the singular values, a lot of irrelevant data is left out and only the most crucial data is used for training, drastically shrinking the area for trainable parameters. Based on the produced model, new images of the target domain can be created. The StyleGAN2-ada repository is where the second path also begins. An encoder is trained to provide the images to the StyleGAN generator via the W latent space using the pixel-to-pixel approach. The result is a 512x1-dimensional vector representing the latent code of the image calculated using a convolutional mesh. The dataset is required for this task in the form of training and test datasets, the modified Dataset 2. The images are kept in the corresponding subfolders without being assigned to a class. This latent code of an image from the trained encoder is used in the paper with the topic Attribute Group Editing. The category relevant as well as irrelevant attributes are extracted based on the average value of the latent code of a class. It is now possible to create new images by dividing the two categories. The irrelevant attributes are processed while the category-relevant attributes are frozen in this procedure. It is feasible to repeat this instruction for the classes where images are still lacking. For this, the algorithm requires data in a different format. As in p2p, the images are kept in the subfolders test and train, but here the images must be renamed in order to capture the class membership of the images. The following structure must be present in the images:

classId_exampleID.jpg

To compare the generated images with each other, meaningful loss functions are applied to the generated images. Since only the path via the psp encoder can generate labeled images, the best results from this path are input to the VGG16 model.

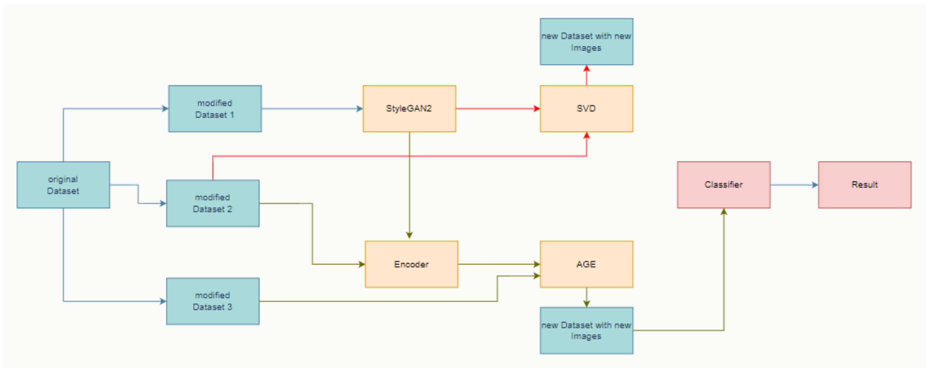

Fig. 15. Base Structure of the thesis, blue ➔ Datasets, orange ➔ Repositories, rot ➔ classifier, arrows in red and green ➔ two paths

5 Implementation

This chapter will demonstrate how to use the design in Figure 15 and list the resources and setting used for this. Readers will learn how to develop the above-described task using some class diagrams. My GitHub repository contains all the models and datasets for the best results.

Development environment

The Python programming language and IntelliJ development environment from JetBrains were used to create the implementations. With a computational capability of 8.6, an NVIDIA GeForce RTX 3060 graphics card was used to create and train the multiple neural networks. The operating system used was Ubuntu 22.04, which is also highly suggested for rebuilding. The implementation need the use of a number of libraries.

Tensorflow 1.15. Deep Learning is the primary emphasis of TensorFlow, an open-source machine learning library. It has a lot of instructions that let users create ML-powered applications very fast and simple. The library's open-source nature enables users to keep pushing the boundaries of machine learning research [62]. TensorFlow

1.15 is utilized for the task of the SVD model. TensorFlow must be installed using nvidia-wheel [63] in order for this version to work with NVIDIA's new graphics cards (RTX 30 GPUs), as this version cannot be installed using pip or anaconda.

Pytorch. Tensor computations may be carried out much more quickly with Pytorch's help and robust GPU acceleration. Additionally, Pytorch enables the construction of deep neural networks. A simple method to get started with Pytorch is the NGC container. Since this container includes all dependencies and it provides a simple starting point for creating typical applications.

Numpy. When working with arrays, Travis Oliphant's Python library is frequently used. Numpy and its source code are openly accessible and working with matrices in linear algebra.

Scikit-learn. Numerous classification, regression, and clustering techniques are offered in this free Python machine learning software package. Examples of algorithms are K-Nearest Neighbor and Random Forest. The library is built on the libraries NumPy, SciPy, and matplotlib and includes effective tools for data analysis.

OpenCV. An open source library for computer vision and machine learning is the Open Source Computer Vision Library. This library's objective is to offer a standard framework for computer vision applications. More than 2500 efficient algorithms in OpenCV are used, among other things, to identify objects or recognize faces.

To execute the repositories, there is a need of a few environments. Environments are recommended, which are also available in the original repositories, but with my computer under Ubuntu they partly do not work though. To get the repositories working, new conda development environments had to be built, which can be found in my GitHub repository as .yml files. ‘Conda env create -f environment.yml’ must be run in order to establish a Conda environment with the libraries specified in the .yml file.

Repository changes

In general, it can be said that not many things in the individual repositories needed to be changed. The most important thing is that the inputs have the right formats and the development environment is built to match the repository, but also my own resources. There were always additional issues on Windows and Ubuntu that may be fixed by switching the libraries used in the development environment. Each repository has a file that correctly formats the specific dataset.

The encoder ResNetGradualStyleEncoder must be included to the file psp encoders.py in order to have an encoder that was not trained on face images. Additionally, an attempt was made to optimize training for parallel processing across many GPUs [64].

The channel multiplier has to be set to one since the reshaped data [65] had a different dimension than what the p2p encoder was expecting.

The StyleGAN in Tensorflow is trained on grayscale images due to the significant time savings. This means that the run generator.py file, which is used to create new images, needs to have one line updated. The following two lines must be added in place of line 36 in the repository of StyleGAN2.

reshape_image = np.reshape(images[0], (256, 256))

```
PIL.Image.fromarray(reshape_image,
'L').save(dnnlib.make_run_dir_path('seed%04d.png' % seed))
```

Transform Datasets

The dataset needs to be transformed into various formats in order to be correctly used with the repositories. For the job, a total of three updated datasets are required.

Modified Dataset 1. The first adjusted dataset is required for the StyleGAN2-ada, which serves as the foundation of our research. For this, the data is kept in a folder without regard to any class. Two passing parameters are required for the script, which are stored in the GitHub Repository, to run so that the computer knows where the data's source and where the intended storage location is. The path of the dataset indicates the folder in which the subfolders of the individual classes are stored. The first For loop iterates over the various classes, and the second nested loop copies the images in the corresponding subfolders in the target path.

Modified Dataset 2. To obtain the classical distribution of a dataset in machine learning, 80% of all images are used as training data and 20% as test data. In this dataset, the names of the individual images can be taken from the original dataset.

Modified Dataset 3. Also for the AGE repository we need the images in a folder structure of 80% training and 20% test data. However, the difference is that for this work we need a class membership of the images. To do this, we rename the images according to the following structure:

CategoryID_SampleID.format

The CategoryID represents the class of the image and the SampleID represents the number of the image within the class. For example, the fifth image of class 1000 would have the following name ⇒ 1000_4.jpg. The format of the image can be saved as .jpg as well as .png.

Two further transformed datasets are needed in the connection of the two paths for the generation of new images as well as the later VGG16 classifier. So that the later Inference model knows from which class new images must be generated, a dataset must develop, which contains all classes, from which new images are to be generated. Within the classes the images to be generated are stored.

Finally, a further dataset is needed, which concatenates the dataset just described with the results of the Inference models. For this purpose, both the front and the back side of an image are copied from the generated images and stored in the resulting dataset. This process is repeated until each class intended for the VGG16 model has the same number of images. Very important in this procedure is that for each front side there is a matching back side since they are concatenated horizontally for the VGG16 model.

Class structure of individual code passages

This subsection will go into more detail about individual important parts of the code and describe their implementations. Since not much has been changed in the repositories from my side, this chapter is relatively short and is intended to clarify a few basic ideas of the implementations of the repositories.

Augmentation. The processing of the images and possibly overfitting are handled by the AugmentPipe class. Figure 16 displays a few potential kinds of image processing augmentation. If an augmentation is to be activated, a value greater than zero must be set. The value for an augmentation specifies how strongly it is to be taken into account for the training of the model.

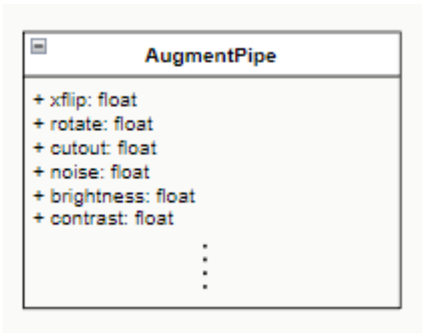

Fig. 16. Class Augmentation, based on the code from [74]

p2p-Encoder. The basic structure of a psp encoder can be seen in Figure 17. The encoder parameter is responsible for extracting features in the form of the StyleGAN pyramid scheme. There are a few options for this, such as the GradualStyleEncoder, which was trained by an ArcFace network for portrait recognition. The used ResNet-GradualStyleEncoder was also added here. The decoder is a generator that has the size of the resolution of the images. After the encoder and decoder have been set, the load_weights function is called. In this function it is checked whether a model should be trained further or a new one has to be built from scratch. If a model should be trained further, the already trained weights are set as initial value for the encoder and decoder.

AGE. With a few exceptions, the structure of the AGE method is reminiscent of the psp encoder. The function "get_code" is needed for the distribution of the images. By means of this function the Gaussian distribution of all sample images is calculated and used for the later generation of new images. With the help of the method "get_test_code" the latent code of an image can be returned. Both with the computation of the average latent code for each class, and for the generation of new images this function is needed.

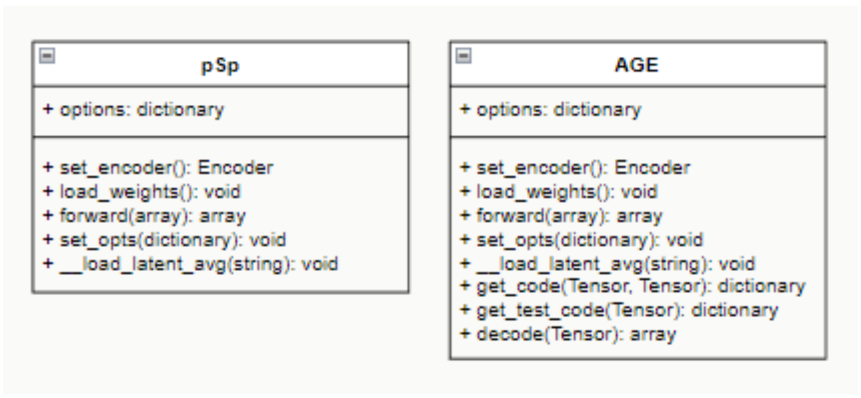

Fig. 17. Structure of models pSp and AGE, based on the code from [64, 75]

What made me select these repositories?

Numerous diverse repositories were examined and compared in the search for appropriate ones that satisfy the needs of this masterthesis. There are numerous works that appear intriguing and promising. I was forced to limit my options to a few repositories because of the limited computational and time resources available. The StyleGAN2-ada architecture is a key part of this work. This repository was selected since it has an architecture that prevents overfitting when training on the little dataset being used here. Additionally, Pytorch promises superior results and a training time that is around 30% faster than the Tensorflow implementation. A Fréchet inception distance record of 2.42 was achieved using the CIFAR-10 dataset, breaking the previous score of 5.59. Because the psp encoder is simple to use and appropriate for the dataset being used here, I choose to build this on the StyleGAN2-ada. Furthermore, the GitHub repository is very well maintained, and some concerns about error messages have already been addressed. A psp encoder was also necessary for the final repository, named AGE. Over other works, the AGE repository and the psp encoder have a significant advantage.

Because the model is already trained for the target domain, it is not necessary to train the model from scratch for each class, in this case the classes that are relevant for the future VGG16 model. In other studies, the model needs to be trained again for every class, which would have led to a runtime that is far too long if there are more than 11,000 classes. Once the AGE model is complete, a for loop can be used to quickly apply the model to all classes. The SVD Repository was still used to get comparative results for the individual image quality.

This work has the minor problem of not using the trained StyleGAN2 generator like the first path because the SVD-Repository is written in TensorFlow. Because of this another StyleGAN generator needs to be train in TensorFlow. The creation of the unclassified images that were just discussed, which would need to be manually labeled in order to be taken into consideration for categorization, is another weakness in this approach.

