Department of Physics and Astronomy
Ruprecht Karl University of Heidelberg

Bachelor Thesis in Physics
submitted by

Tim Küchler

born in Heidelberg (Germany)

2021

Semantic Image Synthesis with Score-Based Generative Models

This Bachelor Thesis has been carried out by Tim Küchler
at the
Heidelberg Collaboratory for Image Processing
and the
Interdisciplinary Center for Scientific Computing
under the supervision of
Prof. Dr. Björn Ommer and Prof. Dr. Tilman Plehn

Acknowledgements

This thesis is the grand finale of a three-year adventure with many ups and downs. But you cannot go through an adventure like this alone. It takes emotional support, competent advice and a lot of great moments of togetherness to finally reach the destination on such a journey. And finally being able to say: You have made it!

I thank my family, my friends, my supervisor Robin Rombach and everyone else who believed in me and was there for me when I needed them. Moreover, I thank all persons who did not get the attention they deserved during this time and yet were patient with me. At the end of the day I can only say one thing:

Without you all, this adventure would have been a different one!

Abstract

Semantische Bildsynthese mit Score-basierten, generativen Modellen:

Die künstliche Bilderzeugung ist ein zukunftsweisendes Forschungsgebiet der modernen Informatik. Wenn die semantische Bildinformation des generierten Bildes beeinflusst werden soll oder die vorgesehene Aufgabe eine pixelweise Kontrolle über das Ergebnis erfordert, wird semantische Bildsynthese benötigt. In dieser Arbeit zeigen wir einen Ansatz, der das neue und vielversprechende Konzept der Score-basierten generativen Modelle für die semantische Bildsynthese verwendet. Dazu trainieren wir zunächst ein rauschbedingtes Score-Netz auf einem Datensatz, das dann mit einem sorgfältig entworfenen rauschbedingten semantischen Segmentierungsnetz kombiniert wird, welches auf demselben Datensatz trainiert wurde. Anschließend vergleichen wir unsere Ergebnisse auf dem Cityscapes-Datensatz [5] quantitativ mit State-of-the-Art-Modellen wie CRN [4], pix2pixHD [44] und SPADE [26] unter Verwendung des FID-Scores sowie der Pixelgenauigkeit und der mittleren IoU. Darüber hinaus generieren wir hochauflösende Landschaftsbilder unter Verwendung von Bildern, die wir von der Bildplatform Flickr gesammelt haben. Abschließend zeigen wir die Grenzen unseres Ansatzes am ADE20K-Datensatz [49] und weisen auf Herausforderungen hin, deren Lösung in der zukünftigen Forschung die semantische Bildsynthese mit Score-basierten generativen Modellen zu einem leistungsfähigen, vielseitigen und einfach zu verwendenden Framework machen könnte.

Semantic Image Synthesis with Score-Based Generative Models:

Artificial image generation is a cutting-edge field of research in modern computer science. If the semantic image information of the generated image is to be influenced or the envised task requires pixel-wise control over the result, semantic image synthesis is needed. In this thesis, we show an approach using the new and promising concept of score-based generative models for semantic image synthesis. For this, we first train a noise-conditional score-network on the dataset, which is then combined with a carefully designed noise-conditional semantic segmentation network trained on the same dataset. We then quantitatively compare our results on the Cityscapes dataset [5] with state-of-the-art models like CRN [4], pix2pixHD [44] and SPADE [26] using FID scores as well as pixel accuracy and mean IoU. Furthermore, we generate high resolution landscape images using images scraped from Flickr. Finally, we show the limitations of our approach on the ADE20K dataset [49] and point out challenges whose solution in future research could make semantic image synthesis with score-based generative models a powerful, versatile and easy-to-use framework.

Contents

1. Introduction

1.1. Generating data

We as humans have been born as intelligent living beings. We use this intelligence to create things of indescribable creativity and complexity. All the more understandable is the drive to understand this creative process and to reproduce it artificially. And it is this drive that gave rise to a new branch of research in computer sciences, which finally led to the concept of *Generative Modeling*.

Generative Modeling is an umbrella term describing artificial approaches to generate realistic data. The current state-of-the-art approaches are dominated by so called *Deep Neural Networks*, complex artificial constructs which are partially inspired by the human brain. These networks are implemented as a computer program, using modern machine learning libraries such as *PyTorch* [27] or *TensorFlow* [1].

In the last years, generative modeling has seen **immense** progress in generating the most diverse types of data. Some results have even reached such a quality that they can hardly be distinguished from realistic data for the human eye (see e.g. Fig. 1.1). Some fields of generative modeling on which outstanding results have been achieved include image generation, video generation and audio and speech generation. In addition to the general push for better generation models, some models are already indispensable for various application purposes such as image processing, anomaly detection in medical context and generation of new data for scientific uses, e.g. the artificial generation of promising molecular candidates for the development of new effective drugs [46].

Figure 1.1.: *Top Left*: Depth-to-Image, *Top Right*: Semantic synthesis, Images from [8] *Bottom*: Class-conditional samples, Images from [3]

1.2. Score-Based Generative Models - The new contenders to GANs?

"Creating noise from data is easy; creating data from noise is generative modeling" [35]. This quote describes the operating principle of a novel model for generating data (Sec. 1.1), which we will explore in various ways in this paper.

The so-called *Score-Based Generative Models* (SGMs) were recently proposed by Yang Song and Stefano Ermon [34] in late 2020. As a generative model SGMs can be used to generate all kinds of data such as images, audio and graphs. They are capable of performing a variety of special tasks such as inpainting, colorization and image-to-image translation. As shown in outstanding results from recent work [35] they are capable to keep up with state-of-the-art generative models such as Generative Adversarial Networks (GANs) [10] and Variational Autoencoders (VAEs) [21]. Therefore, SGMs have gained great attention in the scientific community and are already considered by some to be the "new contenders to GANs" [17].

Score-Based Generative Models work by estimating the score of a given data distribution via score matching [14]. Because the score estimating technique is unstable or ill-defined for a real data distribution, the initial images are perturbed by Gaussian noise. Starting from random noise, SGMs are able to generate completely new data.

SGMs have several advantages to other popular models. They do not rely on adversarial training, which is often unstable, they do not need a special architecture in order to be

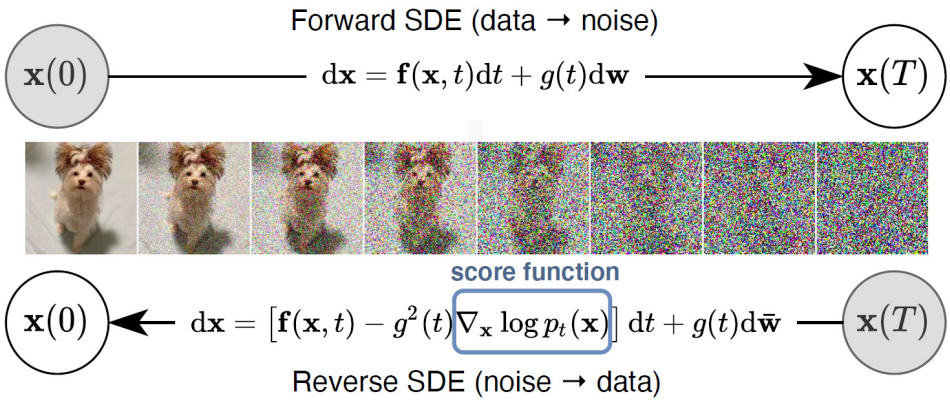


Figure 1.2.: The idea of SGMs: First the data distribution is perturbed via a diffusion process governed by a SDE. The scores of the noise distributions are learned by the score model and are then used to solve a reverse SDE to generate new data from noise. Figure from [35]

tractable, they do not need sampling during training, and they can be used for different tasks such as inpainting, class-conditional generation and colorization without the need to retrain the model. Although SGMs are very slow in sampling, they have the advantage that there is an arbitrary possibility of combinations of sampling techniques and model architectures, allowing the search for a best-possible sampling procedure. All in all, these properties, together with the high quality of the recent results, make SGMs a promising new model for generative modeling.

1.3. Semantic Segmentation and Semantic Synthesis

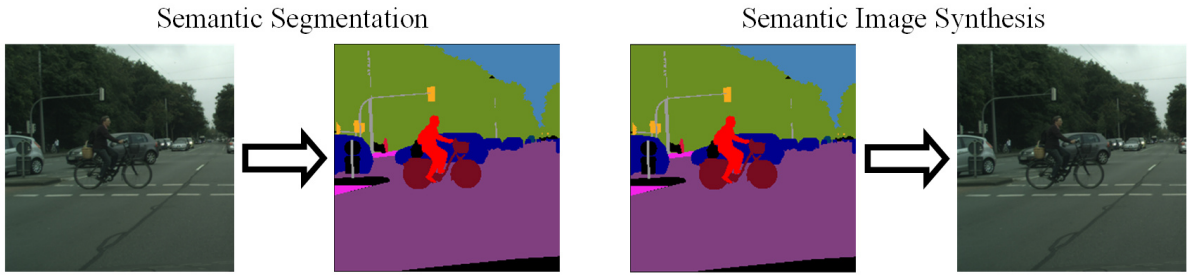


Figure 1.3.: Operating principle of semantic segmentation and semantic image synthesis. Images from Cityscapes dataset [5]

Two large fields in image-based machine learning are semantic segmentation and semantic image synthesis. In semantic segmentation so-called semantic segmentation networks have

the task of a pixel-wise classification of images. In semantic image synthesis – unlike unconditioned image generation [35] – images are generated based on a given semantic label map.

In this thesis we show that Score-Based Generative Models, with the help of semantic segmentation networks, are capable of synthesizing realistic looking images based on semantic label maps for various resolutions up to 1024×512 pixels and datasets such as Cityscapes [5], ADE20K [49] and landscape images scraped from Flickr. We show that the synthesized images for some categories can keep up with state-of-the-art generative models such as CRN [4], pix2pixHD [44], and SPADE [26], although there is still room for further improvement. We therefore point out how the current technique could be optimized in future work to yield even better results.

The implementation for this work is done with PyTorch and the code is publicly available on GitHub at the following links:

- `https://github.com/TimK1998/SemanticSynthesisForScoreBasedModels`
- `https://github.com/TimK1998/SemanticSegmentation`

2. Background

2.1. Neural Networks

2.1.1. The Perceptron

The basis of each artificial neural network is the *perceptron*. A perceptron essentially is an algorithm that is inspired by biological neurons. The perceptron can therefore be understood as an artificial neuron.

Mathematically, a perceptron processes an input vector $\boldsymbol{x} \in \mathbb{R}^n$ and produces an output vector $\boldsymbol{y} \in \mathbb{R}^m$. In the simplified representation of a perceptron in Fig. 2.1 the inputs x_i $(i = 0, \ldots, n)$ are multiplied by weights w_i and then summed. Thereafter a non-linear function is applied to produce the output $y \in \mathbb{R}$. It is the adaption of these weights to a specific problem that is the core of learning in a neural network. A generalization of the transformation described above is given by

$$\boldsymbol{A} \cdot \boldsymbol{x} + \boldsymbol{b} = \hat{\boldsymbol{y}} \tag{2.1}$$

where $A \in \mathbb{R}^{m \times n}$ is a matrix of weights and $b \in \mathbb{R}^m$ is a bias. Then a non-linear transform-

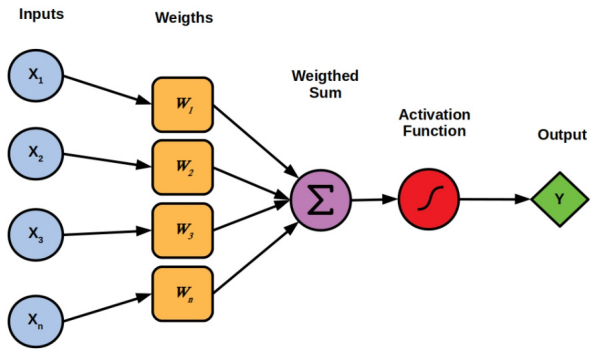


Figure 2.1.: Schematic layout of a perceptron. Figure from https://starship-knowledge.com/neural-networks-perceptrons.

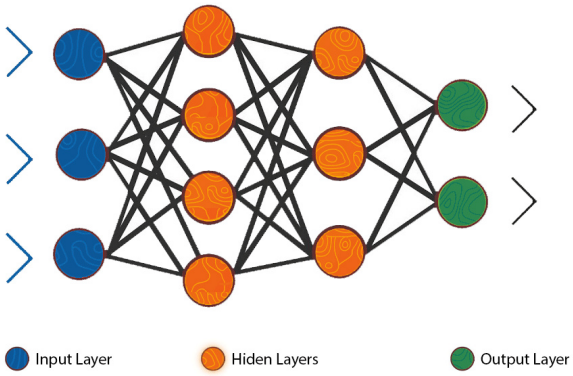


Figure 2.2.: The Multilayer Perceptron. Figure from https://cdn.analyticsvidhya.com/wp-content/uploads/2020/02/ANN-Graph.gif

ation of the form

$$\boldsymbol{y} = \varphi(\hat{\boldsymbol{y}}) \tag{2.2}$$

is applied to the intermediate result $\hat{\boldsymbol{y}}$. The non-linear function $\varphi(\cdot)$ is called the *activation function*. This function has the task of mapping the arbitrary output $\hat{\boldsymbol{y}}$ from the range $(-\infty, \infty)$ to a more appropriate range. The choice of the activation function depends heavily on the specific architecture of a neural network. Typical choices of activation functions are the Sigmoid, Tanh, Softmax and ReLU functions. As activation functions do not play a big role in this thesis the concrete definition of these functions is left as an exploration to the reader.

2.1.2. Multilayer Perceptrons

A *Multilayer Perceptron* is an arrangement of perceptrons in different consecutively interconnected layers. This is what we call a (very basic) *Neural Network*. As depicted in Fig. 2.2 a Multilayer Perceptron is composed of an input layer, one or more hidden layers and an output layer. The output of a layer is connected to the input of the next layer (here: fully connected). The hidden layers can perform various tasks and some of them are presented in Sec. 2.1.3.

During training the learning process of such a neural network consists of three parts which are computed iteratively: the *forward pass*, the *loss calculation* and the *backward pass*. In the forward pass, the input values get processed by the different layers of the neural networks to then produce an output. The loss calculation then computes how good this output is in relation to some optimal output for the given input. This is the concept of *supervised*

learning. There is a variety of so-called *loss functions* for the calculation of losses, some of them will be discussed later in this thesis. The computed loss is then back-propagated through the network to update the parameters of each perceptron in each layer. This is the process of artificial learning. Back-propagation often works in the sense of a gradient descent. Since the loss is to be minimized, the influence of all individual parameters on the loss is then derived and adjusted according to the resulting gradient.

Usually a batch of data is processed simultaneously by the network. While processing the whole dataset at once would lead to faster converging and overall better results in optimization with gradient decent, a larger batch size comes with a much higher vRAM cost during training. Therefore one often decides to use a mini-batch of data which makes the best possible use of the memory of the graphics card. The process of forward pass, loss computation and backward pass of one batch is then called an *iteration* or a *step*. After each batch has been processed in a dataset, an *epoch* has passed.

2.1.3. Common Layers in Neural Networks

Depending on the task of the neural network, different layers of perceptrons are used. The most common ones are *Fully Connected Layers*, *Normalization Layers*, *Convolutional Layers*, *Pooling Layers* and *Dropout Layers*. It is important to note that the choice of layers for a specific problem is **not** trivial! In fact, often it is not even clear why a specific layer is good for a specific task. Furthermore, when training a neural network, it is – in general – impossible to have any insight on how the neural network learned the task.

Fully Connected Layers

A fully connected layer is the easiest layer for neural networks. In this layer, the output of every perceptron in one layer is connected to the input of every perceptron in the next layer. Mathematically, this describes a linear operation between the perceptrons of two layers. Although fully connected layers are in theory able to fit every problem quite well (see Universal Approximation Theorem, e.g. [7]) ,they cannot be used excessively. Considering a 128×128 pixel RGB image leads to $4,831,838,208$ parameters to learn for two fully connected layers. Learning such a high number of parameters (weights) is computationally very expensive and often not feasible.

Normalization Layers

Normalization layers normalize the given inputs. There are several normalization techniques that depend on the choice of what to normalize. Take Batch Normalization (BN) [15] as an example. Suppose the image data is of the form (N, C, W, H) where N is the batch size, C is the channel number (e.g. 3 for RGB images) and W and H are width and height respectively. Then BN normalizes (N, W, H) for each C by transforming the input $\boldsymbol{x}$ in the following way:

$$\mu_B = \frac{1}{N}\sum_{i=1}^{N} x_i, \quad \sigma_B^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_B)^2 \quad \Longrightarrow \quad y_i = \gamma\frac{x_i - \mu_B}{\sigma_B} + \beta, \tag{2.3}$$

where γ and β are learnable parameters of the layer. Depending on the dimensions of the input that get normalized there is also *Channel Norm*, *Instance Norm* and *Group Norm*. Normalization layers are used to improve the stability, speed and performance of neural networks.

Convolutional Layers

The Convolution Layer might be the most important layer in computer vision and gives rise to the category of *Convolutional Neural Networks* (CNNs) [6]. This type of neural networks makes extensive use of convolutional layers and is an important concept in computer vision tasks as it is very good at capturing image details [40].

A Convolutional Layer is especially useful to extract features from images. For example, these features could be lines in different direction but as mentioned above, normally we do not know what information (features) a neural network learns in a (convolutional) layer. In general, one can only say that in a CNN of subsequent Convolutional Layers the upper layers learn more simple features such as lines and the deeper layers learn more complex features, e.g. how a car looks like. This characteristic is called *receptive field* and is due to the fact that a pixel of the first convolutional layer contains information only from, for example, 9 pixels of the image for a 3×3 convolution, while a deeper convolutional layer contains information from many more pixels of the image.

As the Convolutional Layer is most often used for image feature extraction, the *2D-Convolution Layer* is the most popular Convolution Layer and is defined by a 2-dimensional

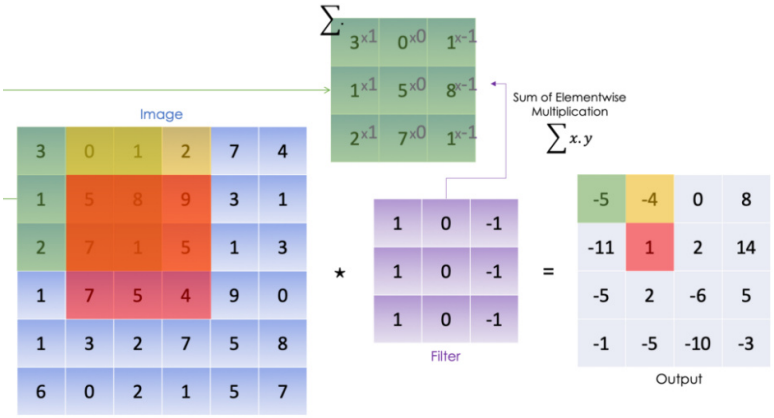


Figure 2.3.: Operation principle of Convolutional Layers. Figure from https://towardsdatascience.com/convolutional-neural-networks-mathematics-1beb3e6447c0.

convolution

$$y_{i,j} = (\boldsymbol{x} * \boldsymbol{f})_{i,j} = \sum_{c=1}^{C} \sum_{h=1}^{H_f} \sum_{w=1}^{W_f} \boldsymbol{f}_{c,h,w} \boldsymbol{x}_{i+h-1,j+w-1,c} \tag{2.4}$$

for an input of size (C, H, W). Essentially this operation can be understood as applying a filter $\boldsymbol{f}$ of size (C, H_f, W_f) to each part of the image. For each position of the filter the corresponding values in the image are multiplied with the learned weights in the filter and then summed up. These summed values for each filter position then form a new output. An illustration of this process is shown in Fig. 2.3.

Pooling Layers

As we have seen, Convolutional Layers summarize and learn specific features in an input image. These feature maps are very sensitive to the location of the features in the input. To make the feature maps more robust to changes in the position of the feature in the image, Pooling Layers are deployed. A Pooling Layer works by dividing the feature map into slices of size $n \times m$. These slices are then condensed to a single scalar value by a pooling operation. Popular pooling operations are *max pooling* and *average pooling*, which given a

slice $\tilde{\boldsymbol{x}} = (x_{ij})$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$ compute the following:

$$\text{max pooling:} \quad f(\tilde{\boldsymbol{x}}) = \max_{i,j}(x_{ij}) \tag{2.5}$$

$$\text{average pooling:} \quad f(\tilde{\boldsymbol{x}}) = \frac{1}{n \cdot m} \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} \tag{2.6}$$

It should be noted that pooling layers are often also used as downsampling layers.

