Heidelberg University
Institute of Computer Science
Computer Vision Group

Master Thesis

Combining Multiple Styles for Neural Style Transfer

Felix Munzlinger

supervised by
Prof. Dr. Filip Sadlo,
Prof. Dr. Björn Ommer,
Matthias Wright

Matriculation Number: 3552207
Date of Submission: 01.05.2022

Abstract

Throughout history, art regularly had to reinvent itself. Now, machine learning detaches paintings from the canvas, making art take a further step. Previous research was able to apply representations of artistic styles to images. In this work, we present an approach to combine multiple styles for neural style transfer, which we perform by applying adaptive instance normalization (AdaIN). For this, we use a generative adversarial network (GAN) with a few-shot image generator with a content conditioned style encoder (COCO-FUNIT). We achieve best results when randomly sampling styles from three paintings, with two of them being from the same artist. Using t-SNE, we show that our model can reproduce real-world relations between styles. However, we find that further methods need to be developed to better measure the quality of style intersections.

Kurzzusammenfassung

Im Laufe der Geschichte war die Kunst einem ständigen Wandel unterlegen. In der Gegenwart löst Machine Learning die Kunst von der Leinwand und ermöglicht es, durch Neural Style Transfer den Stil eines Kunstwerkes auf ein anderes Bild zu übertragen. Gegenstand dieser Arbeit ist, diese Technik zu nutzen und mittels Adaptive Instance Normalization (AdaIN) mehrere Stile miteinander zu kombinieren. Hierfür verwenden wir ein Generative Adversarial Network mit einem Few-Shot Image Generator, der über einen Content-conditioned Style Encoder (COCO-FUNIT) verfügt. Wir erzielen die besten Ergebnisse mit einer zufälligen Stilkombination, bei der wir aus drei Gemälden auswählen, von denen zwei vom selben Künstler stammen. Mittels t-SNE zeigen wir, dass unser Modell imstande ist, Zusammenhänge zwischen Stilepochen zu erkennen. Wir stellen jedoch ebenfalls fest, dass es weiterer Methoden bedarf, mit denen die Qualität von Stilkombinationen gemessen werden kann.

Dedication

Liebe Mama, lieber Papa,
ohne eure Unterstützung, euer Vertrauen und euren Rat wäre weder diese Arbeit noch mein Weg als solcher möglich gewesen. Ihr gebt mir Sicherheit, Liebe und Stabilität. Ich bin euch für immer dankbar, dass Ihr mich zu dem Menschen gemacht habt, der ich heute bin.

Liebe Oma, lieber Opa,
ich bin dankbar dafür, dass ihr Teil meines Lebens seid. Ihr seid für mich Vorbild und bestärkt mich in meinem Glauben an mich selbst. Dank euch habe ich gelernt, auch in widrigen Umständen zurechtzukommen und meinen Weg weiterzugehen.

Liebe Sarah,
ich bin dir nicht nur dafür dankbar, dass du in mühevoller Kleinarbeit meine Thesis Korrektur gelesen hast, sondern auch für die inspirierenden Gespräche und Ratschläge, die du mir gegeben hast. Dein Idealismus, dein Streben nach Erkenntnis und dein Drang, die Welt zu einem besseren Fleck zu machen, werden für immer in mir tiefste Bewunderung hervorrufen. Danke, dass du immer an meiner Seite warst.

Declaration

Herewith I confirm truthfully that I completed this work on my own with the assistance of my supervisors. All used resources are stated precisely and thoroughly. Text, graphs and figures that were adopted, either unchanged or with alteration, from the work of others have been clearly marked with the corresponding reference. Objects that are not marked with references are my own work. This work was compiled to the best of my knowledge.

place and time	Felix Munzlinger

Acknowledgements

I want to thank my whole research group for not only giving me the possibility to pursue my own research interest, but also for supporting me along the road. Whenever I had questions, I got an answer and was able to improve on it. After two years of being part of Ommer Lab, I was constantly able to improve not only due to the work with Matthias, but also Nikolai, who was my previous supervisor. When I started my master's degree, I was still undecided which minor to choose. Without knowing that there was a research group focusing on cultural analytics and artificial intelligence, I chose *European Art History* as a minor because it was what I was interested in. Finding a research group that is focused on the exact same thing I was interested in was one of the most self-fulfilling experiences I had while studying at the University of Heidelberg. Both, my previous research topic ("Self-Supervised Contrastive Learning of Visual Representations for Finding Corresponding Regions across Artworks") and the following work sharpened my understanding of art from an analytical perspective and added to what I was learning in Art History lectures. I want to thank Prof. Ommer not only for this opportunity, but also for his lectures and his seminar that I attended. Unfortunately, in my opinion, the modern university system, does not value teaching nearly as much as it should be. I am grateful that even at the height of Covid-19 Prof. Ommer did not reduce teaching to the minimum and looked for innovative ways to make earning credits possible.

Contents

List of Figures

List of Tables

List of Abbreviations

1D one dimensional

2D two dimensional

3D three dimensional

AdaIN adaptive instance normalization

CNN convolutional neural networks

COCO few-shot image generator with content conditioned style encoder

FID Fréchet inception distance

FUNIT few-shot image generator

GAN generative adversarial networks

L_1 Manhattan distance

L_2 Euclidean distance

MAP energy minimization problem

NeRF neural radiance and density fields

NN neural networks

NST neural style transfer

RGB red, green and blue

SIFID single image Fréchet Inception Distance

t-SNE t-distributed stochastic neighbor embedding

TTUR time-scale update rule

VGG very deep convolutional networks for large-scale image recognition

WCT whitening and coloring transform

1 Introduction

In medieval times, young aspiring artists were educated in a workshop of a major artist. For example, Leonardo Da Vinci was educated in the workshop of Andrea del Verrocchio. There, he learned to prepare colors and paint in his master's style. By this, he gradually developed his skills and with the progression of his training, he was allowed to paint backgrounds in some of del Verrocchio's paintings. While doing so, he was bound to the style of his master to create a coherent painting. In machine learning, such a process is called supervised learning. If the ground truth is already known, the goal can be formulated straight forward.

However, young artists only learn to paint like their master to sharpen their skills or to later use them on other projects. The artist essentially gets inspired by one painting and applies its style to new content. Surprisingly, neural style transfer learns to transfer style similarly, which is called unsupervised learning. Here, a model learns domain-specific knowledge and uses it to perform an image manipulation task.

Now, let's imagine that young Da Vinci was not only supervised by del Verrocchio, but also by Fra Angelico. What would Da Vinci's style have looked like instead?

We project this thought experiment onto the topic of neural style transfer and raise the question:

Is it technically possible to intersect artistic styles using neural style transfer?

And, if it is possible to intersect these styles, we discuss:

To what extent do style intersections create a new artistic style?

Figure 1.1: "Annunciation"(1475) by Leonardo da Vinci und Andrea del Verrocchio (left) and Fra Angelico (1430)(right)

The following work presents a neural style transfer (NST) approach using a GAN architecture. We first cover the theoretical background in machine learning, NST and GAN. Then, we introduce the models we use within the GAN architecture, its objective function, and the approaches by which we train the network. We proceed with one of the most difficult tasks in training a GAN: finding a suitable (hyper-)parameter setting and fixing training instabilities. After this, we discuss our results and compare them to other research results. Finally, we summarize our results, outline future research possibilities and conclude by discussing selected societal implications of neural style transfer.

2 Theoretical background

Computer Vision is a field of research which focuses on image and video processing. This includes applications like object recognition, image classification and the segmentation of objects within an image. In reality, such tasks are virtually effortless for humans, but challenging for computers, making it one of the most active research areas for machine learning applications. [5] Machine learning is a form of applied statistics, helping us to construct algorithms that are able to learn from data, to make predictions or decisions. In this regard, learning means that the algorithm is able to improve its performance by incorporating experience while performing a specific task. A special form of machine learning is deep learning, which focuses on representation learning. In deep learning, the data is transformed by different layers into a more abstract and composite representation. Such representations can be used to detect similar objects or to process data in feature space. [24]

In this chapter, we cover the theoretical background needed to work with NST. First, we introduce neural networks (NN) and enhance their approach to convolutional neural networks (CNN). We then take a look at style transfer methods from a rule-based and NN perspective. Last, we address generative adversarial networks. Here, we consider them from a game theory perspective, discussing how these insights can be implemented by neural networks, improve their stability and demonstrate how we can evaluate these models.

2.1 Neural networks

The idea of a NN was first proposed in 1944 by McCulloch and Pitts [21, 24], who presented the McCulloch-Pitts neuron and were able to show that Turing-computable programs can be computed by a network with a finite number of artificial neurons. The model was inspired by the biological brain and intended to imitate its learning. The reasoning behind this decision was "[...] that the brain provides a proof by example that intelligent behavior is possible, and a conceptually straightforward pass to building intelligence is to reverse engineer the computational principles behind the brain and duplicate its functionality." [24] Today's deep learning goes beyond the neuro-scientific perspective and focuses on learning multiple levels of composition, which are not necessarily inspired by the brain. This section focuses on the

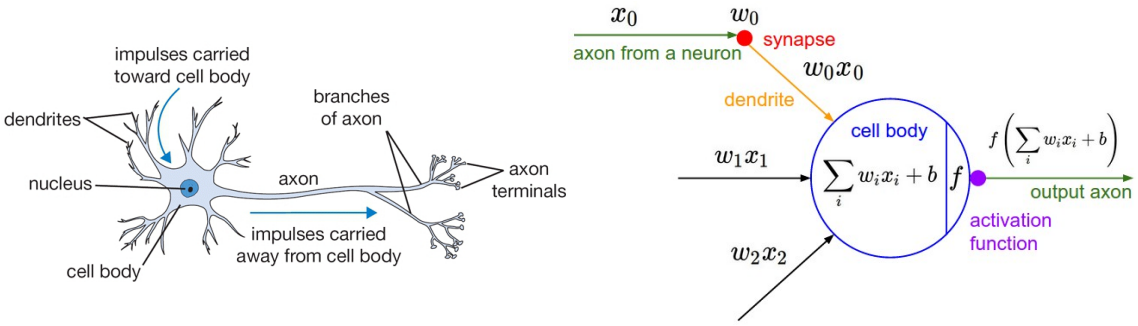


Figure 2.1: Comparison between a biological neuron (left) and its mathematical model (right). Adapted from [49]

fundamental components of NN and explains how they can be used in composition. [24]

2.1.1 Neuron

In the biological brain, the *neuron* is an electrically excitable cell that communicates with other cells. It serves the purpose of a functional switching unit in a biological nervous system. For this, it applies prior knowledge to an input signal to filter its response. Its communication is bound to its ability to process input and produce output signals. Neurons communicate with one another by means of long fibers, which are called *axons*. Through these fibers, electrical signals are sent. If such signals travel along the axons, at some point, they reach a junction, which is called a *synapse*. Here, the electrical signals cause a neurotransmitter chemical to be released, which establishes a connection and alters the electrical activity. The signal can then be processed by a subsequent neuron. [36] We can recreate such a switching unit by applying weights to input signals, add them up and use an *activation function* to rate the weighted input signals. Both processes are displayed in figure 2.1[49].

Let $x \in \mathbb{R}^n$ be an n-dimensional input vector, $w_i \in \mathbb{R}$, $i = 0, \ldots, n-1$ be the weight, which we apply to x_i and $b \in \mathbb{R}$ be a bias term. Then, we can compute a matrix multiplication between x_i and w_i and add a bias term. Its result is then altered by an activation function $f : \mathbb{R} \to \mathbb{R}$, such that we can define the signal processing in a neuron as [8]:

$$y = f(b + \sum_{i=0}^{n} w_i x_i) \tag{2.1}$$

The most commonly used activation function is referred to as Rectified Linear

Unit (ReLU) [29] and is defined as:

$$f(x) = \begin{cases} 0, & \text{if x} < 0 \\ x, & \text{otherwise} \end{cases} \tag{2.2}$$

This activation function, is used to determine whether the neuron response was positive or not. Positive responses can be linked to patterns, which the neuron thinks it is able to detect. The activation functions help the model to adapt to a variety of data (also called *generalization*) and to differentiate between different output signals. A downside of ReLu is, that it always turns negative responses to 0, such that those values can not be mapped, and hence, can not be distinguished.[49, 24] The advanced Leaky ReLU tries to solve this issue by replacing 0 in the first case of equation 2.2 with $0.01x$, to increase its activation range. [62]

2.1.2 Model architecture

Staying in the brain analogy, it is not enough to have processing units like neurons; they also need to be connected to interact with each other. In a biological brain, the synapses take up the job of transferring in- and output signals between neurons. [36] They form a web of neurons, which we also need to build in order to re-engineer the brain. We call this web the *computational graph* and define it using Graph Theory. Let $\mathcal{V}$ be a finite set of neurons, and let $\mathcal{E}$ be a finite set of synapses. We can then formulate a graph $G(\mathcal{V}, \mathcal{E})$, with $uv \in \mathcal{E}$ and $u, v \in \mathcal{V}$. Each neuron $u \in \mathcal{V}$ has weights and biases, which are called *learnable parameters*. We define θ as the vector of all learnable parameters in G. We can utilize neurons by linking them into sequences. In consequence, we formulate G as a directed graph, which has a fixed enter and exit node. In a directed graph, we can only traverse $uv \in \mathcal{E}$ from u to v, but not in the opposite direction. [88]

Within our NN we organize nodes in layers $L_i, i = 0, \dots N-1$, where N is the number of layers in the NN and all L_i are distinct subsets of $\mathcal{V}$. All nodes of layer L_i receive the same input signal from L_{i-1}, or in case of L_0, from the input to the network directly. All layers except L_0 and L_N are called hidden-layers, since their interactions are hidden from us in the sense that we do not use them individually and only access them from their previous layer. We visualize this interaction in figure 2.2. Since we have sequential layers, the inputs to each layer is affected by the parameters of all preceding layer. By this property, small changes to the network parameters amplify as the network becomes deeper [49].

The computational graph G interlinks neighboring layers of neurons and act as a complex non-linear function, which forms the NN architecture. If G is free of cycles, meaning that there is no sub-graph which forms a closed chain, we can call the

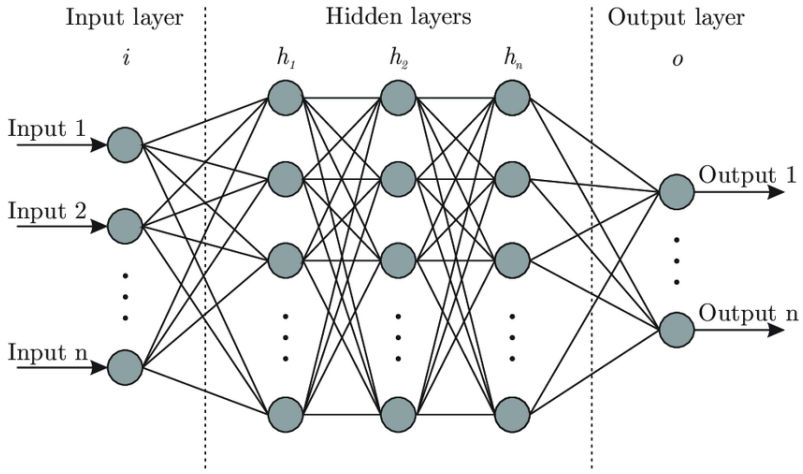


Figure 2.2: Layer structure of the neurons and their connections in a NN. Adapted from [9]

network a *feed-forward neural networks*. [8] Within G we can compute costs C_u for $u \in \mathcal{V}$ by using the neuron activation as defined in equation 2.1. To compute C_u we have to propagate data through G, to receive the input to u, which we call *forward propagation*. After computing costs at all $u \in \mathcal{V}$, we can compute the minimal costs as a discrete energy minimization problem (MAP) in between the entering and exiting node. Let $\mathcal{P}_\mathcal{V}$ be the discrete space of all paths through G, then we can compute the path with minimal total costs as follows [88]:

$$p^* = \underset{p \in \mathcal{P}_\mathcal{V}}{\operatorname{argmin}} \; [E(p, C) = \sum_{u \in \mathcal{V}} C_u(p_u)] \tag{2.3}$$

We are only interested in the minimal cost, as they depict the best possible way of navigating through G and correspond to the best possible solution. Luckily, we do not have to compute the costs at all nodes, since we are only interested in the minimal path. For this reason, we can apply algorithms like Dijkstra or Bellman-Ford to only do those calculations which are necessary to determine the arguments of the minimum. [77].

2.2 Machine learning

To understand how machines learn, it makes sense to first consider how humans learn. Let's take a visual example: A student of art history must name the art period in which a painting shown to her was created. The light reflected from the painting is absorbed by her eyes and processed by her brain. In her brain, the information is addressed by specific neurons of visual cortex and other regions, which can in total address the question for the better or worse, according to her previous practice. After coming up with a solution, she can check if her solution

was right or wrong. Depending on the answer, her brain establishes new synapses between related neurons. If an existing synapse is inefficient or no longer needed, it is degraded again. [78]

A similar process allows machines to learn: Information given to the NN is processed and generates a representation. Based on this representation, a loss is determined, which evaluates the representation. Back propagation then determines the influence of neurons each on the generated representation. An optimization algorithm uses the distributed gradient and adjusts θ accordingly. We repeat this process until we find a complex non-linear function that is able to solve the target problem. The following chapter will elaborate on the methods just discussed and explain how we train a NN.

2.2.1 Loss function

The loss function is an objective function to evaluate the output of a network. It is used to compute an error between the supposed condition and the actual condition. For this, we often compute a distance between both of them. The error will then be propagated back to the involved neurons to update their weights and biases in θ during training. For this loss, we often compute a distance between both of them. A distance can be measured by different metrics, such as the Manhattan distance (L_1) or Euclidean distance (L_2). L_1 computes the absolute distance $||.||_1$, while L_2 proceeds with the euclidean distance $||.||_2$. A norm is a mapping $\mathbb{R}^n \to \mathbb{R}$, that assigns a number to a vector which is intended to describe the size of the object in some way. With a given norm, we can measure the distance between two vectors $x, y \in \mathbb{R}^n$ by the norm of their difference vector. [33] We can only compute a distance vector for symmetrically sized vectors, which happens as follows [71]:

$$L_1(x, y) = ||x - y||_1 = \sum_{i}^{n} |x_i - y_i| \tag{2.4}$$

$$L_2(x, y) = ||x - y||_2 = \sqrt{\sum_{i}^{n} |x_i - y_i|^2} \tag{2.5}$$

Typically, L_1 is more robust than L_2 because L_2 increases the cost of outliers exponentially. However, L_2 tends to be more stable, as it uses the euclidean distance, which only has one solution. Nevertheless, L_1 is often used because it is capable of producing a sparser solution. Such solutions have many large residuals, but also a lot of zeros. This is beneficial because once a variable has a 0 coefficient, it has no impact on the model anymore. [24] As in our brain, we do not need all neurons for any task, but rather have specialized neurons to focus on specific tasks. [36]

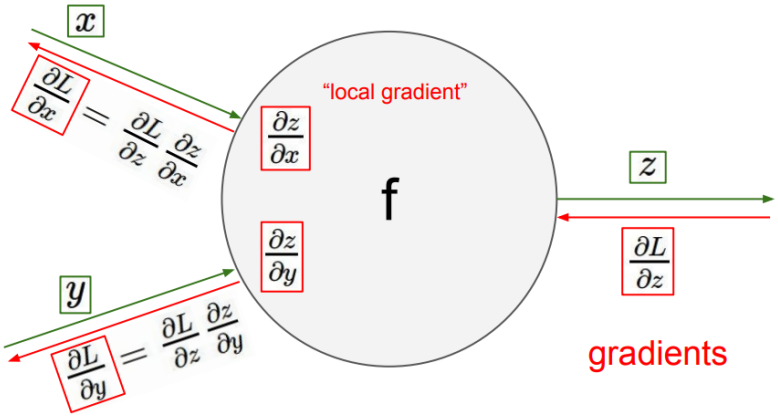


Figure 2.3: Using the chain-rule of calculus to compute the local gradient. Adapted from [51]

2.2.2 Back propagation

Back propagation is an algorithm which has been introduced by Rumelhart et al. in 1986 [81]. It is used to distribute the loss value within the NN and enables us to repeatedly adjusts θ with the target of minimizing the loss function. To determine the factor, by which we adjust θ, we need to establish a backward pass in addition to the forward pass. The backward pass essentially traverses the path chosen by the MAP with the loss value from layer L_{N-1} to L_0. Meanwhile, we compute the gradients for all encountered neurons with respect to the loss value. Such gradients are called *local gradients*. To make this computationally feasible, we use the chain rule of calculus to compute the derivative of a function by composing another function whose derivatives we already know. This process is displayed in figure 2.3. By computing the local gradient, we can express how a change in the input will result in a corresponding change in the output. After computing all gradients, we are able to perform a gradient step on θ to change weights respectively. [24]

2.2.3 Optimizing within a neural network

In this section, we want to demonstrate how to perform gradient steps within a NN. We first introduce a basic algorithm to optimize convex and differentiable functions called *Gradient Descent* and continue with a more advanced approach called *Adam*.