Difficulties in the implementation.

During the work there were some problems regarding the implemented repositories. The start of this master thesis was on a MacBook-Pro with a M1 chip. It quickly became clear that the resources available there were not enough for the required computing power. The RAM of the MacBook was filled up very fast and resulted in permanent crashes of the program. For the same reasons, an implementation via Google Colab could not be done either. In addition, support for TensorFlow 1.x, which is needed for the SVD model, has not been there on Colab since last year. For this reason, I had to use my computer with an RTX3060 graphics card from NVIDIA. However, Linux had to be installed on my PC since the repositories do not provide support for Windows. After Ubuntu was installed, most repositories ran after changes in the respective Conda environment. Only the path via the SVD model required an additional installation, as the range of RTX30 graphics cards only work with the latest drivers from CUDA and these do not provide native support for TensorFlow 1.x. After some research, I came across a tutorial that solves exactly this problem [66]. Via this, it was finally manageable to make the right development environment for the path.

There were also always problems with the execution of the repositories, which even led to the fact that when training a net, unfortunately the results of the previous net were overwritten. In addition, there were always problems with the operating system Ubuntu, such as updates or even new drivers for my GPU.

Furthermore, after three months of work I found out that the StyleGAN2-ada generator was trained with different configurations than recommended in later works. Therefore, after three months I started again to train the generator, which is the basis of my work, with other configurations.

6 Experiments

The outcomes of the implementation are presented in this chapter. Each repository is looked at in further detail, and useful evaluations are provided with particular metrics and visual examples.

Due to time constraints and limited hardware, the goal for even greater variety of tests and outcomes had to be rejected. This made it impossible to properly test the selection of all hyperparameters for the specific repositories. To prevent a long training period, the size of the images to be trained was decreased to 256x256 for the same reason.

With a few exceptions, the initial hyperparameter selection was the same as in the original repository.

6.1 StyleGAN2-ada

The tests begin with the StyleGAN2 model, which enables the development of a generator that can produce images of an ancient coin in high quality. GANs are difficult to analyze since they are typically challenging to train and unstable training can cause a variety of issues. Due to instability, it can definitely be time-consuming and error prone. The FID statistic is used to evaluate the effectiveness of the resulting StyleGAN2-ada generator. FID is a metric used in this case to compare the feature vectors from real images and the fake images produced by the generator. It should be noted that the objective is to decrease the metric's score because a lower score indicates greater image quality and, thus, a closer similarity to the original images. The resulting graph can be seen in Figure 18. The graph rapidly declines after the initial iterations and epochs, but after around 5000 kimg, it begins to asymptotically reach the value of 7.5.

5000 kimgs took on average 82 hours on my NVIDIA 3060 GPU. The generator is already so good by this point that it can keep serving as the foundation for new repositories. The outcomes of the two StyleGAN2-ada implementation strategies on Pytorch and Tensorflow are displayed in Table 1. The tensorflow-based GAN had to be abandoned after nearly 5000 kimg because the resources were not good enough and had to be used for other, more crucial tasks. The tensorflow-based generator was trained with grayscale images because training the entire dataset caused permanent RAM overruns. The dataset's size is decreased from 19.5GB to 6.1GB after the transformation. The Pytorch model did not considerably improve from this value, and it can therefore be inferred that this model would not increase further as well. This information can be used to justify the model's early termination. The Pytorch version is clearly noticeably faster than the Tensorflow version by approximately a factor of three. The reason for this is because Pytorch is still being updated while Tensorflow has been replaced by Pytorch from the authors of the repository and do not get any further updates.

	Resolution	CPU	5.000 kimg	20.000 kimg	sec/kimg
Pytorch auto	256x256	1	80-82 h	13-14 d	58-60
Pytorch stylegan	256x256	1	200-232 h	30-33 d	150-165
TensorFlow	256x256	1	180-190 h	27–29 d	137-145

Table 1. Runtime for all StyleGAN2-ada models

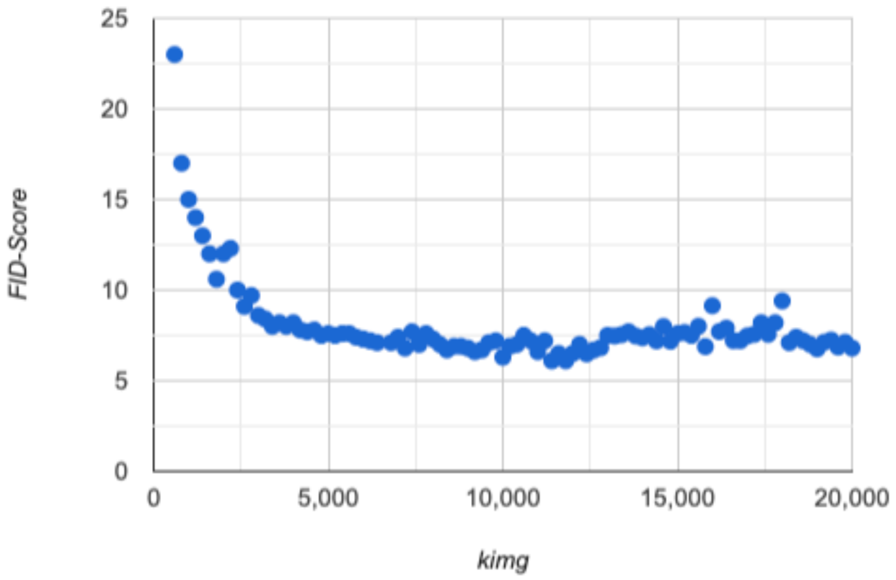

Fig. 18. Time course of the Pytorch StyleGAN2 auto model

It can contrast the findings with those from the original paper to get an understanding of how good the trained generator is. The most significant values of a few evaluation measures are displayed in the table 2. The model, which trained here in this thesis, has an almost 20% higher FID score than the model from the original publication. This is due to the fact that the FFHQ dataset used there does not contain as many significant variations as the dataset utilized here. Additionally, compared to the Coin-dataset, the overall quality of the images from the FFHQ dataset is much higher.

Using the KID metric, the qualities were further evaluated. With the help of KID, better predictions can be made about smaller datasets because the metric is unbiased from the outset and can therefore analyze smaller datasets faster [57]. Table 2 compares the models that were trained here with those that were provided in the study.

An image of each model was created in the figure 19 so that more than just statistics could be used to evaluate the quality of the outcome. Here, the difference in dataset quality that has been previously mentioned numerous times is clear. The generators trained in this paper return images with lower quality, however the FFHQ model can produce images that are extremely sharp. Despite this, all the essential details that should be present in an image of a coin are represented in the generated images. There is a portrait that includes all the features of a human face, including a mouth, nose, and eyes, in both images. The StyleGAN model already has more issues with complex objects, as can be seen in the figure 19. While the Pytorch-written generator makes the object clearly visible, the quality of the tensorflow-based generator must be reduced because some features on the coin are missing. However, the illustration shows clearly the coin's most crucial features.

Therefore, the fundamental structure of this work with the trained generators seems to remain promise despite the slightly poorer values of the evaluation measures.

	FID-Score	KID-Score
SG2-ada FFHQ	6.3	0.002342
SG2-ada Coins	7.59	0.00315
SG2 TensorFlow	8.78	0.00512
SG2-ada Class 20203	132.9	0.12

Table 2. Scores for different StyleGAN2 models

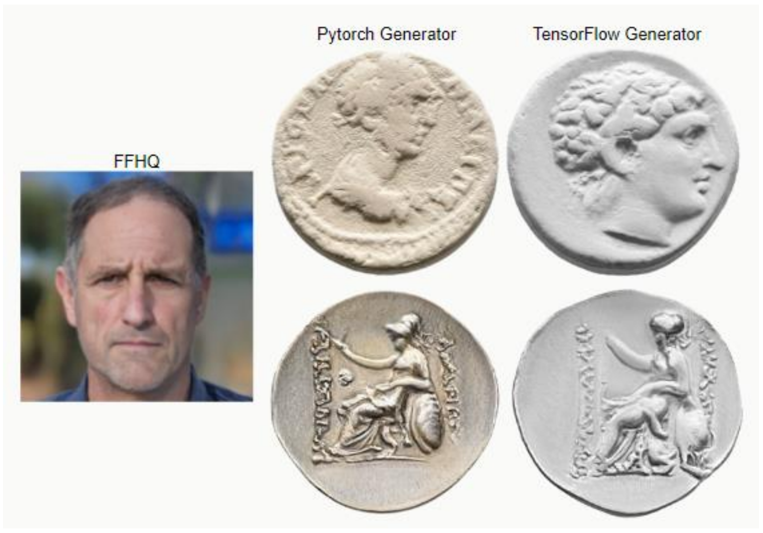

Fig. 19. Visual results for StyleGAN2 models

Types of Augmentations

In order to avoid potential overfitting, the image is processed using a variety of techniques so that the fundamental features remain the same but other attributes, which are not important, are modified. Individual techniques are described and briefly displayed in the parts that follow.

Pixel blitting. Augmentation pixel blitting consists of three primary techniques. All of the approaches rotate the images, and doing so significantly improves the efficiency of a CNN. This is due to the fact that machine learning models learn what pixel combinations and relationships exist between each pixel of an object in an image and how to classify them. The model is given extra information to learn from without needing more images of the object by having the ability to rotate an image in different angles to build several copies of it. A common technique is to rotate an image by 90 degrees.

General geometric transformations. The isotropic and anisotropic scaling are two techniques for geometric transformations. Anisotropic scaling involves at least one dimension being multiplied by a different factor than isotropic scaling, which multiplies all dimensions by the same scaling factor. The shape of the image changes as a consequence of this modification. Arbitrary rotation is another geometric transformation. This technique rotates the image by a random number of degrees.

Color transformations. For instance, the image's contrast, brightness, or color saturation can be changed during the color transformation. It might be feasible to change these values so that the model can see the coin's fonts or even smaller characteristics better.

Image-space filtering. This technique is used to highlight or change specific aspects of the image. The adjustments should result in an improvement in image quality. Similar to color changes, this technique can enhance the visibility of minor details.

Unfortunately, it was not was not possible to test all of the augmentations' hyperparameters and determine which parameter setting produces the best results due to limited resources and the upcoming models and experiments. In this experiment, all augmentations are activated and the model was trained up to epoch 1,000 to observe the effects of augmentation adjustments. Table 3 shows that even though only a few augmentations are triggered during normal training, the outcome barely changes, when all augmentations are on. Based on this outcome, it is presumed that the model and the images that are produced are not significantly affected by changes in the parameters of the augmentations. When a model is initialised, the augmentations are set with a value between 0 and 1. A number greater than zero indicates how likely it is that the specific augmentation will be employed, with zero meaning that it won't be used.

	FID	KID
Normal Training	14.76	0.00512
Training with all Augmentations	14.63	0.0054

Table 3. Training results for different activated Augmentations

Create generator for only one class.

This experiment demonstrates how quickly a generator that belongs to one class can adapt. The dataset is therefore restricted to one class, whose images are once more split into test and training data. A class with enough images is used to stop the model from overfitting so quickly. The class 20203, which has 30 images, was used for the experiment.

Neither the visual results in figure 20 nor the values of the loss functions in table 2 are on the same level as the prior experiment after 400 kimg, which is equal to nearly 7 hours. The experiment was terminated because even after 7 hours the results from the generator were not good enough in comparison to the first experiments and it would have taken 3175 days to train all generators for each class, which is by far to too long fir this thesis.

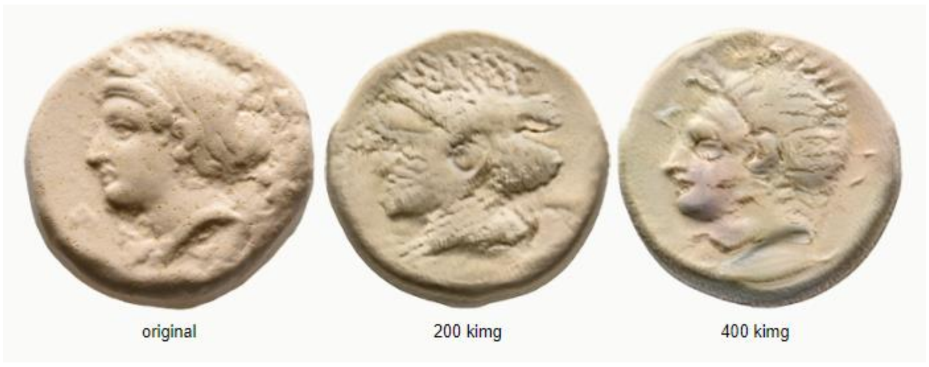

Fig. 20. Visual results from generator trained on one class after 7 hours

6.2 StyleGAN Encoder for Image-to-Image Translation

There are several metrics for the evaluation of the psp encoder, which are briefly explained next.