Dropout ~~Layers~~

Dropout can be applied to any other layer type and has no learnable parameters. Dropout therefore is no real layer. During training, specified by a dropout probability p, parameters are randomly set to 0. The purpose of dropout is to prevent model overfitting, i.e. preventing the model from memorizing instead of learning a dataset.

2.1.4. Model Objectives

The concept of *model* is very important in Deep Learning. A model can be understood as a superordinate concept to a concrete network implementation. We denote a model as $s_\theta(\boldsymbol{x})$, where $\boldsymbol{x}$ is the data input and θ is the set of all learnable parameters.

To define the task of an model, an *objective* is set up. An objective is a function that represents the task the model is trying to accomplish. Usually, the task is represented such that the objective function is to be minimized or maximized. As an example, imagine a model that has the task to approximate (learn) the function $f(x) = x^2$. The objective of this task can be chosen as

$$\theta^* = \arg\min_\theta \left\| x^2 - s_\theta(x) \right\|. \tag{2.7}$$

The notation is interpreted in the following way: The optimal parameters θ^* are the ones that minimize the distance between x^2 and the model $s_\theta(x)$. This gives rise to the definition of the optimal model $s_{\theta^*}(\boldsymbol{x})$ which by definition solves the task perfectly. When describing the learning process of a model it is then stated that the trained model fulfills $s_\theta(\boldsymbol{x}) \approx s_{\theta^*}(\boldsymbol{x})$ or in the case of our example $s_\theta(x) \approx s_{\theta^*}(x) \overset{!}{=} x^2 \, \forall x \in \mathbb{R}$.

If the model is dependent on additional information, e.g. time t, the model is expanded to $s_\theta(\boldsymbol{x}; t)$.

2.2. Data Distributions and Probability Density Functions

Statistical Data Distributions and Stochastic Processes play a large role in the theory behind Score-Based Generative Models. When describing the goal of a model it is often said that the model tries to learn the data distribution of a dataset. The data distribution $p_{data}(\boldsymbol{x})$ of a dataset $D \subset \mathbb{R}^d$ describes how the data $\boldsymbol{x} \in D$ is distributed in relation to certain descriptive variables. The data distribution describes *all* possible data, e.g. all images of cars. Obviously, a real dataset cannot contain an infinite number of images. That is why the data distribution often is impossible to know. The real dataset has the following relation to the data distribution: $D = \{\boldsymbol{x}_i\}_{i=1}^N \overset{i.i.d}{\sim} p_{data}(\boldsymbol{x})$. The $\sim$ means that D is distributed as p_{data} and N denotes the finite number of data in the dataset. $D \subset D^*$ where D^* is the infinite, perfect dataset. In order for the model being able to learn the data distribution from a subset of D^*, the data in D must be *independently and identically distributed* (i.i.d) which essentially means that D should replicate the data distribution as good as possible.

Each distribution can also be represented as a *probability density function* (pdf) $p(\boldsymbol{x})$. While a data distribution shows the frequency of certain variables and often is discrete a pdf is a continuous function describing the probability of the variables in a distribution.

3. Related Work

3.1. Semantic Segmentation

A large field of models in Deep Learning are discriminative models. A simple version of a discriminative model would be a classifier. A classifier given an input $\boldsymbol{x}$ outputs a scalar class value $y \in \boldsymbol{y}$ of the input, where $\boldsymbol{y}$ is a set of classes (which are also called labels). The simplest classifier would be a binary classifier that places an input into one of two classes, e.g. deciding if an image shows a cat or a dog. More advanced classifiers identify way more classes. A classical example would be a classifier trained on the CIFAR-10 dataset [22], which contains millions of 32×32 pixel images categorized by 10 classes (car, airplane, dog, ...).

The above described classifiers always assign one scalar value to an input. Semantic Segmentation Networks also assign scalar values, but not one value per image, but one value per pixel! This already reveals a lot about the application of such networks. They operate on images and their task is to assign each pixel in the image a class. For example, given the image of a street scene a semantic segmentation network tries to classify each individual pixel as part of a car, a street, a building, ...

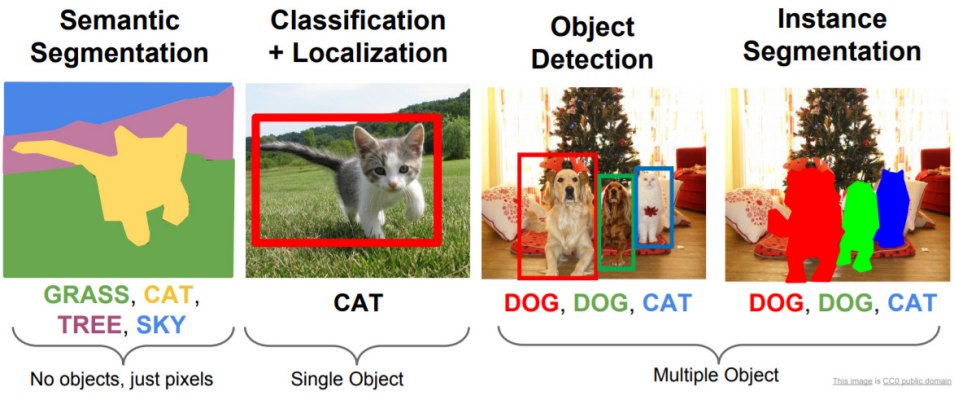


Figure 3.1.: Overview of various computer vision tasks. Figure from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf.

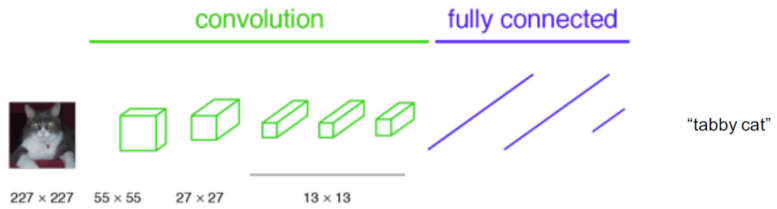


Figure 3.2.: The architecture of classification networks such as VGGNet. Figure from https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1.

There are also modifications of semantic segmentation, for example instance segmentation. In instance segmentation, not only is each pixel assigned a class, but also the different instances of objects in an image are specified. An overview of semantic segmentation and related techniques is shown in Fig. 3.1

As we will see in Sec. 4.5 and Chapter 5, semantic segmentation networks are necessary to make semantic image synthesis possible with Score-Based Generative Models. Therefore, the semantic segmentation models used in later work are briefly explained below:

3.1.1. FCN

FCN [32] stands for *Fully Convolutional Network* and is a semantic segmentation network whose architecture is inspired by successful image classification networks such as VGGNet [33].

VGGNet in principle works by first downsampling an image with several convolution layers, followed by a few fully connected layers. The last fully connected layer then gives the probabilities of the image being of a certain class. An overview of this architecture is presented in Fig. 3.2.

FCN builds on this architecture by replacing the fully connected layers with 1×1 convolution

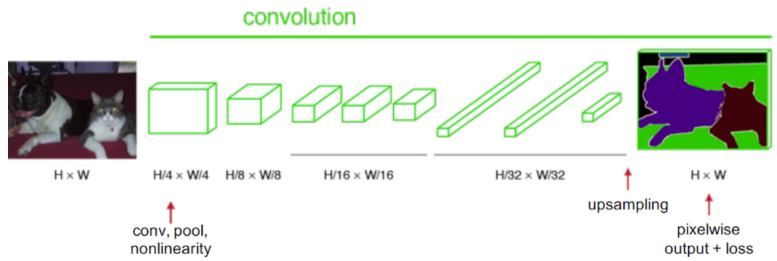


Figure 3.3.: The FCN architecture. Figure from [32].

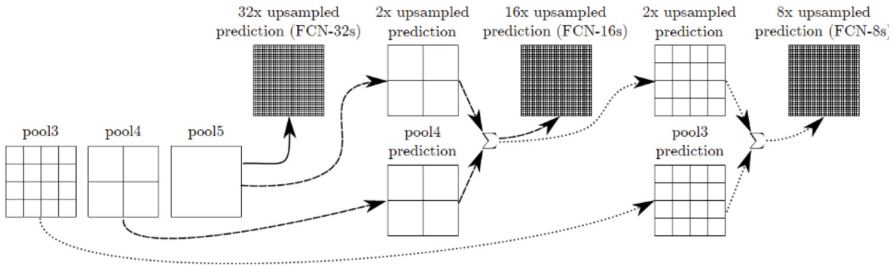


Figure 3.4.: Upsampling by successively combining the output of all max pooling layers. Image from [32]

layers. The entire architecture – seen in Fig. 3.3 – then consists of successive blocks operating on different fractions of the original resolution. Each block consists of a convolution layer, a max pooling layer and an activation function. To move from the pixel-wise $H/32 \times W/32$ prediction to a prediction of the original resolution, an upsampling process (Fig. 3.4) is deployed. The problem with this is that the lower resolutions are needed to capture fine details in the original picture, but the spatial location of these features is almost completely lost. Therefore, the outputs after the pooling layers for *each* resolution are consecutively combined during the upsampling procedure to get back the spatial information. These cross connections between the downsampling and the upsampling part of the network are called *skip connections*.

3.1.2. U-Net

U-Net [30] is an adaption of the fully convolutional structure of FCNs (Sec. 3.1.1) and initially was designed for applications in biomedical image segmentation. Looking at the network architecture in Fig. 3.5 it quickly becomes clear where the name U-Net originates from.

As with FCNs the U-Net architecture consists of a downsampling part and an upsampling part which are connected by skip connections. The main modification in U-Net is that the upsampling part has a lot more feature channels than in FCNs which allows the network to better propagate feature information to higher resolution. In general, the U-Net achieved better results than FCN and needs a lot less training samples to achieve good predictions [30].

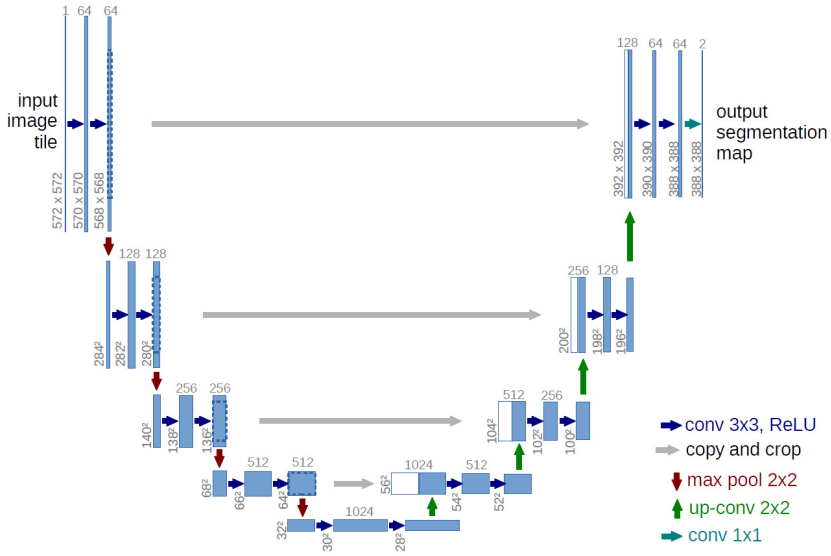


Figure 3.5.: The U-Net architecture. Figure from [30].

3.2. Generative Modeling – VAEs and GANs

There are mainly two types of models: *Generative Models* and *Discriminative Models*. The latter discriminate between different kind of data instances, e.g. discriminate images of cats and dogs. Semantic Segmentation Models (Sec. 3.1) are discriminative models. Given a data instance $\boldsymbol{x}$ and a set of labels $\boldsymbol{y}$ discriminative models learn the conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$. Generative models generate new data instances. They capture the joint probability $p(\boldsymbol{x}, \boldsymbol{y})$ or $p(\boldsymbol{x})$ if there are no labels. In general, generative models are a lot harder than discriminate models. There are various approaches to generative modeling, the most popular being Variational Autoencoders (VAEs) [21] and Generative Adversarial Networks (GANs) [10]:

Variational Autoencoders

Although VAEs [21] do not play a major role in this thesis, they are briefly discussed due to their strong influence in modern generative modeling. In order to understand VAEs one must first understand what an autoencoder is. Autoencoders tackle the problem of encoding and decoding data with minimum information loss. Data generally is described by some abstract features which often form a high dimensional space. The task of an autoencoder is to learn to reduce the dimensionality of this high dimensional features by selecting important old features (selection) or creating less, new features based on the old features (extraction).

The arising feature space is called *latent space* which essentially only contains the most important features of the input data. To learn such an encoding, the autoencoder network consists of an encoder network $E(\boldsymbol{x})$ and a decoder network $D(\boldsymbol{x})$. An input $\boldsymbol{x}$ is first encoded by the encoder to a low dimensional value $\boldsymbol{z}$ which then is decoded by the decoder to an output $\tilde{\boldsymbol{x}}$ of the same dimension as the input. Thereafter, the input $\boldsymbol{x}$ is compared to the output $\tilde{\boldsymbol{x}}$ and the network gets punished for differences between input and output.

An autoencoder thus learns to compress and decompress data in the best possible way without loss of information. A naïve way to now generate new data via a trained autoencoder is to use the decoder to decode a random sample from the latent space. The problem with this approach is that the autoencoder learns to best possible compress the data. As a consequence, we cannot sample from the latent space because the distribution in latent space has no meaning w.r.t the real data. For example, if you decode the image of a car into the latent space and then change the latent representation of the car just a little, after decoding you will most likely see noise instead of an image of a new car. In mathematical terms this means that the latent space of an autoencoder is not regularized.

A VAE solves this problem by ensuring that the latent space has good properties that enable generative modeling while still learning to encode the data in an efficient way. In order to do so the encoder of a VAE does not encode an input $\boldsymbol{x}$ to a single value but to a distribution in latent space $p(\boldsymbol{z}|\boldsymbol{x})$. The decoder then decodes a sample $\boldsymbol{z} \sim p(\boldsymbol{z}|\boldsymbol{x})$ to an output $\tilde{\boldsymbol{x}}$ which is compared to the output to compute a reconstruction loss. Furthermore, a regularization loss is computed assuring that $p(\boldsymbol{z}|\boldsymbol{x}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The regularized latent space has the very useful property that similar data is close together. So now, if the decoder is given a slightly different latent representation of a car, it would decode it into a new image of a car. An overview of the VAE network architecture can be seen in Fig. 3.6

Generative Adversarial Networks

GANs were proposed in 2014 [10] and have since then seen great success and a lot of adaptions. GANs work by combining two models: A generator model and a discriminator model. These two models are trained at the same time and have an adversarial relationship to each other. Adversarial in that sense means that the generator and discriminator have opposing objectives.

The generator $G(\boldsymbol{x})$ has the task of generating samples of the data distribution $p_{data}(\boldsymbol{x})$. The discriminator $D(\boldsymbol{x})$ has the role of distinguishing the generated data $\tilde{\boldsymbol{x}}$ from the real data

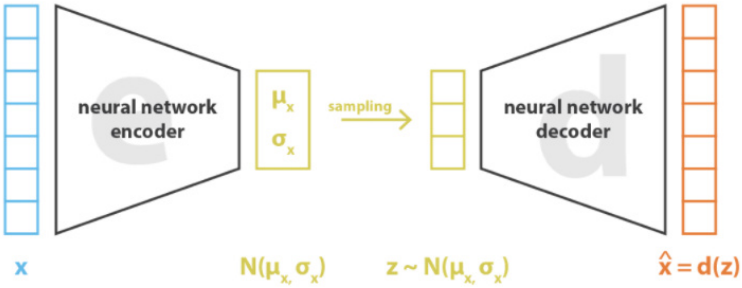


Figure 3.6.: Basic VAE model architecture. Figure from https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

$\boldsymbol{x}$. To be more precise, the task of the generator is to best possible trick the discriminator that at the same time has the task to best possible decide if an image is real or fake (generated). If the discriminator correctly identifies a generated image as fake the generator is punished to produce better images. Obviously, "better" just means that the images are more likely to fool the discriminator, which does not necessarily equate to a real-looking image if the discriminator performs poorly. When the discriminator is deceived by the generator, it is punished to better discriminate real and fake images. While GANs can produce excellent samples and are fast at sampling, it is easy to imagine how hard it is to train two networks at the same time. If not properly adjusted training quickly becomes unstable.

The absolute error of the discriminator can be written in the following way:

$$E(G, D) = \frac{1}{2}\left(\mathbb{E}_{\boldsymbol{x}\sim p_t}[1 - D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_z}[D(G(\boldsymbol{z}))]\right) \tag{3.1}$$

Here $\boldsymbol{x}$ is an input to the discriminator and $\boldsymbol{z}$ is a random input to the generator. Rewriting $G(\boldsymbol{z})$ as an input to the discriminator yields

$$E(G, D) = \frac{1}{2}\left(\mathbb{E}_{\boldsymbol{x}\sim p_t}[1 - D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[D(\boldsymbol{x})]\right), \tag{3.2}$$

where $\boldsymbol{x}$ can either be a real input ($\boldsymbol{x} \sim p_t$) or a generated input ($\boldsymbol{x} \sim p_g$). Therefore, in Equ. 3.2 the left term describes the error of falsely classifying a real image as generated, and the right term describes the error of falsely classifying a generated image as real. From this, it can be deduced that the perfect discriminator $D^*(\boldsymbol{x})$ classifies a real image as 1 and a fake image as 0.

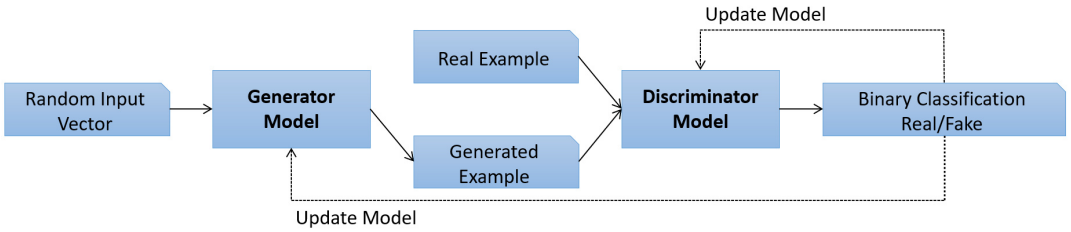


Figure 3.7.: Basic GAN model architecture

The total model objective of GANs can be written as

$$\max_G \left(\min_D E(G, D) \right), \tag{3.3}$$

which can be interpreted as the discriminator trying to minimize its error, while at the same time the generator tries to maximize the discriminator's error. A sketch of the training procedure is shown in Fig. 3.7.

In this thesis, the results of our score-based generative semantic image synthesis model are compared with the results of state-of-the-art models of different architecture. All models used for comparison are briefly explained below:

3.2.1. Cascaded Refinement Networks

Photographic Image Synthesis with Cascaded Refinement Networks (CRN) [4], a work from 2017, proposed one of the first models being able to synthesize convincing photographic looking images from semantic maps. Unlike other popular models to that time, the proposed model does not depend on adversarial training, therefore is no GAN. In fact, CRN is a simple feedforward convolutional network, so it is not a VAE either. Actually, due to the fact that the generated image is fixed for each input (i.e. non-probabilistic), CRN is not even considered a true generative model. The reason to try a feedforward model for image generation is mainly due to the problems of

Figure 3.8.: *Top*: Semantic map, *Bottom*: Sample image

other models (mostly GANs) at that time. There especially were problems of GANs not scaling up to higher resolution [4]. Also, GANs back then were very unstable to train and therefore "remain remarkably difficult to train" [4].

The CRN works as a cascade of refinement modules M^i. Each refinement module consists of three layers: The input layer, an intermediate layer and the output layer. Each of these layers is followed by a 3×3 convolution, a layer normalization and a LReLu activation function.

The input to a refinement module M^i is of size $w_i \times h_i \times (c + d_{i-1})$. Here $w_i \times h_i \times c$ is a downsampled version of the target segmentation map. Normally $w_0 \times h_0 = 4 \times 8$ and the downsampling resolution doubles for every module, i.e. $w_i \times h_i = 2w_{i-1} \times 2h_{i-1}$. c is the channel dimension of the semantic map in one-hot encoding and $d_{i-1} = d_{i-2} + c$. The outputs of a refinement module therefore are convolutional feature layers F_i of size $w_i \times h_i$ which contain $d_i = d_{i-1} + c$ feature maps. In order for this output of size $w_i \times h_i \times d_i$ to be concatenated into the input of layer M^{i+1}, which expects an input of size $w_{i+1} \times h_{i+1} \times (c+d_i)$, the output feature maps are up-sampled to resolution $w_{i+1} \times h_{i+1}$.

