Master's Thesis

An XAI Interface for Diffusion Models

Examiners:
Prof. Dr. Wojciech Samek
Prof. Dr. Klaus-Robert Müller

Supervisor:
Luis Oala

Author:
Elora-Dana Maria Schörverth

A thesis submitted in fulfillment of the requirements
for the degree of

Master of Science

Institute of Computer Science - Faculty IV
Technical University Berlin

and

Department of Artificial Intelligence
Fraunhofer Heinrich-Hertz-Institute

July 19, 2023

Declaration

I, Elora-Dana Maria Schörverth, declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Hiermit erkläre ich, Elora-Dana Maria Schörverth, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, Date:

Signed:

Abstract

Diffusion Models [36, 97, 98] are powerful generative models, widely used for synthetic image generation. Despite their popularity, little research has been conducted to uncover their internal reasoning and representations during the generative process. Based on existing approaches from the field of Explainable Artificial Intelligence (XAI), this thesis introduces an XAI interface for diffusion models. The interface is based on a novel concept called diffusion profile: a unified primitive object combining four analysis axes to inspect the generative process. The interface enables the extraction, visualization, and interpretation of the diffusion model's properties. Using the interface in experiments, we analyze the emergence of the structure during the generative process and find a predetermination of the final sample.

Zusammenfassung

Diffusion Models [36, 97, 98] sind generative neuronale Netze, die unter anderem zur Erzeugung synthetischer Bilder verwendet werden. Trotz ihrer weitreichenden Verwendung ist nicht viel über ihre inneren Prozesse und Repräsentationen bekannt. Das Feld der Explainable Artificial Intelligence (XAI) entwickelt Methoden, um Entscheidungen von künstlicher Intelligenz nachvollziehbar zu machen. Basierend auf relevanten Methoden wird in dieser Arbeit ein XAI Interface für Diffusion Models vorgestellt. Das Interface gründet auf einem neuen Konzept namens Diffusion Profile, welches vier mögliche Analyse-Achsen des generativen Prozesses in einem Objekt zusammenfasst. Das Interface erlaubt es Nutzern, Eigenschaften des Diffusion Models zu extrahieren, visualisieren und interpretieren. Mit dem Interface wird durch Experimente dargestellt, wann sich interne Strukturen im Modell zeigen und inwiefern der generative Prozess vorherbestimmt ist.

Acknowledgements

Firstly, I would like to thank my examiners Prof. Dr. Wojciech Samek and Prof. Dr. Klaus-Robert Müller for their openness and support of my work on this topic.

Secondly, I would to thank my supervisor Luis Oala, for providing countless ideas, discussions, and words of encouragement, for this thesis and our other projects together.

I would also like to thank everyone at Fraunhofer Heinrich-Hertz Institute. Working there for over two years has been a tremendous experience and I highly value the opportunities I had to work in research. I would especially like to thank Gabriel for helping me with parts of this thesis.

Thank you to my friends Leo, Antonia, Lotti, and Konni for proofreading this thesis and making my time in Berlin as amazing as it was.

And last but not least, I would like to thank Louis. Despite being on opposite ends of the world, I feel your support at all times.

Dedication

To my grandparents and my aunt.
Thank you for always being by my side.