L2 Loss Function. The L2 loss function stands for Least Square Errors and is also shortened to LS. The loss function is the squared difference between a prediction and the actual value, in this instance the created image with the original images. The created images' color outputs are compared to the original images' color outputs pixel-by-pixel, and an effort is made to keep the difference between the two as small as possible [67]. The training GAN determines which color to output based on the feedback in order to produce coins that are as similar to the original as possible. This loss function has the advantage that the error is penalized more severely, leading to a bigger correction, which can prevent early model stagnation.

LPIPS. LPIPS is used to evaluate the generated images' quality. For the assessment, LPIPS employs pre-trained networks, and the results are extremely accurate [68]. In this instance, the loss function is based on the pre-trained AlexNet.

MoCo Loss. MoCo is a statistic used to update the network's parameters while it is being trained. A regular encoder and a momentum encoder are both used by MoCo. The update of the parameters is what distinguishes these various encoder types from one another. The momentum encoder uses linear interpolation, whereas the standard encoder uses backpropagation.

ID-lamda Loss. This loss function is used to preserve the identity of the face from the portrait. For this purpose, the cosine similarity of the generated images with their original is calculated. The calculation is based on an ArcFace network [41].

Finding the perfect hyperparameters.

It is not possible to change every parameter and analyse the results due to a lack of resources. I have so restricted myself to a few variables. The frequency of testing was the first parameter, which was changed. Here we want to compare whether a more frequent number of validation steps improves the model. For comparison, 50 and 500 steps will be taken. After 1000 steps, the models were compared to obtain a meaningful result. In the table 4, it is evident that performing more tests does not improve results and actually lengthens the process because each evaluation takes roughly 7 minutes. For this reason, the 500 steps were utilized as the evaluation stage in subsequent tests. The second parameter, the loss functions that are being employed, examines which application of the loss function appears to be most promising. Here, the id-lambda loss defined in the repository is used initially, followed by the MoCo loss in place of the id-lambda loss, and then neither of them are used. The loss functions LPIPS and L2 are used as comparison values to compare the experiments.

The last three models' visual results are displayed in the figure 21 and shown the differences between them impressively. Because the visual impression demonstrates that the MoCo loss function produces images that are by far the highest quality. When the MoCo loss is turned on, the coin's legend and the shape of the person may both be seen more clearly. The visual findings can be confirmed by Table 4, which makes it clear that the model with the MoCo loss also performs the best in the loss functions. So for the following tests, the MoCo Loss and the evaluation step at 500 steps are utilized.

	Overall loss	LPIPS loss	L2 loss	time
Tests every 500 steps	0.2415	0.22 ± 0.02	0.054 ± 0.001	6h
Tests every 5000 steps	0.2476	0.21 ± 0.02	0.052 ± 0.001	10h
ID-lambda loss	0.233	0.25 ± 0.02	0.058 ± 0.001	6h
MoCo loss	0.2195	0.14 ± 0.01	0.023 ± 0.00	6h
None of the losses	0.2531	0.18 ± 0.01	0.035 ± 0.01	6h

Table 4. Results from models with other parameters

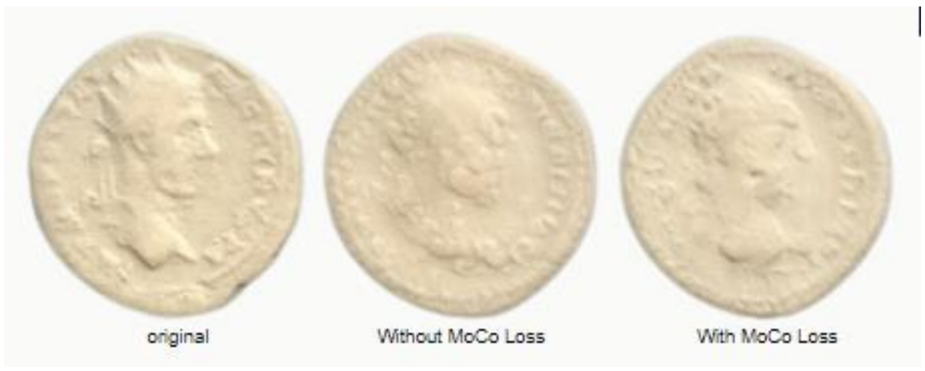

Fig. 21. Generated images for both models

Quality Generator

The testing can now be extended out longer once the ideal parameters have been identified. The loss functions are applied across 100,000 epochs in order to evaluate and compare the qualities of each model. The wandb tool was used to create the graphs, which are seen in figures 22 and 23. A table matching the values of the loss functions used here for each experiment of the psp encoder and the AGE model has been established in Appendix A to allow comparison of all the outcomes of the ongoing experiments.

The generation and processing of portrait-related images served as the basis for Alaluf et al. work's. Due to this, the encoder utilized was developed using the Flickr-Faces-HQ (FFHQ) dataset. I investigate if the data obtained from the encoder is also beneficial for the dataset being utilized here because many coins contain portraits of emperors or other human-like faces. The usability of the portrait information is therefore tested in this experiment. To demonstrate this, an encoder trained on the ImageNet dataset is compared with a GradualStyleEncoder trained on Portraits. The file containing the list of suitable encoders now includes the newly needed ResNet-GradualStyleEncoder.

Comparisons are made with the loss functions mentioned above as well as optical results in the form of images.

The loss functions for the two encoder types asymptotically approach the same values. However, the ResNetGradualStyleEncoder reaches this value earlier, allowing for a shorter training period and correspondingly significant resource reductions. I am using iteration 500 as a validation point, where the ResNetGradualStyleEncoder is already at a value of 0.2 and the GradualStyleEncoder is stuck at about 0.23. Furthermore, the swings of the graph are much smaller with the ResNet-based encoder, so less variation within the model can also be expected. The ResNetGradualStyleEncoder is employed for the subsequent experiments considering these two observations.

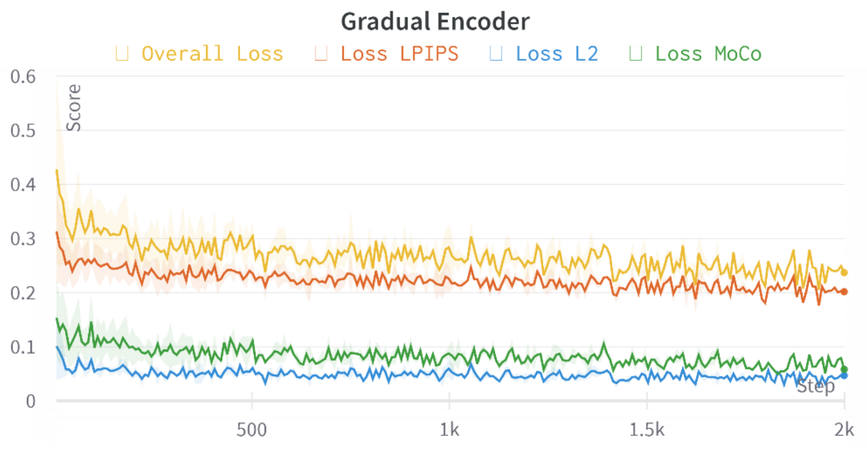

Fig. 22. Time course of the gradual encoder. While the shadows reflect the respective maximum and minimum values, the drawn lines show the average values.

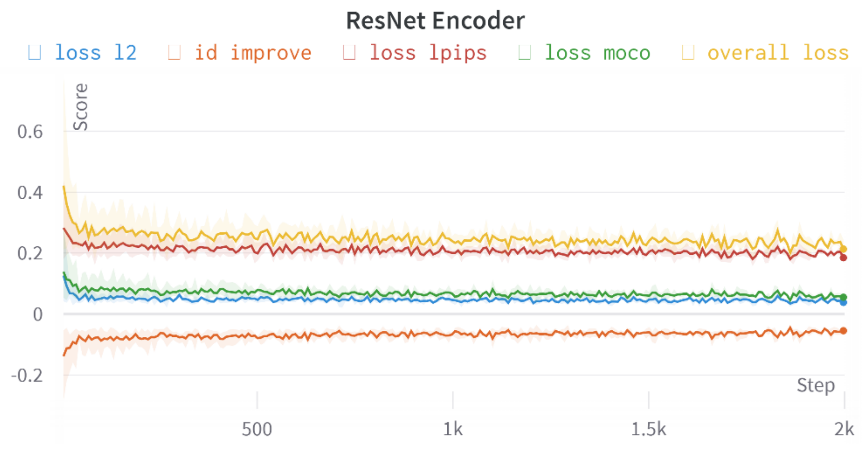

Fig. 23. Time course of the ResNet encoder. While the shadows reflect the respective maximum and minimum values, the drawn lines show the average values.

The visual outcomes and the loss function scores are both identical for the two trained models. The actual generated images of the two models in the figure 24 show a disappointing result, despite the values of the loss functions in the table 5 still indicating good generated images. The contours of the original image are identifiable, but only extremely tiny details allow for the identification of features like the mouth, eyes, and nose. You can see that the ResNetGradualStyleEncoder adds a little more detail to the image in addition to the advantage mentioned before. More structures may be seen in the person's hair than in any other created image.

	LPIPS loss	L2 loss	MoCo loss	Time per 1000 steps
GradualStyleEncoder	0.26 ± 0.06	0.05 ± 0.004	0.09 ± 0.008	6h
ResNetGradualStyleEncoder	0.235 ± 0.05	0.04 ± 0.003	0.08 ± 0.006	6h

Table 5. Results from both encoder types

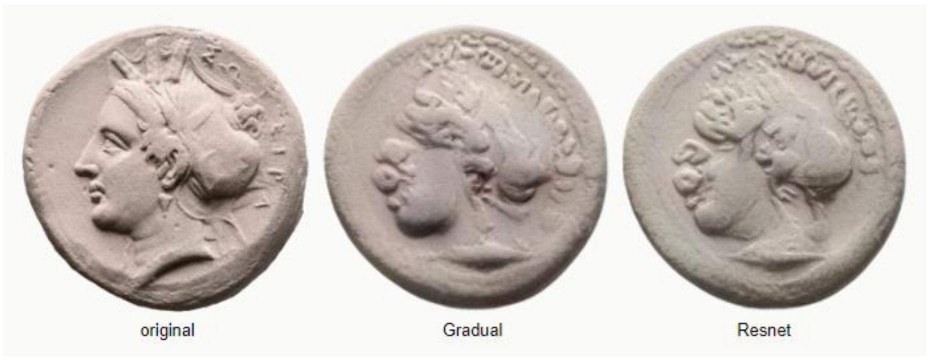

Fig. 24. Visual outcomes of both encoder types

Capacity utilization

The study of the results must take into consideration how the resources are being used. Figure 25 displays the GPU RAM allocated during training together with the CPU and GPU utilization. Since just the GPU is operating for the entire training, this indicates that the development environment is correctly set up on the one hand, but also that a greater performance is not possible with my used resources. The application only uses somewhat less than 30% of the GPU's capacity when building the checkpoints, which are generated every two hours. The usual operation of the used PC explains for the 10% CPU utilization.

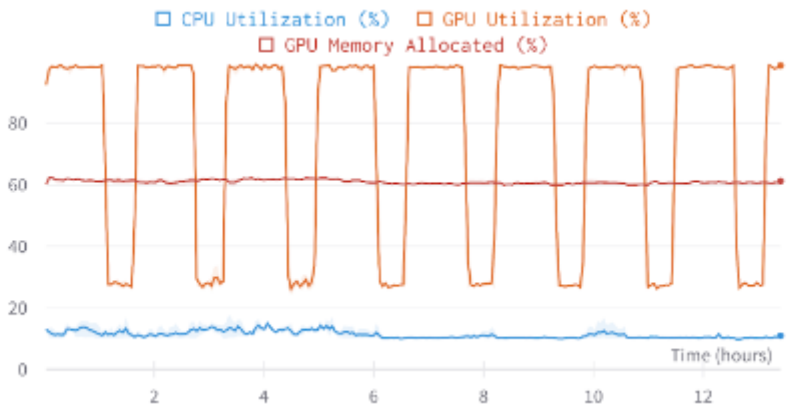

Fig. 25. Capacity utilization during training the psp encoder

Transfer Learning

In this chapter, I try to use knowledge the model already learned from another domain to generate faster and, more importantly, better outcomes for the psp encoder from my target domain. I do this by using both the StyleGAN-FFHQ that the authors have released and the psp encoder that was trained on the same dataset. The two models indicated above were trained on this resolution, so in order to conduct the experiment, I had to increase the resolution to 1024x1024. Sadly, this method quickly proved to be ineffective because a frontal image of a human was still visible in the images. Unfortunately, after approximately 20,000 iterations, nothing had changed, and the experiment had to be abandoned.