Gradient descent

Gradient descent is an optimization algorithm defined by Louis Augustin Cauchy, who laid its foundations in 1847 [10], that is applicable to functions which are convex and differentiable. The algorithm aims at finding a minimum of a given function $f : \mathbb{R}^n \to \mathbb{R}$. If f is convex and differentiable, it has a minimum in $x \in \mathbb{R}^n$ with

$\nabla f(x) = \overline{0}$. We can compute this point using the following iterative rule:

$$x^{t+1} = x^t - \alpha_t \nabla f(x) \tag{2.6}$$

For $t = 0, 1, \ldots, \infty$ we chose $\alpha_t > 0$, which we use as a step-size for the algorithm in equation 2.6. If α is chosen at a constant rate, it might end up oscillating around the minimum. If otherwise α is chosen at a decaying rate, it will slow down training as not only the gradient is vanishing while getting closer to the optimum, so is the step size. Therefore, we should try to find a reasonable compromise between both of them. [88]

Now, let's consider the case of optimizing a NN. Bengio el al. showed that training a NN with multiple hidden layers can be viewed as a convex optimization problem. [7] However, the hidden layers may not actually be differentiable at all points. For instance, ReLu is not differentiable at $x = 0$. In practice, this turns out not to be a problem, as the optimization usually does not arrive at the exact local minimum of the loss function. In a local minimum, the gradient would be 0, making it undefined. We do not expect our training to reach a point where the gradient is 0, making it acceptable for the minimum to correspond to points with undefined gradients. [24].

With gradient descent, we can optimize the learnable parameters θ by our loss function $\mathcal{L}$. For this, the iterative algorithm performs the following optimization step:

$$\theta^{t+1} = \theta^t - \alpha^t \nabla \mathcal{L}(\theta^t) \tag{2.7}$$

By making gradient steps on θ we can apply change weights to the computational graph and optimize it to minimize the loss function $\mathcal{L}$. Elements which have not been part of the *argmin* have zero gradients, resulting in no weight change at iteration t. [88, 24]

Adam optimization algorithm

Adam is a fairly common optimization algorithm introduced in 2014 by Kingma and Ba [45]. It combines *RMSprop* [26] and *Stochastic Gradient Descent* [79], to utilize the squared gradients while scaling the learning rate with a moving average of the gradient. In consequence, it does not use a single learning rate, but estimates the first and second moments of the gradient to compute adaptive learning rates for individual parameters. The concern of the first and second moment are mean m and uncentered variance v, respectively. An uncentered variance does not subtract the mean during its variance calculation. Both are parameterized by β' and β'' respectively, which are used as exponential decay rates for the moment estimation. Both m and v are computed based on the gradient and used to adjust the moving

averages. After computing the bias-corrected first raw moment estimation $\hat{m}$ and the bias-corrected second raw moment estimation $\hat{v}$, both can be used to update the learnable parameters θ. In this regard, a very small constant ϵ is added to the denominator to improve its numerical stability (its default value is 10^{-8}). It is also used to avoid division by zero, while updating variables, whose gradient is almost zero[45]:

$$\theta_{t+1} = \theta_t - \alpha * \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon} \tag{2.8}$$

2.2.4 (Hyper-)parameter

In order to train and improve the performance of a NN we need parameters to control its behavior and thus its performance. We differentiate between parameters which are set internally, and hyperparameter that are set externally. Internal parameters can be, for instance, the activation function or the number of layers within the NN. External hyperparameters, on the other hand, can be parameters, such as the learning rate or maximum number of training steps. While values are typically set manually, their setting can also be seen as an optimization problem itself, which can be computationally approximated. [2]

2.3 Convolutional neural networks

CNN are a form of NN, which are used to process data that has a known, grid-like topology [24]. They were introduced in 1989 by Yann LeCun [52] and are designed for working with two dimensional (2D) image data, although they can be used on one dimensional (1D) and three dimensional (3D) data alike. At its core, they are using the convolution operation instead of matrix multiplication in at least one of their layers. [24]

2.3.1 Convolution

A convolution is an operation on two real-valued functions I and K, that produces a third function C, which expresses how the shape of one is modified by the other[75]:

$$C = I \circledast K \tag{2.9}$$

In the domain of 2D images, this can be used to re-calculate the value of every pixel. Let W and H be the width and height of an image and let (x, y) denote the pixel coordinates with $x \in 1, 2, \ldots, W$ and $y \in 1, 2, \ldots, H$. Also, let I be the discrete image function and K be a weight matrix, which we will refer to as a kernel. Using the convolution of I and K, we can compute the weighted sum of neighboring

pixels in the image. Convolutions are defined by the integral of the product between I and K after having shifted and reversed K. In this regard, we use τ_u and τ_v to index pixel coordinates in I. This operation is typically denoted by an asterisk $\circledast$ [75]:

$$C(x,y) = I(x,y) \circledast K(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(\tau_u,\tau_v)K(x-\tau_u, y-\tau_v)d\tau_u d\tau_v \quad (2.10)$$

The function $K(x-\tau_u, y-\tau_v)$ is simply the image function $I(x,y)$ rotated by 180 degrees around the origin. Since calculating an integral is not trivial, we want to simplify its calculation. We can do this by transforming both integrals to discrete summations over their dimension. Since we are dealing with a finite number of pixel values, the function will be zero at all positions of τ_u and τ_v, for which we do not store a pixel value. Therefore, we are able to implement the infinite summation over a finite number of elements [75]:

$$C(x,y) = \sum_{\tau_u}\sum_{\tau_v} I(\tau_u,\tau_v)K(x-\tau_u, y-\tau_v) \quad (2.11)$$

The above formulation is applicable to 2D inputs. Colored images have a third dimension in which they store their color information, namely red, green and blue (RGB). These color channels are superimposed upon each other, meaning that every pixel value in the image has 3 values associated with it. In practice, we use the same kernel for all color channels and compute the weighted sum of equation 2.11 in each of them. [24]

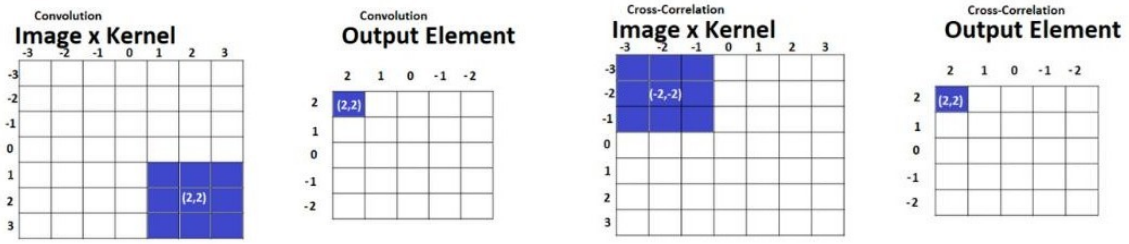
In order not to flip the kernel at every iteration, most machine learning programs use a cross-correlation [24]:

$$C(x,y) = \sum_{\tau_u}\sum_{\tau_v} I(\tau_u,\tau_v)K(x+\tau_u, y+\tau_v) \quad (2.12)$$

Both, convolution and cross-correlation, can be used to compute such an *inner* or *dot product*. Using convolutions over cross-correlations makes sense, when data is given as a time-varying signal because every signal is correlated to its successor. However, in image data there is no time reference we have to obey to. Furthermore, convolutions can cause features to not line up, if we want to match part of a larger image to a smaller one. With cross-correlation, we can visually pleasingly slide the kernel from left to right and top to bottom. We demonstrate their application differences in figure 2.4 and the computation of cross-correlations in figure 2.5. [24]

In fact, many machine learning libraries implement cross-correlation, but call it convolution on 2D images. For convenience reasons, we will continue calling it

convolution, even though we technically mean cross-correlation. [24]



(a) Convolution operation (b) Cross-correlation operation

Figure 2.4: Effect of the convolution operation and cross-correlation operation. Adopted from [13].

2.3.2 Convolutional layer

NN are able to utilize the concept of convolution by a convolutional layer. This layer computes the dot product between a kernel and an image to determine a feature map, as can be seen in figure 2.5. Within a convolutional layer, we have four parameters to control the size of the output volume [50]:

Depth: The depth determines how many kernels we arrange after each other within a convolutional layer. Any subsequent kernel takes the output of its previous kernel as its input. [50]

Kernel size: The kernel size specifies the size of the 2D weight matrix as applied in section 2.3.1. Because we want to use the kernel at different positions of the image, the kernel is intended to be smaller than the image. [50]

Stride: Stride specifies the step size, by which we move the kernel over the input sequence. Considering a stride of one, we move the kernel by one from the left to the right. When we reached the maximum on the right, we use the respective stride to slide the kernel from top to bottom. Using a bigger stride size reduces the spatial size of the feature map more strongly, as we make bigger steps sliding the kernel. [50]

Padding: Padding is used to add pixels to the border and intends to reduce the spatial reduction of the feature map. There are different options on how to add pixels, from which the most popular are zero-padding and reflection-padding. [50]

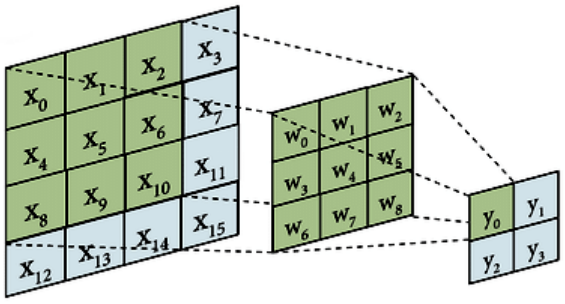


Figure 2.5: Cross-correlation example with stride 1, padding 0 and a kernel size of 3x3. From left to right, we see image I, kernel K and the output of their resulting feature map C. Adopted from [66]

2.3.3 Convolution is superior to matrix multiplication

Using convolutions instead of matrix multiplication within a NN leads to four main benefits, from which machine learning systems can benefit:

Sparse interactions: Using matrix multiplication implies that every output unit interacts with every input unit. For example, if we want to detect small features we do not need to investigate the hole input, but can rather work with the reduced filter size, such that we need fewer parameters. [24]

Parameter sharing: By using the same filter throughout the whole input sequence, we are able to share parameters. So, instead of learning a set of parameters at every location, we are able to only learn one small set of parameters. In practice, this means that as soon as the CNN learned the concept of, e.g. a dog, it can identify one at any location of the image, whereas in case of matrix multiplication, the model might only be able to identify it at a specific location of the image. [24]

Equivariant: The term *equivariant* is used to describe symmetry of one to another function. Let's consider functions f and g. f is equivariant to g if $f(g(x)) = g(f(x))$. This has the practical implication, that changes in the input are related to corresponding changes in the output. In image space, this means, that moving an object within the image corresponds to its respective representation moving in the same direction in feature space. [24]

Variable input size: Last but not least, convolutions are able to deal with different input sizes, as they only have to slide the kernel further over the image. If we on the other hand work with matrix multiplication, we are fixed to a certain input size, because both image and kernel have to be of the same size. This way, we can

train the CNN on smaller input samples, for which we yield fewer gradients than we would if we were using larger input samples. Storing gradients requires most of the resources during training. However, this property does not affect testing, because we only compute gradients during training. By skipping their calculation, we are able to use bigger input samples instead. [**cite {cs231n_CNN}**]

2.3.4 Normalization layer

Training a NN can be difficult because the learnable parameters θ are changing constantly. Since we want to establish a functional connection between those parameters, we want them to change as gradually as possible. However, even small changes amplify the deeper the network becomes. This issue can be addressed by normalizing the output of each convolutional layer. [39] For instance, we are able to normalization data using *instance* and *spectral normalization*[65]. Instance normalization [95] is used to achieve zero means and unit variances, by which we reduce the *internal covariate shift*[39]. Spectral normalization on the other hand is used to operate in continuous space, by which we re-normalize θ whenever it is updated. [65]

Instance normalization

Instance Normalization has been introduced by Ulyanov et al. [95] as a reaction to the publication of Batch Normalization [39]. Both are methods for adaptive re-parametrization and perform the same transformation, except for the number of tensors that are normalized jointly. We use those methods to filter responses of convolutional layers to reduce activation by their mean and standard deviation. Doing so regularizes the response before being given to a subsequent layer. By this, we also regularize the resulting gradient and reduce the internal covariate shift. [39] In return, we achieve a more stable distribution of gradients during training, making the network less prone to higher learning rates and weaker parameter initialization. [24, 51]

To normalize some linear activations of A in A', we use the vector μ containing the mean value of A and σ containing the standard deviation of A. We compute both μ and σ as follows [95]:

$$\mu = \frac{1}{|A|} \sum_{i=0}^{|A|} A_i \tag{2.13}$$

$$\sigma = \sqrt{\frac{1}{|A|} \sum_{i=0}^{|A|} (A_i - \mu)^2} \tag{2.14}$$

We then compute the transformation from A to A' as follows:

$$A' = \frac{A - \mu}{\sigma} \tag{2.15}$$

Spectral normalization

When training a NN, we can achieve a higher training stability if the model operates in continuous space. In continuous space, we obtain arbitrarily small changes in our output by restricting it to sufficiently small changes in its input. This property is captured by *Lipschitz continuity*. If we consider a NN as a function $f : \mathbb{R}^N \to \mathbb{R}^M$, then f is Lipschitz continuous, if there is a constant $L \geq 0$, such that for any $x, y \in \mathbb{R}^N$ it holds:

$$|f(x) - f(y)| \leq L|x - y|, \quad \forall x, y \in \mathbb{R}^N \tag{2.16}$$

The smallest value for L, which satisfies equation 2.16 is called the Lipschitz constant for f on $\mathbb{R}^N$. [88]

In the process of achieving Lipschitz continuity, Arjovsky et al. proposed to clip learnable weights in θ to a parameter $c = 0.01$, such that those weights are in $[-c, c]$ This is, as they put it, a "terrible way to enforce a Lipschitz constraint", but stuck to it due to its simplicity and performance. [4] This lead Gulrajani et al. [28] to propose a gradient penalty, which constrains the gradient norm to enforce Lipschitz continuity. However, this approach is bound to a penalty coefficient λ, that is used to weight the gradient penalty term. Finally, Miyato et al. [65] proposed applying spectral normalization to every layer of the network. This literally constrains Lipschitz continuity to each layer, such that $L = 1$ for each layer and for the whole network, which has shown to consistently perform well in practice. [98] By applying spectral normalization, we do not require any extra (hyper-)parameter tuning to achieve Lipschitz continuity. [65]

2.3.5 Convolutional blocks

A typical convolutional block consists of three parts: First, the input data is run through a convolutional layer, to produce a set of linear activations. Those activations are then normalized and lastly run through a non-linear activation function like ReLu. In practice, most CNN consist of multiple convolutional blocks, in which the number of filters is steadily increased to make up for the loss in spatial size. [24]

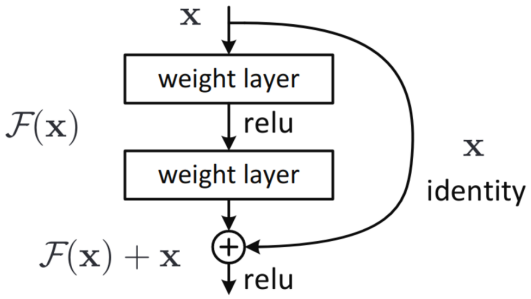


Figure 2.6: The building block of residual networks [30]

2.3.6 Residual blocks

Residual Blocks have been introduced by He et al. [30] and challenged the concept of convolutional blocks, which exclusively feeds their activations to a subsequent block. They introduced the idea, of channeling the data not only into a subsequent, but also into a deeper block. This channel is called *skip connection* or *residual connection*. The area, that is spanned by the residual connection, is called a *building block*. One residual block typically has the length of two convolutional layers, with one activation function in between. The residual block processes data by feeding it through its convolutional layers and into its skip connection. By computing an element-wise product operation between both of them, we obtain a single output. This processing is visualized in figure 2.6.

Let $\mathcal{F}$ be a building block and x be its input, then we can compute the output of the residual block as $\mathcal{H}(x) = \mathcal{F}(x) + x$. This equation can be restructured to $\mathcal{F}(x) = \mathcal{H}(x) - x$, which shows, that the residual block overall is trying to learn $\mathcal{H}(x)$, however, in consequence of the skip connection, it is actually learning the residual $\mathcal{F}(x)$.

He et al. were able to show that it is easier to train deep residual models than comparably deep models that do not have skip connections. This is due to the fact that the model was able to create an identity function for x, in which it simply sets $\mathcal{F}(x) = 0$. Such property is easy to learn and used to propagate larger gradients to the initial layers of the model. This can be used to cope with the problem of *vanishing* or *exploding gradients* and their underlying *degradation problem*.[30]

2.3.7 Pooling layer

A pooling layer is a function that replaces the output of a layer with a statistic summary. It reduces a specific region from its input to a single value, and therefore, helps the representation to become resilient to small changes in the input. Some popular options are max-pooling, average-pooling and adaptive-average-pooling. [24]

2.4 Style transfer

In the following chapter, we introduce *neural style transfer*, which is a method of image manipulation. For this, we start with its beginnings in rule-based systems. We then turn to approaches, that utilize a NN and finish by diving into adaptive instance normalization.

2.4.1 Rule-based systems

Before entering into deep learning, style transfer was performed by rule-based systems or by applying low-level statistics. In particular, methods such as non-parametric sampling, non-photorealistic rendering and image analogy have been used in the past.

Non-parametric sampling is an approach of low-level statistics, which utilizes the color histogram of a reference image to determine its probability distribution. Using this probability distribution, it sets an initial point from which it outwards grows pixels. [16]. However, this approach only replicates histographical data and does not reach any generalization in between.

Non-photorealistic rendering is a rule-based approach, that formally specifies renditions, which should be applied to an arbitrary image. Renditions are grouped into a set to describing a particular style. [90] Determining such a list seems arbitrary to some degree, since even art historians fight over style specifics. This becomes very clear when comparing the style of two artists which contributed to the same epoch to determine their own set of renditions. It's clear that this method is capable of, i.e. transforming a photo to a watercolor painting, but that it is very complex to determine a specific style by using a set of renditions. Also, such renditions have to be set up for each style individually and would not be able to simply adopt a given style. [90]

Image analogy is a multiscale auto regression, that aims at finding analogous image representations. [31] Let A be a picture of a scene and A' be an artwork of this exact scene. We then compute an image B', which relates to B like as A' relates to A. Here, we encounter the problem, that there is not a clear scene for every painting. In paintings, mountains might be moved, trees planted, and houses erased. All this would become a problem if one wanted to derive analogies from it, and might not lead to the best generalization properties to solve a rather complex issue. [96].

2.4.2 Neural style transfer

In order to address the issues listed above, Gatys et al. [22] introduced machine learning to the topic of style transfer in 2016. They wanted to transfer the style of one image to another image by using a CNN. For this, they utilized a pre-trained VGG, which is a well-established model for object-recognition. [89] Their iterative approach performs gradient steps on a content image, until it simultaneously matches its content and style representation. Although the approach by Gatys et al. is powerful in terms of its visual quality, it is a slow iterative optimization process that cannot be used to produce stylizations in real time. Their paper has triggered many subsequent studies trying to increase its computational speed, visual quality, style diversity [80] and intersections between multiple styles.

Computational speed: Johnson et al. [42] increased the computational speed by implementing a feed-forward network. They used a perceptual loss function to train an auto-encoder network, in which they encode an image to feature space and decode it to RGB.

Visual quality: The visual quality of a stylization is determined by the network ability to preserve the global structures of a content image and adopting the local structures of the chosen artistic style. Some artistic styles disregard content, others try to recreate it in a specific manner. Sanakoyeu et al. [86] and Kotovenko et al. [47] were able to increase the visual quality by using a GAN architecture and a specialized loss to learn a smooth space of style representations that display continuous transitions in between different styles.