Acronyms

CelebA CelebFaces Attributes Dataset. 41, 42

CNN Convolutional Neural Network. 4, 21

DDPM Denoising Diffusion Probabilistic Models. 17

DGM Deep Generative Model. 6, 14

DNN Deep Neural Network. 4, 6, 18

FID Fréchet Inception Distance. 8, 9

GAN Generative Adversarial Model. 6, 8, 13–17, 21

INFOGAN Information Maximizing GAN. 16

IS Inception Score. 8, 9

KL Kullback-Leibler. 8, 9

LIME Local Interpretable Model-agnostic Explanations. 12

LRP Layer-wise Relevance Propagation. 12

MCMC Markov Chain Monte Carlo. 6

ML Machine Learning. 5, 10

MSE Mean-Square Error. 34

NLP Natural Language Processing. 15

PSNR Peak-Signal-to-Noise Ratio. 34, 71

SAGAN Self-Attention GAN. 15, 22

SHAP SHapley Additive exPlanations. 12

SSIM Structural Similarity Index Measure. 34–36, 39, 45, 50–53, 56–58, 71

VAE Variational Autoencoder. 6, 9

XAI Explainable Artificial Intelligence. 2, 3, 27, 58

Contents

Chapter 1

Introduction

In 2022 diffusion models took the world of synthetic image generation by storm. Inspired by annealed importance sampling [67] and Langevin dynamics [54], they were initially proposed in 2015 [97], with important contributions in 2019 [98] and 2020 [36]. The general public was made aware of their capabilities with the release of OpenAI's DALLE-2 mini[1] [76] in April 2022. The model is capable of generating images using arbitrary text input prompts (Figure 1.1).

Figure 1.1: Four examples from DALLE-2 mini for the input prompt "a capybara wearing a sunhat and drinking a pina colada".

DALLE-2 was swiftly followed by Google's Imagen [87] and Stable Diffusion [79]. Concerted research efforts are being poured into increasing the resolution of the outputs [38, 79, 86], improving sampling time [62], tools for image editing [6, 127], inpainting [61] or applying diffusion models in different domains, such as video generation [40].

The recent wave of generative models has not been without critique. Most of it is directed at using diffusion models to create digital art, raising copyright [110] and ethical [81] concerns. Diffusion models have also been found to be prone to memorization, enabling the extraction possible private information from the training data [13]. Yet the potential harm extends even further to fields such as medicine, where the trustworthiness of machine learning applications is central when making critical decisions. Diffusion models are already being used to reconstruct [99] or synthesize

[1] https://huggingface.co/spaces/dalle-mini/dalle-mini

[74] medical imaging for analysis, but there is comparatively little understanding of the internal reasoning and representations of diffusion models.

To make a step towards the interpretability of diffusion models, in this thesis we develop an interface capable of inspecting the generative process of diffusion models. We introduce simple explainability methods to approach this complex task and enable extraction and visualization of central attributes. We then use the interface for our experiments, seeking to understand the process of generating realistic images of human faces.

This thesis contains the following structure:

Chapter 2 introduces the foundational theories and examples of deep generative models, as well as common evaluation metrics. Furthermore, the field of Explainable Artificial Intelligence (XAI) and its applicability to deep generative models is introduced.

Chapter 3 covers the theory, architecture, and conditioning of diffusion models.

Chapter 4 details the conceptual design and implementation of the core contribution of this thesis: an XAI interface to track and visualize the generative diffusion process.

Chapter 5 demonstrates the applicability of the developed XAI interface in a series of experiments.

Chapter 6 concludes this thesis by giving a summary of the contributions, discusses current advances, and proposes further use cases of the interface.

Chapter 2

Related Work

In the following chapter, we present an overview of the related work for this thesis. This includes the foundational theory for generative models, including examples and evaluation methods, as well as an overview of the field of XAI. We then introduce a taxonomy of existing XAI approaches for generative models.

2.1 Deep Generative Models

This section introduces the fundamental theories relevant to deep neural networks and generative models. The main focus lies on deep generative models, which combine both aforementioned approaches, in order to generate synthetic data.

2.1.1 Deep Neural Networks

The fascination with artificially intelligent beings can be traced back as far as Greek mythology to the tale about Talos, a giant statue made out of bronze. Built by Hephaestus, the god of blacksmithing, he patrolled the island of Crete and threw boulders onto approaching ships.