Train psp Encoder with less classes

Four separate experiments are included in this area, each of which aims to determine whether the dataset can be transformed to improve the performance of the psp encoder. In the first experiment, the psp encoder is trained using just one class and 30 images as the dataset. In the second trial the dataset is increase with five classes with a total of 112 images and in the final experiments I choose every class that contains more than the threshold of 40 images. In the last the balanced dataset is chosen, so the dataset only take exactly 40 images per class from the previous experiment. These tests are intended to show that there are too many classes and resulting variations of images for the network to be trained.

Additionally, it must be demonstrated that the chosen parameters were actually set correctly and are not to blame for the poor quality of the images that are produced. All experiments up to iteration 20,000 are trained in this case, and the model produced by each experiment's preceding iteration is used as the starting point. Table 6 contains all of the experiment scores.

The results that were anticipated from the other trials were produced in the first experiment with just 30 images from one class. Figure 26 illustrates how, despite the images' high extend of similarity to the original, there are slight variations in the stored individual characteristics. The technique does function on a dataset of coins since the encoder performs the best across all values of the loss function. Similar curves for the loss functions are obtained by training the model just on images with a resolution of 1024x1024, but the generated images perform slightly better.

The output images in the model that was trained on the dataset with five classes are pretty similar, despite the fact that the values for the loss functions are not nearly as high. The model already has worse issues once the number of images in the following model reaches roughly the number of 10,000. The quality of the generated images degrades as the loss function values rise. However, the image's output is still attractive and still conveys the majority of its essential characteristics. By taking only 40 images per class from the previous dataset modify this result to be much better. This is done to avoid a batch of an iteration being entirely made up of images from a big class, because it would be useless to iterate where a batch only contains images from a single class.

There are still 132 classes once the dataset has been changed, but there are now only 5280 images over the experiment. Although the image quality is comparable to the previous experiment, the loss function values are better, and the training process was roughly 15% quicker.

	LPIPS loss	L2 loss	MoCo loss	Time per 1000 steps
One class	0.075 ± 0.02	0.022 ± 0.002	0.014 ± 0.002	4h
Five classes	0.098 ± 0.03	0.025 ± 0.002	0.0165 ± 0.002	4.3h
Classes more 40	0.17 ± 0.04	0.037 ± 0.003	0.048 ± 0.004	5.4h
Balanced dataset	0.14 ± 0.03	0.034 ± 0.003	0.041 ± 0.004	4.7h

Table 6. Results of different datasets

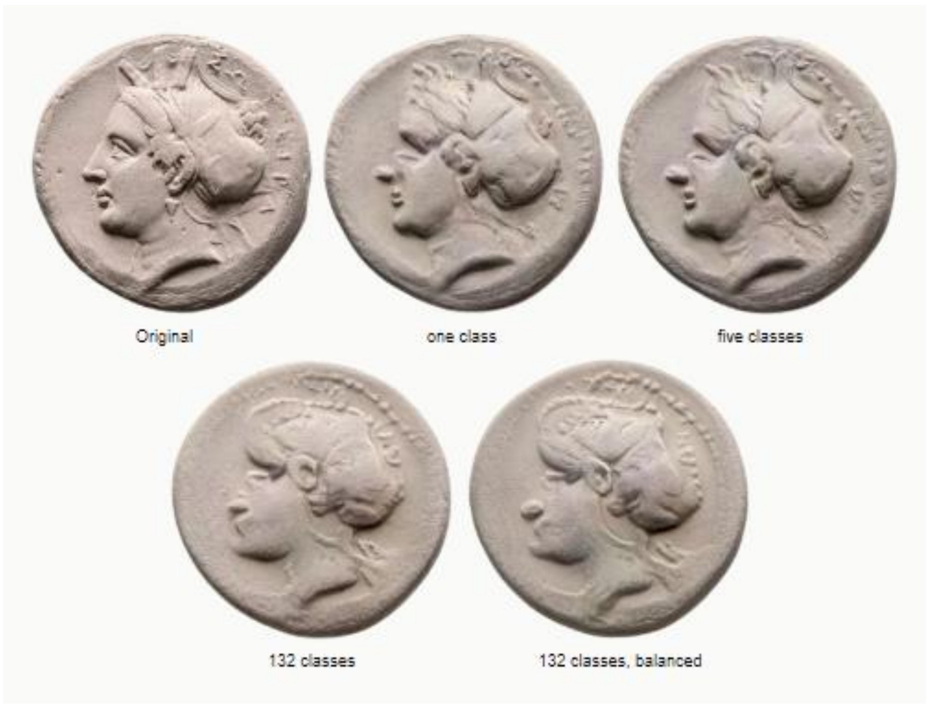

Fig. 26. Visual results of different datasets

6.3 Restyle-Encoder

After the psp encoder produced inconsistent results, GitHub was once more searched for alternatives. After some research, a repository that operates on a similar basis as the psp encoder was discovered. At the conclusion of training for an image, both models output their latent code in W-space.

The intermediate results are improved iteratively with the restyle encoder as opposed to the psp encoder. This is accomplished by using the inversion's outcome as the input for the next iteration. Up until the latent code appears to be similar to the original image, this process is repeated.

However, a modified setup of the StyleGAN2 generator is necessary to execute this repository. For this repository, eight mapping layers are required rather than the normal configuration of two. The model was trained up to a FID score of nine in order to make it competitive.

Due to time restrictions, the restyle encoder was only able to train up to 50,000 iterations before comparing the results to those from the psp encoder after spending the same amount of time. This process took me more than 13 hours for 10,000 iterations. In order to find the correct setting, the output of the psp encoder was used as a guide, which is why the same parameters and the same data set were used. The last state from the psp encoder can be utilized instead of starting from scratch because this repository is related to the psp encoder.

Unfortunately, the results were no better than the actual psp encoder after around 50,000 more repetitions. The scores constantly decreased by 10% for each evaluation metric. In addition to the adverse outcomes, the training time was extended by nearly 700%. In order to these two results, the further work is done for the AGE model with the psp encoder trained on the balanced dataset.

6.4 Attribute Group Editing for Reliable Few-Shot Image Generation

We may examine the visual outcomes and the values of the loss functions, on the one hand, as well as which attributes are essential and which are not for the model, in order to study the outcomes of this repository.

It is first necessary to determine which properties on the coin image are significant to humans and which are irrelevant in order to better understand the model's results. To do this, 10 participants are asked to evaluate how two different coin images from the dataset represent the divide. As an example of a coin, consider below an illustrated object of “other” class and a portrait of an emperor or empress, which can be seen in Figure 27.

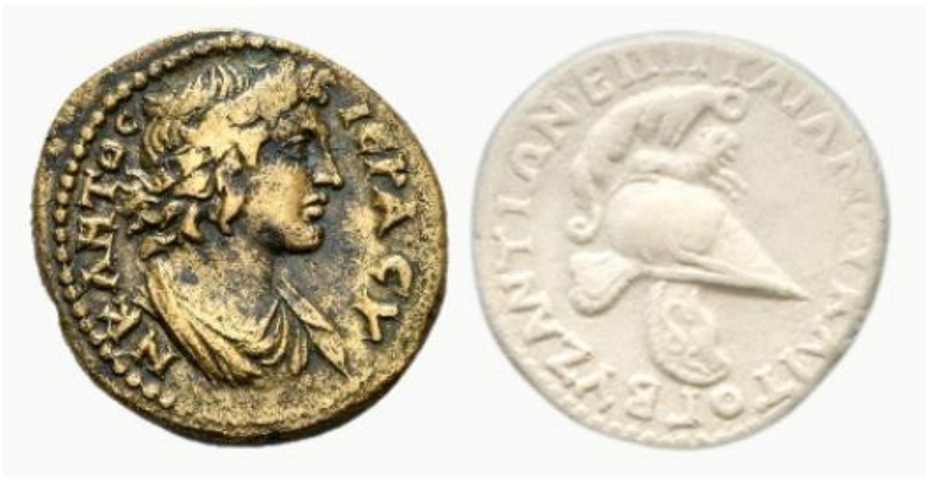

Fig. 27. Two coins from the used dataset

The fact that all responders tended to concentrate primarily on the human traits in the portrait was remarkable. For all participants, the nose, eye, and mouth were the main focus and the point of immediate reference for category-relevant characteristics. Only four out of a total of ten respondents indicated that they thought the human gaze direction was significant. Regarding the relevant characteristics of the other coin from the illustration, we had very different opinions. One half of the respondents tended to concentrate on the object being shown, while the other respondents were more focused on the image's legend. This phenomenon unquestionably demonstrates how difficult and diverse it can be for certain people to separate things into attribute-relevant and irrelevant categories. Rarely people have responded totally in agreement. As a CNN, the foundation upon which all repositories are built, is trained to think like a human, one can only understand the difficulty of the task the AGE model confronts. So, this tendency must be considered while interpreting experimentation results.

Just like the psp encoder, both the GradualStyleEncoder and the ResNetGradualStyleEncoder can be used. The ResNet-based encoder will be used in all of the subsequent experiments because the results from the above experiment suggest that neither type of encoder offers a significant advantage in terms of the score of the loss functions. However, this encoder does approach the score asymptotically faster.

Similar to the experiments in the previous chapter, the images are analyzed using the LPIPS and L2 loss functions. The fundamental idea behind the work is that classes, in this case coin types with fewer than 40 images, can be used to generate images. Due to this, the fundamentally same balanced dataset used for the last experiment of the psp encoder was chosen. The only part of each image that needed changing was its name, as explained in the chapter 5. The model was trained for 100,000 iterations.

In order to be able to classify the results, one must assess the model in a differentiated manner. On the one hand, there are coins with portraits on them, and on the other, there are coins with different figures on them. Two of these coins are seen in the illustration 28. The created images are of an entirely different quality. While the generated image of a portrait can be of decent quality, the quality of the other object is not satisfactory. The existence of a portrait on this coin becomes apparent with closer study. This is because the model separates the category-relevant and irrelevant qualities, and the portrait is given the most weighting because it appears on the majority of coins.

Splitting the dataset

The dataset has to be divided as a result of the preceding experiment's results being verified. The path is ran through the psp encoder and the AGE model with a different dataset in each case to produce a better outcome. With all of the front sides of the images on the one hand, and all of the back sides on the other. This is based on the observation that numerous coins have portraits of Roman emperors on their obverse sides and entire human figures or other items on their reverse sides. As a result, a model's images should to appear more similar, and the AGE model need to provide clearer distinctions between features that are relevant for a given category and those that aren't.

I start with the balanced dataset from the prior findings in order to achieve the greatest performance, with the exception that one dataset excludes the front images and the other excludes the back images.

The original dataset must first be changed before the two models can be trained. While the partial dataset that saves the rear sides receives the images that have the substring "rev" in their names, the partial data that stores the front sides receives the remaining images. The training and test datasets for both the psp encoder and the AGE model can be produced using these two new datasets.

The models degrade in each of the different evaluation metrics, as seen by the values of the loss functions for the two resulting psp encoders in table 7. However, the MoCo loss performance is particularly notable, with a drop of around 35% from the typical balanced dataset. Unfortunately, the resulting images also show the primary negative effects of the loss functions. Figure 28 actually replicates the results of the earlier experiments and doesn't demonstrate any significant improvements in either image. The fact that many coin backs also feature portraits while some fronts do not is the determining factor in the outcome. Because of this, the AGE model continues to keep a variety of the category-relevant features from both pictures and other objects.