Style diversity: Lastly, style diversity wants to access different style as easy as possible, with the goal to transfer an arbitrary style. Li et al. [55] introduced a method that directly transforms the feature map of a target image by a style reference through an approach called whitening and coloring transform (WCT). Simultaneously, Huang and Belongie [37] pursued a different approach, which aligns the mean and variance of the content features with the style features, which they called adaptive instance normalization (AdaIN) (we introduce this approach in section 2.4.2). Both approaches are applicable for arbitrary style transfer, which means that they do not need to train on a specific style to use it during stylization.

Intersecting between styles has also been achieved by Huang and Belongie [37], Kotovenko et al. [47] and Liu et al. [58]. Huang and Belongie [37] provided an early work, which was able to intersect between different styles using NST. However, their approach was still riddled with many stylistic inconsistencies and needed some

further development. In this regard, the approaches presented by Kotovenko et al. [47] and Liu et al. [58] are more advanced and determine better stylization results. Kotovenko et al. obtained a smooth style space and were therefore, able to interpolate across different styles. Meanwhile, Liu et al. [58] were able to interpolate in between styles, while performing an image-to-image translation. Such translation morphs an image from one to another class and is therefore, closely related to NST. Liu et al. averaged multiple style representations into one, which is then used during stylization. The two approaches differ in the fact that Kotovenko et al. were able to combine styles from different artists or data classes, while Liu et al. only averaged the style of identical data classes.

Related work

The ability of performing a NST can be a useful tool for many other tasks unrelated to artistic image stylization. Style transfer has also been used by Chidambaram and Qi [11] to imitate the style of chess player Mikhail Tal on a chess bord by stylizing it with a move. Luo et al. used style transfer for image restoration [60], whereas Liang et al. [56] used it to reduce color variations in histopathological images. Other approaches within the medical domain like Ma et al. [61] applied style transfer to 3D cardiac magnetic resonance (MR) images to stylize their brightness, contrast, texture, etc. Regateiro and Boyer [76] were able to modify the 3D shape of a moving character by an arbitrary static source character using NST. Lastly, Grinstein et al. [27] were able to stylize 1D audio waves by a target class. What all the examples above have in common is that they perform a transformation from one to another domain. Knowledge about the target domains give guidance to the process of learning a metric, which is able to perform the transformation. This is easier than making the model find the desired manipulation task by itself.

Adaptive instance normalization

AdaIN is a normalization layer for arbitrary style transfer, which has been introduced by Huang and Belongie [37]. To apply AdaIN, one needs two models, a CNN, which encodes images to feature space, and a model that decodes a representation back to RGB space. The authors use two images, a content image x and a style image y, which they forward to the CNN to receive their encoded feature representations $\hat{x}$ and $\hat{y}$. As an intermediate step, they then apply the AdaIN normalization layer to align mean and variance of $\hat{x}$ by $\hat{y}$ and to compute feature f [37]:

$$\text{AdaIN}(\hat{x}, \hat{y}) = \sigma(\hat{y}) \ \frac{\hat{x} - \mu(\hat{x})}{\sigma(\hat{x})} + \mu(\hat{y}) = f \tag{2.17}$$

Afterwards, they use a second model to invert f to RGB space. However, AdaIN can be applied as part of the inverting process itself, as demonstrated by Huang et al. [38]. Within the decoding model, they use AdaIN as a normalization layer, allowing them to perform multiple normalization at different stages of the model compared to just one intermediate transformation. [58]

2.5 Generative adversarial networks

In this chapter, we want to introduce generative adversarial networks. [25] GAN are networks of two competing NN which both maximize their own payoff. To understand their behavior, we first present their theoretical principles in game theory. Subsequently, we merge the approach of NN and game theory to develop GAN from this. A well-known problem of training GAN is that they have difficulties to converge. For this reason, we present stabilization techniques that can be applied to deal with this problem. Another issue is that we can not use a single objective function, which the network optimizes. GAN are trained by reaching an equilibrium between both models. With this obstacle, it is difficult to measure GAN performance, since reaching an equilibrium does not mean having a good solution to fulfill the related task. Therefore, we can only approximate its performance. In this regard, we include a final section on methods to estimate GAN performance. [23]

2.5.1 Game-theoretical principles

Game Theory is a branch of economics which has been introduced to the scientific literature in 1944 by von Neumann and Morgenstern. [69] It studies the strategic interaction among rational decision-makers. By using Game Theory, we can formalize the interaction between two NN. In this regard, we will define a strictly competitive two-player game with zero-sum, which reaches an *Nash equilibrium*.

Two-player games

A two-player game is a strategic interaction among two participants, which act as rational decision-makers. A game can be considered strategic, if both players evaluate their own, non-empty set of actions A_i once and for all and simultaneously choose their action. Every action yields a payoff, which the player who chose the action pays to its counterpart. This payoff can be computed using payoff function $v_i : A \to \mathbb{R}$, where $A = A_1 \text{ x } A_2$. Let such interaction be defined as $\mathcal{I} = \langle (A_i), v_i, \rangle_{i=1,2}$. Finally, if A_i is finite for both players, then the game can be considered finite as well. [6, 72]

Zero-sum games

In a zero-sum game of two players, the summation of both players' preferences is zero, resulting in no preference, such that:

$$v_1 = -v_2 \tag{2.18}$$

Both players are said to *maxminimize* if they choose an action which maximizes the payoff they receive, but minimize the payoff they have to pay to their counterpart. An action $a_1^* \in A$ can be considered a *maxminimizer* for player 1 if the following holds:

$$\min_{a_2 \in A_2} \; v_1(a_1^*, a_2) \geq \min_{a_2 \in A_2} \; v_1(a_1, a_2) \;\; \forall a_1 \in A_1 \tag{2.19}$$

Similarly, $a_2^* \in A$ can be considered a *maxminimizer* for player 2 if the following holds:

$$\min_{a_1 \in A_1} \; v_1(a_1, a_2^*) \geq \min_{a_1 \in A_1} \; v_1(a_1, a_2) \;\; \forall a_2 \in A_2 \tag{2.20}$$

Since both players do not only maximize the payoff they receive, but also minimize the payoff they pay to the other player, both can not cooperate and play a strictly competitive game. [72]

Nash equilibrium

The *Nash Equilibrium* [68] captures a steady state of $\mathcal{I}$, in which both players hold the correct expectation about the other player's behavior and acts rationally, such that no player can do better by unilaterally changing its strategy. Both players choose actions which maxminimize, but can not do better than playing a zero-sum game in which the net exchange is 0. [72] In a Nash Equilibrium, we have $a^* \in A$ which holds the property of equation 2.21 for player 1 and equation 2.22 for player 2 [6]:

$$v_1(a_1^*, a_2^*) \geq v_1(a_1, a_2^*), \;\; \forall a_1 \in A_1 \tag{2.21}$$

$$v_2(a_1^*, a_2^*) \geq v_2(a_1^*, a_2), \;\; \forall a_2 \in A_2 \tag{2.22}$$

2.5.2 Adversarial modeling framework

GAN have been introduced by Goodfellow et al. in 2014 [25], who defined them as a framework which consists of a generator and an adversary that compete with each other. Their approach is based on the game theoretical scenario of a two-player game with zero-sum playing a strictly competetive game. Player 1 is a generative

model that attempts to represent its input data in RGB image space. Its input can be random noise. However, in the case of NST it will be a content and a style image, which $\mathcal{G}$ wants to unite. The content image is sampled from content dataset $\mathcal{C}$, while the style image is sampled from style dataset $\mathcal{S}$. Player 2 is a discriminative model $\mathcal{D}$ that attempts to distinguish samples drawn from $\mathcal{S}$ and $\mathcal{G}$. Subsequently, we can use those classification results to regularize both, $\mathcal{G}$ and $\mathcal{D}$. This process is depicted in figure 2.7.

As both play a zero-sum game, we can formulate learning as a payoff function v, in which: $v_1 = -v(\mathcal{G}, \mathcal{D})$ and $v_2 = v(\mathcal{G}, \mathcal{D})$, such that they respect equation 2.18. As they play a strictly competitive game with each other, both players want to maximize their own payoff [24]:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} v(\mathcal{G}, \mathcal{D}) \tag{2.23}$$

The minimax theorem published by Fan [19] states that if v is convex on $\mathcal{G}$ and v is concave on $\mathcal{D}$, then both are interchangeable as follows:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} v(\mathcal{G}, \mathcal{D}) = \max_{\mathcal{D}} \min_{\mathcal{G}} v(\mathcal{G}, \mathcal{D}) \tag{2.24}$$

If this condition holds, we can show that an optimal solution exists, and that it coincides with a Nash Equilibrium of a zero-sum game. [6, 83] Such point optimally solves equation 2.23 and we can compute $\mathcal{G}^*$ as follows:

$$\mathcal{G}^* = \operatorname*{argmin}_{\mathcal{G}} \max_{\mathcal{D}} v(\mathcal{G}, \mathcal{D}). \tag{2.25}$$

Now, let's specify an appropriate payoff function $v(\mathcal{G}, \mathcal{D})$, which we adopt from Goodfellow et al. [24]:

$$v(\mathcal{G}, \mathcal{D}) = \mathop{\mathbb{E}}_{y \in \mathcal{S}} [\log \mathcal{D}(y)] + \mathop{\mathbb{E}}_{\substack{(y) \in \mathcal{S} \\ x \in \mathcal{C}}} [\log (1 - \mathcal{D}(\mathcal{G}(x, y)))] \tag{2.26}$$

By equation 2.26 we channel $\mathcal{G}$ to generate samples, which $\mathcal{D}$ classifies as real. While $\mathcal{D}$ is driven to correctly classify samples as "real" or "fake". At convergence, $\mathcal{D}$ maximizes its discriminative nature, whereas $\mathcal{G}$ finds the optimal way to cope with it, such that the samples of $\mathcal{G}$ are indistinguishable from real data. [23]

By minimizing the probability of $\mathcal{D}$ being correct as a target of $\mathcal{G}$ a network might be suffering from the vanishing gradients problem (see section 4.2.4), as Goodfellow indicated. [25] Therefore, he suggested maximizing the probability of $\mathcal{D}$ being mistaken, which can be computed as follows:

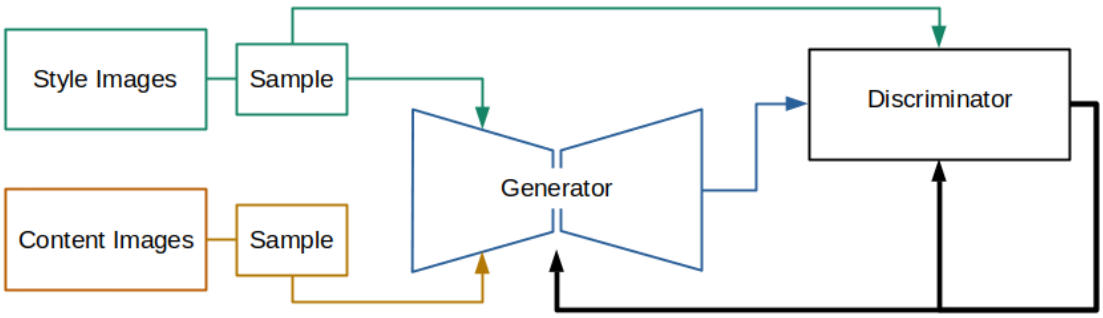


Figure 2.7: Interaction between $\mathcal{G}$ and $\mathcal{D}$ while performing a style transfer.

$$v(\mathcal{G},\mathcal{D}) = \mathop{\mathbb{E}}_{y\in\mathcal{S}} [-\log\ \mathcal{D}(y)] + \mathop{\mathbb{E}}_{\substack{y\in\mathcal{S}\\x\in\mathcal{C}}} [\log\ (1-\mathcal{D}(\mathcal{G}(x,y))] \tag{2.27}$$

In a two-player game, both players are required to choose their action simultaneously, as indicated in section 2.5.1. While training GAN, this step happens in a row which does not constrain a problem, as both models can not "watch" each other. Also, we only perform gradient steps for one model at a time, meaning, that we only compute gradients for one model. After updating one model, we use the same training data to update the other model. We do this to reduce the overall weight changes to learnable parameters in θ and to stabilize training. [25, 23]

Remark 1 *Unfortunately, we can not guarantee that a Nash equilibrium exists. This is because v in general fails to be convex in the first and concave in the second argument. [83, 6, 24] Analogous to non-convex optimization, we focus on finding a local equilibrium, as a tractable solution for non-convex games. In such local equilibrium, both players do not have much of an incentive to switch their strategy. By converging to such local equilibrium, we may encounter failures, such as the mode collapse (see section 4.2.2) or oscillates from one to another mode, which can be the result of a bad local equilibrium. [46]*

Generator

$\mathcal{G}$ is a generative model that uses its input to create an output at the same dimensionality. In our case, this dimensionality is 3 x W x H pixels. To reach the desired state, $\mathcal{G}$ is often trained using further loss functions, which channel its convergence. To reach this state, $\mathcal{G}$ maximizes the probability of $\mathcal{D}$ making a mistake. Because of this, it generates samples, which show identical features, to those drawn from training data. At convergence, the samples generated by $\mathcal{G}$ are indistinguishable from those of $\mathcal{S}$. [23, 24]

Discriminator

$\mathcal{D}$ is a classifier, stacked on a CNN, which attempts to distinguish between samples drawn from training data and such generated by $\mathcal{G}$. The classification is performed by determining a scalar value from its input. Depending on which optimization method is used, we can derive further information from the scalar. $\mathcal{D}$ is trained to maximize the probability of assigning the correct label to its input. At convergence, $\mathcal{D}$ outputs the same scalar value for all samples, as it can not differentiate between them. At this stage, $\mathcal{D}$ and $\mathcal{G}$ reached a Nash equilibrium because both cannot increase their payoff by unilaterally changing their strategy. [23, 24]

2.5.3 Stabilization techniques

A key problem when it comes to training GAN is the risk that the model fails to converge. This can be attributed to the fact that GAN do not minimize a single objective loss function, but to find an equilibrium between the generator and discriminator. [23] Besides the general model architecture, there are further possibilities to stabilize the training and to enable convergence. In the following, we want to highlight some of these possibilities. First, we discuss *Differentiable Augmentations*, which is a method that can be used to work with smaller datasets. We continue with the concept of using unbalanced update ratios for $\mathcal{G}$ and $\mathcal{D}$ to mitigate learning speed differences. Lastly, we raise the possibility of using historic averaging on the learnable parameters of $\mathcal{G}$ to average weight changes in $\mathcal{G}$.

Differentiable augmentation

Differentiable Augmentation has been introduced by Zhao et al. [99] to encounter the problem of limited training data. When training data is limited, $\mathcal{D}$ might be able to memorize the exact training data, which is called *overfitting*. In consequence, the gradient for $\mathcal{D}$ vanishes, while those of $\mathcal{G}$ explode.

A widely used strategy to mitigate this issue is to apply data augmentations to the training data. By scaling brightness, contrast, saturation and hue of an image, we can use it more often and extract more information than we would if trained on the unchanged image. By this, we add artifacts to the image, which diversifies the available information. However, since those artifacts are added before entering $\mathcal{G}$, they might be learned and become parts of the probability distribution. Zhao et al. [99] were able to show this behaviour and demonstrated their harm on translation accuracy. If augmentations are only added while training $\mathcal{D}$, the results become even worse, as $\mathcal{G}$ and $\mathcal{D}$ optimize different objectives.

Zhao et al. combated the issue of data augmentation by augmenting both the style and generated sample, right before entering into $\mathcal{D}$. Because of back-propagation,

Figure 2.8: Differentiable Augmentations applied from left (original painting) to right (*color, translation, cutout*)

those augmentations have to be differentiable. First, they adjusted the color diversity by randomly scaling brightness, contrast and saturation (denoted as *color*). Then, they randomly shifted the image with respect to its width and height and pad the missing pixels with zeros. (denoted as *translation*) Lastly, they masked a random part of the image with respect to its width and height and pad the missing pixels with zeros. (denoted as *cutout*). We demonstrate those augmentations in figure 2.8. By using this augmentation strategy, the authors were able to match state-of-the-art results while only using 20% of the available training data. [99]

Unbalanced updates

When building a GAN, one wants it to be balanced between $\mathcal{G}$ and $\mathcal{D}$. If the training progress of one model is happening much faster than it is for the other, the inferior will perish. Especially in regularized networks, this might be an issue and cause training instabilities. This problem has been addressed by Miyato et al. [65], who proposed to balance how many gradient steps $\mathcal{G}$ and $\mathcal{D}$ make each training iteration. By performing multiple gradient steps each training iteration, the training takes longer, than it would by training on a 1:1 ratio. Heusel et al. [32] addressed this issue by proposing to train on different learning rates, which they called time-scale update rule (TTUR). Zhang et al. showed that both approaches produce similar results, but find training on a 1:1 ratio more desirable, as it takes less computational resources. Meanwhile, Sanakoyeu at al. [86] proposed using a winning rate of $\mathcal{D}$ to determine the model for which they perform gradient steps. A winning rates $\delta \in [0, 1]$ is a barrier, that is contested by an averaged accuracy p of $\mathcal{D}$. If $p \geq \delta$, they perform a gradient step for $\mathcal{G}$ and update $\mathcal{D}$ in all other cases. To compute p at time $t + 1$ we initially set $p_0 = \delta$ and use the current accuracy a_t, to compute p_{t+1} as follows [86]:

$$p_{t+1} = 0.99 * p_t + 0.01 * a_t \tag{2.28}$$

Historic averaging

Historic averaging is a technique related to ensemble learning. In ensemble learning, we reduce the variance of a model by building an ensemble of models, which are utilized to create the final output. The ensemble we use for historic averaging is a deep copy of our model, that adjusts its weights by a moving average after each training iteration. The averaged deep copy is then used during testing. [53] Historical averaging makes the model less prone to rampant loss shifts or irregular data representations because weight changes are averaged during training time. Historic averaging also improves the numerical stability of the averaged model because it regularizes weight changes and levels out fluctuations. [91] The latter leads to a wider optimum and better generalization properties, as shown by Izmailov et al. [41]. However, as the deep copy has to be available during training, we also need more computational resources.

2.5.4 Evaluate performance

The fact, that GAN do not having a single objective loss function has broader implications, than creating instability between its competing models. It also makes it difficult to measure its training performance. By remark 1 we already indicated that we can not guarantee the existence of a Nash equilibrium. Therefore, we have to consider a local equilibrium, as tractable solutions. Hence, we might find an equilibrium, but we can not guarantee that $\mathcal{G}$ produces the desired results. An equilibrium can be just as balanced between two strong counterparts as it can be between two weak ones. Salimans et al. [85] suggest judging the visual quality of training samples to get an idea of how the training is progressing. This can happen by human selection or through a metrics like FID [32] or SIFID [80]. We can also examine $\mathcal{G}$ to learn more about its ability to determine style representations. Such representations can be analyzed by their topology using t-distributed stochastic neighbor embedding (t-SNE)[63] to determine whether paintings from the same artists also have similar representations in feature space.

Fréchet inception distance

The Fréchet inception distance is a metric to evaluate generated images introduced by Heusel et al. [32] The metric compares a set of generated samples F to a set of real samples T to compute their similarity score. To compute their similarity, all images are encoded using the Inception V3 model by Szegedy et al. [92] to capture computer-vision specific features. Those features can be summarized as a multivariate Gaussian distribution, through their mean μ and covariance σ. [14] Both can be computed as described in equation 2.13 and 2.14. Afterwards, the

Fréchet inception distance (FID) score can be computed by the squared Fréchet distance d as shown by Dowson and Landau [15]:

$$d^2 = |\mu_T - \mu_F|^2 + trace(\sigma_T + \sigma_F - 2(\sigma_T\sigma_F)^{\frac{1}{2}}) \tag{2.29}$$

A major drawback to FID is its sample size. Some authors use 50,000 [43, 44], while others use 10,000 [59, 94] images. [12] We might only be able to use FID to rate the image quality after training a model, since creating and processing 50,000 will substantially delay the training. Chong and Forsyth [12] argue that the number of samples $\mathcal{N}$ is related to a bias in FID. This bias vanishes for $\mathcal{N} \to \infty$. If we choose $\mathcal{N}$ too small, we increase the variance in FID.