One of the first real steps towards artificial intelligence, as we know it today, was made in 1943 when McCulloch and Pitts introduced the first mathematical model of a biological neuron, capable of processing binary signals [64]. The concept was extended by Rosenblatt in 1958 who introduced the perceptron - a single-layer neural network able to process any real-valued input [82]. The perceptron consists of n input neurons x, the weights w, and the activation function φ. Each input neuron x_i is multiplied by its respective weight w_i, and the weighted sum of all neurons is applied to the activation function φ. The activation function acts as a threshold, defining whether or not the neuron should be activated:

$$y = \varphi(\sum_{i=1}^{n} w_i x_i) \tag{2.1.1}$$

A key drawback of single-layer perceptrons is their restriction to linear functions [52]. This was resolved by introducing the multi-layer perceptron, a fully-connected

feedforward network that adds an intermediate, hidden layer between the input and output layers. The neurons in each layer l are connected with each neuron in the previous layer $l-1$, resulting in a non-linear mapping between input x and output y.

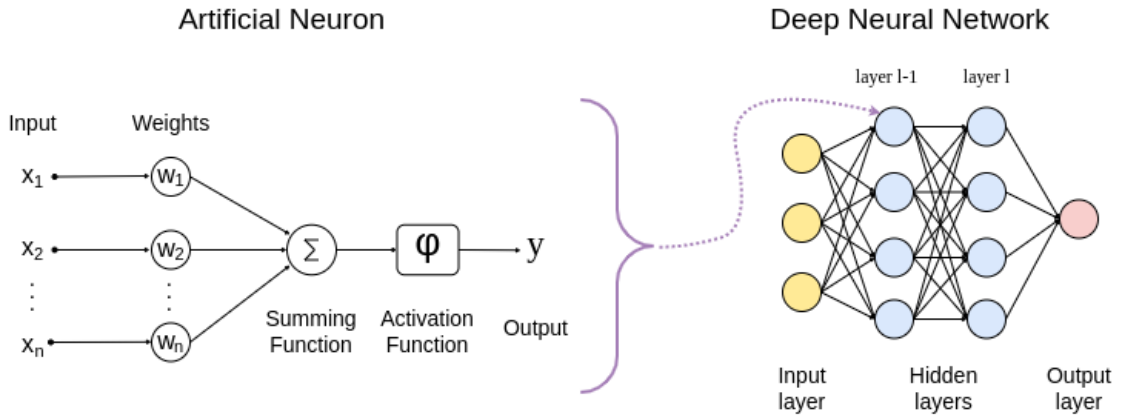


Figure 2.1: DNNs consist of a layer of input nodes, one or more hidden layers, and an output layer. Each node is an artificial neuron, generating activations based on their input and a non-linear activation function. The output layer then produces the final prediction y.

Deep Neural Network (DNN) refers to networks that contain at least two hidden layers. To train such networks, the efficient learning rule Backpropagation [56, 83] was proposed. It iteratively adjusts the weights w of the network to minimize the error function

$$E = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2 \tag{2.1.2}$$

which measures the mean-square error between actual output $\hat{y}_i$ and desired output y_i. E is minimized through gradient descent by computing the partial derivative wrt. each weight $w_{i,l}$ in the network. In other words, the error of the output unit y is backpropagated through the network to determine how each neuron contributes towards E.

A fundamental architecture for DNNs today is the Convolutional Neural Network (CNN) [31, 96, 103], first proposed by LeCun et al. in 1998 [55]. Convolutional layers shift a kernel filter across a larger input image thereby mimicking the receptive field of visual cells [42] and extracting visual features such as edges, corners and endpoints. Furthermore downsampling layers are incorporated, which reduce the input in spatial size [91].

With less expensive and faster computational power, as well as easier accessibility to large amounts of data, the last decade has seen tremendous breakthroughs in computer vision [31, 75, 80], natural language processing [15, 16, 23, 109], reinforcement learning [33, 65] and generative modeling [28, 36, 49, 79].