	LPIPS loss	L2 loss	MoCo loss	Time 1000 steps
AGE	0.47 ± 0.03	0.046 ± 0.002	-	8h
psp obv	0.17 ± 0.01	0.045 ± 0.003	0.028 ± .003	5.5h
psp rev	0.2 ± 0.02	0.047 ± 0.004	0.032 ± .003	5.5h
AGE obv	0.41 ± 0.03	0.041 ± 0.003	-	7h
AGE rev	0.45 ± 0.04	0.037 ± 0.003	-	7h

Table 7. Results of splitted Datasets in Obv and Rev

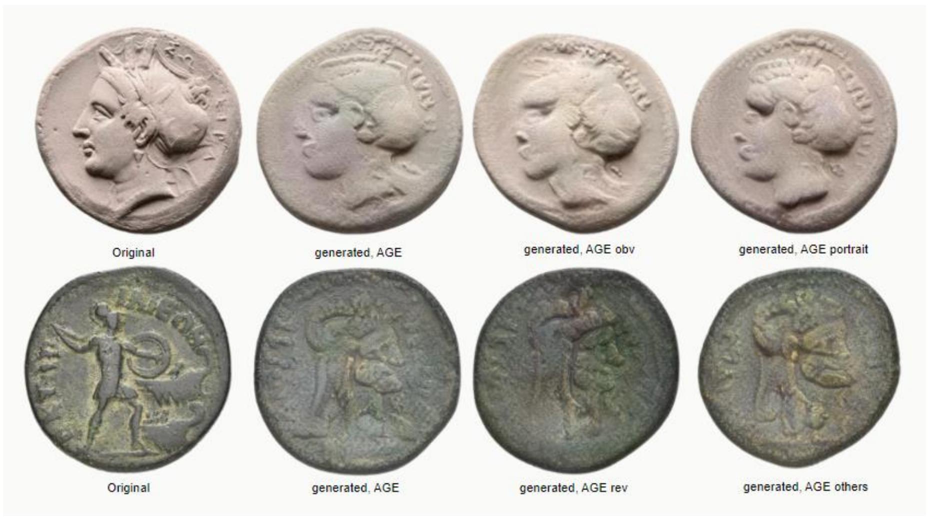

Fig. 28. Results of splitted Datasets in Obv and Rev

Object-detection model

To finally separate the classes portrait and others, the use of an object recognition model that can differentiate these two different classes will be applied. 200 images from each class were labeled and used to train the classifier in order to effectively train the model. Three subsets of the dataset were created for training. To evaluate the final model, a test dataset made up of 20% of all images was used. The validation dataset stores 20% of the remaining images, while the training dataset holds 80% of the remaining 80% images.

After completing the training, which took around 3 hours and involved 10 epochs, the model have an accuracy score of about 86%. The entire original dataset may now be divided into the two classes using this model. This is accomplished by running through each individual class and predicting whether or not each image is a portrait.

With a success rate of 86% and a total of just under 70,000 images to be identified, there are about 10,000 incorrect classifications, which is a far more encouraging result than in the last trial. Two distinct models are utilized with the generated dataset. One model only creates portraits, whereas the other attempts to produce images of other items.

When the entire dataset had been classified, the Portrait class had a total of 38,203 images, whereas the other class had 31,955 images.

Since the psp encoder for the front sides has previously seen many portraits and may therefore shorten training time, the decision was made to start the training from this last checkpoint.

On the other hand, because it has already encountered many classes like them, it was decided to start with the psp encoder of the backs for the class “others”.

However, because our classes now have a different appearance and a different class embedding file as an output, we must train a new model for the AGE model from start.

The loss functions with the psp encoder only slightly improve and show no significant impact. However, since the seperation between portraits and other objects makes it easier to distinguish between features that are pertinent to a certain category and those that are not, the AGE model must nonetheless be the primary focus of the change from this dataset.

The images in figure 28 and the values of the loss functions from table 7, still do not demonstrate any appreciable advancements. This is due to the fact that there are still too many classes for the AGE model to handle (over 10,000).

The datasets can also be reduced here because the values of the VGG16 model under comparison were only trained with 164 classes. The dataset's size must be decreased, but it must also be taken into account that it would be unbalanced without additional restrictions. This results from the fact that some classes only keep one Portrait section image, while some classes store multiple. Of course, the "others" class is also the same in this way. As a result, the model once more finds it difficult to learn all classes equally and once more uses a class with plenty of images as the fundamental building block for producing new images.

In other words, it is s requirement to find classes that present both a large number of images on the one hand and a balanced distribution between the classes "portrait" and "others" on the other. This is the only way to really guarantee that the result is a balanced dataset for the two new generative models.

As a lower bound, the threshold was set to 7, so that a balanced dataset with sufficient data is available for training and subsequent classification by the VGG16 network.

In order to create such a dataset, it is required to repeatedly go through each class in the initial dataset and determine which classes have more than 7 images, both in the dataset for the class "others" and in the dataset for “portraits”.

The dataset was iterated over, and it was found that exactly 168 different coin types contain exactly or more than 7 images from the "portrait" and "others" class. As a result, each dataset for training the generative models contains a total of 1176 images.

While the values for the AGE model do not further increase, the loss function findings from Table 8 demonstrate considerable improvements for the psp encoder. The values of the loss functions cannot be kept up with by the psp encoder's visual output, particularly that of the AGE model. Because portraits are still shown on isolated coins, the psp encoder is still unable to accurately represent the class "others."

The dataset employed here cannot handle the amount of information needed by the AGE model. When comparing the experiments from the original paper, we find that while the number of classes, roughly 130, is nearly identical to that in this experiment, their experiment contains almost 1,000 images per class, which is a factor of 100 more than the number of images per class in this transformed dataset.

The images from Figure 29 clearly demonstrate how difficult it is for the AGE model to identify a class's category-relevant features.

As a result, even after a significant dataset reduction, the results are still lower to those of the psp encoder. Because of this, figure 29 also displays the images that the psp encoder generated. It is obvious that the images not only perform far better than the loss functions but also outperform them in terms of visual impact. The AGE model cannot be utilized to create new images, and instead, we take the psp encoder to creates new images as a result of the experiment's findings.

Even though the optical results and the values of the loss functions with the psp encoder are superior to those with the AGE model, they are still not good enough. For this reason, a step back in the path is taken in this final experiment. So far, the transformed datasets have been applied exclusively to the psp encoder and the AGE model, but still with the same basic framework, StyleGAN2, which trained on all images. For this reason, two new generators are created here, one trained only on "portraits" and the other trained only on the class "others". The transfer learning variation is used for the training in order to significantly reduce training time. This indicates that the StyleGAN model trained previously is used as the starting point for both models, rather than both being trained from scratch. Up to iteration 500, both trained StyleGAN models got extra training.

The loss function values are by far the best that the psp encoder has yet to produce. We have a significant improvement across every value, but unfortunately the figure

29 cannot support the finding results. The generated images are essentially identical to the results of the earlier studies and don't differ much from them.

However, the loss function values indicate that the dataset utilized in this study produces the best results, so the model generated is used to generate new images.

The visual findings and the loss function results diverge because the average generated images perform better across the entire dataset.

A significant disadvantage of the psp encoder is the lack of diversity in the images produced by him. The encoder is unable to provide each generated image with a unique look or set of different features.

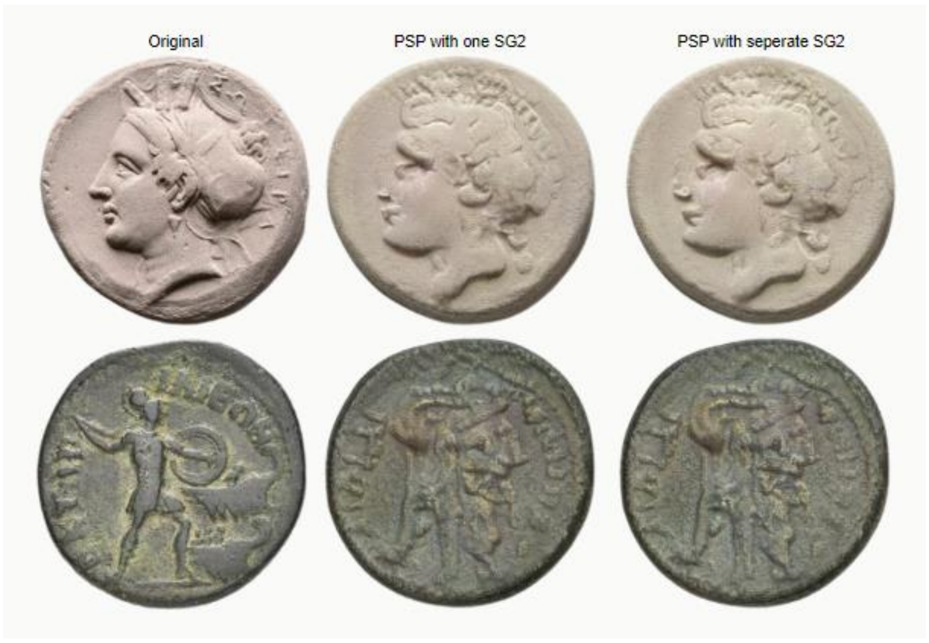

Fig. 29. Results of splitted dataset in classes “portrait” and “others”

The goal is to expand the dataset of 168 coin types after training the two models so that each class contains exactly 42 images, with 21 images displaying the obverse and 21 images displaying the reverse. This is crucial for the later VGG16 model since the corresponding obverse images and reverse images must be concatenated. The dataset will have 7056 images altogether once the generated images have been included.

	LPIPS loss	L2 loss	MoCo loss	Time per 1000 steps
psp portrait	0.17 ± 0.02	0.04 ± 0.003	0.028 ± 0.003	5.7h
psp others	0.19 ± 0.03	0.05 ± 0.003	0.03 ± 0.003	5.2h
AGE portrait	0.42 ± 0.04	0.049 ± 0.003	-	7.2h
AGE others	0.46 ± 0.04	0.052 ± 0.004	-	6.9
psp portrait 168 classes same SG2	0.13 ± 0.01	0.029 ± 0.002	0.02 ± 0.001	4.7h
psp others 168 classes same SG2	0.15 ± 0.02	0.031 ± 0.002	0.025 ± 0.002	4.7h
AGE portrait 168 classes same SG2	0.28 ± 0.04	0.042 ± 0.003	-	6.6h
AGE others 168 classes same SG2	0.33 ± 0.02	0.05 ± 0.003	-	6.6h
psp portrait seperate SG2	0.0985 ± 0.01	0.025 ± 0.002	0.017 ± 0.001	4.7h
psp others seperate SG2	0.106 ± 0.01	0.028 ± 0.002	0.021 ± 0.002	4.7h
SVD	0.009 ± 0.00	0.01 ± 0.001	-	24h

Table 8. Results of the dataset split based on the classifier

6.5 Few-Shot Adaption of Generative Adversarial Networks

The generator, which was trained using grayscale images and TensorFlow, is required for this task.
This repository was only be chosen to compare the quality of the images produced here with the models trained above because the model would have to be retrained for each class and would be outside the scope of this work to be as an input for the VGG16 model.

The same loss functions are also used for this experiment to reveal the differences in quality between the repositories. Before we get to the experiments of this repository, I noticed throughout the training that my GPU was not running at full power. While in the repositories described above the GPU was running at just under 100% power, here it was always between 30 and 40% power during the training. This may well be related to the implemented development environment, which had to be built via extra instructions from the Internet as described in Chapter 5 [66].

The same configurations, config-f, used for StyleGAN were chosen in order to train the SVD model on the complete dataset with around 48 hours for the complete dataset. After training, the model's results reveal a significant improvement over the first path. The images are quite identical to their originals, with the crucially minor differences in the image's component elements. The loss function values from table 8 confirm the successful results.

Sadly, because the images are not labeled, they cannot be used with the VGG16 model. Making a model for each class and generate images would be the only way to produce labeled output images. To demonstrate how much work, it would be to train a new model for each class, I choose one class and train the SVD model exclusively on this class. This enables to examine how many hours are required to train a model for each class in addition to how long it takes the SVD model to adapt to only one class.

It took 50 kimg to train one class, which is close to 10 hours given my resources. If a model were to be trained for each class, the runtime would be close to 13 years. If only 168 classes should produce new images as in the trials mentioned above, it will take about 70 days. This time may undoubtedly be decreased with greater resources or even a fully employed GPU, which could make this repository even more exciting in the future.

Figures 30 and Table 8 both show the excellent results of the SVD model, which was trained on the complete data set and took almost precisely 24 hours to complete. Both results are significantly better than the other models and the model used is able to make minor adjustments using SVD while preserving the essential features of the original image.

Nevertheless, the major drawback of this repository remains the output of unlabeled images.