To the final output feature maps $F^{\bar{i}}$ of module $M^{\bar{i}}$ a 1×1 convolution (linear projection) is applied to get from the output of size $w_{\bar{i}} \times h_{\bar{i}} \times d_{\bar{i}}$ to a color image of size $w_{\bar{i}} \times h_{\bar{i}} \times 3$.

3.2.2. pix2pixHD

pix2pixHD [44] is the successor of the older pix2pix [16] framework and was developed by scientists of NVIDIA Corporation. pix2pixHD is a GAN model which is able to produce realistic high resolution semantic synthesized images and therefore mitigates the two main concerns of previous GAN models: resolution and image quality. In their paper the authors of pix2pixHD show that their model is able to produce samples of higher quality and realism than CRN (Sec. 3.2.1).

Figure 3.9.: *Top*: Semantic map, *Bottom*: Sample image

The pix2pix framework works as a classical conditional GAN model in a supervised setting. The generator G has the task to translate a given

semantic map to a realistic-looking image, while the discriminator D has the task to distinguish between the generated image and the original image for the semantic map.

The pix2pixHD model improves this framework by proposing a coarse-to-fine generator and a multi-scale discriminator. The coarse-to-fine generator essentially is a generator made up of multiple single generators $G_1, G_2, \ldots$ gradually working on higher resolutions of the image. The multi-scale discriminator addresses the need for a high receptive field for high resolution discrimination. Normally to achieve a higher receptive field either a deeper network or larger convolutional kernels are needed, both leading to increased model capacity and potentially overfitting. Therefore, the multi-scale discriminator consists of three individual discriminators D_1, D_2, D_3 each operating on different resolutions. In accordance with the GAN objective in Equ. 3.3 one gets

$$\min_G \max_{D_1,D_2,D_3} \sum_{k=1,2,3} E(G, D_k)\,. \tag{3.4}$$

The generator G is not indexed because the generators G_i are not trained concurrently but one after the other.

3.2.3. SPADE

SPADE [26] actually is no model itself, but the proposal of a new layer for semantic image synthesis: The **SP**atially-**A**daptive (**DE**-)Normalization Layer. In their paper, the authors test their new layer as an extension to the pix2pixHD model (Sec. 3.2.2).

Usually in normalization layers such as Batch Normalization (see Equ. 2.3), the activations of a layer are first normalized to zero mean and unit variance. Thereafter the normalized activation is denormalized using learned parameters. These learned parameters have no information of the semantic map and are the same for all values in $w_i \times h_i$, i.e. they are not spatially-adaptive. This

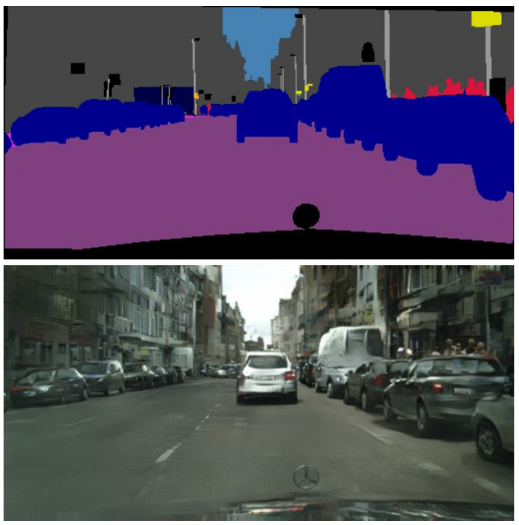


Figure 3.10.: *Top*: Semantic map, *Bottom*: Sample image

behavior leads to information loss during normalization.

The SPADE layer solves the normalization problem where segmentation map information is not preserved. Like the Batch Normalization the spatially-adaptive normalization normalizes the activation in a channel-wise manner and then denormalizes the activations with learned scale and bias parameters. When $\boldsymbol{h}^i$ are the activations after the $i{-}th$ layer the (de-)normalized activations $\boldsymbol{h}^i_{norm}$ after spatially-adaptive (de-)normalization are described by the following equation:

$$h^i_{n,c,y,x,norm} = \gamma^i_{c,y,x}(\boldsymbol{m})\frac{h^i_{n,c,y,x} - \mu^i_c}{\sigma^i_c} + \beta^i_{c,y,x}(\boldsymbol{m}). \tag{3.5}$$

This equation is in index notation where $n \in [1, N]$ is the batch size, $c \in [1, C^i]$ is the channel size, $y \in [1, H^i]$ the vertical pixel position and $x \in [1, W^i]$ the horizontal pixel position. $\boldsymbol{m}$ is the target segmentation map and μ^i_c and σ^i_c are the mean and standard deviation of $\boldsymbol{h}^i$ in channel c, defined by

$$\mu^i_c = \frac{1}{NH^iW^i}\sum_{n,y,x} h^i_{n,c,y,x} \tag{3.6}$$

$$\sigma^i_c = \sqrt{\frac{1}{NH^iW^i}\sum_{n,y,x}\left((h^i_{n,c,y,x})^2 - (\mu^i_c)^2\right)}. \tag{3.7}$$

For better understanding, note the similarities to Equ. 2.3 defining Batch Normalization. Condensed to one sentence, spatially-adaptive normalization is a batch normalization that is conditioned on the segmentation map $\boldsymbol{m}$ and is spatially dependent on the pixel positions x and y. Replacing the normal normalization layers in pix2pixHD by SPADE layers yields better image quality and realism [26].

4. Score-Based Generative Models

As we have seen in the preceding chapters there is quite a large range of models designed to generate data such as images, audio, or video. Despite this high diversity, only some of them – in particular GANs and VAEs – achieved outstanding success [3, 28], on the basis of which they were heavily improved in the last years. It is therefore quite difficult to come up with a new model that is able to quickly keep up with these highly adapted models. The fact that in this harsh environment *Score-Based Generative Models* [34, 35, 37] were able to achieve state-of-the-art results explains why they recently gained a lot of attention and makes them a new promising contender to the field of well-established generative models.

As Score-Based Generative Models were going through a lot of changes since the first publication, the next sections chronologically explain Score-Based Generative Models and their evolution in detail. In Sec. 4.1 a brief overview of the core concepts of Score-Based Generative Models is given, followed by Sec. 4.2 explaining why this core concept fails when being used without adaptions. Sec. 4.3 then presents how adding noise of discrete noise scales to the data distribution makes Score-Based Generative Modeling feasible, after which Sec. 4.4 generalizes this concept by introducing continuous noise governed by a Stochastical Differential Equation (SDE). Finally, in Sec. 4.5 the theory behind controllable generation is presented.

4.1. The idea behind Score-Based Generative Models

In general, the most basic Score-Based Generative Model consists of two core ideas: First, estimating the score $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$ of an unknown data distribution $p_{data}(\boldsymbol{x})$ of a dataset $\{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^N$ of N i.i.d samples. This process is referred to as *Score Matching* [14]. And second, using *Langevin Dynamics* [29, 45] to sample from the data distribution by starting with an initial value $\tilde{\boldsymbol{x}}_0$ from a prior distribution $\pi(\boldsymbol{x})$. This value is then gradually transformed by T recursive steps of Langevin Dynamics and will finally come out as $\tilde{\boldsymbol{x}}_T$

where $p(\tilde{\boldsymbol{x}}_T) \approx p_{data}(\boldsymbol{x})$, which makes it a nearly perfect sample of $p_{data}(\boldsymbol{x})$. The resulting generative modeling framework is called *Score Matching Langevin Dynamics* (SMLD).

4.1.1. Score Matching

This section gives an overview of Score Matching and how this technique can be used to get an objective for a score estimator, i.e. the model that is trained to estimate the score $\nabla_{\boldsymbol{x}} \log p_{data}(\boldsymbol{x})$ of the data distribution $p_{data}(\boldsymbol{x})$. The section is mathematically fairly intensive, and for that reason the main goal of this section is to provide a good feeling for why using the estimation of the score function as a model objective is a reasonable idea.

Score Matching [14] is a technique that originates from probabilistic models and their difficulties to be trained on an unnormalized probability density function $\tilde{p}(\boldsymbol{x})$ of a data distribution $p_{data}(\boldsymbol{x})$. The task of such a model $p_\theta(\boldsymbol{x})$ is to learn parameters θ such that $p_\theta(\boldsymbol{x}) = p_{data}(\boldsymbol{x})$ where $p_{data}(\boldsymbol{x})$ is the normalized density function defined as

$$p_{data}(\boldsymbol{x}) = \frac{\tilde{p}(\boldsymbol{x})}{Z}, \quad Z = \int \tilde{p}(\boldsymbol{x}) d\boldsymbol{x}\,. \tag{4.1}$$

Z is called the partition function and can be understood as a normalization constant. It is this partition function Z that causes the models difficulty to achieve its task. This is because the computation of Z is intractable, meaning that it cannot be solved in polynomial time or rather its complexity is at least $\mathcal{O}(k^n)$, where $k > 1$ is some constant and $n \in \mathbb{N}$ is the length of an input. The reason to use the score function to overcome this problem can be easily seen by using the calculation rules for logarithms to get $\log p(\boldsymbol{x}) = \log \tilde{p}(\boldsymbol{x}) - \log Z$. It is therefore obvious that the score function $\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})$ does not depend on the intractable partition function Z.

Score Matching uses this fact by minimizing the Fisher divergence between $p_{data}(\boldsymbol{x})$ and $p_\theta(\boldsymbol{x})$, which is defined as

$$L(\theta) \triangleq \frac{1}{2}\mathbb{E}_{p_{data}}[\|s_\theta(\boldsymbol{x}) - \nabla_{\boldsymbol{x}} \log p_{data}(\boldsymbol{x})\|_2^2], \tag{4.2}$$

where $s_\theta(\boldsymbol{x}) \triangleq \nabla_{\boldsymbol{x}} \log p_\theta(\boldsymbol{x})$ and $\|\cdot\|_2$ is the Euclidean norm. From here on, we now begin to transition from the probabilistic model $p_\theta(\boldsymbol{x})$ to the score model $s_\theta(\boldsymbol{x})$. Equ. 4.2 is the formulation of Score Matching which can be used in an integrated form as part of the model objective for probabilistic models. But beyond that Score Matching can also be used as a

standalone model objective for the so-called *Score-Based Models*.

Ultimately, both probabilistic models and score-based models learn the data distribution so that it is possible to generate samples from it. However probabilistic models aim to learn the data distribution itself. Score-based models ($s_\theta(\boldsymbol{x})$) aim to learn the score of the data distribution. In order to train a score-based model, we have to reformulate the objective in Equ. 4.2 as it is still not readily usable for leaning score-models because the data distribution $p_{data}(\boldsymbol{x})$ is typically unknown and so is $\nabla_{\boldsymbol{x}} \log p_{data}(\boldsymbol{x})$.

To address the problem of the unknown data distribution, *Denoising Score Matching* [43] was proposed. Denoising Score Matching introduces noise to Equ. 4.2 so that $\nabla_{\boldsymbol{x}}$ does not operate directly on the unknown distribution. A noise distribution $q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x})$ is applied to the data distribution to get a perturbed data distribution $q_\sigma(\boldsymbol{x}) = \int q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x}) p_{data}(\boldsymbol{x}) d\boldsymbol{x}$. When the noise is small ($q_\sigma(\boldsymbol{x}) \approx p_{data}(\boldsymbol{x})$) then $s_\theta(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} \log q_\sigma(\boldsymbol{x}) \approx \nabla_{\boldsymbol{x}} \log p_{data}(\boldsymbol{x})$ holds true. The objective of Denoising Score Matching follows as

$$\theta^* = \arg\min_\theta \frac{1}{2} \mathbb{E}_{q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x}) p_{data}(\boldsymbol{x})} [\| s_\theta(\boldsymbol{x}) - \nabla_{\boldsymbol{x}} \log q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x}) \|_2^2]. \tag{4.3}$$

In conclusion, Equ. 4.3 is the training objective for score-based models that is further used throughout this thesis. Nevertheless, there are also other Score Matching techniques such as *Sliced Score Matching* [36] that could be used, but it turns out that Denoising Score Matching is considerably faster. The objective of Score-Based Models (Equ. 4.2) has several desirable properties which makes it a reasonable approach for a generative model. As shown above the objective does not rely on the intractable partition function Z. Therefore, the objective is almost always tractable which has the implications that there is no special model architecture necessary. Furthermore, the objective can be optimized without adversarial training (Sec. 3.2) making it easier and more stable to train.

4.1.2. Langevin Dynamics

As described in Sec. 4.1.1 a score-model is tasked to estimate the score of the unknown data distribution $p_{data}(\boldsymbol{x})$ of a dataset. However, the score of a distribution is not yet a generated image. Instead, as it is the *gradient* of the (log) data distribution it can be utilized to sample from $p_{data}(\boldsymbol{x})$ by transforming a value $\tilde{\boldsymbol{x}}_0$ of a prior distribution $\pi(\boldsymbol{x})$ to be part of the data distribution, following the gradient (the score) of the data distribution. There are some restrictions to what the prior distribution should look like but for the considerations of

score-based models the prior distribution is just Gaussian noise, so $\pi(\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{x}, \mathbf{0}, \boldsymbol{I})$.

The process used for sampling is based on *Langevin Dynamics*, a concept from physics, that was adopted to machine learning [29, 45]. In Physics, the *Langevin Equation* describes particles moving in a potential which additionally are subject to random forces. This process is called *Brownian Motion*. The Langevin equation reads as

$$\lambda \frac{\boldsymbol{x}(t)}{dt} = -\nabla_{\boldsymbol{x}} V(\boldsymbol{x}(t)) + \boldsymbol{\eta}(t), \tag{4.4}$$

where V is the potential the particle is moving in, $\boldsymbol{x}(t)$ is the particle's position and $\boldsymbol{\eta}(t)$ is white noise. Commonly Equ. 4.4 is written as an (Itô)-Stochastical Differential Equation (SDE)

$$d\boldsymbol{x}(t) = \underbrace{-\nabla_{\boldsymbol{x}} V(\boldsymbol{x}(t))\, dt}_{\text{drift term}} + \underbrace{\sqrt{2}\, d\boldsymbol{w}}_{\text{diffusion term}}, \tag{4.5}$$

with a drift term and a diffusion term. $\boldsymbol{w}$ denotes the *Wiener Process* [9] which can be interpreted as the derivative of white noise. It is important to note that the positions of particles described by Equ. 4.5 are distributed according to some probability density function $p(\boldsymbol{x})$ for all times $t \geq 0$. This has some very useful consequences: We can choose the potential V in such a way that the underlying probability density function is p_{data}. In this case we can sample from p_{data}.

To achieve the state where the particles position described by the Langevin Equation are distributed by p_{data} we have to choose $V(\boldsymbol{x}) = -\log p_{data}(\boldsymbol{x})$. Inserting $V(\boldsymbol{x})$ into Equ. 4.5 results in

$$d\boldsymbol{x}(t) = \nabla_{\boldsymbol{x}} \log p_{data}(\boldsymbol{x}) dt + \sqrt{2}\, d\boldsymbol{w}. \tag{4.6}$$

Now, the connection between Sampling with Langevin Dynamics and Score Matching (Sec. 4.1.1) can be seen. If the score of the data distribution is known we can sample from this distribution by using Equ. 4.6. For now, $\boldsymbol{x}(t)$ was the position of a physical particle but Equ. 4.6 is applicable for any d-dimensional data $\boldsymbol{x}(t) \in \mathbb{R}^d$, e.g. images. To use Equ. 4.6 in the context of machine learning we cannot directly use the SDE as it describes continuous steps of infinitesimal size. We therefore discretize the SDE to discrete steps which can then be calculated by a computer. The resulting discretization in Equ. 4.7 is called the Euler-Maruyama discretization of the SDE.

$$\tilde{\boldsymbol{x}}_t = \tilde{\boldsymbol{x}}_{t-1} + \frac{\epsilon}{2} \nabla_{\boldsymbol{x}} \log p_{data}(\tilde{\boldsymbol{x}}_{t-1}) + \sqrt{\epsilon}\, \boldsymbol{z}_t \tag{4.7}$$

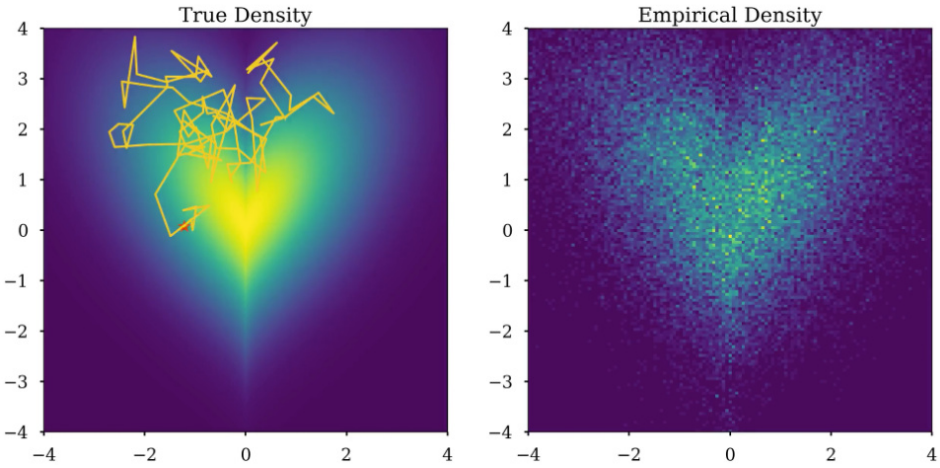


Figure 4.1.: Application of Langevin Dynamics sampling. Figure from https://abdulfatir.com/Langevin-Monte-Carlo/

Equ. 4.7 is a recursive algorithm that can be used to sample from p_{data}. Here $\tilde{\boldsymbol{x}}_0$ is a value from the prior distribution given above and the "$\sim$" above the $\boldsymbol{x}$ shows that $\tilde{\boldsymbol{x}}$ is not in the dataset nor described by the data distribution $p_{data}(\boldsymbol{x})$ but a random initial sample. Furthermore $z_t \sim \mathcal{N}(0, I)$ is Gaussian noise and ϵ is the step size of the algorithm. The sample after T steps is a perfect sample of p_{data} when $T \to \infty$ and $\epsilon \to 0$. For application of this algorithm $T \gg 0$ and $\epsilon \ll 1$ are assumed.

Recapitulating what the algorithm does, one can look at Fig. 4.1. The lines in the left image show a particle moving randomly according to Equ. 4.5. In the background the underlying (heart-shaped) data density function is represented and in the right image samples are shown, where each of the dots is an initial sample from the prior distribution that was transformed using Equ. 4.7. When applying Equ. 4.7 to infinite values from the prior distribution this process can be thought of as transforming one distribution to another one. As a side note, the formulation of this integrated process is given by the *Fokker-Planck-Equation* [9] – another important equation in physics. For the purposes of score-based models only a one-by-one application of Equ. 4.7 is needed, essentially transforming one image from a noise distribution to an image that looks like it belongs to the data distribution of the target dataset.

4.2. Hindrances of a naïve application

While this general concept looks quite appealing to use as the basis for a generative model, in its naïve application it might fail for datasets containing real world data. The authors of

[34] discuss two hindrances preventing this naïve application, which both are bypassable by adding random noise to the data. These bypasses also have some consequences on Score Matching and Langevin Dynamics which will be discussed in Sec. 4.3. It should be noted, however, that the final score-based model (Sec. 4.4) which we will derive in the following two sections can also be derived from considerations other than those described in this section.

4.2.1. The manifold hypothesis

The problems in both Sec. 4.2.1 and Sec. 4.2.2 originate from a similar source: The data density of a dataset of real world data. The manifold hypothesis states that real world high-dimensional data lies on low-dimensional manifolds embedded within the high-dimensional space which is called the ambient space. To further explain this hypothesis with an example, one can imagine a dataset of black and white images, each image being $m \times n$ pixels in size. The ambient space – the space of *all* possible black and white images with that size – has therefore $m \times n$ dimensions. Datasets normally do not contain all possible data of a kind which would make them useless but rather contain data that is very special, e.g. images of cars. As there is some similarity in cars the data in such a dataset is assumed to only cover a small, connected volume in the high dimensional space – a manifold.

With this hypothesis two problems arise for score-models as they are described above. First, the score $\nabla_{\boldsymbol{x}} \log p_{data}(\boldsymbol{x})$ is calculated in the whole ambient space, i.e. for all dimensions, not only for the lower dimensions of the manifold. This results in the score being undefined when $\boldsymbol{x}$ is confined to such a low dimensional manifold. Second, it was shown in [14] that the objective from Equ. 4.2 can only be used for defining a consistent score estimator when the data, the data distribution describes, has the same dimensionality as the whole space.