2.1.2 Generative Models

Generative Modeling is a learning task aiming to approximate an intractable probability distribution to generate new, synthetic samples. Given a finite number of independent and identically distributed (i.i.d) training samples from an intractable probability distribution $x \sim \mathcal{X}$, we seek to obtain a generator g, mapping samples from a tractable distribution $z \sim \mathcal{Z}$ such that $g(z) \approx x$. $\mathcal{Z}$ is commonly referred to as the latent space and assumed as a univariate Gaussian [85].

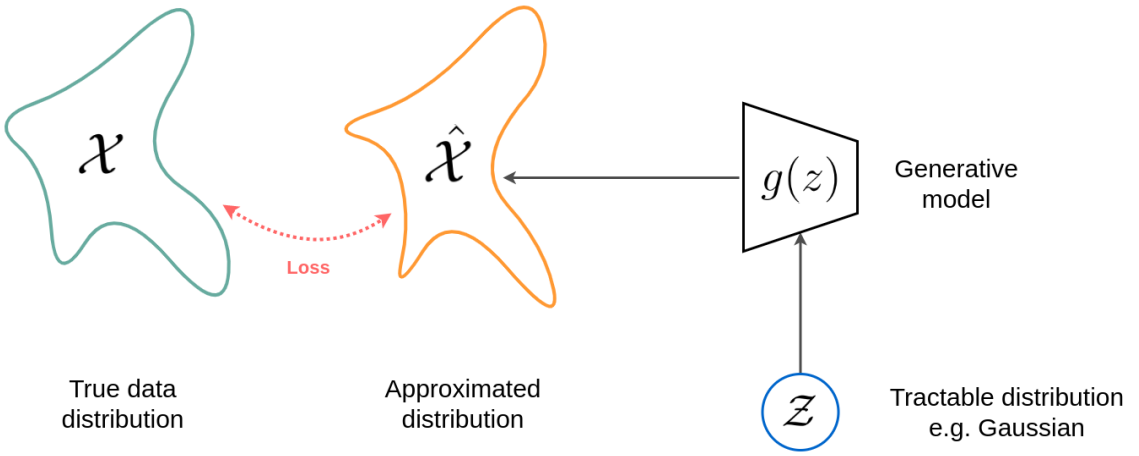


Figure 2.2: Conceptual design of a generative model. The generator g maps a data point $z \sim \mathcal{Z}$, sampled from a tractable distribution, to the true data distribution $\mathcal{X}$, such that such that $g(z) \approx x$. The training objective is generate an approximate data distribution $\hat{\mathcal{X}}$ which closely matches the true data distribution. The figure was adapted from [85] and [46].

Generative modeling can be applied to a large variety of problems, including traditional Machine Learning (ML) tasks such as text, image, or audio analysis, or further problems such as reinforcement learning, graph analysis, and medical imaging. Tomczak introduced a general taxonomy of generative models [107, p.5-7], distinguishing four different types:

Autoregressive models are commonly used to predict the next time step in a series by applying explicit density estimation on all previous steps. They represent the distribution over x using autoregressive modeling

$$p(x) = p(x_0) \prod_{i=1}^{D} p(x_i | x_{<i}). \tag{2.1.3}$$

As modeling all conditional distributions $p(x_i|x_{<i})$ would be computationally inefficient, ARMS make use of causal convolutions [72]. A popular example is the PixelCNN [71], a convolutional variant of the PixelRNN [70], designed for conditional image generation. The autoregreassive connections are used to model the image in a pixel-wise fashion and decomposing the joint image distribution as a product of

conditionals.

Flow-based models are constructed as an invertible transformation mapping between the observed data x and a latent variable $z = f(x)$. The model f is formed by stacking simple, invertible transformations and composing a series of invertible flows as $f(x) = f_1 \circ ... \circ f_L(x)$. Each f_i has a tractable inverse and a tractable Jacobian determinant, enabling efficient sampling [39].

$$p(x) = p(z = f(x))|J_{f(x)}| \tag{2.1.4}$$

This way of expressing the density of a random variable through invertible transformation is called the *change of variables formula*.