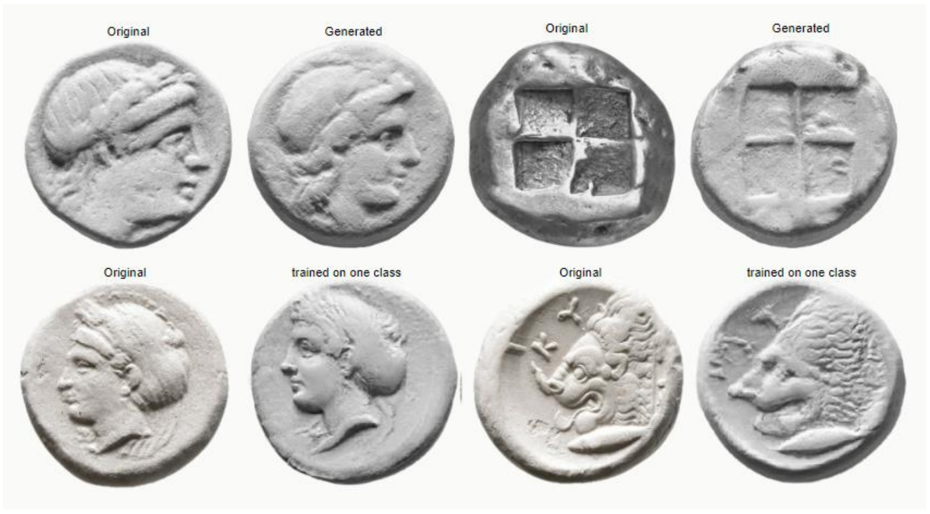

Fig. 30. Results of the SVD model

6.6 VGG16 Classifier

The data set resulting from the inference model of the psp encoder must be converted into a format, which can be seen in Figure 31, to be used for the VGG16 model. An image must be created by horizontally concatenating the front and back sides. The training dataset receives 80% of the concatenated images, whereas the test dataset receives 20%.

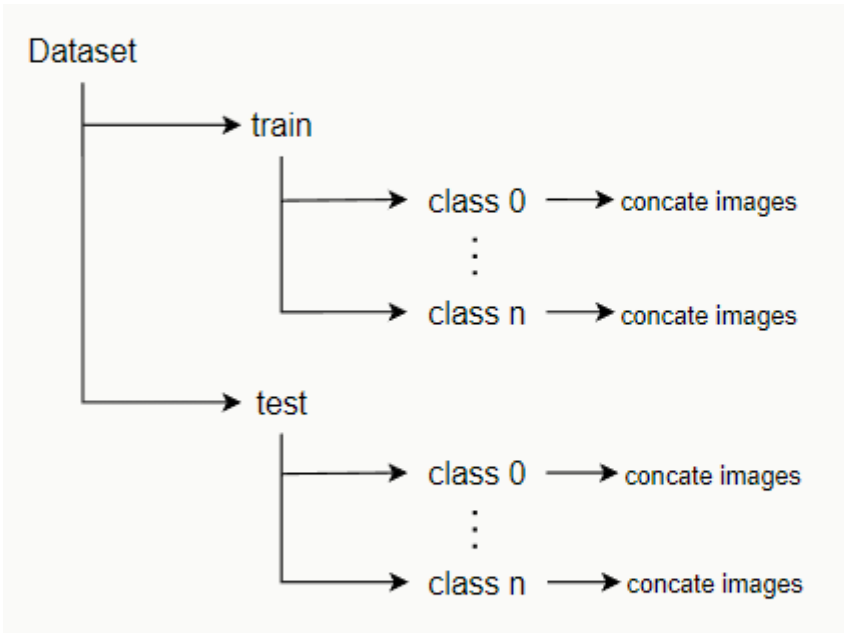

Fig. 31. Structure of dataset used for VGG16 model

The VGG16 model can be trained after the dataset has been transformed. As a starting point, the VGG16 model, which is pretrained on the ImageNet dataset is employed. The model is trained over 10 epochs, with x, in our case 441, steps per epoch. The formula 6 is used to determine how many steps there are in each epoch.

$$x = \frac{\#images}{\#batches} = \frac{7056}{64} = 441 \quad (6)$$

For one epoch of the classification model you need about 1 hour in Google Colab. As a comparison value, the result of the VGG16 classifier is compared with the dataset by taking only classes that contain more than 40 images. Here, with exactly 164 classes, the VGG16 model achieved a top-1 value of 79% and a top-5 value of 97%.

To test the classifier, three different experiments were executed. In the first experiment, the dataset taken from the last psp experiment, with the following conditions. Only coin types that store more than 7 images each from both classes "portrait" and "others" are taken. To make the dataset balanced, also only exactly 7 images of a coin type are taken per class. In the end we have the above resulting dataset of 168 classes with exactly 14 images per coin type with a total of 2,352 images. This also means that there are still no generated images in the dataset here. At the end of the training, the VGG16 model has a Top1 value of 16.3% and a Top5 value of 41.6%. This result should provide a better comparison value for the following experiments.

Now, to see if there is an improvement in performance by expanding the dataset with generated images, each image from the dataset is generated twice, which means we now store a total of 7056 images. The results from this experiment are surprisingly much better than the results from previous experiment and also from the original classifier. However, this is not related to the quality of the generated images, but to the ratio of generated to original images as well as the lack of diversity from the inference model of the psp encoder described above. Figure 32 shows two generated images of an example image and illustrates the low diversity in the output. Because the dataset now consists of ⅔ generated images, fewer original images fall into the test dataset, which means that the model must classify generated images. Since there are many images in a class in the training data that are almost the same, it is very easy for the VGG16 model to classify the image correctly. Only in the unlikely outcome that an original image is included in the test set, the classifier has difficulties.

For exactly this reason of the distribution in the dataset, another experiment was performed in which the distribution is 50-50. That means that for each image we generate only one output image, resulting in a dataset with a total of 4704 images.

The result clearly indicates that the suspicion just expressed is quite true. Due to the redistribution of the dataset, the test set is also balanced and thus also contains more original images that have to be classified, resulting in more errors which can be seen in the values in table 9.

Despite the slightly worse values of the individual score, the results of the classifier show, despite the weaker generated outputs, that the generated images contribute to significantly outperform the results from the first experiment. With better quality, the model can be improved much further and become an excellent alternative for the future.

	Top-1	Top-5	Time
Original VGG16 model	79%	97%	-
VGG16 model 42 images per class	92.9%	98.6%	10h
VGG16 model 14 images per class	16.3%	41.6%	3h
VGG16 model 28 images per class	61%	83.3%	5h

Table 9. Results of all VGG16 models

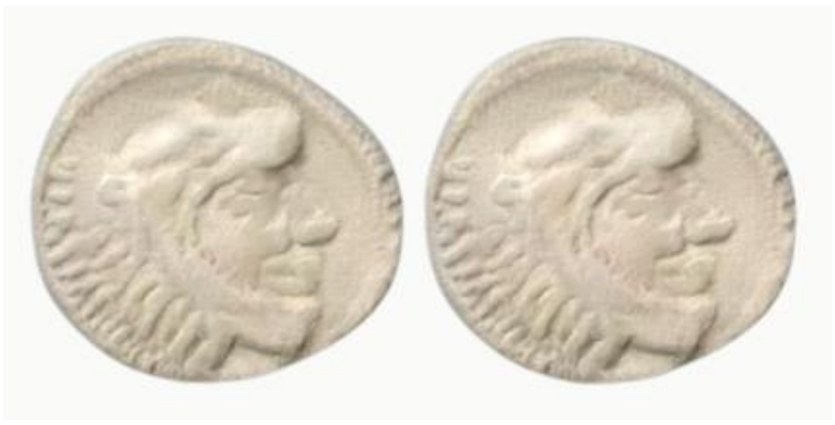
Fig. 32. Two generated images from the same example

7 Discussion and Conclusion

I discuss possible problems and how they might affect the outcomes in this subsection. Additionally, I go over ideas for improving the trained models.
Unfortunately, the outcomes fell short of what was hoped for and expected qualitatively. There are numerous causes for the models' bad performance, which will be covered in greater detail in this chapter.

Quality of Dataset

The quality of the data source is by far the biggest issue for generative models. When first seeing the dataset, the dataset's great diversity of classifications is immediately noticeable. Since only one binary dataset was frequently utilized, this dataset has more than 11,000 distinct classes, making it substantially larger in terms of class size than the individual repositories used. The AGE model is particularly affected by the partially significant discrepancy within a class and the occurring large similarities under the classes since it finds it so challenging to draw a distinction between the classes. The results decrease as a result of the difficulty in separating category-relevant attributes from irrelevant ones.

A model cannot deliver good results if it is fed with suboptimal inputs [69]. The findings of the experiments also support this trend. The quality of the individual images has a significant negative impact on both the values of the loss functions and the visual outcomes. The detection of object recognition can be quite challenging when there is a noticeable corrosion or heavy wear. Loss of substance, particularly in the coin's relief, is a sign of wear and coins have a history of being used as currency, thus because of this use, they frequently have significant wear. Corrosion is another issue, which manifests itself in the relief as holes. Additionally, the handling of the coin in nature, where it interacts chemically with numerous things, might cause corrosion. For instance, oxidation may result in a coin's color changing [15]. The figure 33 provides an illustration of the points being made here. This figure shows the wear, corrosion, and the differences between these images that go along with it.

The fundamental goal of a CNN, upon which all repositories are built, is to accurately imitate the human brain. If persons find it difficult to understand and analyze the differences within a class that was just mentioned, how should such a model not experience difficulty?

The involves reducing of the unbalanced dataset might be shown by certain tests. While some classes only have one image on each coin, others have more than one hundred images on each type. The model then spends a lot of time in just those classes and does not learn the classes with less examples. If the batch size were to be so small that each batch contained just images from a single class, the gradient of the iteration would not be as useful. I had to accept batch sizes of 4 in some circumstances because I had such a large dataset and didn't want to go over my resource limit.

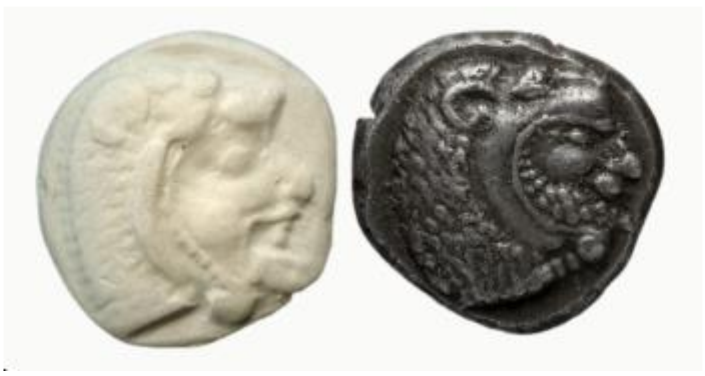

Fig. 33. Different images from the same class

Image resolution

The resolution of the images to be processed was a key factor in the analysis of the results. I intentionally went with a slightly lower resolution because raising it would have resulted in immediate execution aborts. Unfortunately, Google Colab's resources were insufficient for a higher resolution as well. The outcomes of the work by Karras et al. show that substantially better outcomes can be obtained with a greater resolution. The authors obtained a value of 6.3 with Frechet Inception Distance at a resolution of 256x256 and the result of 1.5 at a resolution of 1024x1024 [57]. One potential would be the ability to validate these experiments using better or just more available resources.

Portraits

There is a close relationship between portrait collaboration and all elaborated repositories. In several of them, additional loss functions have also been included to improve portrait editing. For this reason, portraits are more likely to include details that are important to the subject of the image. Here, features such the lips, nose, eyes, and hair, among others, determine which person is represented. These characteristics also make it simple to edit and modify the images. Due to the lack of a specific property that the model may use for all images, these features are missing from the dataset used in this study.

7.1 Future Outlook

In this subsection, I will address some issues that may have a positive effect on the outcome of the models.

Optimization of the classifier. Already several experiments that have helped the generative model were mentioned in this work. Despite this, the classification performed by a classifier can still be improved since it currently has an accuracy rating of 86%. If this value could be increased further, it would improve the distribution of the dataset. Thus, the individual models would have fewer conflicts across the classes and could concentrate more on a specific class.

StyleGAN-Model. The division of the dataset was used in the final stages of the work for both the foundation and the construction of the two paths. There was a generator that was trained using portraits, and another that was trained using the "others" class. Due to time constraints, it was not possible to train the model entirely from scratch, instead, it was continued via transfer learning. As a result, some images from the other class are sometimes included in each model. The model has to be completely retrained in order to be able to solve the issue and another improvement would be to use the upgraded classification system mentioned above here to divide the datasets even more.