4.2.2. Low data density region

A second problem that arises with the data distribution of a dataset of real world data is that there is just not enough data to fully cover the data distribution. Recall that the data distribution is unknown and describes *all* data of a kind. To make an example, if a model should learn to generate cars, then the data distribution would contain *all* images of cars one can imagine. Certainly, we are not able to create a dataset containing images of all cars which in reverse means that the dataset most likely focuses on data that is particularly representative for the data the model should learn.

This has some consequences for score estimation. For data distributions, there are regions of high density, where a lot of data is expected. In datasets – as they try to represent the data distribution – most data should appear in this high density regions. But there are also low density regions where little or no data is available. In the car example from above, this means that there might be a lot of images of blue or black cars but little or no images of yellow cars. In that regions of low density, the score cannot be estimated as there is no data to learn from. In mathematical representation this means that when considering any region $\mathcal{R} \subset \mathbb{R}^d$ in a dataset $\{\boldsymbol{x}_i\}_{i=1}^N \overset{i.i.d}{\sim} p_{data}(\boldsymbol{x})$ such that $p_{data}(\mathcal{R}) \approx 0$ the intersection $\{\boldsymbol{x}_i\}_{i=1}^N \cap \mathcal{R}$ is often equal to the empty set $\varnothing$.

4.3. Introducing noise to the data distribution

To mitigate the hindrances stated in Sec. 4.2.1 and Sec. 4.2.2 the data is perturbed with random Gaussian noise. For the problem in Sec. 4.2.1, adding small Gaussian noise to the data ensures that the perturbed data distribution has the same dimensionality as the whole space fixing the problem with the inconsistent score estimator. In addition, the disturbed data distribution no longer concentrates on a low dimensional manifold so the score is now defined everywhere in ambient space.

The solution to the problem in Sec. 4.2.2 is similar, but requires large Gaussian noise. This large noise perturbs the data distribution in such a way that low density regions in the original data distribution are filled which makes it possible for the score-models to improve on learning the score in low density regions.

The idea is then to use multiple noise scales with decreasing magnitude to transform the maximum noise-perturbed data distribution via a sequence of lower noise-perturbed data distributions to the true (unnoised) data distribution. This idea provokes several changes for the score-model $s_\theta(\boldsymbol{x})$ and the Langevin Dynamics sampling procedure.

4.3.1. Noise Conditional Score Networks

As there is now a sequence of data distributions with decreasing noise, the score-model no longer aims to learn the score of one data distribution but to jointly learn the score of all data distributions. The noise can be thought of as a parameter which is given to the network additionally to the data $\boldsymbol{x}$. Such a model $s_\theta(\boldsymbol{x}, \sigma)$ is called a *Noise Conditional Score Network* (NCSN) [34]. In practice when giving the model a value $\boldsymbol{x}_\sigma$ from a noisy data distribution

$q_{\sigma_i}(\boldsymbol{x}) \triangleq \int p_{data}(\boldsymbol{t})\mathcal{N}(\boldsymbol{x}|\boldsymbol{t}, \sigma_i^2 I)d\boldsymbol{t}$ the variance σ_i of this distribution is given as a parameter to the model, so it can learn the difference between various noise scales applied to data $\boldsymbol{x}$. The authors of [34] choose the noise scales $\{\sigma_i\}_{i=1}^L$ as a positive geometric series, i.e. $\frac{\sigma_1}{\sigma_2} = \cdots = \frac{\sigma_{L-1}}{\sigma_L}$ where σ_L denotes the smallest noise scale and σ_1 the largest noise scale. A high σ_1 was found to be significant for the diversity of the samples [37]. However, a high σ_1 comes with the flaw of high computational expenses of Langevin Dynamics. [37] shows that choosing σ_1 as the maximum Euclidean distance between all pairs of training data points is a good choice. In general, if the score-model is properly trained, then $\forall\sigma \in \{\sigma_i\}_{i=1}^L$ applies $s_\theta(\boldsymbol{x}, \sigma) \approx \nabla_{\boldsymbol{x}} \log q_\sigma(\boldsymbol{x})$, making $s_\theta(\boldsymbol{x}, \sigma)$ a nearly perfect score estimator on all noise scales.

To achieve high quality samples, the model architecture has to be well adapted to the task of noise-conditional Score Matching. The model the authors of [34] use is called NCSN (Noise Conditional Score Network) and is built upon successful model architectures from dense semantic segmentation a.o. U-Net (Sec. 3.1.2). When advancing score-models to continuous noise in Sec. 4.4 the authors of [35] introduce an improved model called NCSN++ which is strongly based on the model implementation of [12].

In order to train a NCSN the objective of noise-conditional Score Matching must be known. The noise distributions get chosen as $q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x}) = \mathcal{N}(\tilde{\boldsymbol{x}}|\boldsymbol{x}, \sigma^2 I)$; therefore $\nabla_{\tilde{\boldsymbol{x}}} \log q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x}) = -{}^{(\tilde{\boldsymbol{x}}-\boldsymbol{x})}/_{\sigma^2}$. For a given $\sigma \in \{\sigma_i\}_{i=1}^L$ the objective can be formulated as an adaption of Equ. 4.3:

$$\ell(\theta; \sigma) \triangleq \frac{1}{2}\mathbb{E}_{p_{data}(\boldsymbol{x})}\mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathcal{N}(\boldsymbol{x},\sigma^2 I)}\left[\left\| s_\theta(\tilde{\boldsymbol{x}}, \sigma) + \frac{\tilde{\boldsymbol{x}} - \boldsymbol{x}}{\sigma^2}\right\|_2^2\right] \tag{4.8}$$

To get one unified objective Equ. 4.8 is summed for all noise scales $\sigma \in \{\sigma_i\}_{i=1}^L$:

$$\mathcal{L}(\theta; \{\sigma_i\}_{i=1}^L) \triangleq \frac{1}{L}\sum_{i=1}^{L} \lambda(\sigma_i)\ell(\theta; \sigma_i) \tag{4.9}$$

Here $\lambda(\sigma_i) > 0$ is a coefficient function depending on σ_i. As it is advantageous when the scores of all levels of noise have roughly the same order of magnitude and empirically it is observed that $\|s_\theta(\boldsymbol{x}, \sigma)\|_2 \propto {}^1/_\sigma$ [37], $\lambda(\sigma)$ is chosen to be σ^2. With that choice the term $\lambda(\sigma_i)\ell(\theta; \sigma_i)$ in Equ. 4.9 is equal to $\sigma^2\ell(\theta; \sigma_i) = \frac{1}{2}\mathbb{E}[\left\|\sigma s_\theta(\tilde{\boldsymbol{x}}, \sigma) + \frac{\tilde{\boldsymbol{x}}-\boldsymbol{x}}{\sigma^2}\right\|_2^2]$. Because $\frac{\tilde{\boldsymbol{x}}-\boldsymbol{x}}{\sigma^2} \sim \mathcal{N}(0, I)$ and $\|\sigma s_\theta(\tilde{\boldsymbol{x}}, \sigma)\| \propto 1$ we can deduce that the order of magnitude of $\lambda(\sigma)\ell(\theta; \sigma)$ is independent of σ.

4.3.2. Annealed Langevin Dynamics

After introducing noise scales it is clear that also the Langevin Dynamics algorithm has to be adjusted. In Sec. 4.1.2 we saw that Langevin Dynamics transforms a sample $\tilde{\boldsymbol{x}}_0$ from a prior distribution $\pi(\boldsymbol{x})$ to a sample $\tilde{\boldsymbol{x}}_T$ from the target data distribution p_{data} by applying T steps of Langevin Dynamics algorithm (Equ. 4.7). Now the goal remains the same, but it is not possible to directly transform the prior distribution to the data distribution because the score-model only knows the scores for certain noise levels. Instead, Equ. 4.7 is applied L times in a sequence, once for each transition from one noise distribution $q_{\sigma_i}(\boldsymbol{x})$ to another $q_{\sigma_{i+1}}(\boldsymbol{x})$. This means, that starting from an initial sample $\tilde{\boldsymbol{x}}_0$ from a prior distribution that is perturbed with the maximum noise σ_1 – i.e. $\pi(\boldsymbol{x}) \sim q_{\sigma_1}(\boldsymbol{x})$ – this sample is then transformed to be a sample of the noise distribution $q_{\sigma_2}(\boldsymbol{x})$. The procedure is then repeated by using the final sample $\tilde{\boldsymbol{x}}_T$ from the last step as the new initial sample $\tilde{\boldsymbol{x}}_0$ that is then be transformed to be a sample of the noise distribution $q_{\sigma_3}(\boldsymbol{x})$. After L of such steps we therefore have a sample of $q_{\sigma_L}(\boldsymbol{x}) \approx p_{data}(\boldsymbol{x})$. The final algorithm can be seen in Alg. 1.

Algorithm 1: ANNEALED LANGEVIN DYNAMICS (adapted from [34])

Require : $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$
1: Initialize $\tilde{\boldsymbol{x}}_0$
2: **for** $i \leftarrow$ *to* L **do**
3: $\quad \alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ ▷ `` $\alpha_i$ is the step size ``
4: $\quad$ **for** $t \leftarrow 1$ *to* T **do**
5: $\quad\quad$ Draw $z_t \sim \mathcal{N}(0, I)$
6: $\quad\quad \tilde{\boldsymbol{x}}_t \leftarrow \tilde{\boldsymbol{x}}_{t-1} + \frac{\alpha_i}{2} s_\theta(\tilde{\boldsymbol{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\boldsymbol{z}_t$
7: $\quad \tilde{\boldsymbol{x}}_0 \leftarrow \tilde{\boldsymbol{x}}_T$
8: **return** $\tilde{\boldsymbol{x}}_T$

Here α_i denotes the outer step size that is tuned down gradually by $\alpha_i = \epsilon \cdot \sigma_i^2/\sigma_L^2$. Since the noise distributions $\{\sigma_i\}_{i=1}^{L}$ are perturbed with Gaussian noise, the scores encountered during sampling are all well-defined. Also, when σ_1 is sufficiently large there are less low density regions in $q_{\sigma_1}(\boldsymbol{x})$, preventing the algorithm to encounter regions where the score has not been learned due to a lack of training data.

With the release of [34] the authors found that their *FID (Fréchet inception distance)* [11] scores – a metric for evaluating the quality of samples – was higher (worse) than state-of-the-art FID scores even though the samples looked better to human eyes. Thereupon [19] found that the final sample generated by Alg. 1 contains some noise which is not visible to the human eye but noticeably decreases the FID score. To solve this problem the authors of

[19] proposed to add an extra denoising step after the original Annealed Langevin Dynamics which significantly improves FID scores. This improved algorithm can be found in Alg. 2.

Algorithm 2: IMPROVED ANNEALED LANGEVIN DYNAMICS (adapted from [34] and [19])

Require : $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$
Initialize $\tilde{\boldsymbol{x}}_0$
for $i \leftarrow$ *to* L **do**
  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$
  for $t \leftarrow 1$ *to* T **do**
    Draw $z_t \sim \mathcal{N}(0, I)$
    $\tilde{\boldsymbol{x}}_t \leftarrow \tilde{\boldsymbol{x}}_{t-1} + \frac{\alpha_i}{2} s_\theta(\tilde{\boldsymbol{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\boldsymbol{z}_t$
  $\tilde{\boldsymbol{x}}_0 \leftarrow \tilde{\boldsymbol{x}}_T$
if *denoise* $\tilde{\boldsymbol{x}_T}$ **then**
  return $\tilde{\boldsymbol{x}}_T + \sigma_L^2 s_\theta(\tilde{\boldsymbol{x}}_T, \sigma_L)$
else
  return $\tilde{\boldsymbol{x}}_T$

4.4. The way to continuous noise

In Sec. 4.2 a sequence of discrete noise scales was used to perturb the data distribution. Although this score-model produced first visually appealing samples [34], the training image and sample size of the model was limited to $\sim 64 \times 64$ pixels. In [37] 5 techniques were proposed which allowed the authors to attain resolutions up to 256×256 pixels. Most likely it is possible to scale up this model to even higher resolutions but the authors of [35] instead proposed a new model which uses continuous noise: Introducing a continuous noise [35] instead of discrete noise scales allowed the authors to reach resolutions up to 1024×1024 pixels, introduces a mathematically pleasant generalization of different Score-Based Models and creates the possibility for a lot of more sampling methods than just Langevin Dynamics.

4.4.1. Score-Based Generative Modeling with SDEs

The discrete noise scales $\{\sigma_i\}_{i=1}^{L}$ are now replaced by a diffusion process $\{\boldsymbol{x}(t)\}_{t=0}^{T}$ indexed by a continuous time variable $t \in [0, T]$ that evolves according to a SDE. For the distribution p_{data} of a dataset of i.i.d samples this means that $\boldsymbol{x}(0) \sim p_{data}(\boldsymbol{x})$ and $\boldsymbol{x}(T) \sim \pi(\boldsymbol{x})$ where $\pi(\boldsymbol{x})$ is a prior distribution that in *completely independent* of p_{data}, e.g. a Gaussian

distribution with fixed mean and variance. The diffusion process can be modeled by the following (Itô-)SDE:

$$d\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t)dt + g(t)d\boldsymbol{w} \tag{4.10}$$

In Equ. 4.10 the vector-valued function $\boldsymbol{f}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ is called the *drift coefficient*, the scalar function $g(\cdot) : \mathbb{R} \to \mathbb{R}$ is called the *diffusion coefficient* and $\boldsymbol{w}$ denotes the *Wiener process*. In general $g(\cdot)$ could also be a $d \times d$ matrix, but for the ease of presentation a scalar diffusion coefficient is assumed. How to choose the diffusion coefficient and the drift coefficient such that $\boldsymbol{x}(t)$ diffuses to $\pi(\boldsymbol{x})$ is explained in Sec. 4.4.4.

A remarkable result [2] is that such a SDE has a reverse SDE that models a reverse time diffusion process, diffusing a sample $\boldsymbol{x}(T) \sim \pi(\boldsymbol{x})$ to $p_{data}(\boldsymbol{x})$:

$$d\boldsymbol{x} = [\boldsymbol{f}(\boldsymbol{x}, t) - g(t)^2 \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})]dt + g(t)d\bar{\boldsymbol{w}} \tag{4.11}$$

Here dt is an infinitesimal timestep backwards in time and $\bar{\boldsymbol{w}}$ is the Wiener process running backwards in time. For the application of this reverse diffusion process, it comes very handy that it only depends on the score $\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$. So, if the score of each noisy distribution $p_t(\boldsymbol{x})$ of $\boldsymbol{x}(t)$ ($t \in [0, T]$) is known, it is possible to sample from p_{data} by solving the reverse SDE.

4.4.2. Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models (DDPM) [12] are closely related to SMLD (Sec. 4.3.1). They very briefly will be presented here, as they can be generalized together with SMLD to continuous NCSN (Sec. 4.4.3, Sec. 4.4.4). DDPMs are trained on Markov chains whose transitions are learned by the model to reverse a diffusion process by reversing the Markov chain. This will especially work if the diffusion consists of small amounts of Gaussian noise [12]. The objective of such a model is given as

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} (1 - \alpha_i) \mathbb{E}_{p_{data}(\boldsymbol{x})} \mathbb{E}_{p_{\alpha_i}(\tilde{\boldsymbol{x}}|\boldsymbol{x})} \left[\| s_\theta(\tilde{\boldsymbol{x}}, i) - \nabla_{\tilde{\boldsymbol{x}}} \log p_{\alpha_i}(\tilde{\boldsymbol{x}}|\boldsymbol{x}) \|_2^2 \right], \tag{4.12}$$

where $\alpha_i \triangleq \Pi_{j=1}^{i}(1 - \beta_j)$ and $0 < \beta_1 < \beta_2 < \cdots < \beta_N < 1$ are positive noise scales. The Markov chain $\{\boldsymbol{x}_i\}_{i=0}^{N}$ is constructed in such a way that the perturbation kernels $p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1}) = \mathcal{N}(\boldsymbol{x}_i; \sqrt{1 - \beta_i}\, \boldsymbol{x}_{i-1}, \beta_i \boldsymbol{I})$. Starting with an initial sample $\tilde{\boldsymbol{x}}_N$ and a trained

DDPM $s_\theta(\boldsymbol{x}_i, \beta_i)$ the following reverse Markov chain can be used to sample from p_{data}:

$$\boldsymbol{x}_{i-1} = \frac{1}{\sqrt{1-\beta_i}}(\boldsymbol{x}_i + \beta_i s_\theta(\boldsymbol{x}_i, i)) + \sqrt{\beta_i}\boldsymbol{z}_i, \quad i = N, N-1, \dots, 1 \tag{4.13}$$

4.4.3. Generalizing the Score Matching objective

Like in the previous sections, some adaptions to the objective of Score Matching have to be made. Without further mathematical derivation, a continuous generalization of the previous objectives (Equ. 4.3, Equ. 4.9, Equ. 4.12) is given as

$$\theta^* = \arg\min_\theta \mathbb{E}_t \left\{ \lambda(t) \mathbb{E}_{\boldsymbol{x}(0)} \mathbb{E}_{\boldsymbol{x}(t)|\boldsymbol{x}(0)} \left[\left\| s_\theta(\boldsymbol{x}(t), t) - \nabla_{\boldsymbol{x}(t)} \log p_{0t}(\boldsymbol{x}(t)|\boldsymbol{x}(0)) \right\|_2^2 \right] \right\}. \tag{4.14}$$

$\lambda(t) : [0, T] \to \mathbb{R}_{>0}$ is a positive weighting function. Similarly to the weighting function in Equ. 4.9 it is chosen in a way such that the magnitude of the score is independent of the noise – here characterized by t – which implies a choice of the form $\lambda \propto 1/\mathbb{E} \left\| \nabla_{\boldsymbol{x}(t)} \log p_{0t}(\boldsymbol{x}(t)|\boldsymbol{x}(0)) \right\|_2^2$.

Moreover, note the similarities of this objective with the one in Equ. 4.3. The objective in general is now dependent on t and the discrete noise distributions have been replaced by transition kernels $p_{st}(\boldsymbol{x}(t)|\boldsymbol{x}(s))$ describing a transition from $\boldsymbol{x}(s)$ to $\boldsymbol{x}(t)$ where $0 \leq s \leq t \leq T$.

4.4.4. VE and VP SDEs

In general, there is are infinite ways to choose the drift coefficient $\boldsymbol{f}(\boldsymbol{x}(t), t)$ and the diffusion coefficient $g(t)$ in Equ. 4.10. Having said that, there are two especially interesting ways to choose $\boldsymbol{f}$ and g. It is namely that the noise pertubations used in SMLD and DDPM correspond to discretizations of two types of SDEs. With that knowledge, the concepts of SMLD and DDPM can be transformed to two different implementations of the continuous noise model.

For SMLD, perturbation kernels of the form $p_{\sigma_i}(\tilde{\boldsymbol{x}}|\boldsymbol{x}) = \mathcal{N}(\tilde{\boldsymbol{x}}|\boldsymbol{x}, \sigma^2 \boldsymbol{I})$ with $i \in \{1, \dots, N\}$ are deployed. Applying these perturbation kernels in series on a variable $\boldsymbol{x}$ can be written as the following Markov chain:

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2}\boldsymbol{z}_{i-1}, \tag{4.15}$$

where $\boldsymbol{z}_{i-1} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. $\boldsymbol{x}_0 \sim p_{data}$ and σ_0 is chosen as 0. $\boldsymbol{x}_i$, σ_i and z_i are now reformulated

as $\boldsymbol{x}(i/N)$, $\sigma(i/N)$ and $z(i/N)$ for $i = 1, 2, \ldots, N$. With $\Delta t = 1/N$ and $t \in \{0, 1/N, \ldots, N-1/N\}$ Equ. 4.15 can be rewritten as

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t) + \sqrt{\sigma^2(t + \Delta t) - \sigma^2(t)}\, \boldsymbol{z}(t). \tag{4.16}$$

With the assumption $\Delta t \ll 1$ Equ. 4.16 can be approximated as

$$\boldsymbol{x}(t + \Delta t) \approx \boldsymbol{x}(t) + \sqrt{\frac{d\sigma^2(t)}{dt}\Delta t}\, \boldsymbol{z}(t), \tag{4.17}$$

and in the limit of $\Delta t \to 0$ (continuous), Equ. 4.17 converges to

$$d\boldsymbol{x} = \sqrt{\frac{d\sigma^2(t)}{dt}}\, d\boldsymbol{w}. \tag{4.18}$$

This SDE is called the *VE SDE* and describes the continuous stochastic process $\{\boldsymbol{x}(t)\}_{t=0}^{1}$ of the former discrete Markov chain $\{\boldsymbol{x}_i\}_{i=1}^{N}$. In the representation of Equ. 4.10 this means that $\boldsymbol{f}(\boldsymbol{x}, t) = 0$ and $g(t) = \sqrt{\frac{d\sigma^2(t)}{dt}}$.