Single image fréchet inception distance

single image Fréchet Inception Distance has been introduced by Shaham et al. [80] as a method to measure image similarity in case where we only have a single image. It uses the feature distribution of two images right before applying the second pooling layer of the Inception v3 model [92] from which they compute the Fréchet distance as in section 2.5.4. The authors used single image Fréchet Inception Distance (SIFID) to calculate individual scores over a list of images and subsequently averaged those scores. Since this score does not rely on a high $\mathcal{N}$, we can also compute it during training.[80]

t-distributed stochastic neighbor embedding

t-distributed stochastic neighbor embedding is a non-linear dimensionality reduction technique introduced by Van der Maaten and Hinton. [63] It is used to represent high-dimensional data in a lower dimension, most commonly in 2D or 3D. To perform such dimensionality reductions, the algorithm first assigns a probability distribution over pairs of high-dimensional vectors such that similar objects have a higher probability, while dissimilar objects have a lower probability. The iterative algorithm then minimizes the Kullback–Leibler divergence with respect to the locations of the points in their target space. As a result, we obtain a low-dimensional representation of our high-dimensional data. [63]

2.6 Summary

In this chapter, we introduced neural networks and demonstrated how they are able to learn. We use those methods to set up the model architecture in chapter 3.1. Afterwards, we introduced the idea of neural style transfer, which is the main concern of this work. Lastly, we specified a generative adversarial networks architecture.

Here, we introduced various stabilization techniques, which we are going to analyze in chapter 4 to cure training instabilities. Furthermore, we presented performance approximations, which we utilize while selecting the most promising stabilization techniques and evaluating our training performance.

3 Methods

In the previous chapter, we introduced the theoretical background in machine learning, neural style transfer and generative adversarial networks.

Now, we present the network, loss function and approach we take to perform the neural style transfer. For this, we first decompose the network into its inner models and specify their data processing. Afterwards, we present the loss function and indicate deviations from Saito et al. [84] Lastly, we specify how we train those models.

3.1 Model

Our network adopts the architecture provided by Saito et al. [84] and consists of a Few-shot Image Generator $\mathcal{G}$ and a Patch GAN Disciminator $\mathcal{D}$. The two models play a strictly-competitive game against each other.

3.1.1 Few-shot image generator with content conditioned style encoder

The few-shot image generator with content conditioned style encoder (COCO) has been introduced by Saito et al. [84] and is based on the few-shot image generator (FUNIT) by Liu et al. [58]. Saito et al. argue that FUNIT suffers from a content loss problem. Accoording to them, the generator fails to preserve domain invariant content information, when content and style are not aligned with each other. They propose to condition the style on the content feature, such that both are consistent with each other. As the content loss problem only becomes an issue if content and style are not aligned with each other, they can use this approach to circumvent the problem. [84] Their generator $\mathcal{G}$ contains a content encoder E_c, a content conditioned style encoder E_s and a decoder D, as can be seen in figure 3.1.

Content encoder: E_c is the model, which they use to encode content images. They first utilize a convolutional block to map the color channels to a latent dimensionality of 64, but leave the spatial size untouched. Afterwards, they apply four convolutional blocks, which double the latent dimensions and bisect the spatial size

of E_c. Subsequently, they add two residual blocks, that hold on to the spatial size and latent dimensions. Therefore, if we feed an image at size 3 x 256 x 256 to E_c, then E_c will return a vector of size 1024 x 16 x 16.

Style encoder: E_s is the model which they use to encode style images. They also utilize a convolutional block to transform its color channels to a dimensionality of 64 and leave the spatial size unchanged. Afterwards, they apply two convolutional blocks which double its latent dimensions and bisect its spatial size. Hereafter, they add three convolutional blocks which keep the latent dimensionality, but bisect its spatial size. Its grid of activation is then be mean pooled, making it reach a spatial size of 1. Lastly, its latent dimensionality is mapped to 64 dimensions, without changing its spatial size. Therefore, if we feed an image at size 3 x 256 x 256 to E_s, then E_s will return a vector of size 64 x 1 x 1.

Decoder: D is the model which they use to decode the encoded content image, while normalizing it with the AdaIN parameters of the encoded style image. This normalization happens within its first two residual blocks that do not change the latent dimensionality or spatial size. Afterwards, they apply four convolutional blocks, which double its spatial size and bisect its latent dimensions. Finally, another convolutional layer is added, which keeps the spatial size, but reduces the latent dimensionality to 3. Therefore, if we feed a content feature representation at size 1024 x 16 x 16 and a style feature representation at size 64 x 1 x 1 to D, then D will return a vector of size 3 x 256 x 256, which corresponds to RGB space.

As a key element of COCO, Saito et al. [84] want to align the style by the content to circumvent the content loss problem of FUNIT. They therefore mean pool the feature representation of E_c to reach a spatial size 1 and map it to a vector ζ_c, with 64 dimensions. Simultaneously, they concatenate the feature representation of E_s with a constant style bias at a dimensionality of 1024. The concatenation is mapped it to a vector ζ_s, with 64 dimensions. As both, ζ_c and ζ_s have the same dimensionality, their mappings can be combined by an element-wise product operation. Finally, their result is mapped to a vector ζ_{AdaIN} with 256 dimensions, that provide the AdaIN parameters to D.

This architecture is further visualized in figure 3.1. [84, 58] To generate a stylized image, we encode content image x and style image y, condition style by content and decode the stylized image $\overline{x}$:

$$\overline{x} = \mathcal{G}(x, y) = D(E_c(x), E_s(y)) \tag{3.1}$$

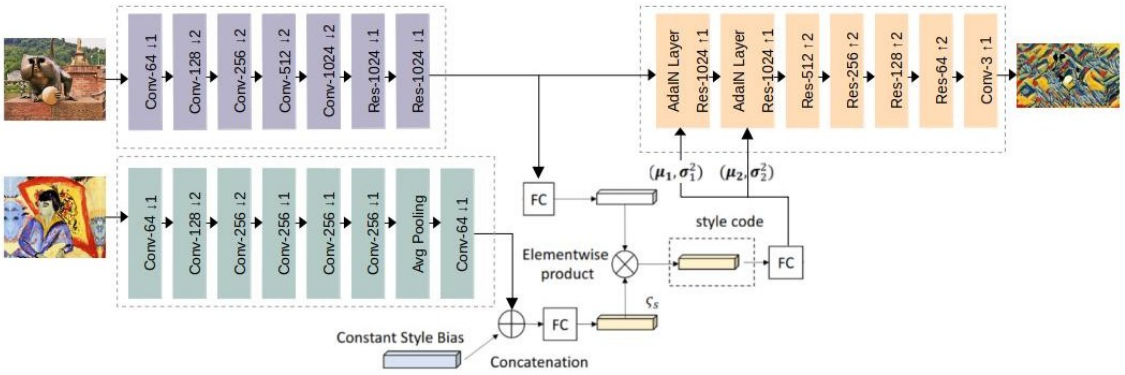


Figure 3.1: Model architecture of COCO-FUNIT. Modified from Saito et al. [84] and Liu et al. [58]

3.1.2 Patch GAN discriminator

The Patch GAN discriminator $\mathcal{D}$ is a special form of a CNN, which processes individual patches of an image identically, but independently. The difference to a regular discriminator is that it does not map an image to a single scalar output. It rather produces a 2D matrix of scalars which are the classification results of their respective patch. By this property, $\mathcal{D}$ effectively considers the image as a Markov random field. [40] The discriminator used by Saito et al. [84] is constructed from two incremental parts, a CNN $\mathcal{D}_m$ and a classifier $\mathcal{D}_c$.

CNN: Within $\mathcal{D}_m$ the authors use spectral normalization, as this has shown to stabilize the training of a discriminator. [65] $\mathcal{D}_m$ first utilizes a convolutional block to map the color channels to a latent dimensionality of 64, but leaves the spatial size untouched. Now, $\mathcal{D}_m$ is enhanced with further layers, of which each has two residual blocks. The first residual block upholds, while the second doubles its latent dimensionality. After passing through those layers, the output is padded with reflection padding and pooled using average pooling to make the model invariant to local translations and to bisect the spatial size. They add six of those layers, but double the latent dimensionality only at its first three, such that the model reaches a latent dimensionality of 512. They do not add padding or pooling to the last layer because otherwise they would summarize the data, which is not helpful to a classifier. Therefore, if we feed an image at size 3 x 256 x 256 to $\mathcal{D}_m$, then $\mathcal{D}_m$ will return a vector of size 512 x 16 x 16.

Classifier: $\mathcal{D}_c$ processes the output of $\mathcal{D}_m$ with a convolutional block that reduces the latent dimensionality to 1. The model architecture is further visualized in figure 3.2. Because they use paintings of different artists, they are able to condition $\mathcal{D}$ with an artist label l, by which they can direct the classification process. [64] To enable

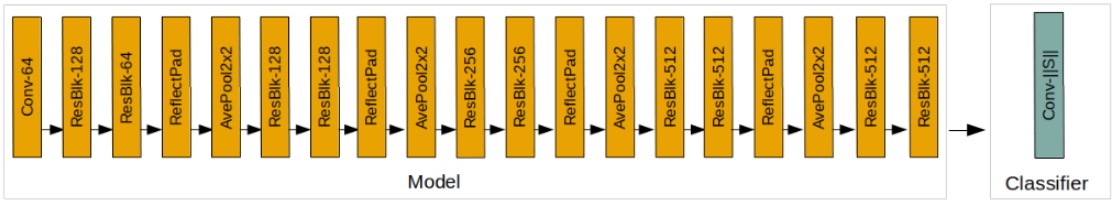


Figure 3.2: Visualization of $\mathcal{D}$ architecture. The data is fed from the first to the last layer of the model and afterwards classified by the classifier, that maps the classification result to class labels.

this processing, they feed l into a lookup table f where they get a fix label embedding at 512 latent dimensions. They then calculate a mean over the activation maps of $\mathcal{D}_m$, which is denoted as $\mathcal{D}_m^{1x1}$. As $\mathcal{D}_m^{1x1}$ and $f(l)$ now have the same dimensionality, they can combine them by multiplication and summation. Adding this value to the classification result gives us control over the classification output. [84, 58] Now, let's consider an image x and a label l, then we can compute the classification result as follows:

$$\mathcal{D}(x, l) = \mathcal{D}_c(\mathcal{D}_m(x)) + \sum \mathcal{D}_m^{1x1}(x) * f(l) \tag{3.2}$$

Therefore, if we feed a feature representation by $\mathcal{D}_m$ at size 512 x 16 x 16 to $\mathcal{D}_c$, then $\mathcal{D}_c$ will return a vector of size 1 x 16 x 16. We denote the logit produced by $\mathcal{D}(x, l)$ as ϕ_{true}, if the logit was obtained using training data and ϕ_{fake} if its source was generated data.

3.2 Loss funtion

In this section, we present the loss function $\mathcal{L}$ by which we channel the convergence of $\mathcal{G}$. As we want to reduce training instabilities, we train $\mathcal{G}$ and $\mathcal{D}$ apart from each other, but with counterpart regularization. [24] In practice, this means that we only compute gradients for the focus model. As we also use the Hinge formulation [98, 57] of the GAN loss, we can not use a single loss function, as being done in equation 2.27, since $\mathcal{L}_{GAN}^{\mathcal{G}}$ and $\mathcal{L}_{GAN}^{\mathcal{D}}$ are computed differently. Using the Hinge loss does have stabilizing effects to our GAN training, as reported by [98, 57, 58]. We compute $\mathcal{L}$ as follows:

$$\mathcal{L} = \min_G \max_D \mathcal{L}_{\mathcal{G}}(\mathcal{G}, \mathcal{D}) + \mathcal{L}_{\mathcal{D}}(\mathcal{G}, \mathcal{D}) \tag{3.3}$$

In this formulation, we combine both objective functions $\mathcal{L}_{\mathcal{G}}$ and $\mathcal{L}_{\mathcal{D}}$ into one, which we use as an objective function. In both functions, we use respective λ values to control their impact on the optimization goal:

$$\mathcal{L}_{\mathcal{G}}(\mathcal{G},\mathcal{D}) = \lambda_{GAN} * \mathcal{L}^{\mathcal{G}}_{GAN}(\mathcal{G},\mathcal{D}) + \lambda_R * \mathcal{L}_R(\mathcal{G}) + \lambda_F * \mathcal{L}_F(\mathcal{G}) \tag{3.4}$$

$$\mathcal{L}_{\mathcal{D}}(\mathcal{G},\mathcal{D}) = \lambda_{GAN} * \mathcal{L}^{\mathcal{D}}_{GAN}(\mathcal{G},\mathcal{D}) \tag{3.5}$$

In the following, we introduce the individual parts of $\mathcal{L}$. We first introduce the Hinge Loss ($\mathcal{L}_{GAN}$), which we use to compute the GAN loss. Then, we channel the convergence of $\mathcal{G}$ by introducing the feature matching loss ($\mathcal{L}_F$) and the reconstruction loss ($\mathcal{L}_R$). Because we want to achieve sparse representations, in which we switch style features on and off, we use L_1 (see section 2.2.1) to compute distances during our loss computation.

3.2.1 Hinge GAN loss

The GAN loss was introduced by Goodfellow et al. [25] as a method to penalize $\mathcal{D}$ for misclassifying samples and to create a feedback loop to $\mathcal{G}$. We use the Hinge version of this loss, which has been introduced by Lim and Ye. [57] This loss has demonstrated to be more stable and less affected by mode collapse than the one introduced by Goodfellow et al. The Hinge loss itself is a function which approximates the loss with a convex function. If the scalar value y was yielded from a sample drawn from style dataset $\mathcal{S}$, then $t = 1$, whereas $t = -1$ if it was generated by $\mathcal{G}$. The Hinge loss is defined as a maximization problem, which we can turn into a minimization problem:

$$\max(0, 1 - t * y) = -\min(0,\ \ t * y - 1) \tag{3.6}$$

To generate new samples $\mathcal{G}$ utilizes $\mathcal{S}$, as well as the content dataset $\mathcal{C}$. Using this notation, we can adjust equation 2.27 accordingly [98]:

$$\mathcal{L}^{\mathcal{D}}_{GAN}(\mathcal{G},\mathcal{D}) = \mathop{\mathbb{E}}_{(y,l)\in\mathcal{S}} [-\min(0,\ \ \mathcal{D}(y,l) - 1)] + \mathop{\mathbb{E}}_{\substack{(y,l)\in\mathcal{S} \\ x\in\mathcal{C}}} [-\min(0, -\mathcal{D}(\mathcal{G}(x,y),l) - 1)] \tag{3.7}$$

As defined by Zhang et al. [98] we use equation 3.6 to compute the loss while updating $\mathcal{D}$ and the first part of equation 2.27 to update $\mathcal{G}$:

$$\mathcal{L}^{\mathcal{G}}_{GAN}(\mathcal{G},\mathcal{D}) = \mathop{\mathbb{E}}_{\substack{(y,l)\in\mathcal{S} \\ x\in\mathcal{C}}} [-log\ \ \mathcal{D}(\mathcal{G}(x,y),l)] \tag{3.8}$$

3.2.2 Feature matching loss

The feature matching loss regularizes $\mathcal{G}$ with the feature response of $\mathcal{D}_m$. It requires $\mathcal{G}$ to generate samples which trigger a feature representation that is similar to the feature representation of its style sample. As the sole purpose of $\mathcal{D}$ is to distinguish real and generated data, one would expect D_m to yield distant feature representations for both. Naturally, $\mathcal{D}$ wants to find features that are most discriminative of both cases. If both feature representations are similar, then $\mathcal{G}$ generated data that matches the probability distribution of the real data in $\mathcal{D}_m$. We compute the feature matching loss as follows [85]:

$$\mathcal{L}_F(\mathcal{G}) = \mathop{\mathbb{E}}_{\substack{(y,l)\in\mathcal{S} \\ x\in\mathcal{C}}} [||\mathcal{D}_m(y) - \mathcal{D}_m(\mathcal{G}(x,y)||_1] \tag{3.9}$$

3.2.3 Pixel reconstruction loss

The pixel reconstruction loss is a loss that measures pixel distances in between images. Those images are a content image x and the image that $\mathcal{G}$ generates, when x is fed to E_c and E_s. The intuition behind this loss is that $\mathcal{G}$ should be able to apply style information to some content features and produce an output that resembles both. Adjusting an image by itself should change its pixel in a way that it resembles itself, in other words, no changes should be visible. We use this to guarantee that the information is transmitted, but to not constrain its transmission to a certain model. We compute the pixel reconstruction loss as follows [84]:

$$\mathcal{L}_R(\mathcal{G}) = \mathop{\mathbb{E}}_{x\in\mathcal{C}} [||x - \mathcal{G}(x,x)||_1] \tag{3.10}$$

3.2.4 Classification of reconstructed images

Focusing on the task of image-to-image translation, Saito et al. were able to operate on a single dataset for both content and style. In image-to-image translation, one translates an image of one class to another class. Saito et al. could translate a sheep to a dog and a dog to a sheep. By operating on a single dataset, all data classes are known to $\mathcal{D}$, which is why they were also able to classify a reconstructed image. Such classification adds gradient to the learning process. If we also wanted to get a classification result for $\mathcal{G}(x,x)$, we could do one of the following, but since none of these options is a viable possibility, we refrain from forwarding reconstructed images to $\mathcal{D}$:

Add a second classifier to $\mathcal{D}$: Adding a second classifier to $\mathcal{D}$ has also been done by Kotovenko et al. [47]. We could train this classifier on x and $\mathcal{G}(x,x)$. Empirically,

we find that using a second classifier creates a disturbing set of gradients, from which both, $\mathcal{G}$ and $\mathcal{D}$ suffer. Furthermore, this also increases computational costs, as we need to compute more forward passes in $\mathcal{D}$.

Add content to style labels in $\mathcal{D}$: Adding content labels to $\mathcal{D}$ would be an applicable solution, if we wanted to transfer the texture of a content image onto a style image. However, we would only train $\mathcal{D}$ on $\mathcal{G}(x, x)$. Thus, $\mathcal{D}$ would quickly learn to always classify samples with content labels as fake, because all samples of this data class can be considered fake.

Reconstruct a style image: By reconstructing a style image, meaning that instead of using a content image x, we use a style image y, to compute $\mathcal{G}(y, \{y\})$. Empirically, we find that this leads to bad content preservation properties, under which $\mathcal{G}$ is not able to preserve the global structures of x in $\overline{x}$. We hypothesize that this is due to artifacts, which y adds to E_c.

3.3 Training

We train the GAN by feeding it with training samples. Those samples are taken from a content dataset $\mathcal{C}$ and a style dataset $\mathcal{S}$. Given these datasets, we sample (y, l) from $\mathcal{S}$, where y is a style image and l its data class, and x from $\mathcal{C}$, where x is a content image. First, we augment those training images and then continue by feeding them into $\mathcal{G}$. Here, we differentiate between four different modes, which we evaluate in the subsequent chapter :

1. single style sample: $(y_1, l_1) \sim \mathcal{S}$ [84]
2. two style samples from the same artists: $(y_1, l_1), (y_2, l_2) \sim \mathcal{S}$, with $l_1 = l_2$ [84]
3. two style samples from the different artists: $(y_1, l_1), (y_2, l_2) \sim \mathcal{S}$, with $l_1 \neq l_2$
4. randomly use mode 1-3

To process multiple style samples, we linearly interpolate their style embeddings to compute an intermediate latent vector:

$$\overline{x} = \mathcal{G}(x, \{y_1, \ldots, y_n\}) = D(E_c(x), \frac{1}{n} \sum_{i=0}^{n} E_s(y_i)) \tag{3.11}$$

We continue by feeding (y_1, l_1) and $(\overline{x}, l_1)$ to $\mathcal{D}$, to receive their logits and feature representations. Depending on which model we currently want to update, we compute $\mathcal{L}_{GAN}^{\mathcal{G}}$ or $\mathcal{L}_{GAN}^{\mathcal{D}}$. Afterwards, we use the feature representations of $\mathcal{D}$ to determine

the feature matching loss $\mathcal{L}_F$. Finally, we reconstruct the content image by computing $\mathcal{G}(x, x)$ and compute the image reconstruction loss $\mathcal{L}_R$. After computing all loss values, we can use back propagation to determine the gradients and perform gradient steps based on the loss.

4 Experiments

In the previous chapters, we introduced the theoretical background in machine learning, neural style transfer and generative adversarial networks. We then presented the network which we use to perform the neural style transfer, its loss function and the approach by which we train our network. In this section we present our experiments and results.

First, we introduce our dataset and its augmentations. Then, we report on instabilities which we find using the (hyper-)parameter setting by Saito et al. [84]. We propose several stabilization techniques to encounter those instabilities and determine their most promising combination as part of a separate optimization problem. Afterwards, we use the newly determined (hyper-)parameter setting to compute the approaches, which we presented in the previous chapter. Subsequently, we record further training observations, of which some are not directly related to the results we obtained, but suggest valuable insides into FUNIT and COCO. Finally, we analyze the alignment of our results with previous research results, which were also tackling the issue of intersecting between artistic styles for neural style transfer.