Oversampling instead of Undersampling. The current use case for creating balance in the dataset being used here is undersampling. This process involves deleting as many images from each class as necessary to leave each class with the same number of images saved. Another strategy would be what is known as oversampling. In this method, pictures from classes with fewer examples are copied in order to bring them up to parity with the other classes. Thus, in order to achieve a balanced dataset, no significant information is deleted and as a result, the model has more data from which to extract knowledge gained.

Viewing angle for portraits. Another issue is depicted in the illustration 34. There are some portraits in the dataset that look to the left, some that are frontal, and still others that look to the right. In the generated images, it appears that almost every depicted person is looking to the right. This is due to the fact that there are significantly more portraits looking to the right than to the left, causing the model to recognize these particular images as the standard and produce them as a result. This issue might be resolved if the above model had to recognize and differentiate between more classes. In this case, we would have had more than two classes and would have expanded the class for portraits to include every possible viewing angle.

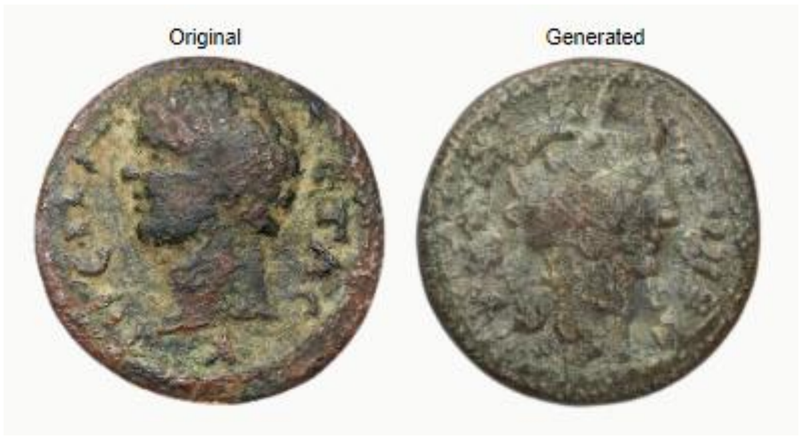

Fig. 34. Generated image from portrait with left viewing angle

Multidisciplinary images by Kaiser. There are several well-known kings and other illustrations that appear in just more than one class and can be used to retrain the object recognition model. In this case, rather of the look directions of the portraits, the class portrait, and all other objects, we would have all kings as a different class. This could be the most challenging option for improving results but consequently, the quality of generated images, could be the best over all experiments.
Quantencomputer. A QC uses quantum mechanics to operate and replaces traditional bits with so-called quantum bits, Qubits. One method of combining ML and QC is to have the QC run multiple solution paths at once, saving time. Another strategy is to optimize the parameters because QC could help identify the optimal parameters for the model quickly and easily. In general, it can be said that quantum machine learning will have a significant impact on the future and will revolutionize the training of models [70]. This work can also benefit from it because it became clear throughout the work that choosing the right configuration and parameters for the model is crucial.
Meta Learning. Meta Learning aims to assist the computer in the learning process in the same way that humans do. In this case, an algorithm receives a variety of tasks and generates a learner that can generalize the model using a limited number of examples. Discriminative models were the initial use cases, but in recent years, generative models have been used more frequently. For instance, the few-shot learning problem is a common meta-learning issue. MAML and Reptile are examples of meta learning algorithms. MAML is an algorithm that trains its parameters in a way that just a small number of gradient steps are required to learn a new task in a different domain [38]. For this reason, the dataset might be divided between classes with more than 40 images and classes with fewer than 40. The larger classes would make up the training set, and the algorithm would be modified to fit the smaller classes with fewer than 40 images.
Similar to MAML, the Algorithm Reptile seeks a general initialization of the parameters, although it uses far less processing power and memory during calculation than MAML does.

Many studies on deep networks have already used MAML and Reptile but only with discriminative models [38]. It would be interesting to see if this approach yielded better results if these algorithms are more available in the domain for generative models.

Many-Classes Few-Shots Learning. The majority of datasets used to solve few-shot learning problems have only a few examples, often just one single image. In the dataset used here, things are slightly different because we have more than 70.000 images total across the dataset. For this reason, this alternative of Few-Shot learning would be a potentially better approach. Unlike the usual many-class many-shots or few-class few-shot models, this approach focuses on a dataset with many classes, each containing only a few examples. During my research, however, I did not come across any implemented models that focused on generative models. Only implementations of discriminative models were found [71, 72].

Few-Shot Classification. Another strategy to achieve the goals of this work would be to focus more on the classification model. As was already mentioned in the previous subsection, the VGG16 model can be replaced with a Many-Classes Few-Shots model. With this approach, it may be tested to see if the classification system can be improved without the need of newly created images. These results can then be compared to those from a dataset with artificially generated images.

Optimization of parameter selection. As was already mentioned with regard to the topic of quantum computing, choosing the parameters is a good way to enhance machine learning models. Due to time and resource constraints, not all parameters could be optimized and their effects on the model could be examined.

The difficulty in determining the ideal parameters is that one does not know in advance which learning rate will produce the greatest results for the various loss functions. After each iteration, the output must be checked and the accuracy compared in order to analyze the parameter selection and, if necessary, make parameter changes.

The Brute-Force approach is an algorithm for parameter selection that enables extensive exploration of all possible options. This algorithm is simple to use and perhaps the most logical approach, but it takes a very long time to process a large number of options.

The evolutional algorithm is another method for parameter optimization. We have a number of models with predefined hyperparameters and are examining the results of each model individually. We only include the models that have the best results in the following. The choice of new models with comparable hyperparameters will depend on the models that were kept. This process will be repeated until the desired outcome has been achieved [78].

Transformer-Based Models. Transformers were first introduced in 2017 [73] and have grown in popularity over the past few years in the field of deep neural networks. Especially when using Natural Language Processing (NLP) in challenging applications, like the Google search engine. This architecture is now being used more and more for computer vision programs, slowly replacing the CNNs for more difficult tasks.

A transformer model is a neural network that can learn the context and significance of connections in sequential data. Transformer models consist of an encoder and a decoder that process data, similar to most neural networks [73].
Although there are several significant similarities between Transformers and CNN, the difference is primarily in the architecture. Changing the architecture of each model from CNNs to transformers could have a positive impact on the generated images.
Technological development. The technological advancement and the subsequent increase in better resources is a very important factor. In the near future, it's possible that better and more speedy results will be produced by raising the quality of the resources available. Additionally, by increasing the batch sizes for training, the chance that a batch may contain only images from a single class will be reduced. A significant milestone for ML would undoubtedly be the inclusion of QC during model training. Additionally, the resolution of the images might be increased with increased resource capabilities, giving the images used for training a higher quality. Additionally, the above improvement suggestions could be implemented more quickly and effectively with the aid of high-quality resources.

7.2 Conclusion

In this work, Few-Shot Learning was used to try to generate new high-quality images from a non-optimal in order to expand the usable dataset for a classification system. To do this, several approaches were taken based on the StyleGAN2 architecture. One approach was to use latent code to edit the image using category-irrelevant attributes so that all the elementary characteristics of the class could still be recognized. Here, it was quickly determined that the AGE model had difficulty with this dataset. For this reason, images were ultimately generated from this path using the psp encoder.

In the second approach, the singular vectors of the singular value decomposition of an image were frozen during training and only the singular values were continuously updated to result in a model to generate high quality images. Only the SVD model was able to meet the high expectations and generate satisfactory images from the dataset. Since this model cannot generate classified images and each image would have to be classified manually, the generated images could not be used for the classification system. The other approach provided images with class membership but could not generate high-quality images. Consequently, the VGG16 model trained on the generated dataset could not keep up with the given values of another model. Only when two new examples are generated for each image, the VGG16 model performs better than the original results. However, as described above, the generated images are too similar, causing the classifier to memorize the image.

Despite the unsatisfactory results, this work has shown what approaches exist for generating ancient coins and where their strengths and weaknesses are.

What was visible across all experiments is that the generative model has less difficulty generating coins with portraits but has even more difficulty with images depicting other objects on the coin. Neither splitting the dataset by obverse and reverse nor using the VGG19 classifier helped improve the generative model. The only appealing results always occurred when the dataset was reduced to a few hundred images. The way of splitting portraits and all others, limiting the number of classes to 168 classes, gave the best results. However, even this model could not match the values of the current accuracy of the VGG16 classifier.

One finding of this work shows how dependent ML models are on a qualitative dataset. As described in the last chapter, the dataset used here has clear disadvantages compared to the usual datasets used by each repository.

To this end, it was shown that generative models require a lot of computation time to learn to the dataset used here. The resulting training time is strongly dependent on the resources, the number of data and the choice of parameters.

When choosing the parameters, a middle ground between computational power and quality gain must be taken.

It has been demonstrated through some experiments that high quality images can be produced, but only for a subset of the dataset. The potential future implications discussed above may help to increase the working subset of the dataset.

References

1. Bayerisches Münzkontor 2021, Münzkunde, accessed 2. January 2023 https://www.muenzkontor.de/muenzkunde-numismatik
2. Karl Christ: Antike Numismatik. Einführung und Bibliographie. Wissenschaftliche Buchgesellschaft Darmstadt 1991
3. University of Zürich 2023, Münzen als Geld, accessed 2. January 2023, https://www.adfontes.uzh.ch/tutorium/masse-zahlen-und-geld/muenzen-als-geld
4. Bayerisches Münzkontor 2017, Lexikon, accessed 2. January 2023, https://www.muenzwissen.com/lexikon
5. IBM 2022, What is machine learning?, accessed 6. January 2023, https://www.ibm.com/topics/machine-learning
6. Berkeley School of Information 2022, What is Machine Learning (ML) ?, accessed 2 January 2023, https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/
7. Laurenz Wuttke 2021, Was ist Unsupervised Learning?, accessed 2. January 2023, https://datasolut.com/wiki/unsupervised-learning/
8. Ouali, Y., Hudelot, C. & Tami, M. 2020, An Overview of Deep Semi-Supervised Learning
9. Buffet, O., Pietquin, O. & Weng, P. 2020, Reinforcement Learning
10. Steinberg, E., Dyuldina, M., Kaptsov, N. & Emelyanov, T. 2021, How to recognize coins with deep learning visual model
11. Conn, B & Arandjelovic, O. 2017, Towards Computer Vision Based Ancient Coin Recognition in the Wild – Automatic Reliable Image Preprocessing and Normalization
12. Schlag, I. & Arandjelovic, O. 2017, Ancient Roman Coin Recognition in the Wild using Deep Learning Based Recognition of Artistically Depicted Face Profiles
13. Anwar, H., Anwar, S., Zambanini, S. & Porikli, F. 2020, Deep Ancient Roman Republican Coin Classification via Feature Fusion and Attention
14. Gutberlet, F. 2021, Simulation von Abnutzung und Korrosion antiker Münzen mittels Cycle Generative Adversarial Networks Verfahren
15. Anwar, H., Sabetghadam, S. & Bell, P. 2020, An Image-Based Class Retrieval System for Roman Republican Coins
16. IBM 2022, What is a neural network?, accessed 21. December 2022, https://www.ibm.com/topics/neural-networks
17. DeepAI 2022, What is a Feed Forward Neural Network?, accessed 21. December 2022, https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network
18. O`Shea, K. & Nash, R. 2015, An Introduction to Convolutional Neural Networks
19. Dwarampudi, M. & Reddy, S. 2019, Effects of padding on LSTMs and CNNs
20. Riad, R., Teboul, O., Grangier, D. & Zeghidour, N. 2022, Learning strides in convolutional neural networks
21. Datawow 2020, Interns Explain CNN, last accessed 15. December 2022, https://datawow.io/blogs/interns-explain-cnn-8a669d053f8b
22. LeCun, Y., Kavukcuoglu, K. & Farabet, C. 2010, Convolutional networks and applications in vision
23. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A & Bengio, Y. 2014, Generative Adversarial Networks
24. Google Developers 2022, The Discrimator, last accessed 16. December 2022, https://developers.google.com/machine-learning/gan/discriminator?hl=en