Using the same procedure also the perturbation kernels of DDPM can be transformed to a continuous stochastic process leading to

$$d\boldsymbol{x} = -\frac{1}{2}\beta(t)\boldsymbol{x}\, dt + \sqrt{\beta(t)}\, d\boldsymbol{w}, \tag{4.19}$$

which is called the *VP SDE*. The names VE and VP originate from the observations that the VE SDE always gives a process with **E**xploding **V**ariance while the VP SDE yields a process with bounded variance, therefore **P**reserving the initial **V**ariance. The variance of the SDEs are given as the variance of their perturbation kernels $p_{0t}(\boldsymbol{x}(t)|\boldsymbol{x}(0))$, which are $[\sigma^2(t) - \sigma^2(0)]\boldsymbol{I}$ for the VE SDE and $\boldsymbol{I} - \boldsymbol{I}\exp\left(-\int_0^t \beta(s)ds\right)$ for the VP SDE. As the VE SDE is better suited to produce high resolution images [35] and is easier to implement, we exclusively use the VE SDE in our experiments.

4.4.5. Sampling via the Reverse SDE

As mentioned in Sec. 4.4.1, samples of p_{data} can be generated by solving the reverse SDE. In order to do so, the scores $\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$ are needed which are provided by the trained score model $s_\theta(\boldsymbol{x}(t), t)$. In theory, to solve the reverse SDE any kind of numerical SDE solver can be used. In practice, however, it was determined that most off-the-shelf SDE solvers are ill-

suited for generative modeling [18]. These off-the-shelf SDE solvers often exhibit divergence, slow data generation and worse quality than the specialized SDE solvers proposed beneath.

Predictor Corrector samplers

The first sampling strategy - which is also the one we use for all experiments – consists of a predictor and a corrector. This category of samples is therefore called *Predictor-Corrector Samplers* [35]. The sampler works in such a way that the predictor gives an estimate of the sample at the next step, and the corrector corrects this estimated sample. The idea to use a combination of a predictor and a corrector instead solely using a SDE solver comes from the realization that we have additional information in the reverse SDE (Equ. 4.11): the score. We have seen in Sec. 4.3.2 that the score can be used to sample from p_{data}, so we know how to process this extra information which is normally not available when solving arbitrary SDEs. In the implementation, a variation of Langevin Dynamics samplers plays the role of the corrector and a numerical SDE solver plays the role of the predictor.

The authors of [35] propose to use the *Euler-Maruyama Method* as the SDE solver which is a very simple method to numerically solve SDEs and is based on the well-known Euler method for solving ordinary differential equations (ODEs). For a general SDE (Equ. 4.10) the Euler-Maruyama method computes the evolution of $\boldsymbol{x}$ as the following recursive formula:

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + \tau \boldsymbol{f}(\boldsymbol{x}_{i-1}, t_{i-1}) + g(\boldsymbol{x}_{i-1}, t_{i-1})\boldsymbol{z}_{i-1}, \tag{4.20}$$

where τ is the step size defined as $\tau = \frac{T}{i}$, $\boldsymbol{z}_{i-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, $i = 1, \ldots, N$ and $0 = t_0 < t_1 < \cdots < t_n = T$ is the discretization of time t. We have seen such a discretization of a SDE before in Equ. 4.15. From this realization follow two points: First, the discretization in Equ. 4.15 is called Euler-Maruyama discretization and second, the Euler-Maruyama discretization is a Markov chain.

The full predictor-corrector algorithm using Langevin Dynamics and the Euler-Maruyama method can be seen in Alg. 3. In this algorithm N denotes the number of sampling steps and snr is the signal-to-noise-ratio which adjusts the radio between the scores and the noise. snr has a large effect on sampling quality and is experimentally chosen to be between ~ 0.15 and ~ 0.075 depending on the sampling resolution.

Algorithm 3: PREDICTOR-CORRECTOR SAMPLER

Require : N, snr

Initialize $\tilde{\boldsymbol{x}}_0$

for $i \leftarrow 1$ *to* N **do**

  Draw $z_i \sim \mathcal{N}(0, I)$ ▷ `Langevin Dynamics`

  $\epsilon_{i-1} \leftarrow \left[\frac{snr \cdot \|\boldsymbol{z}_{i-1}\|}{\left\|s_\theta(\tilde{\boldsymbol{x}}_{t_{i-1}}, t_{i-1})\right\|} \right]^2$

  $\tilde{\boldsymbol{x}}_{langevin} \leftarrow \tilde{\boldsymbol{x}}_{i-1} + 2\epsilon_{i-1}\, s_\theta(\tilde{\boldsymbol{x}}_{i-1}, t_{i-1}) + \sqrt{2\epsilon_{i-1}}\, \boldsymbol{z}_{i-1}$

  Draw $z_i \sim \mathcal{N}(0, I)$ ▷ `Euler-Maruyama`

  $\Delta t_i \leftarrow t_{i-1} - t_i$

  $\tilde{\boldsymbol{x}}_i \leftarrow \tilde{\boldsymbol{x}}_{langevin} + \boldsymbol{f}(\tilde{\boldsymbol{x}}_{langevin}, t_{i-1}) \cdot \Delta t_i + g(\tilde{\boldsymbol{x}}_{i-1}, t_{i-1}) \cdot \sqrt{\Delta t_i}\, \boldsymbol{z}_{i-1}$

return $\tilde{\boldsymbol{x}}_N$

Probability Flow ODE sampler

This sampling strategy is based upon the mathematical finding that every diffusion process has a corresponding *deterministic process* which is governed by the following *ODE*:

$$d\boldsymbol{x} = \left[\boldsymbol{f}(\boldsymbol{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x}) \right] dt. \tag{4.21}$$

Deterministic in this sense means that the trajectories of the process are not affected by some kind of randomness but rather can be predicted beforehand. This special ODE which needs the score in order to be determined from the SDE shares the same probability densities $\{p_t(\boldsymbol{x})\}_{t=0}^T$ as the SDE.

An advantage is that the ODE can be solved with numerical **O**DE solvers, which are normally a lot faster than numerical SDE solvers. Nevertheless, we do not use this sampling technique because it produces samples of worse quality [35] and the other advantages of solving this ODE do not apply for our purpose of use.

Other sampling methods

There were also other sampling methods proposed, of which we want to name a few. *Reverse Diffusion Samplers* were proposed in [35] together with Predictor-Corrector samplers and Probability Flow samples and are a kind of numerical SDE solvers which discretize the reverse SDE in the same way the forward SDE is discretized.

A promising sampler was proposed as a result of an in-depth sampler analysis and a lot of

fine-tuning in [18]. These samplers are non-general-purpose SDE solvers which means they are specially tailored to solve the SDE in Equ. 4.10. This results in sampling speed $2 - 10$ times higher than all other mentioned samplers and also in higher sampling quality. However, we leave the exploration of this sampler in context of our work to future research.

4.5. Controllable Generation

Controllable Generation is the modulation of the generation process. Popular controllable generation tasks include inpainting, colorization, class conditional generation or generation conditioned on semantic maps. For continuous-noise Score-Based Generative Models (Sec. 4.4), controllable generation can be achieved in an advantageous way. Instead of retraining the model for each controllable generation task, additional information can be used for SGMs during the sampling process to model the controllable generation task.

Basically, this means that when we have a trained score-model $s_\theta(\boldsymbol{x}, \sigma)$, it is not only possible to sample from $p_{data}(\boldsymbol{x}(0))$ but also from $p_{data}(\boldsymbol{x}(0)|\boldsymbol{y})$ if $p_t(\boldsymbol{y}|\boldsymbol{x}(t))$ is known. Here $p_{data}(\boldsymbol{x}(0)|\boldsymbol{y})$ is the data distribution of images with zero noise which fulfill the condition $\boldsymbol{y}$ and $p_t(\boldsymbol{y}|\boldsymbol{x}(t))$ is a probability density function for $\boldsymbol{x}(t)$ fulfilling the condition $\boldsymbol{y}$ at timestep t.

Starting with $p_T(\boldsymbol{x}(T)|\boldsymbol{y})$ it is possible to solve the following conditional reverse SDE to sample from $p_{data}(\boldsymbol{x}(0)|\boldsymbol{y})$:

$$d\boldsymbol{x} = \{\boldsymbol{f}(\boldsymbol{x}, t) - g(t)^2[\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{y}|\boldsymbol{x})]\}dt + g(t)d\bar{\boldsymbol{w}}. \tag{4.22}$$

For class-conditional or semantic map-based generation, the consequence is that a second model is needed to solve the reverse SDE. In Equ. 4.22 the score-model is responsible for the unconditional score $\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})$, while a second, independently trained model is responsible for the conditional scores $\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{y}|\boldsymbol{x})$. What seems very mathematical at first is actually very easy to implement:

1. Train the continuous-noise score-model $s_\theta(\boldsymbol{x}, \sigma)$ on a dataset of choice.

2. Train a second model conditioned on the same noise which is able to output $\boldsymbol{y}$ from $\boldsymbol{x}(t)$. For semantic image synthesis, this would mean training a noise-conditional semantic segmentation network $p_\theta(\boldsymbol{x}, \sigma)$ (Sec. 3.1) with noisy images $\boldsymbol{x}(t)$ where the semantic map used for error computation is $\boldsymbol{y}$ of the original image $\boldsymbol{x}(0)$.

3. During training, add the scores of the score-model to the scores of the noise-conditional segmentation network. To obtain scores from the segmentation network, compare the prediction $\boldsymbol{y}(t)$ from the sample $\boldsymbol{x}(t)$ at the current sampling step t to the target segmentation map $\boldsymbol{y}$ and compute the gradients that would make $\boldsymbol{y}(t)$ look more like $\boldsymbol{y}$.

5. Implementation and Experiments

5.1. Class-Conditional Sampling

Class-conditional sampling works in the same way as semantic synthesis sampling but in general is much easier to do as there is only one label per image instead of one label per pixel. As we will see, the quality of the resulting samples heavily depends on how good the classification network or the segmentation network perform on high noise and in general it is easier to estimate a label from a noisy image than a label for every pixel, where each pixel is transformed by noise up to two magnitudes higher than the individual pixel information.

That SGMs can do class-conditional sampling has been already shown in [35] but this capability was exclusively shown for an implementation in Tensorflow. To close this gap, we ported the Tensorflow implementation to PyTorch. As a classification network, a noise conditional Wide Residual Network (WideResNet) [48] was used.

Interestingly we found that the gradients computed on the WideResNet prediction were around 1000 times smaller than the gradients of the score-model. Therefore, the class conditional gradients had no effect on the image generation and had to be scaled with an experimentally motivated factor of 1000 in order to get good results. Since there was not such a scaling factor in the Tensorflow implementation, we would have been very interested to know if the class-conditional gradients in this implementation were larger natively. Unfortunately, due to TensorFlow's jit (just in time) technique, there is no way to read out any tensors during runtime. As it turns out, these scaling factors are also important in later experiments and must be determined individually for each dataset. Also, it is often not sufficient to choose a static scaling factor, but a scaling factor described by a function must be chosen.

As a qualitative proof of concept, we test the implementation on the MNIST [23] dataset, which contains handwritten digits in the range $0 - 9$ and was initially designed for handwriting recognition. Some of our results are shown in Fig. 5.1. We can conclude that the implementation works in general, although there are some samples that strongly deviate from the requested numbers. This is very likely due to the fact that for the implementation a very

Figure 5.1.: MNIST class-conditional results. *Left:* 2, *Middle:* 5, *Right:* 8

basic model and SDE were used. In further experiments the significantly more complicated NCSN++ model and the VE SDE were deployed.

5.2. Semantic Synthesis Implementation

As described in Sec. 4.5 a semantic segmentation network is needed to learn the forward process $\log p_t(\boldsymbol{y}|\boldsymbol{x}(t))$ which can be then used to solve the $\boldsymbol{y}$-dependent reverse SDE in Equ. 4.22. Effectively, this means that the semantic segmentation is learned on noisy images $\boldsymbol{x}(t)$, where the noise for timestep t is governed by the forward SDE. Since for each training image $\boldsymbol{x}(0)$ there also is a segmentation map $\boldsymbol{y}$, the guess $\tilde{\boldsymbol{y}}$ of the segmentation network can be compared to the true map to then compute an error.

5.2.1. Generating noisy training images

As mentioned above – because the forward SDE is tractable – we can easily generate as much noisy training images as we want by adding noise to the initial training images $\{\boldsymbol{x}_i(0)\}_{i=1}^N$:

$$\boldsymbol{x}(t) = \boldsymbol{x}(0) + \sigma(t) \cdot \boldsymbol{z}\,. \tag{5.1}$$

In Equ. 5.1 $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$ is Gaussian noise and $\sigma(t)$ is the noise scale for timestep t which for the VE SDE is given by

$$\sigma(t) = \sigma_{min} \cdot \left(\frac{\sigma_{max}}{\sigma_{min}}\right)^t. \tag{5.2}$$

σ_{min} corresponds to $\sigma(t = 0)$ and is always chosen as 0.01. Since the image information is in the range $[0, 1]$, a noise of 0.01 is not visible to the human eye. σ_{max} corresponds

to $\sigma(t = T \triangleq 1)$ and is chosen as the maximum pairwise distance for the dataset under investigation. During training, for each not noisy training image in a batch we then randomly choose noise scales $\sigma(t)$ with $t \in [0, 1]$ and transform the not noisy training batch into a noisy training batch by using Equ. 5.1.

5.2.2. Choosing a noise encoding

With the noisy images we can now train a semantic segmentation network. Obviously, a semantic segmentation network is expected to perform increasingly worse on increasingly noisy images. To give an idea of the difficulty of the task, the maximum noise scale for the Cityscapes dataset is 338, i.e. in images with maximum noise, the noise is 338 higher than the underlying image information! Therefore, we cannot simply train a regular segmentation network $p_\theta(\boldsymbol{x}(t))$ on noisy images, but we need to condition the network on noise $p_\theta(\boldsymbol{x}(t), \sigma(t))$, so that it can better distinguish the noise scales. There are some advanced ways to encode this noise information, including *Sinusoidal Positional Embeddings* [42] or *Gaussian Fourier Embeddings* [39]. These encoding techniques are used in NCSN++ to encode the noise information but for the segmentation network, we work with a much simpler type of encodings.

Our idea to condition the semantic segmentation is to encode the noise information as a $4th$ channel dimension in addition to the 3 color channels for each pixel. The size of a batch of (noisy) training image has the size $N \times C \times H \times W$. We want to expand the noise information of size N to size $N \times 1 \times H \times W$ and then concatenate the noise with the images to obtain a resulting batch of size $N \times (C + 1) \times H \times W$. This batch is then fed into the network.

However, there are several plausible options to express the noise of an image. The first option would be to use the noise scale itself, the second option would be to use the logarithm of the noise scale and the third option would be to use the timestep which directly correlates to noise. Recalling to the noise function in Equ. 5.2 we can plot these three options in Fig. 5.2.

In order to evaluate these possibilities, one needs to know the input format of the images into the network. Normally, the pixel values are displayed in the range $[0, 1]$ rather than in the range $[0, 255]$. But since there is noise added to the initial images the pixel ranges can be everywhere from $[0, \sigma_{min}]$ and $[0, \sigma_{max}]$, depending on the current noise scale. When we tried to learn a segmentation network from images in various pixel ranges, we got very bad

results. Therefore, we transform any perturbed image $\boldsymbol{x}(t)$ into the range $[0, 1]$ by using the following formula:

$$\boldsymbol{x}_{[0,1]}(t) = \frac{\boldsymbol{x}(t) - \min(\boldsymbol{x}(t))}{\max(\boldsymbol{x}(t)) - \min(\boldsymbol{x}(t))}. \tag{5.3}$$

It is plausible that the noise encodings should be in a related order of magnitude than the image information. Therefore, we do not use noise scales as encodings because they are up to two magnitudes larger. Training one segmentation network conditioned on the logarithm of the noise scale $\log \sigma(t)$ and another conditioned on timesteps $t \in [0, 1]$ revealed no significant differences. After 100 epochs on the cityscapes dataset, we got a peak IoU of ~ 0.212 for noise encodings and a peak IoU of ~ 0.213 for timestep encodings. Since NCSN++ is conditioned on the logarithm of the noise scale, we also choose this encoding to condition our semantic segmentation network $p_\theta(\boldsymbol{x}(t), \log \sigma(t))$.

5.2.3. Choosing a Semantic Segmentation Network

There exists a wide variety of semantic segmentation networks that we could use. The problem with state-of-the-art segmentation networks is, that they often use pretrained classification networks in their architecture which makes them hard to condition on noise, because these pretrained networks are obviously not trained on noisy images.

Therefore, we first tried to use simpler models such as U-Net (Sec. 3.1.2) which are easy to implement and easy to condition on noise. The results we got were not really promising, but it turned out that they actually were good enough to use them for semantic image synthesis. Concrete analyses of the trained segmentation networks in terms of pixel accuracy and IoU is presented in the dedicated experiment sections Sec. 5.4, Sec. 5.5 and Sec. 5.6.

We also tried to condition more complicated networks such as FCDenseNet [13], which significantly outperforms U-Net on non-noisy images, but we could only achieve a peak IoU of ~ 0.146 on noisy images, so we finally decided to use U-Net as semantic segmentation network for further experiments.

5.2.4. Choosing an error function

To compare the estimated semantic map $\tilde{\boldsymbol{y}}$ with the ground truth semantic map $\boldsymbol{y}$, we need to specify an error function. For U-Net the cross-entropy loss function is a good choice. Cross-entropy calculates the loss for an output given as probability value between 0 and 1 which is exactly what we (pixel-wise) have for a semantic segmentation network. The cross

entropy operates on the logarithm of the output probability $\log p$. If the true label of a pixel is 1 the loss is relatively low for probabilities > 0.2 and grows rapidly for lower losses. For multiclass classification as for semantic segmentation the cross-entropy loss can be written as

$$\Delta_{xy} = -\sum_{c=1}^{C} \begin{cases} 0, & \text{for } c \neq c_t \\ \log p(c_t)_{xy}, & \text{otherwise} \end{cases} \tag{5.4}$$

where C is the total number of labels, x and y are the pixel-wise positions in the segmentation map, c_t is the true label for position xy and $p(c_t)_{xy}$ is the probability the network outputs for the pixel at position xy having label c_t.

5.2.5. Choosing a scale factor function

As already touched upon in the previous section Sec. 5.1 a scale factor for the gradients of the semantic segmentation network is necessary to obtain good sampling results. When using a static scale factor, we observe that the pixel values often go into saturation, i.e. converging to 1, which leads to overexposed samples (see e.g. sample for $T = 0$ in Fig. 5.3).

We are therefore interested in finding the time interval $T \subset [0, 1]$ in which the gradients of the semantic segmentation network are the most needed, i.e. in which time interval important structural image information is formed and consolidated. With this information we hope to reduce the saturation effect on the final sample, because the semantic segmentation network is influencing the sample for a lower amount of time. We propose that for low noise (t near 0) the image structure is already consolidated and the semantic segmentation network only plays a minor role for generation. We further propose that for high noise (t near 1) the gradients of the segmentation network in theory would be beneficial for generation, but only if the network is capable to operate on high noise scales which we could not achieve up to maximum noise.

Therefore, as a scaling function we propose a function that is 0 in regions where the network performs badly and 1, in regions where the networks performs reasonable, followed by a linear decrease to 0 for regions where we suggest that the image structure has already consolidated. To find out these regions we perform two experiments. In the first experiment, we set the scale factor to 1 for $t > t_{stop}$ and to 0 for $t \leq t_{stop}$, where t_{stop} is subsequently reduced from 1 to 0 for each sample. In the second experiment we introduce t_{start} for which for $t < t_{start}$ the scale factor is 1 and for $t \geq t_{start}$ the scale factor is 0. The results – exemplary for one semantic map – can be seen in Fig. 5.3 resp. Fig. 5.4.