4.1 Data

For the content dataset $\mathcal{C}$, we choose the Places365 dataset by Zhou et al. [100] which has been released in 2018 for scene recognition. It consists of 10^7 images divided into 434 classes. Training images have a minimum edge length of 512 pixel and are available in grayscale or with RGB color channels. For the style dataset $\mathcal{S}$, we choose a dataset provided by our research group. The dataset contains 20 artist data classes with a total of 4.940 images, which are available in 720 x 720 pixels with RGB color channels. The list includes artists from Abstract art, Cubism, Expressionism, Impressionism, Mannerism and Romanticism. Many artists work not only with one technique but use different materials, surfaces and objects. Also, it is quite normal to make several preliminary studies before an artist begins to paint a larger painting. While these can outline the artwork and shape its global structures, they usually do not include the artist's typical texture. To avoid confusing our model, we remove these and other atypical works.

All images are pre-augmented in the sense that we aligned their appearance.

Figure 4.1: The painting by Camille Pissarro is resized, padded using reflection padding and center cropped, from left to right.

By this process we ensure, that we do not remove any style information and only replicate style that is already there. For this, we reduce image borders when they have disruptive factors, such as white boards, shadows, visible parts of the frame, etc. We compresse images with a maximum edge length exceeding 720 pixels to a maximum edge length of 720 pixels. In addition, we apply reflection padding to all images until they reach a minimum edge length of 720. When processing an image at size 3 x 1000 x 500 pixels, we receive an image at size 3 x 1080 x 720 pixels. Now, we center crop the image at 3 x 720 x 720 to reach its final size. This whole transformation is visualized in figure 4.1.

4.1.1 Validation

To validate the training we select 100 images from content dataset $\mathcal{C}$ and style dataset $\mathcal{S}$ from which we generate plots. During validation, we restrain from adding randomness to make validations at different stages of the training progress comparable to each other. However, there are some random augmentations which we do not want to suppress, like differentiable augmentation, as their suppression may not paint a truthful picture. For this reason, we reset the random seed right before starting the validation. This approach has the advantage that we apply the same augmentations at any validation stage, even though they are designed to be random. After finishing a validation epoch, we reset the random state to the state it was at, right before we validated the model.

4.1.2 Data augmentation

Data augmentation is a technique used to increase the variety of existing data by modifying it. This approach helps the model to develop better generalization properties. We augment training images from both datasets $\mathcal{C}$ and $\mathcal{S}$ the same way. Saito et al. [84] suggest resizing images to have a minimum edge length of 270 pixels. We

find that this hinders finer details in artistic style transfers. Therefore, we resize images to have a minimum edge length of 512 pixels as proposed by Kotovenko et al. [48]. Afterwards, we randomly scale the image width and height by a factor in [1.0, 1.1] and crop a random 256 x 256 pixels section of the image. This helps the model develop better generalization properties, as it becomes harder to simply remember training data. If the image is only available in grayscale, we repeat the color channel three times to make it fit to the input channels of the models. Then, we randomly flip the image on its horizontal and vertical axis with a probability of 0.5. Lastly, we normalize its pixel values to fit a common scale. This improves the numerical stability of the model and might also reduce overall training time. [24] For this, we first normalize the color range from [0, 255] to [0,1] by dividing all pixels values by 255. We then standardize the data to [-1, 1] by applying equation 2.15 with $\mu = \sigma = 0.5$. By this, we symmetrically center the data around 0.

We also run experiments with manually calculating μ and σ in $\mathcal{C}$ and $\mathcal{S}$, which we calculate by using equation 2.13 and 2.14. Here, we compute:
$\mu_{\mathcal{C}} = (0.4571, 0.4406, 0.4073)$ and $\sigma_{\mathcal{C}} = (0.2745, 0.2717, 0.2897)$,
$\mu_{\mathcal{S}} = (0.5057, 0.4780, 0.4313)$ and $\sigma_{\mathcal{S}} = (0.2602, 0.2470, 0.2589)$.
After computing these values, we standardize samples fed to E_c with $\mu_{\mathcal{C}}$, $\sigma_{\mathcal{C}}$ and samples fed to E_s with $\mu_{\mathcal{S}}$, $\sigma_{\mathcal{S}}$. This approach would technically be better suited to capture the data distribution. However, since we standardize pixel values from [0, 1] to [-1, 1], we have to undo this standardization, once we receive the generated sample by D. Technically, this can easily be done by solving equation 2.15 for A, to reverse the standardization in A':

$$A = A' * \sigma + \mu \tag{4.1}$$

However, since $\mu_{\mathcal{C}} \neq \mu_{\mathcal{S}}$ and $\sigma_{\mathcal{C}} \neq \sigma_{\mathcal{S}}$ we can not resort to equation 4.1. We have to decide on a single value for μ and σ to reverse the standardization process. Therefore, we compute $\mu = \frac{\mu_{\mathcal{C}} + \mu_{\mathcal{S}}}{2} = (0.4814, 0.4593, 0.4193)$ and $\sigma = \frac{\sigma_{\mathcal{C}} + \sigma_{\mathcal{S}}}{2} = (0.2674, 0.2594, 0.2743)$, which we use for standardization. After evaluating both approaches we find that standardization in range [-1, 1] produces better results, which is also the widely used approach in computer vision related research [48, 84, 58].

4.2 Encountered instabilities

In this section, we report on instabilities which we observe in prior experiments. We find a broad range of issues which we subsequently confront by suggesting appropriate stabilization techniques. First, we approach the issue of early convergence in

$\mathcal{D}$. We then move on to the case of mode collapse, which might be rooted in an early convergence of $\mathcal{D}$. We continue by focusing on the issue of over- and underfitting on our training data. Afterwards, we concentrate on the problem of vanishing gradients, meaning that a model only obtains few gradients, from which it can not sufficiently improve. Then, we make some first comments on the issue of numerical instabilities, which we also cover later in this chapter. Lastly, we discuss the weight of the reconstruction loss λ_R.

4.2.1 Early convergence in $\mathcal{D}$

By examining the training of our network, we find that $\mathcal{D}$ might issue the same logit for samples generated by $\mathcal{G}$ and such drawn from style dataset $\mathcal{S}$ early in the training. We call the respective logits ϕ_{fake} and ϕ_{true}. In such cases, we have $r := \phi_{fake} \approx \phi_{true}$ with $r \in [-1, 1]$, while the GAN loss $\mathcal{L}^{\mathcal{D}}_{GAN} \to 2$ and the feature matching loss $\mathcal{L}_F \to 0$. A corresponding plot is presented in figure A.7.

The reason why $r \in [-1, 1]$ can be found in the loss setup of $\mathcal{L}^{\mathcal{D}}_{GAN}$. Let's consider $|r| > 1$ in equation 3.6: Here, $\mathcal{L}^{\mathcal{D}}_{GAN}$ effectively computes $1 + |r| > 2$. If we consider $r \in [-1, 1]$, we find that the Hinge loss effectively computes $\mathcal{L}^{\mathcal{D}}_{GAN} = (1+r)+(1-r) = 2$. It is, therefore, most optimal for $\mathcal{D}$ to predict $r \in [-1, 1]$, if $\phi_{true} \approx \phi_{fake}$.

Possibly, $\mathcal{D}$ is unable to differentiate both samples due to $\mathcal{L}_F \to 0$. We find that feature representations in $\mathcal{D}_m$ of the same data class have an average feature distance of 0.0003. Thus, $\mathcal{D}_m$ yields almost the same feature representation for all styles under one label. Meanwhile, the averaged feature distance between ϕ_{fake} and ϕ_{true} is smaller than 0.007. Thus, $\mathcal{D}$ is not able to differentiate between samples drawn from $\mathcal{G}$ and $\mathcal{S}$. Since both are almost identical in $\mathcal{D}_m$, the feedback is not informative. The loss is, therefore, mainly driven by the reconstruction loss $\mathcal{L}_R$, which can lower the translation accuracy, as argued by Liu et al. [58].

4.2.2 Mode collapse

A well-regulated network is characterized by the fact that the training progress is balanced in both models. A mode collapse is a scenario in which a model maps different inputs to the same output. At its core, a mode collapse is caused by an imbalance in the training progress. In practice, this means that there is a learning speed difference between both models. Because one model is superior to the other, it is also able to choose an efficient over a well distributed solution. This behavior can result in a partial or complete collapse. In a complete collapse, all outputs are nearly identical, whereas in a partial collapse, outputs share some common properties. [23] As the superior has no incentive to switch things up, it will over-optimize on this data distribution. Meanwhile, the regularization from its counterpart is not informative,

causing the network to converge to a bad local equilibrium. By chance, the inferior might later recognize its mistake and correct it. Depend on the training progression, the superior might not be able to cope with this because it might already reached a local optimum. Thus, we are interested in both models competing for supremacy, but not actually achieving it.

In prior experiments, we find this issue to strongly affect $\mathcal{G}$ which, in this setting, is the superior model. It is able to generate its samples, such that they fit the real data distribution in $\mathcal{D}_m$. Empirically, we find that the GAN loss $\mathcal{L}_{GAN}^{\mathcal{G}} \to 0$ early in the training indicates a mode collapse because it suggests that $\mathcal{G}$ exploits a misclassified data point in $\mathcal{D}$.

4.2.3 Over- and underfitting the data distribution

To have good generalization properties, a model must identify an underlying trend in the data without considering residual variation as part of the trend. When residual variations are part of the trend, we characterize the model as overfitting on the data distribution. On the other hand, we characterize it as underfitting on the data distribution, when the model is not able to recognize the trend.

The overfitting problem describes a statistical phenomenon in which a model corresponds too closely to a dataset and therefore, fails to fit unknown data. Its essence is that the model learned some residual variations, as if they were part of an underlying trend. Those residual variations most certainly do not occur in unknown data, as they can be thought of as noise. Since those variations are not existent during testing, the model can not reliably predict the distribution of unknown data. Thus, the model is too complex. In practice, overfitting often occurs when the model is trained for too many iterations.

The opposite of overfitting is called underfitting. A model underfits the data distribution when it did not learn enough about the underlying trend to make reliable predictions. We want the model to develop good generalization properties which capture the underlying trend to make reasonable predictions on unknown data. In practice, underfitting often occurs when the model is trained for too few iterations.

4.2.4 Vanishing gradients

Within latent space, there might be suboptimal regions where gradients get too small to make reasonable progress in gradient-based algorithms like Adam (see section 2.2.3). While optimizing the latent space, those algorithms can get stuck at regions with unfavorable parameters. This syndrome is called the *vanishing-gradient problem*, which was defined by Hochreiter et al. [34, 35]. When a gradient based

optimizer reaches such region, further training might not be useful. Vanishing gradients can be caused by any factor, which influences the latent space. [97]

4.2.5 Numerical instabilities

While performing prior experiments, we also regularly encounter visually appearing numerical instabilities in generated samples. Those instabilities come in the form of texture free holes, which become larger and more noticeable as training progressed. In this regard, we also conduct experiments with different settings for ϵ. This hyperparameter is added to the denominator in equation 2.8 to improve the numerical stability of the Adam. We find that if ϵ is set too high, the denominator will almost act as a constant, which leads Adam to evolve into a momentum optimizer. However, we stick to the Adam parameterization by Saito et al. [84] because we cannot solve the issue by tuning the ϵ parameter. Saito et al. set $\epsilon = 10^{-8}$, which has also been done by Sauer et al. [87] and Zhang et al. [98].

4.2.6 Weighting of the reconstruction loss

Saito et al. [84] propose setting the weight of the reconstruction loss $\lambda_R = 0.1$, while setting both, the weighting of the feature matching loss and GAN loss $\lambda_F = \lambda_G = 1.0$. Such choice has its root in image-to-image translation. Here, they want to translate the texture of one animal to another, to generate a hybrid between both of them. To do so, they only keep the global structure of a content image and add texture information to the local structures. Saito et al. use the ImageNet dataset [82] and extract different animal classes from it. ImageNet is a dataset of visual recognition, which includes image classification, single-object localization and object detection. All the above require at least one object which is the focus of the image. This setting has the advantage that a distinct object can be seen in both the content and the style image. Some samples of dogs in ImageNet can be seen find in Ahmed et al. [1]. Art, on the other hand, often not only presents a center object but can cover the hole image with texture that can be versatile, strongly deviate in combination possibilities and diverse in color variety. In the image-to-image translation tasks, they have the comfort of morphing from one to another data class. The generation process, therefore, is not only regularized by texture, but also by the content, which is known to $\mathcal{D}$. This makes translating a painting onto a content image a more challenging problem than translating between data classes. Liu et al. argue that a larger λ_R induces a lower translation accuracy. In prior experiments, we find that keeping $\lambda_R = 0.1$ omits most of the global structures of our content image, as well as most local structures of our style image, as can be seen in figure A.8. This is congruent with the findings Liu et al. made when examining

$\lambda_R = 0.01$.

4.3 Stabilization techniques

In this section, we present stabilization techniques which we examine to counteract training instabilities. First, we discuss regularizing weight changes in $\mathcal{G}$, since $\mathcal{G}$ has been the superior model in prior experiments. Afterwards, we propose balancing updates between $\mathcal{G}$ and $\mathcal{D}$. Finally, we suggest different augmentation strategies to cope with the limited style dataset.

4.3.1 Regularization of weight changes

We consider historic averaging and using spectral norm as normalization layers to regularize weight changes in $\mathcal{G}$. Historic averaging adjusts a deep copy of $\mathcal{G}$ by a moving average of its weight and biases after each training iteration. To evaluate more broadly, we explore a linearly (used by Saito et al.[84]) and an exponentially moving averaging (used by Sauer et al. [87]). Here, we find the exponentially moving average to produce the visually more pleasing results. Spectral normalization, on the other hand, assures Lipschitz continuity to the training of $\mathcal{G}$. By this property, we obtain arbitrarily small changes in our output by restricting it to sufficiently small changes in its input.

4.3.2 Balanced updates

To balance updates between $\mathcal{G}$ and $\mathcal{D}$ we investigate two methods: TTUR [32] and a winning rate [47, 86]. TTUR has been used by Zhang et al.[98] who indicate learning rates of $\alpha_\mathcal{G} = 0.0001$ and $\alpha_\mathcal{D} = 0.0004$ to train $\mathcal{G}$ and $\mathcal{D}$ at a 1:1 ratio. Winning rates have been used by Kotovenko et al. [47] and are applied to regularize the interaction of $\mathcal{G}$ and $\mathcal{D}$. Kotovenko et al. suggest a winning rate of $\delta = \{80\%, 90\%\}$. However, in prior experiments we find visually pleasing results with a winning rate of $\delta = 70\%$. In these experiments we find that winning rates are able to transfer the texture of a style very convincingly, but sacrifice content. We hypothesize, that this is because most early updates are done for $\mathcal{D}$. By receiving disproportionately many updates, $\mathcal{D}$ is able to determine a metric strongly reliant on texture. $\mathcal{G}$ learned to cope with this property and therefore, generates convincing textures. However, $\mathcal{D}$ can not regularize $\mathcal{G}$ for disregarding content. This hypothesis is also confirmed by the fact that in such experiments, the model was not able to reconstruct the image without artifacts.

4.3.3 Data augmentation

Data augmentation is used to bypass the problem of overfitting. We evaluate two different methods: using a color jitter as part of the augmentation strategy and using differentiable augmentation by Zhao et al. [99]

A random color jitter is used by Saito et al. [84]. They apply a color jitter that randomly scales brightness, contrast, saturation and hue by a factor in [0.8, 1.2]. Training $\mathcal{G}$ on a broader range of colors, contrasts, etc. could be a useful property considering the fact that we want to intersect styles. Furthermore, it might help the model determine a broader knowledge of colors. This is an important property because it ensures that $\mathcal{G}$ does not need to extrapolate colors while applying them. The issue with using a color jitter is that it might populate artifacts in the data distribution of $\mathcal{G}$, which is shown by Zhao et al.

In contrast, differentiable augmentations can be used to avoid these artifacts. We set the parameters to those provided by Sauer et al. [87], as we empirically find that they produce better results, compared to the parameters provided by Zhao et al. [99]. In details, we change the cutout ratio from 0.5 to 0.2, which masks a smaller area of the image. Using a color jitter, as well as differentiable augmentations the same time does not only worsen our results [99], it also applies a similar color transformations twice.

4.4 (Hyper-)parameters

In this section, we determine the right (hyper-)parameter setting to train our network. As we adopt the network architecture from Saito et al. [84] we also utilize some of their (hyper-)parameters. However, as outlined in section 4.2, we experience various training instabilities using those settings. Therefore, we consider various stabilization techniques which we pointed out in section 4.3, as well as the volume of learnable parameters in $\mathcal{D}$ and its dimensionalit to obtain a stable training setting. Stabilization techniques are evaluated by considering their application in an optimization problem. After finding the most promising stabilization techniques, we present the final (hyper-)parameter setting in section 4.4.3 which we use to train our network.

4.4.1 Basis (hyper-)parameter setting

By considering some (hyper-)parameters as given, we can approximate the optimization problem in chapter 4.4.2 more easily. We, therefore, set the momentum parameters of Adam to $\beta^{'} = 0.0, \beta^{''} = 0.99, \epsilon = 10^{-8}$, as being done by Sauer et al. [87], Zhang et al. [98], Liu et al. [58] and Saito et al. [84]. As discussed in

section 4.2.5, we also perform some prior experiments with different settings but observe that they do not improve our results. Furthermore, we randomly initialize our learnable parameters and use a batch size of 8, as being done by Saito et al. [84]

4.4.2 (Hyper-)parameter optimization

Now, we want to determine the right (hyper-)parameters setting for our network, which is a problem-dependent issue. [71] Nevertheless, we utilize the setting by Saito et al., which we present in section 4.4.1 and focus on (hyper-)parameters, which we consider "stabilizing". Choosing the right (hyper-)parameters is often done by trial and error or grid search. However, finding that we can consider selecting the right (hyper-)parameters as an optimization problem itself, for which we want to minimize a score function. [2] To compute such score, we can use a metric that measures the similarity between samples generated by $\mathcal{G}$ and samples drawn from $\mathcal{S}$. One would typically use FID [32] to compute such score, however, FID is affected by a bias that only vanishes for $\mathcal{N} \to \infty$, with a reasonable $\mathcal{N} \in \{10,000, 50,000\}$. Computing the FID score at $\mathcal{N} = 50,000$ takes approximately 2,5h on a NVIDIA Quadro RTX 6000. As we also want to prune underperforming trials, we have to evaluate each trial at multiple stages of their training. Computing the FID score would quickly become computationally expensive, which is why we use the SIFID [80]. We validate the training every 5,000 training iterations and train for a maximum of 50,000 iterations. To prune trails, we use a Successive Halving Pruner [54] and sample new trails from a tree-structured parzen estimator [73]. Within this sampler, we use a partitioned search space that enables the sampler to recognize correlations between parameters. [18]. We perform 100 optimization trials using Optuna [2] for the parameters shown in table 4.1.

Here, ID 0 determines the learning rate of Adam in $\mathcal{G}$, which we use. If the sampler chooses TTUR, we use the learning rate $\alpha_{\mathcal{D}} = 0.0004$ [32, 98], otherwise we set $\alpha_{\mathcal{D}} = \alpha_{\mathcal{G}} = 0.0001$. ID 1 is intends to evaluate whether it is advantageous to apply a winning rate. If we apply such winning rate, we set the winning rate $\delta = 0.7$. In ID 2, we evaluate an augmentation strategy. We analyze differentiable augmentations by Zhao et al. [99] (see section 2.5.3) and using a color jitter as part of the augmentation strategy in section 4.1.2. ID 3 evaluates which regularization we apply to weight changes in $\mathcal{G}$. We can either constrain Lipschitz continuity to all layers using spectral norm (see section 2.3.4) or average weight changes over time using historical averaging (see section 2.5.3). In ID 4 and 5 we determine the size and dimensionality of $\mathcal{D}$, which we want to set appropriately to cope with $\mathcal{G}$. Lastly, in ID 6, we determine λ_R, which is the weight we use for the image reconstruction loss.

Table 4.1: (Hyper-)parameters, which we analyze as part of an separate optimization problem with their optimal value. The best trial finishes with an SIFID-score of 23.33

ID	set	optimized
0	{TTUR, none}	TTUR
1	{win-rate, none}	none
2	{diff. augm., color jitter in dataloader, none}	diff. aug
3	{spectral norm., historical averaging, none}	spectral norm
4	[4, 6]	6
5	{512, 1024}	512
6	[0, 1.0]	0.857

4.4.3 (Hyper-)parameter setting

After computing the optimized (hyper-)parameters in table 4.1, we make first experiments with the obtained setting. We compare this approach to a trial in which we set $\lambda_R = 1.0$. Here, we find that the setting of ID 6 was hindering the training progress and that the latter approach was able to generate samples in a higher quality on the long run. Therefore, we continue with this setting and trained our network for 200,000 training iterations.