25. Dubey, S., Singh, S. & Chaudhuri, B. 2021, Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark
26. Katte, A. 2018, Choosing between GAN or Encoder Decoder Architecture for ML Applications is Like Comparing Apples to Oranges, last accessed 21. December 2022 https://analyticsindiamag.com/choosing-between-gan-or-encoder-decoder-architecture-for-ml-applications-is-like-comparing-apples-to-oranges/
27. Google Developers 2022, The Generator, last accessed 16. December 2022, https://developers.google.com/machine-learning/gan/generator?hl=en
28. IBM Technology 2021, What are GANs?, last accessed 6. January 2023, https://www.youtube.com/watch?v=TpMIssRdhco
29. Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J. & Aila, T. 2019, Analyzing and Improving the Image Quality of StyleGAN
30. Karras, T., Laine, S. & Aila, T. 2018, A Style-Based Generator Architecture for Generative Adversarial Networks
31. Max-Planck-Institut 2017, Singulärwertzerlegung, last accessed 5. January 2023, https://resources.mpi-inf.mpg.de/departments/d1/teaching/ss10/MFI2/kap47.pdf
32. Gramlich, G. 2004, Anwendung der Linearen Algebra mit MATLAB
33. Data Science Team 2020, Singulärwert-Zerlegung, last accessed 6. January 2023, https://datascience.eu/de/maschinelles-lernen/singularwert-zerlegung/
34. Microsoft 2022, Bewerten des SVD-Empfehlungsmoduls, last accessed 6. January 2023, https://learn.microsoft.com/de-de/azure/machine-learning/component-reference/score-svd-recommender
35. Dr. Kumar, V. 2020, Singular Value Decomposition (SVD) & Its Application in Recommender System, last accessed 6. January 2023, https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/
36. Dilmegani, C. 2022, What is Few-Shot Learning?, last accessed 6. January 2023, https://research.aimultiple.com/few-shot-learning/
37. Bennequin, E. 2019, Meta-learning algorithms for Few-Shot Computer Vision
38. Finn, C., Abbeel, P. & Levine, S. 2017, Model-Agnostic Meta-Learning for Fast Adaption of Deep Networks
39. Chen, W., Liu, Y., Kira, Z., Wang, F. & Huang, J. 2019, A closer look at Few-Shot classification
40. Robb, E., Chu, W., Kumar, A. & Huang, J. 2020, Few-Shot Adaption of Generative Adversarial Networks
41. Richardson, E., Alaluf, Y., Patashnik, O., Nitzan, Y., Azar, Y., Shapiro, S. & Cohren-Or, D. 2020, Encoding in Style: a StyleGAN Encoder for Image-to-Image Translation
42. Pidhorskyi, S., Adjeroh, D. & Doretto, G. 2020, Adversarial Latent Autoencoders
43. Zhu, J., Shen, Y., Zhao, D. & Zhou, B. 2020, In-Domain GAN Inversion for Real Image Editing
44. Karras, T., Laine, S. & Aila, T, 2018, A Style-Based Generator Architecture for Generative Adversarial Networks
45. Sridhar, A. 2022, Meta-GAN for Few-Shit Image Generation
46. Robb, E., Chu, W., Kumar, A. & Huang, J. 2020, Few-Shot Adaption of Generative Adversarial Networks
47. Tov, O., Alaluf, Y. & Nitzan, Y. 2021, Designing an Encoder for StyleGAN Image Manipulation
48. Alaluf, Y., Patashnik, O. & Cohen-Or, D. 2021, ReStyle: A Residual-Based StyleGAN Encoder via Iterative Refinement

49. Ding, G., Han, X., Wang, S., Wu, S., Jin, X., Tu, D. & Huang, Q. 2022, Attribute Group Editing for Reliable Few-shot Image Generation
50. Liu, B., Zhu, Y., Song, K. & Elgammal, A. 2021, Towards Faster and Stabilized GAN Training for High-Fidelity Few-Shot Image Synthesis
51. Zhao, S., Liu, Z. Lin, J., Zhu, J. & Han, S. 2020, Differentiable Augmentation for Data-Efficient GAN Training
52. Shaham, T., Dekel, T. & Michaeli, T. 2019, SinGAN: Learning a Generative Model from a Single Natural Image
53. Sushko, V., Gall, J. & Khoreva, A. 2021, One-Shot GAN: Learning to Generate Samples from Single Images and Videos
54. CORPUS NUMMORUM ONLINE 2022, last accessed 20. October 2022, https://www.corpus-nummorum.eu/search/types
55. Karras, T. & Hellsten, J. 2021, StyleGAN2-ADA – Official PyTorch implementation, https://github.com/NVlabs/stylegan2-ada-pytorch
56. NVIDIA DEVELOPER 2023, CUDA Zone, last accessed 17. December, https://developer.nvidia.com/cuda-zone
57. Karras, T., Aittala, M., Hellsten, J., Laine, S., Lehtinen, J. & Aila, T. 2020, Training Generative Adversarial Networks with Limited Data
58. Kynkäänniemi, T. 2019, Evaluation Metrics of Generative Adversarial Networks
59. Omertov, 2022, encoder4editing, last accessed 5. January 2023, https://github.com/omertov/encoder4editing
60. Mo, S., Cho, M. & Shin, J 2020, Freeze the Discriminator: a Simple Baseline for Fine-Tuning GANs
61. Rebuffi, S., Bilen, H. & Vedaldi, A. 2017, Learning multiple domain with residual adapters
62. TensorFlow, 2023, last accessed 8. January 2023, https://www.tensorflow.org
63. Dr. Kinghorn, D. 2021, last accessed 8. January 2023, https://www.pugetsystems.com/labs/hpc/How-To-Install-TensorFlow-1-15-for-NVIDIA-RTX30-GPUs-without-docker-or-CUDA-install-2005/
64. Alaluf, Y., Richardson, E. & Morozov, A. 2021, last accessed 6. January 2023, https://github.com/eladrich/pixel2style2pixel/issues/154
65. Schultz, D. 2022, last accessed 6. January 2023, https://github.com/dvschultz/stylegan2-ada-pytorch
66. Dr. Kinghorn, D. 2020, last accessed 16. January 2023, https://www.pugetsystems.com/labs/hpc/How-To-Install-TensorFlow-1-15-for-NVIDIA-RTX30-GPUs-without-docker-or-CUDA-install-2005/
67. Zhao, H, Gallo, O., Frosio, I. & Kautz, J. 2015, Loss Functions for Image Restoration with Neural Networks
68. Tang, H., Liu, H. & Sebe, N. 2019, Unified Generative Adversarial Networks for Controllable Image-to-Image Translation
69. Dodge, S. & Karam, L. 2016, Understanding How Image Quality Affects Deep Neural Networks
70. Sultanow, E., Bauckhage, C., Knopf, C. & Piatkowski, N 2022, Sicherheit von Quantum Machine Learning
71. Liu, L., Zhou, T., Long, G., Jiang, J. & Zhang, C. 2020, Many-Class Few-Shot Learning on Multi-Granularity Class Hierarchy
72. Anonymous authors 2019, MAHINET: A Neural Network For Many-Class Few-Shot Learning With Class Hierarchy

73. Vaswani, A., Shazeer, N., Parmar, N., Jones, L., Kaiser, L. & Polosukhin, I. 2017, Attention Is All You Need
74. Karras, T & Hellsten, J. 2021, last accessed 20 January 2023, https://github.com/NVlabs/stylegan2-ada-pytorch
75. Ding, G. 2022, last accessed 20 January 2023, https://github.com/UniBester/AGE
76. Crawford, M. 1974, Roman Republican Coinage
77. Zisserman, A. & Simonyan, K. 2014, Very Deep Convolutional Networks for Large-Scale Image Recognition
78. Menzies, T., Kocagüneli, E., Minku, L., Peters, F. & Turhan, B. 2015, Sharing Data and Models in Software Engineering, Chapter 24

8 Appendix

8.1 Appendix A

	LPIPS loss	L2 loss	MoCo loss	Time per 1000 steps
One class	0.075 ± 0.02	0.022 ± 0.002	0.014 ± 0.002	4h
Five classes	0.098 ± 0.03	0.025 ± 0.002	0.0165 ± 0.002	4.3h
Classes more 40	0.17 ± 0.04	0.037 ± 0.003	0.048 ± 0.004	5.4h
Balanced dataset	0.14 ± 0.03	0.034 ± 0.003	0.041 ± 0.004	4.7h
AGE	0.47 ± 0.03	0.046 ± 0.002	-	8h
psp obv	0.17 ± 0.01	0.045 ± 0.003	0.028 ± .003	5.5h
psp rev	0.2 ± 0.02	0.047 ± 0.004	0.032 ± .003	5.5h
AGE obv	0.41 ± 0.03	0.041 ± 0.003	-	7h
AGE rev	0.45 ± 0.04	0.037 ± 0.003	-	7h
psp portrait	0.17 ± 0.02	0.04 ± 0.003	0.028 ± 0.003	5.7h
psp others	0.19 ± 0.03	0.05 ± 0.003	0.03 ± 0.003	5.2h
AGE portrait	0.42 ± 0.04	0.049 ± 0.003	-	7.2h
AGE others	0.46 ± 0.04	0.052 ± 0.004	-	6.9
psp portrait 168 classes same SG2	0.13 ± 0.01	0.029 ± 0.002	0.02 ± 0.001	4.7h
psp others 168 classes same SG2	0.15 ± 0.02	0.031 ± 0.002	0.025 ± 0.002	4.7h
AGE portrait 168 classes same SG2	0.28 ± 0.04	0.042 ± 0.003	-	6.6h
AGE others 168 classes same SG2	0.33 ± 0.02	0.05 ± 0.003	-	6.6h
psp portrait seper-ate SG2	0.0985 ± 0.01	0.025 ± 0.002	0.017 ± 0.001	4.7h
psp others seperate SG2	0.106 ± 0.01	0.028 ± 0.002	0.021 ± 0.002	4.7h
SVD	0.009 ± 0.00	0.01 ± 0.001	-	24h

Table 10. All results for the psp encoder and AGE model

8.2 Appendix B

To download the individual trained models from the work, they have been uploaded to the Goethe University Hessenbox cloud storage service. The link to the models, as well as the corresponding scripts for the transformation of the datasets and the Conda environments are located in my GitHub repository. To keep the number of models in a manageable range, only the models that gave the best results are uploaded. For the StyleGAN2 generator, the models can be found for the SVD model and also the configuration with two mapping layers implemented in Pytorch for each of the class “portrait” and “others”. For the psp encoder as well as for the AGE model, the models and datasets are uploaded from the experiment with the classes Portrait and others separated from the classifier with 168 classes. The only difference between the models is that the psp encoder was trained on the separated StyleGAN models and the AGE models were trained on the non-separated StyleGAN. Due to the weak results of the restyle encoder, neither the StyleGAN model used for this purpose nor the restyle encoder is uploaded. For the SVD model, on the other hand, the best SVD model in terms of results is also uploaded with the StyleGAN.

QR-Code:

Bitly-Link: https://bit.ly/JannikHockMasterthesis

The GitHub repository is set to private, if you are interested in taking a closer look at the repository, feel free to send me an email: jannikhock30@googlemail.com

8.3 Deutsche Zusammenfassung

Spätestens seit der Website thispersondoesntexist sind generative Modelle bekannt dafür, realistische Bilder einer Zieldomäne generieren zu können. Ein starker Nachteil dieser Modelle ist jedoch die benötigte große Menge an Daten, die in diversen Domänen allerdings schwer zu holen sind. Unter anderem aus diesem Grund haben sich Deep-Learning Algorithmen im Laufe der Zeit immer weiter verbessert, um sie effizienter im Umgang mit Daten und Ressourcen zu nutzen. Das Resultat waren Few-Shot Learning Probleme, die sich aus nur wenigen Beispielen einer Zieldomäne anpassen können.
Die Erforschung antiker Münzen ist ein Themengebiet, das genau in diese Problemstellung reinfällt. Der hier benutzte Datensatz erreicht nur bei knapp 1% aller Klassen den kritischen Schwellenwert von 40, welcher angibt, ab wann eine Klasse für das Klassifizierungssystem verwendet werden kann.

Daher liegt der Schwerpunkt dieser Arbeit auf der Erstellung neuer Bilder von antiken Münzen unter Verwendung bereits bekannter GitHub-Repositories und deren Ansätze des Few-Shot-Learnings. Es werden verschiedene Techniken und Algorithmen verwendet, um neue Bilder auf der Grundlage der StyleGAN2-Architektur zu erzeugen. Zahlreiche Experimente zeigen die Vor- und Nachteile der einzelnen Modelle sowie Verbesserungspotenziale auf. Für die Analyse werden individuelle Metriken sowie visuelle Resultate herangezogen, um die Ergebnisse der einzelnen Experimente zu vergleichen. Ziel der Arbeit ist es, den Datensatz so zu vergrößern, dass jede Klasse mit dem VGG16-Modell klassifiziert werden kann.