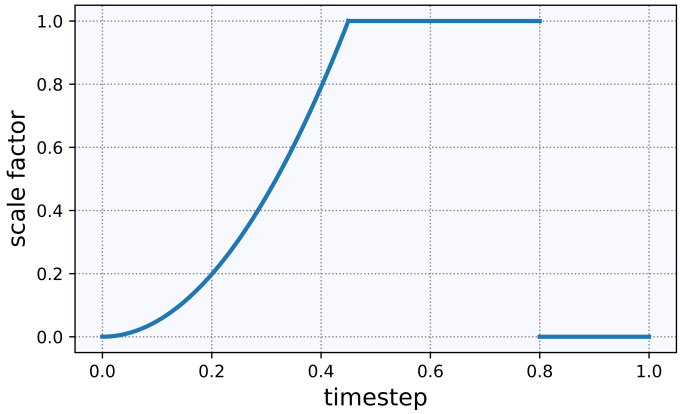


Figure 5.2.: The proposed scale factor function

We come to the qualitative result that in order to get samples that match the semantic map, the semantic segmentation network has to be started at $t > 65$ and can be stopped/decreased at $t < 0.45$. Based on the findings for these experiments, supplemented by the accuracy and mIoU curves in Fig. 5.8, we generously conclude that for $t > 0.8$ the segmentation network performs to badly to be helpful for generation and that for $t < 0.45$ the structures in the image have already consolidated. We therefore propose the following scale function:

$$s(t) = \begin{cases} 0, & \text{for } t > 0.8 \\ s_0, & \text{for } t \leq 0.8 \wedge t > 0.45 \\ \frac{s_0}{0.45^2} t^2, & \text{for } t \leq 0.45 \end{cases}, \tag{5.5}$$

where s_0 is the maximum scale factor for a given dataset, which is determined experimentally. As function in the region $0 \leq t \leq 0.45$ any kind of function that decreases sufficiently fast could be used. The final gradients for sampling are therefore calculated via

$$\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x}) + s(t) \cdot \nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{y}|\boldsymbol{x}). \tag{5.6}$$

5.3. Improvements and Adaptations

As initial experiments before testing semantic image synthesis on a large scale, we test the impact of two improvements/adaptations we made to the original score model architecture [35].

Figure 5.3.: Switching off the semantic segmentation network at different timesteps $T \in [0, 1]$

Figure 5.4.: Switching on the semantic segmentation network at different timesteps $T \in [0, 1]$

5.3.1. Mixed precision learning

vRAM usage and training speed are key aspects in training deep neural network. Complex models might consist of several million up to a billion parameters to optimize. This requires an immense amount of computing power but at least equally important, vRAM. vRAM – video random access memory – is the memory that a GPU has available during operation. Modern high-end consumer graphic cards typically have around $10GB$ of vRAM. The crux of the matter is that reaching more than $10GB$ when training a moderately complex model is really easy and when training such complex models as NCSNs, the vRAM demand can increase beyond $100GB$, making training deep neural networks a very cost intensive business. Also, when training complex models one can expect to need one week or more of training time, making testing and finetuning of models very exhausting.

To mitigate these problems, the concept of *Mixed Precision Learning* [25] was developed, decreasing both vRAM usage and training time with simultaneous retention of training quality, i.e. training loss. Normally, each parameter of a network is stored as a float with 32 bits of precision. Considering a network with ~ 250 M parameters, this would be equivalent to ~ 1 GB. What sounds not so much at first quickly becomes unmanageable when thinking of the series of matrix arithmetic's the parameters undergo and each calculation storing extra data such as gradients.

The idea of mixed precision learning is to use floats with a 16 bit precision wherever possible. In order to not lose model accuracy, two concepts are applied. First, there is always a 32 bit master copy of the model weights. For the forward pass, this master copy is converted to 16 bit, where the complex gradient calculations happen. In the backward pass these 16 bit gradients are converted back to 32 bit where they are then used by the optimizer to update the master weights. But this improvement during gradient calculation comes with the flaw of losing some gradients as every number $< 2^{-24}$ is equal to 0 for 16 bit precision but in most models there is at least some relevant gradient information in the range $[2^{-27}, 2^{-24})$. To mitigate this effect the concept of loss scaling was proposed. Loss scaling multiplies the 32 bit loss after the forward pass by a scale factor to move them in the 16 bit range in which the gradients are then computed in the backward pass. Thereafter the scaled 16 bit gradients are converted to scaled 32 bit gradients and then divided by the same scale factor as before. Finally, these scaled gradients are used by the optimizer to update the master weights. The full concept can be seen in Fig. 5.5.

As the mixed precision technique was not used in the original SGM paper [35] it was now

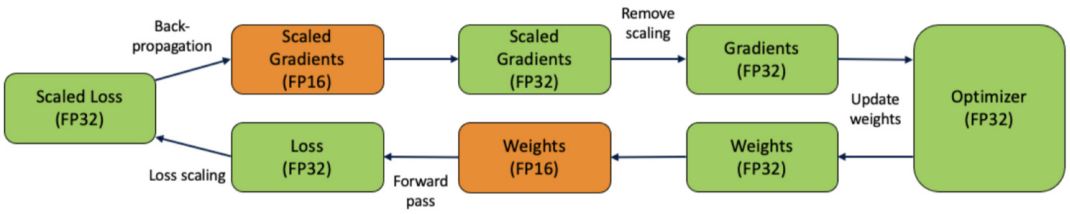


Figure 5.5.: Concept of Mixed Precision

GPU type	Mixed precision **Off**	Mixed precision **On**
RTX 2080 Ti	7791MB	7367MB
A100	39039MB	29481MB

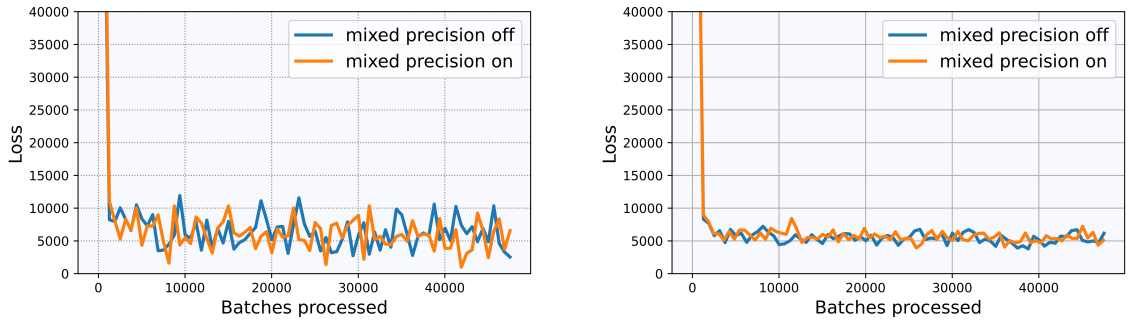
Table 5.1.: vRAM usage w/ and w/o mixed precision

implemented using off-the-shelf PyTorch tools for mixed precision, to check how much this technique improves vRAM usage and training time for the NCSN. To do so we trained the NCSN++ on the Cityscapes dataset on two different GPUs with the same model settings. For each GPU 2×100 epochs were passed, once with mixed precision on and once with mixed precision off. The first GPU was a *Nvidia GeForce RTX 2080 Ti* with 11 GB of vRAM and the second GPU was a *Nvidia A100* with 40 GB of vRAM. For both GPUs we used the maximum possible batch size for non-mixed precision training which is 1 for the RTX 2080 Ti and 5 for the A100. The effects of mixed precision training on vRAM usage and training time can be seen in Tab. 3.1 resp. Tab. 3.2. For the vRAM usage the total vRAM needed is shown and for the training time the average training time for one epoch is shown.

For the A100 serious improvements on vRAM usage can be observed. For the RTX 2080 Ti there are only little improvements for vRAM usage but good improvements on training time. In general, mixed precision is expected to perform way better on the new Ampere GPU architecture (A100) than on the older Turing GPU architecture (RTX 2080 Ti). To ensure that the training quality does not suffer from the use of mixed precision, the loss curves for all test setups are compared in Fig. 5.6. The higher fluctuations for the RTX 2080 Ti are due to the fact that the batch size is smaller, therefore leading to major loss differences

GPU type	Mixed precision **Off**	Mixed precision **On**
RTX 2080 Ti	1494s	1206s
A100	965s	987s

Table 5.2.: Training time per epoch w/ and w/o mixed precision



(a) Loss curve RTX 2080 Ti (b) Loss curve A100

Figure 5.6.: Comparing loss curves for mixed precision **on** an **off** for both GPUs

in a batch-to-batch comparison. In general, it can be seen that mixed precision has no notable effect on training loss. Also, a qualitative comparison of generated samples shows no difference perceptible by a human. We therefore conclude that – at least for non-FID records breaking attempts – mixed precision should be used and we do so for all upcoming experiments. It should be noted, however, that in a few cases, unstable behavior occurred during mixed-precision learning, resulting in nan values at training loss. In these cases, the run had to be discarded.

5.3.2. Training on arbitrary image sizes

When training a generative model, the image size the model gets as input for learning can have a large effect on what the model learns. Here not the total image size of the images is meant but the image size the model gets as input. As an example, the cityscapes dataset (downscaled) consists of 256×512 pixel images. To reduce vRAM during training, the images from the dataset often are randomly cropped to another resolution, here 256×256 pixels. For some datasets this has only little negative effects on the model accuracy, e.g. for landscapes where the relative position of objects to each other only plays a minor role. But for datasets as the cityscapes dataset it is quite important for realistic results that the model learns the logic of images showing street scenes, i.e. learning that the street is in the middle, that there are parking cars on the left and right side of the image and so on. Training on too small crops can hinder the model to learn such logic.

As the version of NCSN [35] does not support training on/sampling of non-square images, the model was adapted to do so. Actually, the model already was able to process non-square images natively, but there was a small bug that we fixed to make non-square training/sampling

Figure 5.7.: Unconditional samples of NCSN trained on cropped images (*top*) and full size images (*bottom*)

possible. To show the importance of this bugfix, we investigate the influence of the above described effect on NCSN. For this purpose, we trained two identical NCSN, one on 256×256 pixels crops and one on the 256×512 pixels original images. Some example results can be seen in Fig. 5.7. It is clearly visible that the full image model outperforms the cropped image model when the focus of evaluation is on street scene logic. It must be mentioned that for semantic image synthesis – the task for our experiments – this logic information actually is given by the semantic map which initially gives rise to the idea of training on crops at all. Nevertheless, we suspect that a model knowing the logic without a semantic map also performs somewhat better than if it did not. For that reason, models for further experiments on the cityscapes dataset were always trained with the full size images. For larger images such as landscape images from Flickr which reach resolutions up to 2048×1024 pixels this strategy is computationally not feasible and we chose a crop size of 512×512 pixels.

5.4. A competitive experiment on the Cityscapes dataset

As a first experiment, we want to test semantic image synthesis with SGMs on the Cityscapes dataset [5] (Sec. 5.4.1). Because of the similar image scenery in every picture and the small number of classes the Cityscapes dataset is a relatively easy and therefore popular dataset for tasks as semantic segmentation and semantic image synthesis. For computational reasons

we will work on downsampled images of size 512×256 pixels but later experiments on landscape images (Sec. 5.5) are also tasked to generate higher resolution images. We will investigate the influence of various model and sampling parameters on sampling quality in Sec. 5.4.3. Thereafter, we will compare our resulting samples with state-of-the-art semantic image synthesis models in Sec. 5.4.4 using the evaluation metrics described in Sec. 5.4.2 and a human perceptual study.

5.4.1. The Cityscapes dataset

The Cityscapes dataset [5] is a dataset containing pictures of street scenes shot in various German cities in daylight and good/medium weather conditions. It contains 5000 2048×1024 8 bit color images with a fine annotated label map for each image. Moreover, it contains $20{,}000$ images with coarse annotated label maps which were not used for the experiments. 500 of the 5000 images/label maps are only for validation. The maps of the validation images are later used as basis for semantic image synthesis and the generated images are then compared to the 500 original images to compute FID scores (Sec. 5.4.2). Each semantic label map consists of 30 classes, e.g. street, car, bus, ... We combine 11 of these classes to one common background class, so we work with 20 training classes in total.

5.4.2. Metrics

In the course of the following experiments, different evaluation metrics are used to quantify the quality of the samples:

Fréchet inception distance

The Fréchet inception distance (FID) [11] is a metric for evaluating the quality of generated images in comparison to the real images and was initially developed to evaluate GAN performance. The goal of such score is to capture the similarities best possibly between real images and samples.

FID uses the outputs of generated and original images of the Inception-v3 [38] image recognition model. For each image, the output consists of 2048 activation features. Between the two resulting collections of 2048 feature vectors for generated and real images, the Fréchet

distance defined as

$$d^2 = \|\mu_1 - \mu_2\|^2 + \mathrm{Tr}(C_1 + C_2 - 2\sqrt{C_1 \cdot C_2}) \tag{5.7}$$

is calculated. Here μ_i are the feature-wise means of the real and generated images and C_i are the covariance matrices for the real and generated feature vectors. In comparison to other evaluation scores, the FID score shows higher (worse) values for all kind of mutations such as noise, blur and distortions. However, it should be noted that a score is never able to truly capture the similarities a human would see between images. Even noise so small that it cannot be noticed by the human eye can lead to a worse FID score [19].

Pixel accuracy

To evaluate the performance of a semantic segmentation network, the pixel accuracy for the predicted semantic map in relation to the true map can be used. The pixel accuracy is calculated via

$$acc = \frac{TP + TN}{TP + TN + FP + FN}, \tag{5.8}$$

where TP is true-positive, TN is true-negative, FP is false-positive and FN is false-negative. One problem with the accuracy score is that is does not capture the performance for classes which do not occur very often. If the network performs poorly on a class, this has nearly no effect on the accuracy score when there are only few pixels with this class in the image.

Mean Intersection over Union

The Mean Intersection over Union (mIoU) also is a measure for the performance of semantic segmentation network. The mIoU fixes the problem of the pixel accuracy score by calculating the intersection over union

$$IoU_i = \frac{|A_i \cap B_i|}{|A_i \cup B_i|} \tag{5.9}$$

for each class i where A_i is the predicted region and B_i the real region for class i. For the values IoU_i then the mean value is calculated.

5.4.3. Training the networks

Training NCSN++

First, the unconditional NCSN++ has to be trained. We train the model on downsampled but not cropped 512×256 pixel images. The choice for this resolution is only a matter of available compilation power and GPU vRAM. For resolutions of 1024×1024 pixels, the authors of [35] trained their model on 8 Nvidia Tesla V100 32GB GPUs with a batch size of just 8. Also, the sampling procedure of one image of this resolution took them around 50 m which is infeasible if one wants to sample a lot of images or wants to test a lot of configurations.

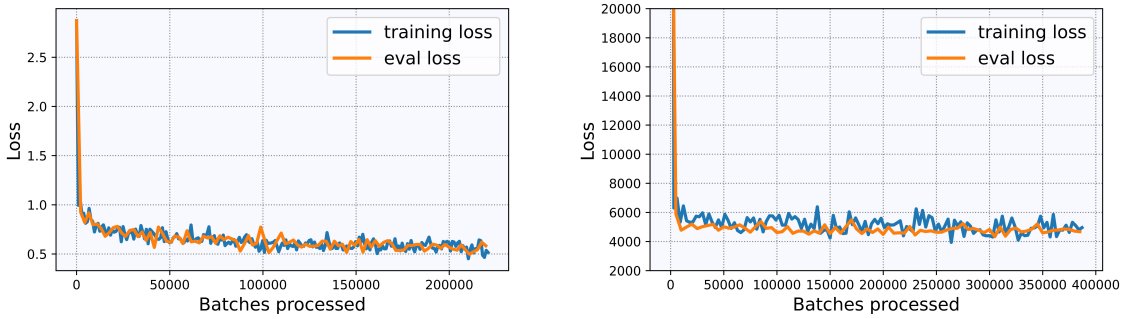
There are roughly 25 parameters describing the architecture design and training procedure of NCSN++. For the complex model architecture parameters, we adopt the parameters the authors of [35] found best working for resolutions of $\sim 256 \times 256$ pixels.

One important parameter we had to change is the parameter for the maximum noise scale σ_{max}. As explained in Sec. 4.3.1 the maximum noise scale should be chosen as the maximum Euclidean distance between the training images. For a dataset with n images the number of distances calculations is $1 + 2 + \cdots + n = \sum_{k=1}^{n} k = \frac{n^2+n}{2}$ and therefore the complexity of this computation is $\mathcal{O}(n^2)$. With such a high complexity it is computationally infeasible to compute the distances between all images, i.e. to find the true maximum Euclidean distance, for large datasets. As a compromise, we decided to calculate $20{,}000{,}000$ distances between randomly chosen training images. We then get the maximum noise scale value for the cityscapes dataset as $\sigma_{max} \approx 338$.

With the final parameters set up we trained our model on one Nvidia A100 40GB GPU with a batch size of 8 for 1000 epochs which took ~ 11 days. Note that, as justified in Sec. 5.3.1, the network was trained with mixed precision. The loss curve of the network is shown is Fig. 5.8. The loss curves suggests that the network did not memorize the training images, i.e. is not subject to overfitting.

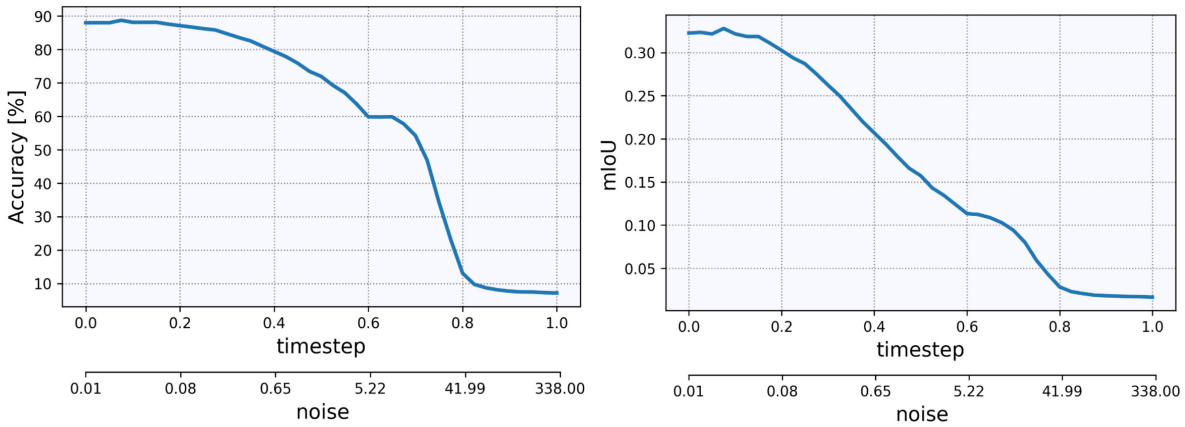
Training noise-conditional U-Net

Following Sec. 5.2 on implementation of semantic image synthesis, we implement the classical U-Net from [30] and adopt it to be conditioned on $\log \sigma(t)$. The network is then trained on noisy cityscapes training images whose noise scales are calculated via Equ. 5.2 with random timesteps $t \in [0, 0.8]$. The network is not trained on the noise scales corresponding to



(a) U-Net (b) NSCN++

Figure 5.8.: Training and evaluation loss of U-Net and NCSN++ trained on the Cityscapes dataset



(a) Pixel Accuracy (b) Mean IoU

Figure 5.9.: Pixel accuracy and mean IoU for noise-conditional U-Net trained on the Cityscapes dataset

$t \in (0.8, 1]$. This is due to the realization that the network performs poorly on such high noise scales and is therefore no help in the image generation process. For this reason, according to the scaling factor function in Equ. 5.5, the segmentation network does not need to predict an image with such high noise anyway, so there is no need to train it for high noise $t > 0.8$.

Given the same maximum noise scale $\sigma_{max} \approx 338$ as for NSCN++ and a learning rate of 0.0002 for the Adam optimizer [20] we use for optimization, we trained the network for 600 epochs with a batch size of 8. In Fig. 5.8 the loss curve is shown and as with NCSN++ there is no sign of overfitting. In order to get a quantitative impression of the performance of the segmentation network, we evaluated the pixel accuracy and the mean IoU separately

Figure 5.10.: Unconditional samples of NCSN++ trained on the Cityscapes dataset

for 41 timesteps in $[0, 0.8]$. The resulting curves in Fig. 5.9 show, as expected, a decrease of pixel accuracy and mIoU towards higher noise scales. It can also be seen that the values for $t = 0.8$ are not significantly higher than for $t > 0.8$, where the network was not trained. This indeed shows that for $t >= 0.8$ the network is just guessing. A surprising finding in the two curves is the plateau region $\approx 0.6 < t < 0.7$ for which we do not have a satisfying explanation.

5.4.4. Finding optimal sampling parameters

There are two parameters which are essential for high quality samples. The signal-to-noise ratio snr of the Langevin sampler and the maximum scale factor s_0 from Equ. 5.5.

According to [34] a high snr produces samples of higher image quality while a lower snr produces samples of higher diversity. Moreover, the authors of [34] recommend choices of $snr \in [0.05, 0.2]$. We qualitatively tested values in this range on unconditional samples of the Cityscapes dataset. It is really hard to see any differences in the samples, but we think that the higher sample quality manifests itself in sharper edges which in our opinion does look worse for higher values of snr for the Cityscapes dataset. We therefore decide to go with a value of $snr = 0.1$, but the reader is invited to form his or her own opinion by looking at the samples in Fig. 5.11. The influence of snr was also investigated on conditional samples but we were not able to find any differences in sampling quality at all.