4.5 Results

Now, we present the results of evaluating of four different approaches to train the network, which we introduced in section 3.3. First, we train our GAN using a single style image (denoted as Model A). We hypothesize that using a single style image may work best to preserve specific style features. However, a style is made from stylistic elements that are distinctive of a style. We might find those stylistic elements, when intersecting between two paintings of the same artist (denoted as Model B). Sanakoyeu et al. [86] even go so far to say, that "[...] it is insufficient to only use a single artwork, because it might not represent the full scope of an artistic style". [86] However, a specific style is not something a single artist determines in isolation, but it is shaped from previous developments and a product of the evolution of style. Therefore, we train a model on intersecting between two paintings of different artists (denoted as Model C). Lastly, we combine all of those approaches and let a model randomly use one of them at each training iteration (denoted as

model	single	multi	multi class	∅
A	43.135	40.792	36.041	39,989
B	43.870	41.532	40.201	41,867
C	45.758	44.042	41.220	43,673
D	47.173	43.119	36.732	42,341

Table 4.2: FID computed for 10,000 samples

model	single	multi	multi class	∅
A	39.517	37.375	32.067	36,319
B	39.414	37.844	36.717	37,991
C	41.118	39.736	36.992	39,282
D	43.045	40.144	33.356	38.848

Table 4.3: FID computed for 50,000 samples

Model D). For all models, we use the same (hyper-)parameters, as determined in section 4.4.2. We compute t-SNE [63] for each style encoder to analyze their style embedding. Here, we reduce the 64 style dimensions of E_s to 2D, such that we can plot them on a graph. For this, we find 1000 iterations at a learning rate of 200 to produce reasonable results. We also calculate FID scores for all four models using both 10,000 (see table 4.2) and 50,000 (see table 4.3) samples, but find its limitations in intersecting styles.

We display some of our results in figure 4.2 and evaluate each training mode individually in the following.

4.5.1 Model A

Training the network with a single style sample achieves the lowest FID score with $\mathcal{N} \in \{10,000, 50,000\}$, as can be seen in figure 4.2 and 4.3, respectively.

We evaluate the style embedding using t-SNE and find that the style of similar artists are represented by similar style features. To measure the deviation of all styles to their centroid, we compute a centroid for each artist. Here, we find the highest average deviation compared to all other training methods. This is not a good property, as it means that the style embeddings are widely scattered in the style space.

It is noticeable that the model is not able to transfer the style of Nicholas Roerich to an image while preserving its global structures. This data class also contains the biggest chunk of paintings. However, since we sample paintings by equally distributing the artist appearance during training, this issue is unlikely caused by overtraining on the artist class, otherwise we would find a similar behaviour in other

Figure 4.2: Results obtained during evaluation. As style we use two paintings by Pablo Picasso and one by Berthe Morisot. Model A has been trained with a single style image, model B with two style samples from one artist, model C also with two samples from two artists and model D by randomly sampling from the approaches in model A, B, C.

data classes. By investigating the style embedding through t-SNE we expose the style of Roerich to be spatially distant to other styles. One explanation for this behavior could be that this condition might be caused by the style itself. If we take a closer look at Roerich's work, we see that paintings often consist of larger areas of color. Neighboring areas often adopt this color tone with minor adjustments. Besides this issue, other data classes are able to reasonably approximate style.

Lastly, the model also sufferes the strongest from numerical instabilities, which we later cover in section 4.6.1

4.5.2 Model B

Training the network with two styles from one artist has achieves the second lowest FID-score. We also find that this approach has the most balanced GAN loss. $\mathcal{L}_{GAN}^{\mathcal{D}}$ is nearly the same as in the other approaches, however $\mathcal{L}_{GAN}^{\mathcal{G}}$ is about 50% lower, meaning that $\mathcal{D}$ is more susceptible to making a mistake. However, we also observe that $\mathcal{D}$ suffers from an early convergence in the first 50,000 training iterations, as can be seen in figure A.7. It seems like the model is able to recover from it, but this incident might have hindered its early development. In this regard, we also notice strong indications for a mode collapse, as some stylizations are almost identical, although we use completely different styles.

Numerical instabilities which have been very present in the previous approach have become less obvious. However, the stylization is rather flat, with little depth effect. We also observe less pronounced brush strokes, which appear almost rhythmical, like a layer which has been laid on top of the image. Nevertheless, the issue that the model is not able to transfer the style of Nicholas Roerich to an image has been reduced, but we still find that the model has trouble transferring it.

As both the local structures of the style images and the global structure of the content image are not preserved to a sufficient extent, we conclude that the model has converged to a bad local equilibrium.

4.5.3 Model C

Training the network with two styles from two artists results in the highest FID-score. Furthermore, we find the highest divergence between $\mathcal{L}_{GAN}^{\mathcal{G}}$ and $\mathcal{L}_{GAN}^{\mathcal{D}}$. The model can only transfer the brush stroke to a limited extent. Even in cases where the brush stroke is visible, the model looks like it needs more training. However, this would also not be beneficial, since $\mathcal{D}$ is already very confident in its classification. The model seems to lose track of global structures and often omits them to a far extent. In addition to these limitations, we find the biggest areas of numerical instabilities.

Because the model does not generate stylizations, which uphold the global structures of the content image while applying the local structures of the style images, we conclude that the model has converged to a bad local equilibrium.

4.5.4 Model D

Training the model by randomly selecting one of the above settings at each training iteration reached the second highest FID score in $\mathcal{N} \in \{10,000, 50,000\}$. The only model with a higher FID score is model C. However, we come to the conclusion that this model was able to subjectively generate the best stylizations. A reason for the comparatively high FID score might be that the model combines the training of all the above models. Model A achieves the lowest FID score, while model B combines two paintings by one artist. In the latter, we assume that an artist uses recurring colors and brush strokes, making its paintings somewhat similar in color and style. For similar styles, the intersection should also be close to both styles, whereby little style information is lost. In contrast, Model C has the highest FID score. Since model D randomly chooses between these settings, it is not surprising that it also achieves a medium FID score. A second reason on why we were not able to determine FID results that corresponds to our subjective findings is due to the fact that FID is used to compare two datasets. However, in our example, we use more than one style sample per style transfer. If we evaluate a NST that uses style elements from two paintings on how closely it resembles the style elements of one style, we can only determine a partial result. Also, if we compute a score for both styles individually, we can not just average them. To compute such score, we either need a single style image or have to intersect both style representations in the Inception v3 model [92]. Since only the latter is applicable, we evaluate this approach, but find that it is also unable to provide satisfying results.

When analyzing the style embedding using t-SNE in figure 4.3, we find that this model is spatially the most closely distributed model. This is supported by the fact that the model has the lowest average deviation from the centroid of each artist. Furthermore, the intersections of the convex hulls become smaller, meaning that styles are less broadly distributed in the style space.

Subsequently, we analyze the centroids in figure 4.4. The highlighted paintings are representatives of the respective artists' style. Again, we find that artists who use a similar style are also spatially close to each other. Interestingly, we even find that Claude Monet's early and late work are spatially close to each other, although Monet nearly went blind in his late work, which was the original reason for splitting the dataset. We mark this in figure 4.4 with a grayish box. Originally, we have split the data class to prevent different styles from confusing the style embedding. This

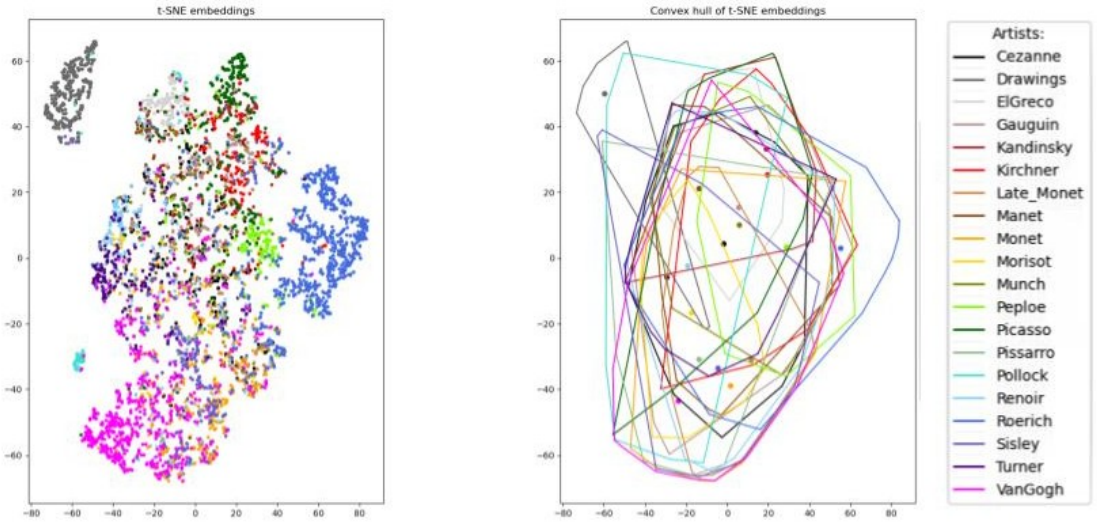


Figure 4.3: t-SNE embedding for model D (left) vs. convex hull of t-SNE embedding with centroids (right)

indicates that we did not need to do this, since although the style is different, the style encoder sees similarities in both of them.

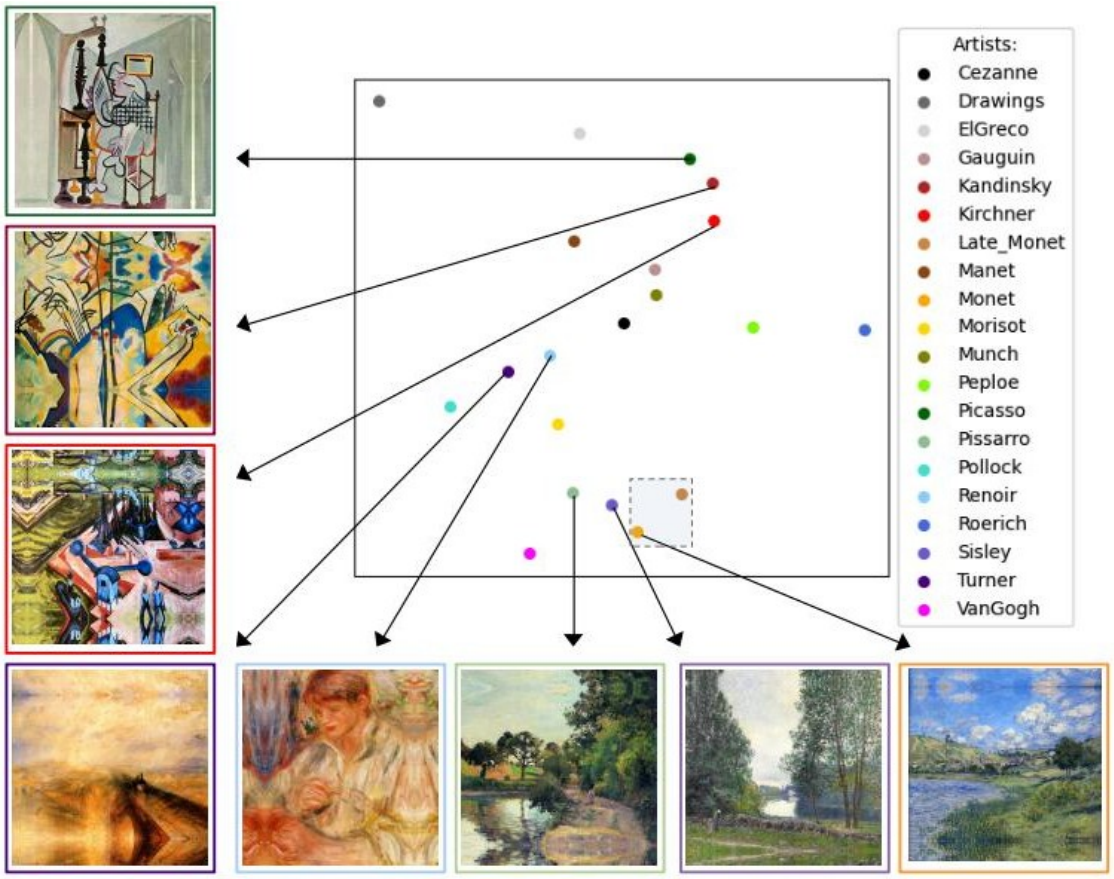


Figure 4.4: Comparison of the spatial relation between paintings and centroids of the t-SNE embedding for each artist in model D. Also, Monets early and late work are spatially close to each other (marked with grayish box)

Let us now consider the style of two particular artists, the Impressionists Berthe Morisot and Edouard Manet. Morisot married Manet's brother Eugène, so one would think that the two knew each other well and were aware of each other's work. We select their style coordinates in t-SNE, add convex hulls and compute a centroid, as can be seen in figure 4.5. Here, we determine the artwork, that is closest and furthest to the centroid for both artists. The paintings close to the centroid show strong similarities. Both depict a young woman, with pale skin in a dress, painted in natural colors with visible brushstrokes. The artwork fa from the centroid are different in color, brush strokes and content. This provides evidence for the assumption that similar images are close in style space, while different images are far from each other.

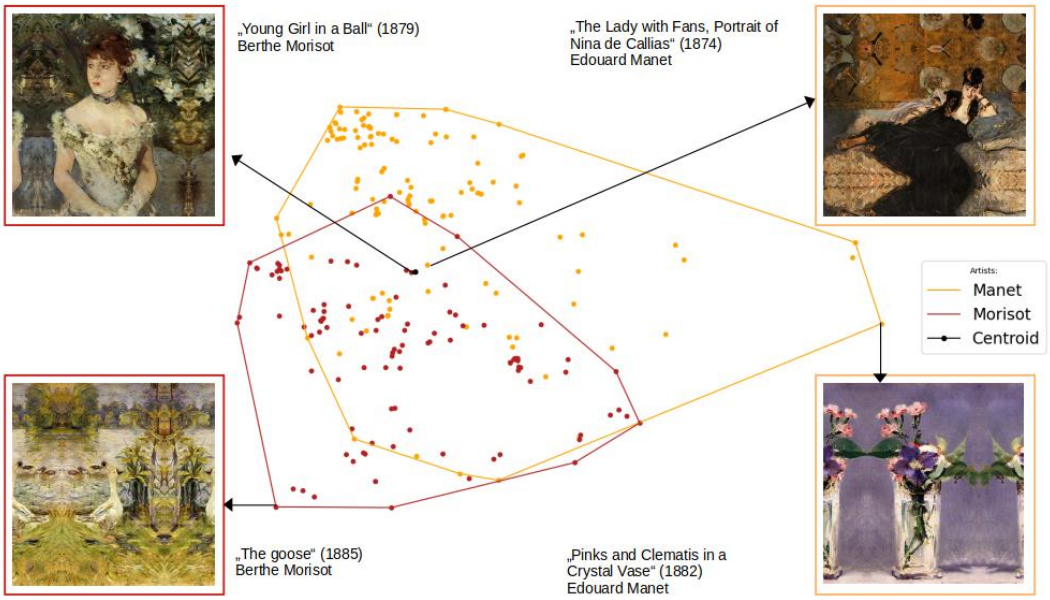


Figure 4.5: Comparing the style embeddings in t-SNE of Berthe Morisot (firebrick red) and Edouard Manet (orange)

We provide a detailed style intersection in figure 4.6 and included further stylization results to the appendix, which can be seen in figure A.2, A.3, A.4, A.5 and A.6

4.6 Training observations

In this section, we cover some training observations. First, we discuss numerical instabilities which we encounter in our experiments. Then, we document our findings when using a pre-trained model in COCO and FUNIT. Afterwards, we comment on FUNIT, the content loss problem described by Saito et al. [84] and our findings regarding an early convergence in $\mathcal{D}$. Last but not least, we describe a condition in

Figure 4.6: Style intersection between Berthe Morisot and Pablo Picasso (top) vs. style intersection between two paintings by Pablo Picasso (bottom).

which FUNIT abandons stylization almost completely in some cases.

4.6.1 Numerical instabilities in extreme pixel values

In our evaluation results, we experience numerical instabilities, as can be seen in figure A.1. We already identified those instabilities in chapter 4.2.5, but find that they do not entirely disappear even with stabilization measures. We analyze their appearance and find that they correlate with strong increases and decreases in color frequencies. For this, we compare the color histogram of a style image, which causes numerical instabilities to one, which does not cause these instabilities, as can be seen in figure 4.7. Here, we find that the one experiencing numerical instabilities had a sharp increase in its color frequency.

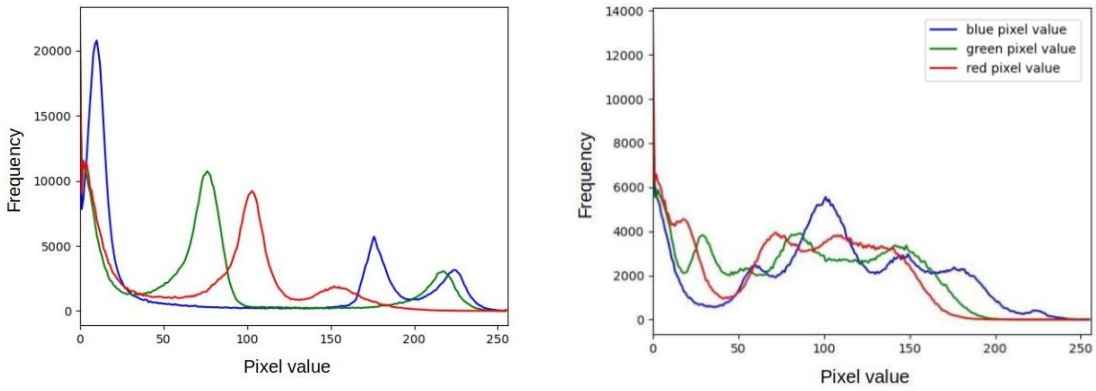


Figure 4.7: Color histogram which causes numerical instabilities (left) vs. color histogram which does not cause numerical instabilities (right)

Now, let's consider a content image, as it is the second part of a NST. Such images have features like as shadows, edges, etc. Our goal is to apply the texture according

(a) Content image

(b) Masked content image

(c) Styled image

(d) Masked style image

Figure 4.8: We compute a extreme pixel values mask (upper and lower 10%) for image (a), which are marked turquoise in (b). Then, we compute the stylized image (c) on which we also attach the extreme pixel value mask, which can be seen in (d). This allows us to show that extreme pixel values correlate with numerical instabilities.

to the union of features and style. Colors, for example, are darkened when moving further into a shadow. For this, the model uses the color information available in its style feature. In cases where numerical instabilities become visible, we do not have a smooth color frequency distribution. Therefore, we hypothesize that the model extrapolates the color gradient and fails to select the correct texture.

Empirically, we find that we can approximate the appearance of numerical instabilities in areas where pixel values reach extremities. For this, we build a mask from a content image, which indicates its upper and lower 10% of pixel values. We then stylize the content image and attached the mask to the stylized image. Here, we find that regions of extreme pixel values correspond to numerical instabilities, as can be seen in figure 4.8.

However, their appearance at pixel value extremities may not be the root, but rather a syndrome of the underlying issue. By computing an averaged color histogram for all style images in our style dataset $\mathcal{S}$, we find that the histogram presents a strong frequency increase and decrease for pixel values in $[0, 25] \wedge [250, 255]$, as

can be seen in figure 4.9. It might be that because of the average color frequency, our training suffers from numerical instabilities, which may, in consequence, have become artifacts in $\mathcal{G}$.