As seen in Sec. 5.2.5 the maximum scale factor s_0 is weighting the relation between the gradients of the score network and the gradients of the semantic segmentation networks. In qualitative experiments, we found that values of s_0 in the range $[0.0005, 0.0035]$ produce visually appealing results that also comply to the given semantic map. We noticed that for

Figure 5.11.: Unconditional samples of NCSN++ for different values of snr

s_0	0.0005	0.001	0.0015	0.002	0.0025	0.003	0.0035
FID	98.78	100.32	100.59	102.48	104.52	104.59	106.40
Mean IoU	0.2422	0.2668	0.2879	0.3076	0.3023	0.3192	0.3263
Pixel acc	70.46%	72.33%	73.78%	74.45%	75.13%	75.33%	75.30%

Table 5.3.: Evaluation of maximum scale factor s_0 on various scores for the Cityscapes dataset

higher values the samples looked slightly worse, i.e. over-saturated, and for lower values the compliance with the semantic map decreases. To quantify these effects we sampled 100 images for $s_0 = 0.0005, 0.001, \ldots, 0.0035$ using the same 100 segmentation maps for each value of s_0. The results in Tab. 5.3 confirm that the images quality (FID) decreases and the pixel accuracy as well as the mean IoU increases with higher s_0. We think that $s_0 = 0.002$ is a good compromise between image quality and semantic segmentation map compliance, and therefore use this value for sampling.

5.4.5. Results and Quantitative Comparisons

With the trained models and the sampling parameters $snr = 0.1$ and $s_0 = 0.002$ we sampled images on all 500 testing semantic label maps of the Cityscapes dataset. We further did the same for the models CRN [4], pix2pixHD [44] and SPADE [26], either using pretrained models (CRN) or training them ourselves if no pretrained models were provided (pix2pixHD, SPADE) for resolution 512×256. It is worth noting that these 3 models (and adaptations of them) are state-of-the-art models, occupying 6 of the top 10 places in semantic image synthesis benchmarks (sorting by mIoU) on the Cityscapes dataset (see `https://paperswithcode.com/sota/image-to-image-translation-on-cityscapes`).

Figure 5.12.: Visual comparison of semantic image synthesis results on the Cityscapes dataset

	CRN [4]	pix2pixHD [44]	SPADE [26]	NCSN++ [35]	Oracle
Mean IoU	0.5345	0.5172	**0.6191**	0.3089	0.6181
Pixel acc	74.86%	79.59%	**81.50%**	74.95%	81.00%

Table 5.4.: Semantic segmentation scores on results by different methods on the Cityscapes dataset. Oracle describes scores on original images

To compare our results to the ones of CRN, pix2pixHD and SPADE we use several metrics: Qualitative visual comparisons, image quality comparisons using the FID score (Sec. 5.4.2) and semantic segmentation comparisons using pixel accuracy (Sec. 5.4.2) and mean IoU (Sec. 5.4.2). To calculate the FID scores, a python package called *pytorch-fid* [31] was used. To produce the semantic maps from the sampled images to calculate pixel accuracy and mean IoU, a pretrained version of the semantic segmentation network *Dilated Residual Network* (DRN) [47] was used. The reason for this choice is that DRN generally performs very well on the Cityscapes dataset, that DRN was used in similar model comparisons in [16] and [26] and that we want to prevent bias by using another network than U-Net for our analysis.

The visual comparison of some explicitly selected samples is presented in Fig. 5.12. In our opinion, our results have the possibility to outperform the visual appearance of the other models. However, there are two very important restrictions to this opinion: First, our samples do not always look better, therefore we provide more, uncurated samples in the appendix in Tab. A.1. And second, apparently our model is very bad at sampling fine structures of the semantic map, e.g. persons or street signs.

This poor performance on fine structure can also be seen in the pixel accuracy and mIoU scores in Tab. 5.4. While the pixel accuracy score is acceptable in comparison to the other models, the mIoU is not. For the quality comparison using FID scores in Tab. 5.5 we conclude that our model perform quite good in comparison to the other models.

All in all we conclude to things for this experiment: First, our approach of semantic images synthesis using SGMs is capable to keep up with state-of the-art models. Second, there is room for improvements, especially in semantic map compliance for fine structures. We think that a more complex noise-conditioned semantic segmentation network than the relatively simple U-Net could increase mean IoU and pixel accuracy scores if being better for higher noise scales. We also think that such a network would increase sample quality, as we suppose that the main factor for bad quality is not NCSN++ but the semantic segmentation network performing poor on high noise scales.

	CRN [4]	pix2pixHD [44]	SPADE [26]	NCSN++ [35]
FID	71.43	83.91	**56.03**	66.46

Table 5.5.: FID scores on results by different methods on the Cityscapes dataset

5.5. Synthesizing high resolution landscapes

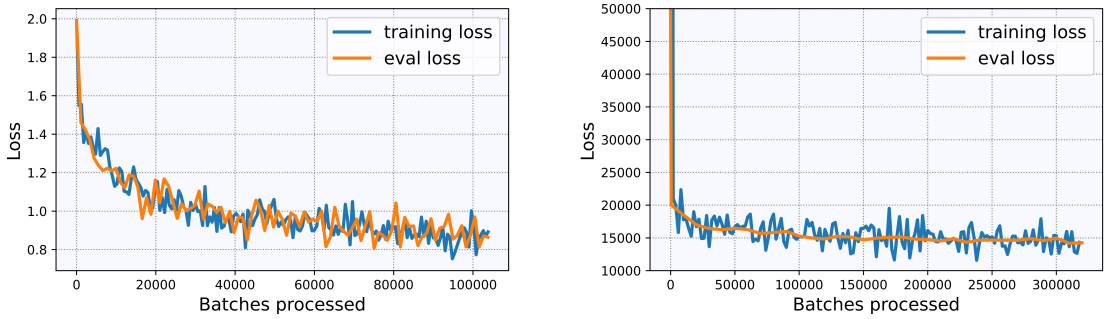
In the experiment on the Cityscapes dataset in Sec. 5.4 we found that our semantic image synthesis model performs poor on fine structures like persons and traffic lights but good on coarse structures like streets, cars and vegetation. For that reason, we want to test our model on a dataset with fewer labels and more coarse structures. Inspired by impressive results from Esser and Rombach [8] on a dataset called *S-FLCKR* which contains 79266 landscape images from the image platform Flickr, we chose to test our model on this dataset. The semantic maps for these images were generated with a COCO-stuff [24] segmentation network which also contains some landscape labels. Since these generated semantic maps are far from perfect, they do not represent *ground truth* and therefore most quantitative measures cannot be applied to our results.

5.5.1. Training and Sampling

For training of our models, we use the same model parameters as described in Sec. 5.4, except for the maximum Euclidean distance, which was calculated to be $\sigma_{max} \approx 440$. With a batch size of 8 resp. 22, NCSN++ and U-Net were trained for 30 resp. 10 epochs. One important deviation from the training procedure for the Cityscapes dataset is that the landscape images are of various resolutions up to 1024×2048 and so the models were trained on 512×512 randomly cropped and horizontally flipped images. This higher training resolution will allow us to sample images of higher resolution but also drastically increases vRAM usage during NSCN++ training to $\approx 80GB$ and sampling time to ≈ 1 h per sample. We present loss curves as well as mIoU and pixel accuracy curves in Fig. 5.13 and Fig. 5.14.

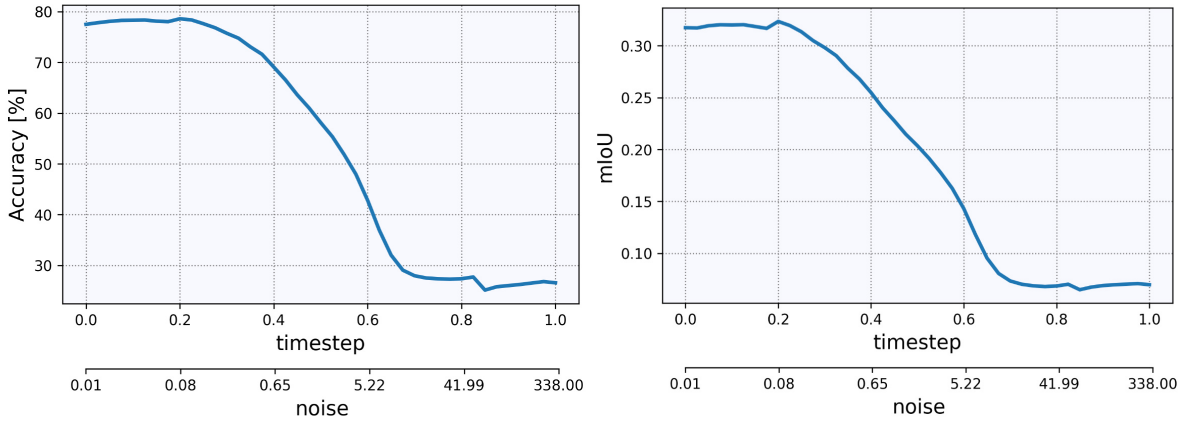
In order to get a semantic segmentation network that works well on landscapes, we condensed the 182 class labels of COCO-stuff to 6 landscape classes – Sea, Mountain, Grass, Forest, Sky, Clouds – and one "unlabeled" class. During training, the model is penalized only for incorrect predictions on the 6 landscape images. For sampling, we drew our own semantic maps using the 6 classes described above.

For the sampling parameters s_0 and snr, no quantitative experiments were conducted,



(a) U-Net (b) NSCN++

Figure 5.13.: Training and evaluation loss of U-Net and NCSN++ trained on the S-FLCKR dataset



(a) Pixel Accuracy (b) Mean IoU

Figure 5.14.: Pixel accuracy and mean IoU for noise-conditional U-Net trained on the S-FLCKR dataset

because the enormous sampling time of $\approx 1\,\mathrm{h}$ per image did not allow us to accumulate enough samples for an extended analysis. Qualitative examinations together with the insights from Sec. 5.4.4 let us conclude that $s_0 = 0.00002$ and $snr = 0.1$ work okay for the S-FLCKR dataset. Furthermore, we adjusted the start value in Equ. 5.5 from 0.8 to 0.7 because the curves in figure Fig. 5.14 show that the semantic segmentation network performs poorly for $t > 0.7$. One unconditional sample can be found in Fig. 5.15, more are presented in Tab. A2.

Figure 5.15.: An unconditional sample of NCSN++ trained on the S-FLCKR dataset. More samples in Tab. A2

5.5.2. Results

We show one of our conditional samples in Fig. 5.16 and some more in Tab. A3. As one can see the resulting image is not perfect in terms of texture quality and semantic map compliance. In this result we see one of the biggest disadvantages of our approach for semantic image synthesis: The long sampling time and the strong influence of small parameter changes on the sampling quality and semantic map compliance. Besides the parameters snr and s_0 there are also the parameters defining the cases in Equ. 5.5. With a sampling time of $\approx 1\,\mathrm{h}$ it is hard to find a setting that works flawlessly. Moreover, the samples for the S-FLCKR dataset were found to be much more sensitive to parameter changes than the samples for the Cityscapes dataset. For example, while for Cityscapes a parameter $s_0 \in [0.001, 0.004]$ produces results with only minor visible differences, for S-FLCKR a change from $s_0 = 0.00002$ to 0.00004 changes the images quality by a lot. The same finding holds true for the other parameters. We show some samples with different parameters in Fig. 5.17.

For us, it remains an open question for further research why the gradients of the semantic segmentation network affect the sampling quality in such an unintuitive way and how to mitigate these negative effects.

Figure 5.16.: A conditional sample of NCSN++ trained on the S-FLCKR dataset. Semantic map in bottom left corner. More samples in Tab. A3

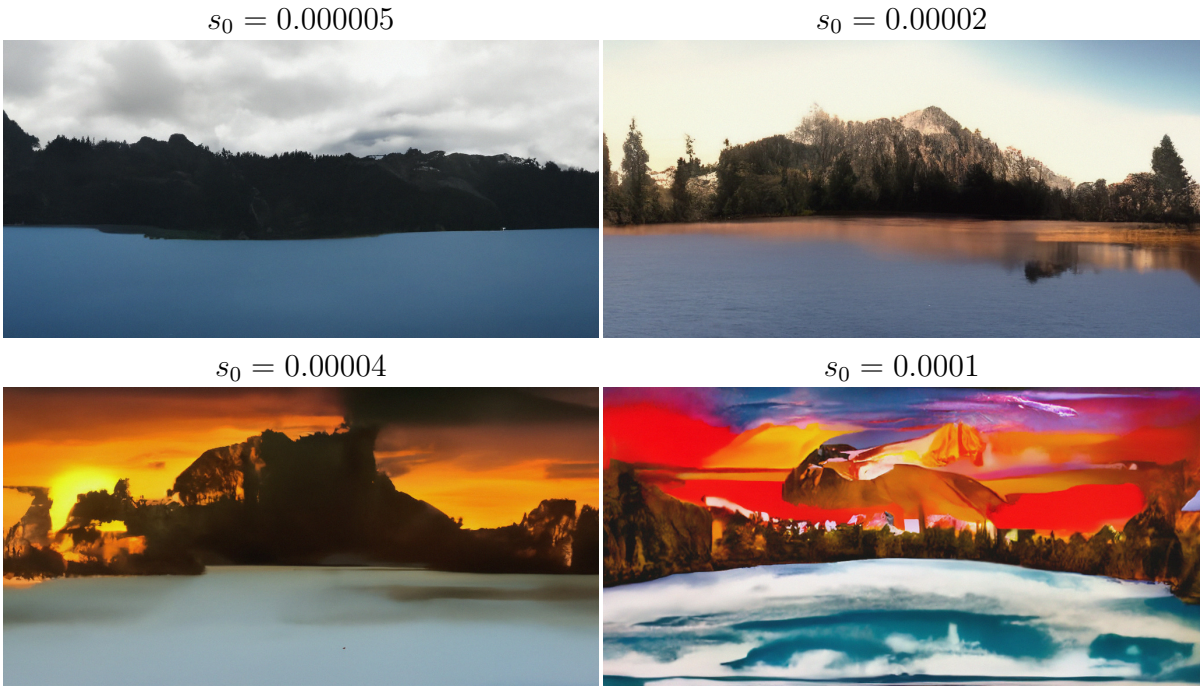


Figure 5.17.: Conditional S-FLCKR samples for different values of s_0

5.6. ADE20K - Pushing the model to its limits

In this experiment, we want to qualitatively evaluate the performance of the proposed semantic image synthesis model on the ADE20K dataset [49]. The goal is to get a feeling for the limits of the current model, especially for the relatively simple U-Net semantic segmentation network. The ADE20K dataset is generally difficult to learn due to its large variety of scenes and also due to the small image sizes, and the sample quality is to be expected poorer than for simple datasets such as Cityscapes.

5.6.1. The ADE20K dataset

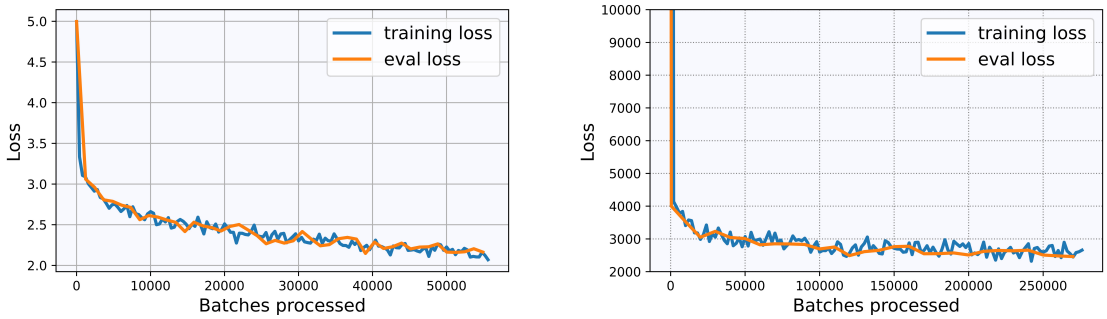
The ADE20K dataset [49] is designed for semantic segmentation and semantic image synthesis of images with no general type of scene. It consists of 25574 training and 2000 testing images of various resolutions which all come with a fine annotated label map. The images span 365 different scenes and contain $707{,}868$ unique objects from $3{,}688$ categories. The 365 scenes are very diverse, containing scenes of urban buildings, landscapes, rooms, such as bathrooms and kitchens, scenes of daily life and many more.

5.6.2. Training and Sampling

As in the previous experiments, we did not change any complex model parameters. The maximum Euclidean distance between the images was calculated to $\sigma_{max} \approx 394$. We then trained both the NSCN++ and the U-Net on 256×256 pixel randomly cropped images and chose a batch size of 16 for the NCSN++ and 64 for U-Net. NCSN++ and U-Net were trained for 220 resp. 150 epochs.

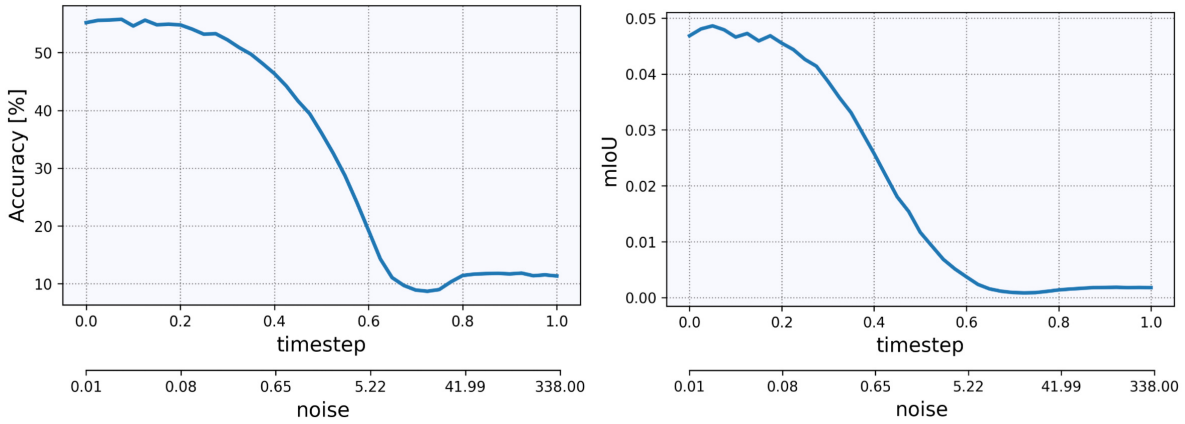
Sampling parameters were only qualitatively investigated, and it was found that a value of $snr = 0.075$ for the signal-to-noise-ratio and $s_0 = 0.0025$ for the maximum scale factor produces decent results. As we will see in Sec. 5.6.3, there is no need for parameter fine-tuning since the resulting samples defy any quantitative analysis.

The loss curves of U-Net and NCSN++ are presented in Fig. 5.18. We also show some unconditioned samples in Fig. 5.20 which convincingly show why the ADE20K is considered a dataset for semantic image synthesis and not for general image generation.



(a) U-Net (b) NSCN++

Figure 5.18.: Training and evaluation loss of U-Net and NCSN++ trained on the ADE20K dataset



(a) Pixel Accuracy (b) Mean IoU

Figure 5.19.: Pixel accuracy and mean IoU for noise-conditional U-Net trained on the ADE20K dataset

Figure 5.20.: Unconditional samples of NCSN++ trained on the ADE20K dataset

Figure 5.21.: Semantically synthesized images for the ADE20Kdataset.
Bottom: Synthesized Images, *Top*: Corresponding real images

5.6.3. Results

Some final samples of semantic image synthesis on the ADE20K dataset are presented in Fig. 5.21. It is quite clearly visible that the sample images have only remotely anything to do with the original images. We do not see the reasons for this in the image quality of the NCSN++. It is true that the unconditional samples are not very realistic, but this was expected for a dataset such as ADE20K. Conversely, we even find that the unconditional samples of NCSN++ capture the character of the dataset quite well, and besides, the logical information should be provided by the semantic segmentation network anyway. Therefore, we suspect that the poor sample quality is more likely due to the semantic segmentation network not performing well on noisy images, which is supported by the findings Fig. 5.19. However, we are not entirely sure why this poor segmentation performance leads to the very artistic-looking image style, as we see no evidence of this behavior in unconditional samples.

From the results of this experiment, we conclude that a simple noise-conditioned U-net simply does not perform well enough to be used in semantic image synthesis with SGMs for complex datasets. In order to generate good samples on such datasets, we assume that a semantic segmentation network specifically designed for noise-conditioned semantic segmentation must be developed.