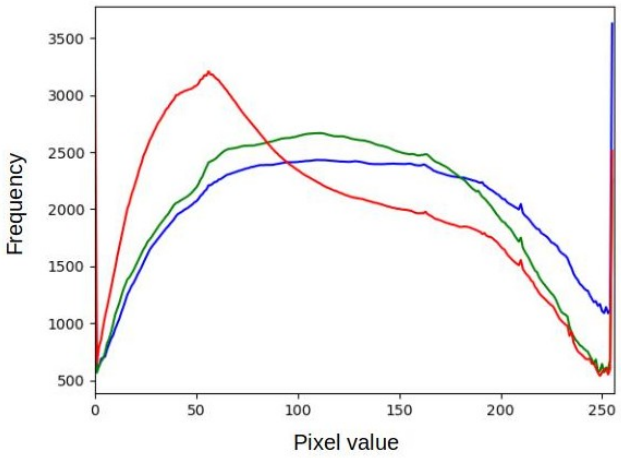


Figure 4.9: Color histogram of style dataset

4.6.2 Pre-trained model cannot be integrated to FUNIT / COCO-FUNIT

Empirically, we find that using pre-trained models in both $\mathcal{G}$ and $\mathcal{D}$ does not produce good results. However, we find that the COCO and FUNIT architecture consistently produces poor results by integrating a pre-trained model into its architecture.

We use VGG [89] as a content encoder in E_c and find that this leads to mode collapse in all cases. Another problem we encounter in the context of VGG is standardization. To work with VGG, images should be standardized with $\mu = (0.485, 0.456, 0.406)$ and $\sigma = (0.229, 0.224, 0.225)$. However, since we use a detached style encoder E_s that is not based on VGG, the subsequent reversed standardization leads to the same problem we have already pointed out in section 4.1.2.

In a second case, we implement $\mathcal{D}$ based on the projected GAN discriminator of Sauer et al. [87] and use their hyperparameter setting. Here, we find that $\mathcal{G}$ loses both content and style and perishes from an early mode collapse. By investigating the logs produced by the model, we find that $\mathcal{D}$ is perfectly able to separate samples drawn from $\mathcal{G}$ and the style dataset. We also conduct experiments in which we disable parts of the $\mathcal{L}$ in multiple combinations, namely $\mathcal{L}_R$ and $\mathcal{L}_F$. However, in none of those cases we are able to generate good quality samples.

4.6.3 FUNIT suffers from early convergence

In experiments, we deal not only with COCO, but also with FUNIT. In contrast to COCO, the training of FUNIT suffers from an early convergence issue in the case of model A, model B and model D. The feature representations in $\mathcal{D}_m$ for the generated and training samples are nearly identical. In such cases, we find the feature matching loss $\mathcal{L}_F < 0.0001$, the reconstruction loss $\mathcal{L}_R < 0.04$ and the GAN loss $\mathcal{L}^G_{GAN} \in [-0.2, 0.2]$, which cause a vanishing gradient problem in $\mathcal{G}$. We also see this confirmed by the evaluation results. However, we find a reasonable regularization between $\mathcal{G}$ and $\mathcal{D}$ for model C, which has not suffered from early convergence in $\mathcal{D}$ and, in return, omits the vanishing gradient issue in $\mathcal{G}$. However, we are actually able to obtain reasonable stylizations not only for model C, but also for model D. Nevertheless, those stylizations do not compare favorable with the stylizations obtained while using COCO in $\mathcal{G}$.

4.6.4 FUNIT abandons stylization in some cases

In the process of evaluating FUNIT, we find that, in some cases, $\mathcal{G}$ completely disregards the local structures of its style sample. $\mathcal{G}$ develops this property with the progression of training, which we observed during validation. As discussed in section 4.6.3, we find that FUNIT suffers in some training modes from an early convergence issue. Interestingly, a subset of this group also does not apply stylization in some samples. We find missing stylizations in model A and model B.

Because of the early convergence issue in $\mathcal{D}$, $\mathcal{G}$ relies less on $\mathcal{L}_F$ and $\mathcal{L}_{GAN}$, meaning that $\mathcal{L}_R$ had a bigger impact on its optimization. We hypothesize that this property leads $\mathcal{G}$ to have an identity function for its content image. This identity function may then be used if $\mathcal{G}$ considers the style information as not useful.

We do not find both issues using COCO for $\mathcal{G}$. In general, $\mathcal{G}$ achieves a more stable training when using COCO, which we are able to observe by the feature matching loss $\mathcal{L}_F$ and the GAN loss $\mathcal{L}_{GAN}$. It might be the case that conditioning the style with content makes the stylization process more difficult for $\mathcal{G}$, which establishes a better regularization between $\mathcal{G}$ and $\mathcal{D}$.

4.7 Comparison to other approaches

In this section, we want to draw a comparison of our approach to those of other research results. We visualize this comparison in figure 4.10. Here, we find that our approach is able to outperform the approach of Huang and Belongie [37], but lacks behind the more advanced approach by Kotovenko et al. [47]. We find that their approach only lacks in the texture of the sky. Other paintings by Vincent van Gogh

and Paul Cezanne display the sky either as a smooth surface or with visual brush strokes. The style intersection of Kotovenko et al. displays in this location cracks in the texture. Our approach does not show these cracks. However, their approach is able to better preserve the global structure of the content image and adds finer details of the style samples. Also, our approach has issues stylizing the stones lying on the beach. The approach by Huang and Belongie exhibits a rhythmical flickering in style. In addition, the outlines of the mountain are highlighted by a variation in texture.

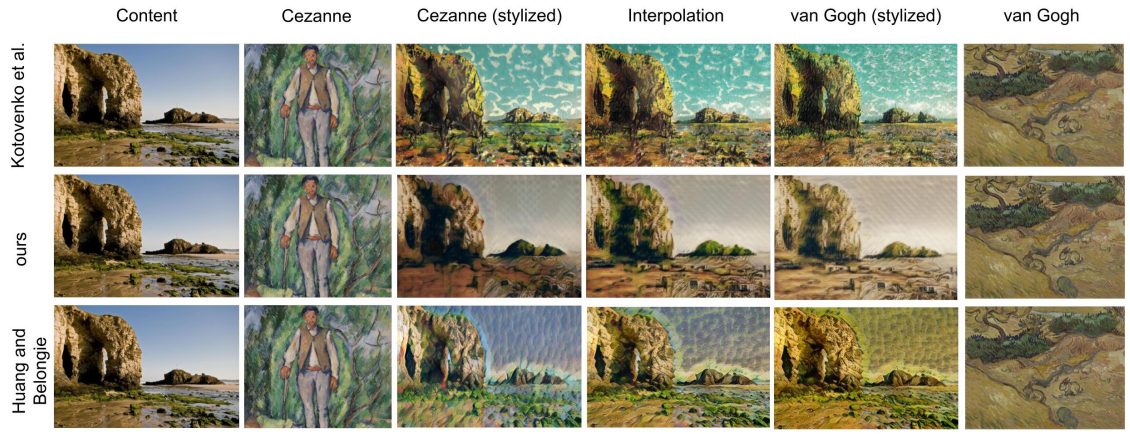


Figure 4.10: Comparinf our approach to Kotovenko et al. [47] Huang and Belongie [37]. Adopted from Kotovenko et al. [47].

5 Conclusion

The presented thesis had the goal to combine multiple styles using neural style transfer. In this last chapter, we recap the chosen approach to achieve the named objective and summarize the main findings. We then point our further improvements to the presented approach and future research possibilities in the context of NST. Eventually, we conclude by discussing our findings and put both the results and the chosen approach in a broader context: Which implications do these findings have on our society? To which extent can machine learning help us to understand artistic styles?

5.1 Summary

At the beginning of this thesis, we raised the question, whether it is technically possible to intersect representations of artistic styles using neural style transfer. We presented an approach to combine multiple styles for neural style transfer. For this, we used a GAN architecture with a few-shot image generator with content conditioned style encoder. While training our GAN we have dealt with various training instabilities, which we were able to fix by considering different stabilization techniques as part of a separate optimization problem.

To perform a NST, we investigated four different approaches: First, we analyzed an approach in which we only use a single style image during training. Then, we examined how the training behaves when we intersect two styles from one artist. We extended this approach to two styles from different artists in our third approach and finally, we analyzed an approach in which we randomly apply one of the above approaches at each training iteration.

In our evaluation in section 4.5, we found that we achieve the best stylization results when resorting to the fourth approach. Here, we examined the style embedding of our model using t-SNE. In this analysis, we found that there is a spatial relationship between a style and its style representation. This analysis allowed us to spatially identify art epochs and to draw conclusions about the interaction between artists. By this we were able to qualitatively evaluate our results. We also evaluated our results using FID, but reached its limitations using multiple style samples. The results which we were able to obtain did not outperform the state-of-the-art

by Kotovenko et al. [47] Samples by Kotovenko et al. better preserved the global structures of the content image and had finer texture details. However, we were able to show that intersections between multiple styles from one or two artists are possible, which was the target of this work.

In terms of computational costs, we were able to reduce the required resources. We trained each of our models for approximately 14 days on a single GPU (Quadro RTX 6000) with 24GB GRAM. Saito et al. on the other hand, parallelize their training on 8 GPUs (V100) each with 32GB GRAM. [84]

5.2 Possibilities for future research

After presenting our contributions to current research, we want to point out parts of our work that could be improved, as well as others that might be beyond the scope of this work. We also look at current research developments and how style transfer could be applied here.

5.2.1 Improvements to this approach

In this section, we highlight some extensions that could potentially improve our approach. First, we indicate that by the end of the training we are dealing with an overly confident discriminator and propose to mitigate this with increasing noise signals. Then, we present a training approach in which two random game partners are chosen, which could reduce the occurrence of mode collapse. Afterwards, we propose to diversify the style dataset and distribute it in a more balanced way. This could also lead to the possibility of a new training approach: training the model with two styles of one style epoch.

Adding noise to $\mathcal{D}$

Throughout our experiments, we find that the discriminator $\mathcal{D}$ is initially often weaker than the generator $\mathcal{G}$. However, this changes as the experiments progress. We find that the discriminator becomes too confident in its classifications after 200,000 training iterations. This means that $\mathcal{D}$ has determined a disjoint probability distribution for samples of the style dataset $\mathcal{S}$ and samples generated by $\mathcal{G}$. Arjovsky and Bottou [3] make the point that low dimensional manifolds are almost certainly disjoint. Having a disjoint probability distribution makes them easily separable, from which we encounter the vanishing gradient problem. Adding noise to $\mathcal{D}$ can spread out the distribution and makes probability distributions overlap. However, if we start adding noise too early in the training process, we might also cause $\mathcal{G}$ to exploit such data point, causing a mode collapse. Therefore, we propose adding noise with

the progression of the training. For this purpose, we could increase the probability of noise signals with a linear function. By this we could continue training and might obtain more useful gradients in the long term. In practice, adding noise can happen by randomly switch labels while computing $\mathcal{L}_{GAN}^{D}$ as proposed by Goodfellow [25]

Random game participants

One way to prevent $\mathcal{G}$ from mapping many input data points to fewer output data points could be to train an ensemble of models. For this, we would use $|\mathcal{G}| > 1$ and $|\mathcal{D}| > 1$ and randomize the playmates. This should encourage each network to develop better generalization properties as they have to cope with more diverse counter strategies. However, this would have the effect that the training would take significantly longer, as training time grows according to the number of $|\mathcal{G}|$ and $|\mathcal{D}|$. By this approach it might be possible to achieve greater success in less time. However, these improvements come at the cost of requiring more computational resources.

Balanced style dataset

We would also like to further balance the style dataset $\mathcal{S}$. The size of $\mathcal{S}$ seems to be reasonable, if we apply differentiable augmentations, to work on smaller datasets. However, we find that we have imbalances in the number of samples available for each artist. These range in [52, 803], making some style more diverse than others. Considering the fact that our style encoder is not conditioned, those dominant style specifics may create "style" artifacts. We also find that 2.816 of the 4.920 images have been painted by Impressionists, such as Monet, Manet, Moriset, VanGogh, etc. Here, we could sort out the more cubistic artists, leaving us with 2.290 images. We would like to create a more balanced dataset, that considers epochs like Cubism, Expressionism, Impressionism, Romanticism, Dark Romanticism and even back to Baroque. We choose this combination because all of these styles work at their core with the concept of light and the re-composition of reality. Here we would like to see a balanced number of artists, with a balanced number of paintings. By doing so, we would help the model develop better generalization properties.

Combination of styles from the same epoch

If we were able to balance the style dataset $\mathcal{S}$, we could use the epoch information during training. We could replace training a model with two styles from the same artists with two styles from two artists of the same epoch. This approach also respects the notion of Fernie, that style is "[...] an expression of a collective spirit". [20]

5.2.2 Future research with neural style transfer

In this section, we will address future research topics that could be enhanced by the use of neural style transfer. First, we focus on an artistic approach that could enable style transfer to 3D objects. This area has recently seen a lot of progress in the field of neural radiance and density fields (NeRF). In the context of these results, we could also use NST to adjust not only a singular 3D object but also a scene, such that we can represent it e.g. at different day times.

3D style transfer using NeRF

After dealing with 2D artistic style transfer in this work, we would like to transfer this approach to the 3D space as well. In art, sculpturing is often seen as one of the most difficult tasks, as it exchanges perspective with proportions. Artists go to extreme length to get them right by measuring exact proportions or creating pre-studies which they cover by water to later reduce water equivalent to their progress on a marble block. Nevertheless, sculpturing is not necessarily about creating an exact copy of a human being. In fact, Auguste Rodin was once accused of using a direct plaster cast for his figure "L'Âge d'airain" (1877), which he angrily denied, followed by a monograph of its re-creation. If we take a look at "David" (1504) by Michelangelo Buonarroti we also find some characteristics, which do not display reality. For example, its right hand is slightly dis-proportionally increased. This is because the statue should have originally been place in great height on the Cattedrale di Santa Maria del Fiore in Florence. However, after releasing the finished statue, it became a landmark of Florence and was placed in front of the Palazzo Vecchio. In contrast to the pictorial representation, sculpture is more about which details are omitted and which are incorporated. This shapes the individual style of an artist and distinguishes its work. It would be interesting to analyze whether we are able to transfer the style from one 3D object to another. For a dataset, we could draw on recent research by Müller et al. [67] that allows us to almost instantly determine 3D objects from images. This technique, which is also called NeRF, could help us to create a style dataset, which we could transfer to arbitrary objects. The target for such research topic could be to create a pipeline in which photos of a person are used to create a 3D object of this person, to which we then transfer a style and thus depict the person as a sculpture. By using a 3D printer, we could print this sculpture, create a negative and have it produced as a brocade casting. However, we could also use those techniques to render computer graphics in game development or other areas where we are dealing with volume graphics.

Scene adjustment for scene recognition

If we were able to successfully apply an artistic style transfer to 3D object - which should be possible considering Regateiro and Boyer [76] and Ma et al. [61] - we could use those insides, to apply style transfer methods to 3D scenes. Lately, Tancik et al. [93] created a 3D model of San Francisco by the use of NeRF. They were able to display the city at day and night because the respective images were taken during night and day. Such models are of interest because they can be used to simulate the real world. By using reinforcement learning, we can train models to operate in these simulations and transfer their learnings onto the real world. Specifically for self-driving cars, it would be incredibly important to train such models in diverse settings. If we could apply style transfer methods to such scenes, it could enable us to simulate conditions like rain, darkness, dazzling light and others. Certainly, the difference between day and night could also be depicted in such a way that we stylize the underlying images. However, it cannot be ruled out that, for example, shadows are changed in a way that they no longer represent their underlying light rays. Another problem would be that we would have to create a scene for each variant, whereas with 3D style transfer we could even represent the smooth transition between them.

5.3 Societal implications of machine learning for neural style transfer

After summarizing our results and outlining future research possibilities for NST, we also want to raise awareness on the risks and unresolved problems due to the use of machine learning. We conclude with some lessons learned by this work with regard to our initially raised question: "To what extent do style intersections create a new artistic style?"

5.3.1 Reflections on the energy consumption

The energy consumption of modern machine learning applications is horrendous. For example, foe example, the power consumption of our approach on a Quadro RTX 6000 GPU has a peak power consumption of 295 watt according to NVIDIA. This amounts to a kilowatt-hour (kWh) consumption of 7.08 / day. Thus, the entire training requires about 99.12 kWh. Since we probe four different approaches, we require at least 396.48 kWh to compute our results. However, this does not take into account that most experiments do not achieve satisfying results with the first trial. Often it requires many training trials with different (hyper-)parameter settings,

through which we slowly approach the desired result. In times of climate change, this energy consumption should make us reflect and question if we need larger or more efficient models for future research. To run the hardware that computes our models, we should rely on renewable energy resources. If this is not possible, we should ask ourselves whether the substantive benefits we derive from this research legitimize the risks posed by climate change.

5.3.2 Social risks due to neural style transfer

Advancing on technologies such as neural style transfer is clearly an interesting research topic with useful applications in science, health, game development and many more. Clearly, in the context of art we can use NST to access paintings in entirely new ways, which can offer a broader understanding of the artwork. However, such technologies can also be used for other purposes, such as DeepFakes. DeepFakes are images or videos, in which a neural network often morphs a face onto a body. As we cited in our introduction, NST is also capable of adjusting audio waves, meaning that it is able to transfer the sound of a specific voice onto the spoken words of someone else. When both are combined with each other, applications can be created that generate fake recordings. Unfortunately, it takes a trained eye to distinguish these recordings from real ones. We have already seen these applications being used for political attacks in the past [70] and present [17]. Luckily, there is accompanying research such as Prajapati and Pollett [74] that manages to detect DeepFakes in image data with 91% test accuracy. However, we cannot assume that this information will reach all consumers, which may draw a distorted reality in certain segments of the population. It is up on computer scientists to take this problem serious and work to ensure that our work is not misused for political gain.

5.3.3 Concluding reflections on the nature of artistic styles

In this work, we were able to show that it is possible to intersect artistic styles. However, it is not trivial to determine the extent to which a style intersection creates a new artistic style. To approach an answer, we should first be clear about the definition of "style".

For sure, one can argue that style represents, to a certain degree, what was aesthetically pleasing to a society at a given point in history. Machine learning applications are able to comprehend these observable characteristics, such as shapes, colors and contrasts.

However, as stated above, some, such as Fernie, see style as "[...] an expression of a collective spirit". [20] The latter reflects, for example, what is troubling the minds of a society. In the past, this might have been religion, but also the devasta-

tion caused by wars, the suffering from epidemics, the loss of individuality through industrialization or, in the present, topics such as feminism and environmental destruction. These realities shaped the artists and are, therefore, an essential part of their work. Machine learning cannot recognize these thoughts and emotions because it only sees the painting, not the stories that contextualize it.

In addition to that, style is not only composed of colors, shapes and contrasts, but invokes an iconography through which meaning is attributed to what is depicted. For example, historically, the lily indicates the virgin Mary. But, even though machine learning cannot (yet) represent the iconographic level of styles, it can still help us in this regard. The field of art history deals, among others, with the evolution and connections to other styles. However, one of the biggest downsides of interpreting art is our cultural bias. Through their education, art historians are equipped with a certain view on art, which is certainly not harming them, but also hinders them in their free thinking. In this specific regard, machines are less biased. In the past, some of the greatest discoveries were only made possible because researchers were able to develop independently of conventional teaching and thus achieved groundbreaking insights. Thus, similarities that machines are able to detect can be counter-intuitive to us, but they can also be viewed in a manner that is detached from our previous understanding and can provide valuable insights to previously hidden connections. As can be seen in figure 4.4 our approach was able to show a similarity between William Turner and Pierre-Auguste Renoir, both artists from two different epochs and countries. Nevertheless, this connection expanded our understanding of their relation, as both use a blurry brush stroke and barely hinting at elements of the painting. In this regard, machine learning can support as a kind of "research assistant".

Coming back to the original question of the extent to which style intersections create a new artistic style, we therefore conclude the following: Machine learning can produce new styles to the same extent that they can capture the complexity of style. For now, with this work, we know, that in terms of visually observable characteristics, they are able to create something new.

Bibliography

[1] Muhammad Ahmed, Omar Reyad, and Basma El-Rahiem. "An Efficient Deep Convolutional Neural Network for Visual Image Classification". In: Mar. 2019.

[2] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[3] Martin Arjovsky and Léon Bottou. *Towards Principled Methods for Training Generative Adversarial Networks*. 2017. arXiv: 1701.04862 [stat.ML].

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[5] DH Ballard, GE Hinton, and TJ Sejnowski. "Parallel visual computation". In: *Nature* 306.5938 (1983), pp. 21–26. URL: https://doi.org/10.1038/306021a0.

[6] Samuel A. Barnett. *Convergence Problems with Generative Adversarial Networks (GANs)*. 2018. arXiv: 1806.11382 [cs.LG].

[7] Yoshua Bengio et al. "Convex Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by Y. Weiss, B. Schölkopf, and J. Platt. Vol. 18. MIT Press, 2006. URL: https://proceedings.neurips.cc/paper/2005/file/0fc170ecbb8ff1afb2c6de48ea5343e7-Paper.pdf.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[9] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. "Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks". In: *Energy and Buildings* 158 (Nov. 2017).

[10] Augustin Cauchy et al. "Méthode générale pour la résolution des systemes d'équations simultanées". In: *Comp. Rend. Sci. Paris* 25.1847 (1847), pp. 536–538.

[11] Muthuraman Chidambaram and Yanjun Qi. *Style Transfer Generative Adversarial Networks: Learning to Play Chess Differently*. 2017. arXiv: 1702.06762 [cs.LG].

[12] Min Jin Chong and David A. Forsyth. "Effectively Unbiased FID and Inception Score and where to find them". In: *CoRR* abs/1911.07023 (2019). arXiv: 1911.07023. URL: http://arxiv.org/abs/1911.07023.

[13] *Convolution vs. Cross-Correlation*. [Online; accessed 7. Apr. 2022]. July 2019. URL: https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation.

[14] Chuong Do. "The Multivariate Gaussian Distribution". In: (2008). URL: https://cs229.stanford.edu/section/gaussians.pdf.

[15] D.C Dowson and B.V Landau. "The Fréchet distance between multivariate normal distributions". In: *Journal of Multivariate Analysis* 12.3 (1982), pp. 450–455. URL: https://www.sciencedirect.com/science/article/pii/0047259X8290077X.

[16] Alexei Efros, Alexei A. Efros, and Thomas K. Leung. "Texture synthesis by non-parametric sampling". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1033–1038 vol.2.

[17] *Fake-Video von Selenskyj im Umlauf*. [Online; accessed 25. Apr. 2022]. Mar. 2022. URL: https://www.zdf.de/nachrichten/video/panorama-fake-video-selenskyj-100.html.

[18] Stefan Falkner, Aaron Klein, and Frank Hutter. "BOHB: Robust and Efficient Hyperparameter Optimization at Scale". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 1437–1446. URL: https://proceedings.mlr.press/v80/falkner18a.html.

[19] Ky Fan. "Minimax Theorems". In: *Proceedings of the National Academy of Sciences of the United States of America* 39.1 (1953), pp. 42–47. URL: http://www.jstor.org/stable/88653.

[20] E. Fernie et al. *Art History and Its Methods*. Phaidon Press, 1995. URL: https://books.google.de/books?id=rZkYAQAAIAAJ.

[21] Frederic B. Fitch. "Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133." In: *Journal of Symbolic Logic* 9.2 (1944), pp. 49–50.

[22] L. A. Gatys, A. S. Ecker, and M. Bethge. "Image Style Transfer Using Convolutional Neural Networks". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2414–2423.

[23] Ian Goodfellow. *NIPS 2016 Tutorial: Generative Adversarial Networks*. 2017. arXiv: 1701.00160 [cs.LG].

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

[25] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[26] Alex Graves. "Generating Sequences With Recurrent Neural Networks". In: *CoRR* abs/1308.0850 (2013). arXiv: 1308.0850. URL: http://arxiv.org/abs/1308.0850.

[27] Eric Grinstein et al. "Audio Style Transfer". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 586–590.

[28] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028 [cs.LG].

[29] Richard H. R. Hahnloser et al. "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". In: 405.6789 (June 2000), pp. 947–951.

[30] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[31] Aaron Hertzmann et al. "Image Analogies". In: 2001, pp. 327–340.

[32] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018. arXiv: 1706.08500 [cs.LG].

[33] H. Heuser. *Lehrbuch der analysis*. Lehrbuch der analysis Bd. 1. Teubner, 2003. URL: https://books.google.de/books?id=ztGZltmfK3EC.

[34] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: (Apr. 1991).

[35] Sepp Hochreiter et al. "Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies". In: *A Field Guide to Dynamical Recurrent Neural Networks* (Mar. 2003).

[36] *How do neurons work?* [Online; accessed 6. Apr. 2022]. Nov. 2017. URL: https://qbi.uq.edu.au/brain-basics/brain/brain-physiology/how-do-neurons-work.

[37] Xun Huang and Serge J. Belongie. "Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization". In: *CoRR* abs/1703.06868 (2017). arXiv: 1703.06868. URL: http://arxiv.org/abs/1703.06868.

[38] Xun Huang et al. *Multimodal Unsupervised Image-to-Image Translation*. 2018. arXiv: 1804.04732 [cs.CV].

[39] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, pp. 448–456.

[40] Phillip Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. arXiv: 1611.07004 [cs.CV].

[41] Pavel Izmailov et al. "Averaging Weights Leads to Wider Optima and Better Generalization". In: *CoRR* abs/1803.05407 (2018). arXiv: 1803.05407. URL: http://arxiv.org/abs/1803.05407.

[42] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". In: *CoRR* abs/1603.08155 (2016). arXiv: 1603.08155. URL: http://arxiv.org/abs/1603.08155.

[43] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE].

[44] Tero Karras et al. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2018. arXiv: 1710.10196 [cs.NE].

[45] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014. URL: http://arxiv.org/abs/1412.6980.

[46] Naveen Kodali et al. *On Convergence and Stability of GANs*. 2018. URL: https://openreview.net/forum?id=ryepFJbA-.

[47] Dmytro Kotovenko et al. "Content and Style Disentanglement for Artistic Style Transfer". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2019.

[48] Dmytro Kotovenko et al. "Rethinking Style Transfer: From Pixels to Parameterized Brushstrokes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 12196–12205.

[49] Stanford Vision Lab. *A cartoon drawing of a biological neuron (left) and its mathematical model (right).* https://cs231n.github.io/neural-networks-1/. (Accessed on 20/09/2021). 2017.

[50] Stanford Vision Lab. *Convolutional Neural Networks (CNNs / ConvNets).* https://cs231n.github.io/convolutional-networks/. (Accessed on 20/09/2021). 2017.

[51] Stanford Vision Lab. *Neural Networks Part 2: Setting up the Data and the Loss.* https://cs231n.github.io/neural-networks-2/. (Accessed on 20/09/2021). 2017.

[52] Y. LeCun. "Generalization and network design strategies". In: 1989.

[53] Herbert Kui Han Lee III. *Model selection and model averaging for neural networks.* Carnegie Mellon University, 1998.

[54] Liam Li et al. *A System for Massively Parallel Hyperparameter Tuning.* 2020. arXiv: 1810.05934 [cs.LG].

[55] Yijun Li et al. "Universal Style Transfer via Feature Transforms". In: *CoRR* abs/1705.08086 (2017). arXiv: 1705.08086. URL: http://arxiv.org/abs/1705.08086.

[56] Hanwen Liang, Konstantinos N. Plataniotis, and Xingyu Li. *Stain Style Transfer of Histopathology Images Via Structure-Preserved Generative Learning.* 2020. arXiv: 2007.12578 [eess.IV].

[57] Jae Hyun Lim and Jong Chul Ye. *Geometric GAN.* 2017. arXiv: 1705.02894 [stat.ML].

[58] Ming-Yu Liu et al. "Few-Shot Unsupervised Image-to-Image Translation". In: *IEEE International Conference on Computer Vision (ICCV).* 2019.

[59] Mario Lucic et al. *Are GANs Created Equal? A Large-Scale Study.* 2018. arXiv: 1711.10337 [stat.ML].

[60] Xuan Luo et al. *Time-Travel Rephotography.* 2020. arXiv: 2012.12261 [cs.CV].

[61] Chunwei Ma, Zhanghexuan Ji, and Mingchen Gao. "Neural Style Transfer Improves 3D Cardiovascular MR Image Segmentation on Inconsistent Data". In: *CoRR* abs/1909.09716 (2019). arXiv: 1909.09716. URL: http://arxiv.org/abs/1909.09716.

[62] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing.* 2013.

[63] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[64] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *CoRR* abs/1411.1784 (2014). arXiv: 1411.1784. URL: http://arxiv.org/abs/1411.1784.

[65] Takeru Miyato et al. *Spectral Normalization for Generative Adversarial Networks*. 2018. arXiv: 1802.05957 [cs.LG].

[66] Lukas Mosser, Olivier Dubrule, and Martin Blunt. "Stochastic Reconstruction of an Oolitic Limestone by Generative Adversarial Networks". In: *Transport in Porous Media* 125 (Oct. 2018).

[67] Thomas Müller et al. "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding". In: *arXiv:2201.05989* (Jan. 2022).

[68] J.F. Nash. "Non-cooperative Games". In: *Annals of Mathematics* 54.2 (1951), pp. 286–295.

[69] John von Neumann, Oskar Morgenstern, and Ariel Rubinstein. *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton University Press, 1944. URL: http://www.jstor.org/stable/j.ctt1r2gkx.

[70] *Obama Deep Fake*. [Online; accessed 14. Mar. 2022]. Mar. 2022. URL: https://ars.electronica.art/center/de/obama-deep-fake.

[71] Björn Ommer. "Deep Vision, Lecture Slides, ANNs - The basics". In: (2020).

[72] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. Vol. 1. MIT Press Books 0262650401. The MIT Press, Aug. 1994. URL: https://ideas.repec.org/b/mtp/titles/0262650401.html.

[73] Yoshihiko Ozaki et al. "Multiobjective Tree-Structured Parzen Estimator for Computationally Expensive Optimization Problems". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. GECCO '20. Cancún, Mexico: Association for Computing Machinery, 2020, pp. 533–541. URL: https://doi.org/10.1145/3377930.3389817.

[74] Pratikkumar Prajapati and Chris Pollett. *MRI-GAN: A Generalized Approach to Detect DeepFakes using Perceptual Image Assessment*. Feb. 2022.

[75] Sarah Price. *Convolution in Two Dimensions*. [Online; accessed 17. Feb. 2022]. Jan. 2004. URL: https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/space/convol.htm.

[76] Joao Regateiro and Edmond Boyer. *3D Human Shape Style Transfer*. 2021. arXiv: 2109.01587 [cs.CV].

[77] Gerhard Reinelt. "Effiziente Algorithmen 1". In: *sript* University of Heidelberg, Institute of Computer Science (2019).

[78] Elissa M. Robbins et al. "SynCAM 1 Adhesion Dynamically Regulates Synapse Number and Impacts Plasticity and Learning". In: *Neuron* 68.5 (2010), pp. 894–906. URL: https://www.sciencedirect.com/science/article/pii/S0896627310009104.

[79] Herbert E. Robbins. "A Stochastic Approximation Method". In: *Annals of Mathematical Statistics* 22 (2007), pp. 400–407.

[80] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. "SinGAN: Learning a Generative Model from a Single Natural Image". In: *Computer Vision (ICCV), IEEE International Conference on*. 2019.

[81] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Neurocomputing: Foundations of Research*. Cambridge, MA, USA: MIT Press, 1988, pp. 696–699.

[82] Olga Russakovsky et al. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV].

[83] Arda Sahiner et al. "Hidden Convexity of Wasserstein GANs: Interpretable Generative Models with Closed-Form Solutions". In: *CoRR* abs/2107.05680 (2021). arXiv: 2107.05680. URL: https://arxiv.org/abs/2107.05680.

[84] Kuniaki Saito, Kate Saenko, and Ming-Yu Liu. "COCO-FUNIT: Few-Shot Unsupervised Image Translation with a Content Conditioned Style Encoder". In: *arXiv preprint arXiv:2007.07431* (2020).

[85] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG].

[86] Artsiom Sanakoyeu et al. "A Style-Aware Content Loss for Real-time HD Style Transfer". In: *CoRR* abs/1807.10201 (2018). arXiv: 1807.10201. URL: http://arxiv.org/abs/1807.10201.

[87] Axel Sauer et al. "Projected GANs Converge Faster". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[88] Bogdan Savchynskyy. "Discrete Graphical Models — An Optimization Perspective". In: *Foundations and Trends in Computer Graphics and Vision* 11.3-4 (2019), pp. 160–429. URL: http://dx.doi.org/10.1561/0600000084.

[89] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[90] Thomas Strothotte and Stefan Schlechtweg. "Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation". In: *Morgan Kaufmann* (Jan. 2002).

[91] Hang Su and Haoyu Chen. "Experiments on Parallel Training of Deep Neural Network using Model Averaging". In: *CoRR* abs/1507.01239 (2015). arXiv: 1507.01239. URL: http://arxiv.org/abs/1507.01239.

[92] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].

[93] Matthew Tancik et al. "Block-NeRF: Scalable Large Scene Neural View Synthesis". In: *arXiv* (2022).

[94] Ilya Tolstikhin et al. *Wasserstein Auto-Encoders*. 2019. arXiv: 1711.01558 [stat.ML].

[95] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *CoRR* abs/1607.08022 (2016). arXiv: 1607.08022. URL: http://arxiv.org/abs/1607.08022.

[96] Berkeley University of California. *CS294-158-SP20 Deep Unsupervised Learning*. https://sites.google.com/view/berkeley-cs294-158-sp20/home. (Accessed on 20/09/2021). 2020.

[97] Leni Ven and Johannes Lederer. *Regularization and Reparameterization Avoid Vanishing Gradients in Sigmoid-Type Networks*. 2021. arXiv: 2106.02260 [cs.LG].

[98] Han Zhang et al. *Self-Attention Generative Adversarial Networks*. 2019. arXiv: 1805.08318 [stat.ML].

[99] Shengyu Zhao et al. *Differentiable Augmentation for Data-Efficient GAN Training*. 2020. arXiv: 2006.10738 [cs.CV].

[100] Bolei Zhou et al. "Places: A 10 million Image Database for Scene Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).

Appendices

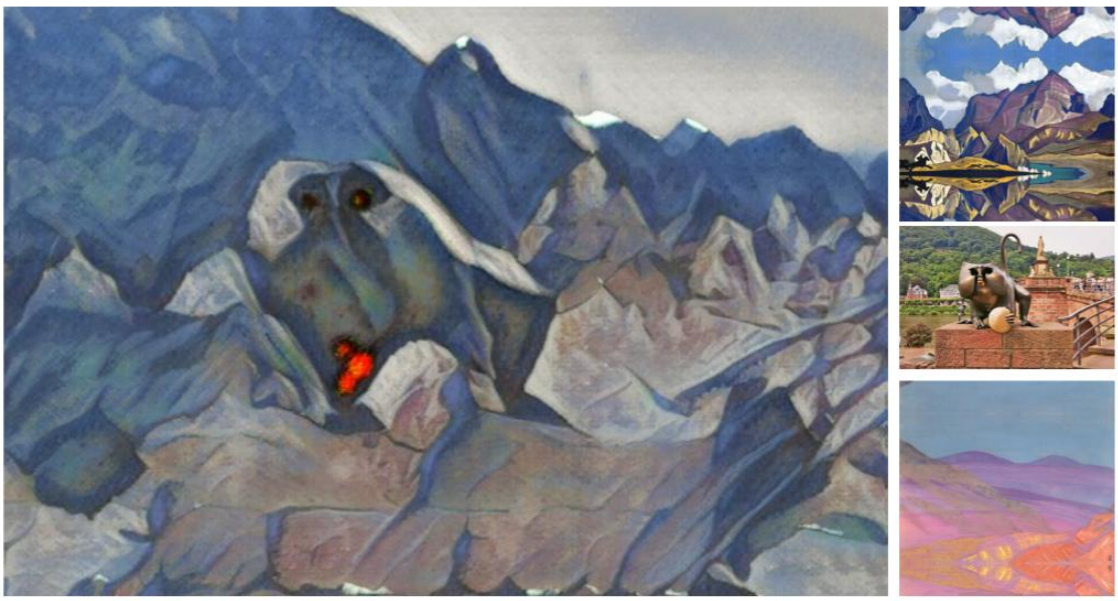


Figure A.1: Model D: Image stylized in the style of Nicholas Roerich demonstrating a numerical instability. Right in the middle, the numerical instability takes the form of a red, texture free hole.

Figure A.2: Model D: Image stylized in the style of Pablo Picasso.

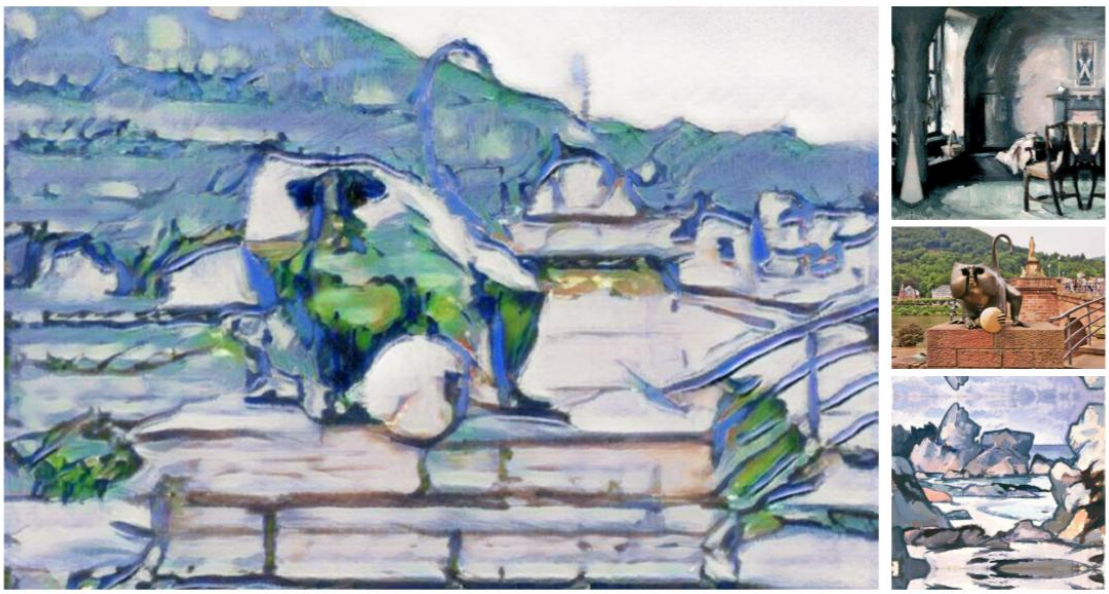
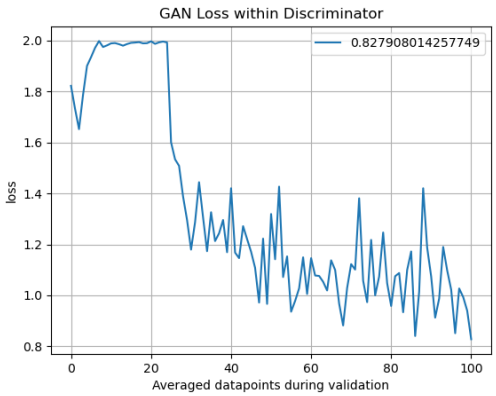


Figure A.3: Model D: Image stylized in the style of Samuel Peploe.

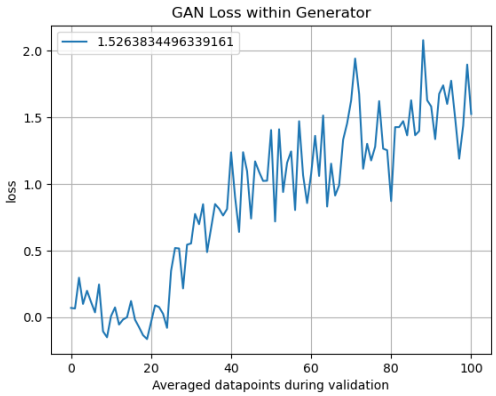
Figure A.4: Model D: Image stylized in the style of Pablo Picasso and Édouard Manet.

Figure A.5: Model D: Image stylized in the style of Jackson Pollock and Ernst Ludwig Kirchner.

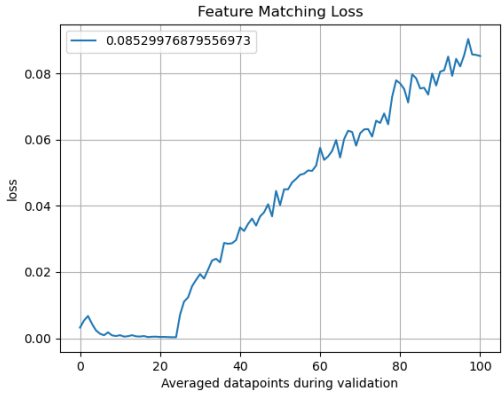
Figure A.6: Model D: Image stylized in the style of Camille Pissarro and Pablo Picasso.



(a) $\mathcal{L}_{GAN}^{\mathcal{D}}$



(b) $\mathcal{L}_{GAN}^{\mathcal{G}}$



(c) $\mathcal{L}_F$

Figure A.7: Model C suffers from an early convergence in $\mathcal{D}$, from which it was able to recover.

Figure A.8: Evaluation results while using $\lambda_R = 0.1$