6. Conclusion and Outlook

In this thesis, we presented the use of *Score-Based Generative Models* [34, 37, 35] for *Semantic Image Synthesis*. Semantic image synthesis provides human control over the semantic composition of an image, making image generation more versatile and useful for image generation applications. Furthermore, it enables image generation on datasets that would otherwise be hard to learn by unconditional models. Semantic image synthesis with score-based generative models brings a number of benefits that makes it particularly desirable to use: Unlike standalone models such as CRN [4], pix2pixHD [44], and SPADE [26], which are explicitly designed for semantic image synthesis and are often not further developed after publication, the approach presented in this thesis is not bound to a concrete architecture, but is a modular system that can evolve over time and has many parts that can be adapted and improved independently. Any future advancement in noise-conditional semantic segmentation networks, score-based generative models or sampling techniques can be used for semantic image synthesis in an easy-to-implement plug-and-play fashion. After training an unconditional score model like NSCN++ [35], one can then decide how to use this model. Combined with other networks conditioned on noise, – e.g. our noise-conditional U-Net for semantic image synthesis – the trained model can be used for example for colorization, class-conditional sampling, or semantic image synthesis, or presumably all at once.

In the conducted experiments (see Sec. 5.4, Sec. 5.5, Sec. 5.6), we show not only that semantic image synthesis with score-based generative models is generally possible, but also that we can achieve good results even with a very simple semantic segmentation model like U-Net [30]. The comparison of our samples on the Cityscapes dataset [5] with those of the state-of-the-art models CRN, pix2pixHD and SPADE yielded promising results, although they could not beat the favorite SPADE. As a result of this experiment, our model reached the second best FID score [11] and the third place in pixel accuracy. However, our model was found to perform very poor on fine structures of the semantic map. This problem is due to the limited capability of the simple noise-conditional U-Net on high noise scales, which was confirmed on the ADE20K [49] dataset where the poor U-Net performance allows for

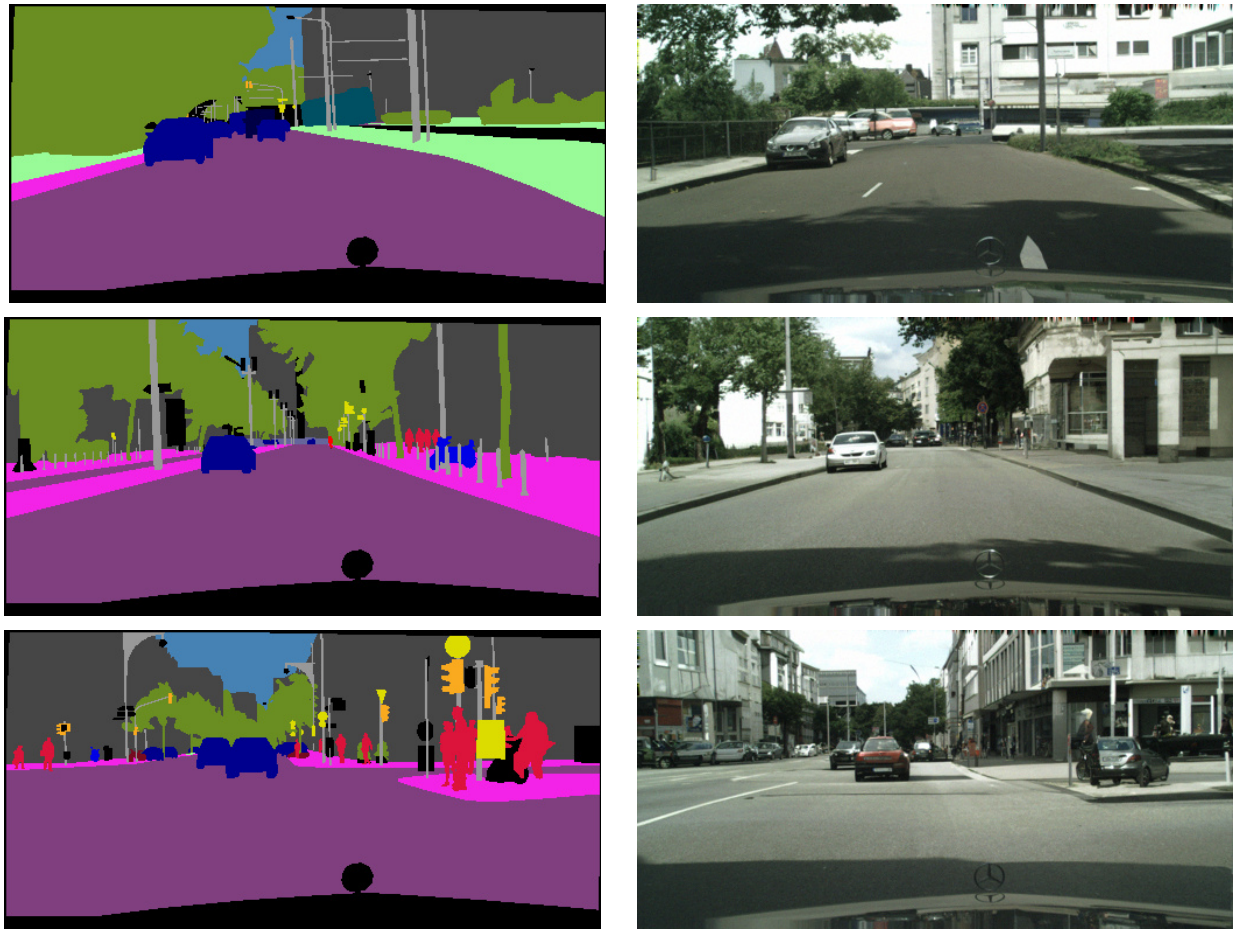
no realistic semantic image synthesis. In turn, on a simpler dataset with landscape images scraped from Flickr with only few labels and coarse structures in the semantic maps, our approach is able to generative good-looking images on high resolutions up to 1024×512 pixels.

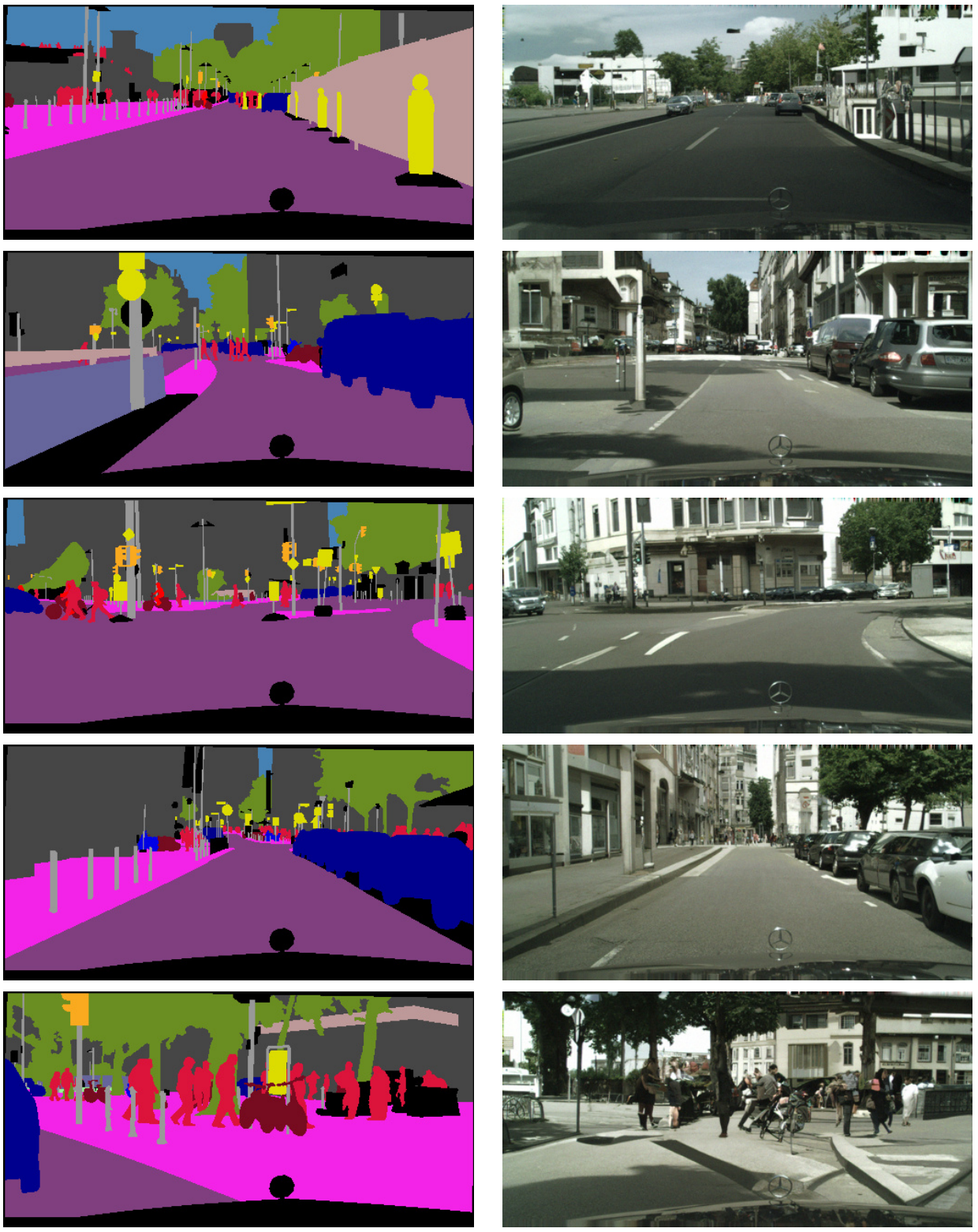
In conclusion we see two main issues that have to be tackled in future research to really make semantic image synthesis with score-based generative models an attractive state-of-the-art approach: First, one needs to find a semantic segmentation model that works better on high image noise. We suggest that such a model will most likely need to be developed specifically for this task, as our attempts to use more complex models such as FCDenseNet [13] performed worse than U-Net. Also, most modern semantic segmentation models rely on pretrained models which obviously are not conditioned on noise. And second, and there already is a lot of research in this direction, the sampling time of score-based generative models should be decreased, as it is a large hindrance for parameter fine-tuning and does not allow for real time image generation tasks. In [18], a promising new sampler was presented that can reduce sampling times by a factor of up to 10. Another interesting approach is the use of score-based generative models in latent space, which was recently achieved in [41] and allows for up to $600\times$ faster sampling speed than before. However, whether our approach is also applicable in latent space is an open question and left to future research.

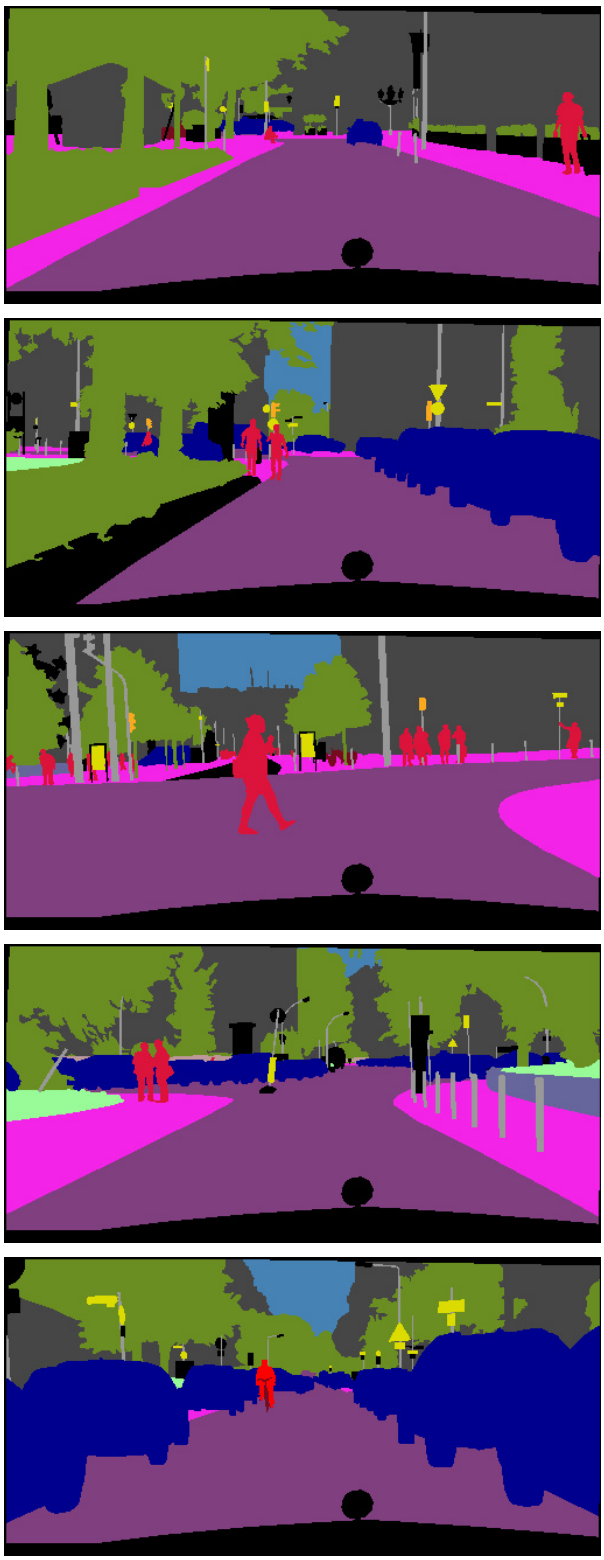
Altogether, our approach combines the versatility and high sampling quality of score-based generative models with semantic image synthesis, making it a powerful framework for future research in this field.

A. Supplementary Materials

A.1. Cityscapes samples







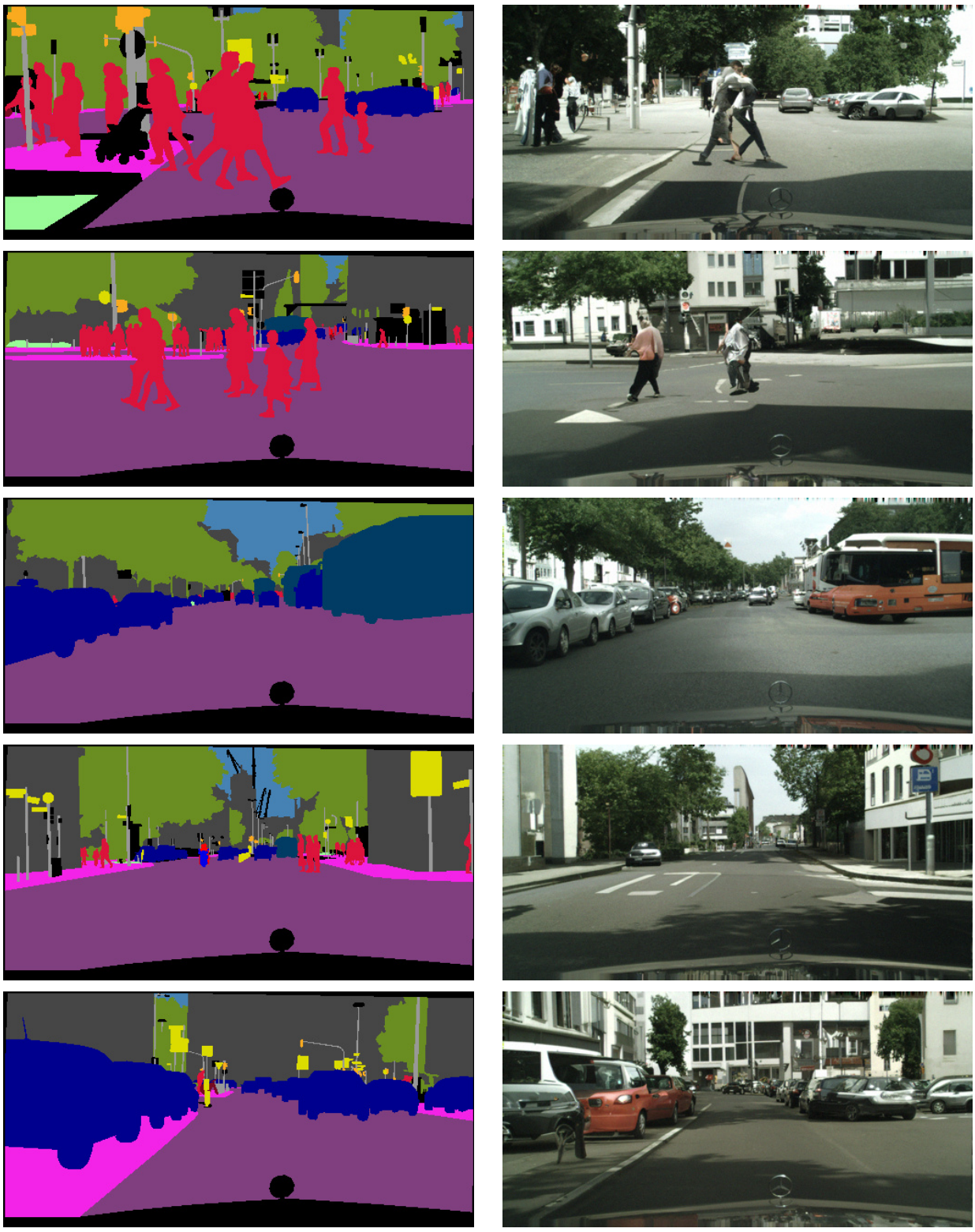


Table A.1.: Uncurated conditional samples of NCSN++ for the Cityscapes dataset

A.2. S-FLCKR samples

Table A.2.: Unconditional samples of NCSN++ for the S-FLCKR dataset

Table A.3.: Conditional samples of NCSN++ for the S-FLCKR dataset

B. Lists

B.1. List of Figures

B.2. List of Tables

B.3. List of Algorithms

C. Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] B. D. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

[3] A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

[4] Q. Chen and V. Koltun. Photographic image synthesis with cascaded refinement networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1520–1529, 2017.

[5] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] L. Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404. Morgan Kaufmann, 1990.

[7] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[8] P. Esser, R. Rombach, and B. Ommer. Taming transformers for high-resolution image synthesis, 2020.

[9] C. W. Gardiner. *Stochastic Methods : a Handbook for the Natural and Social Sciences*. Springer, 2009.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3, 06 2014.

[11] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 6629–6640, 2017.

[12] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020.

[13] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[14] A. Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005.

[15] S. Ioffe and C. 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*, page 448–456, 2015.

[16] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, 2017.

[17] A. Jolicoeur-Martineau. The new contender to gans: score matching with langevin sampling. `https://ajolicoeur.wordpress.com/the-new-contender-to-gans-score-matching-with-langevin-sampling/`.

[18] A. Jolicoeur-Martineau, K. Li, R. Piche-Taillefer, T. Kachman, and I. Mitliagkas. Gotta go fast when generating data with score-based models. *ArXiv*, abs/2105.14080, 2021.

[19] A. Jolicoeur-Martineau, R. Piché-Taillefer, I. Mitliagkas, and R. T. des Combes. Adversarial score matching and improved sampling for image generation. In *International Conference on Learning Representations*, 2021.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[21] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

[22] A. Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 2012.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2015.

[25] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu. Mixed precision training, 2018.

[26] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu. Semantic image synthesis with spatially-adaptive normalization. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2332–2341, 2019.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[28] A. Razavi, A. van den Oord, and O. Vinyals. Generating diverse high-fidelity images with vq-vae-2, 2019.

[29] G. O. Roberts and R. L. Tweedie. Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341 – 363, 1996.

[30] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[31] M. Seitzer. pytorch-fid: FID Score for PyTorch. `https://github.com/mseitzer/pytorch-fid`, August 2020. Version 0.1.1.

[32] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017.

[33] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[34] Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[35] Y. Song and S. Ermon. Improved techniques for training score-based generative models. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems*, 2020.

[36] Y. Song, S. Garg, J. Shi, and S. Ermon. Sliced score matching: A scalable approach to density and score estimation. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence*, page 204, 2019.

[37] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

[38] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[39] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.

[40] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep image prior. *arXiv:1711.10925*, 2017.

[41] A. Vahdat, K. Kreis, and J. Kautz. Score-based generative modeling in latent space, 2021.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, 2017.

[43] P. Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011.

[44] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[45] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, page 681–688, 2011.

[46] P. Wollenhaupt. Molgrad: Moleküle generieren und optimieren mit ki. 2020.

[47] F. Yu, V. Koltun, and T. Funkhouser. Dilated residual networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[48] S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016.

[49] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5122–5130, 2017.

D. Deposition

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

I hereby assure that I have written this thesis independently and that I have not used any sources or aids other than those indicated.

Heidelberg, den 20. Juli 2021

Heidelberg, July 20, 2021