Energy-based models are inspired by physics and learn an energy function $E(x)$, assigning low energy values to inputs from the data distribution and high energy values to other inputs. In particular, the Boltzmann distribution

$$p(x) = \frac{exp{-E(x)}}{Z} \tag{2.1.5}$$

is used to define the probability distribution, where $Z = \sum_x exp{-E(x)}$ is the partition function to be approximated. The sampling procedure is implicit through Markov Chain Monte Carlo (MCMC) sampling $x \sim e^{-E(x)}$ [25].

Latent Variable Models model the probability distribution $p(x)$ through latent variables assuming a lower-dimensional latent space for the data points. The generative process

$$\begin{aligned} z &\sim p(z) \\ x &\sim p(x|z) \end{aligned} \tag{2.1.6}$$

first samples a latent factor from the probability distribution $p(z)$ and then samples a data point from the conditional distribution $p(x|z)$, which can be treated as a generator. Latent variable models will be the focus of this thesis.

2.1.3 Deep Generative Models

Deriving a generative model is a challenging task. With advances in deep learning, it has become common practice to design the generator g_θ as a DNN, trained to learn parameters θ such that new synthetic samples are statistically indistinguishably from the training data $g_\theta(z) \approx x$. This is called a Deep Generative Model (DGM). To establish the correspondence between a latent z and a true datapoint x we can either use statistical inference or invert a generator [85]. Two popular latent variable architectures are the Variational Autoencoder (VAE) and the Generative Adversarial Model (GAN).

Variational Autoencoders [49, 50] are a popular architecture for deep latent-variable models $p_\theta(x, z)$. They introduce a parametric inference model $q_\phi(z|x)$, also called an encoder, with ϕ being the variational parameters. These parameters are then optimized such that the inference model approximates the true but intractable posterior of the generative model

$$q_\phi(z|x) \approx p_\theta(z|x) \tag{2.1.7}$$

Similarly to a traditional autoencoder, q_ϕ maps from the data space $\mathcal{X}$ to the latent space $\mathcal{Z}$, but rather than mapping an individual point, the mapping is a probability distribution [85].

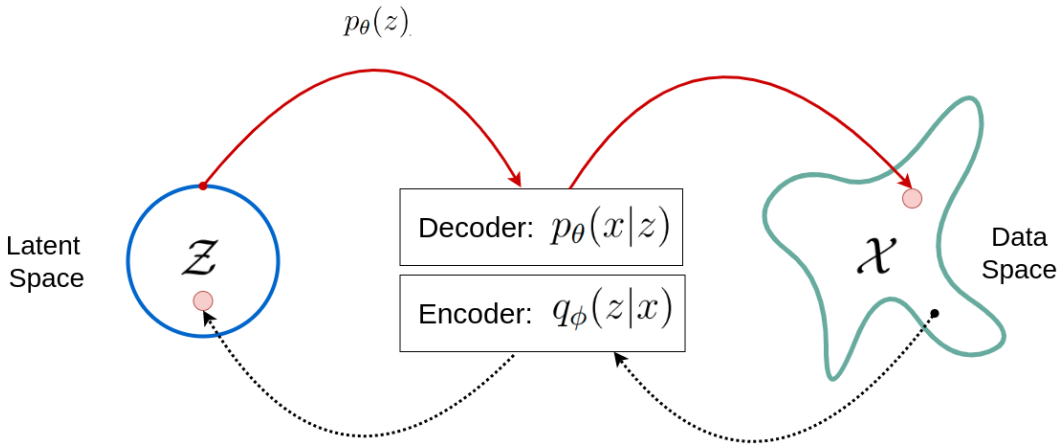


Figure 2.3: Figure adapted from [50, 116]. The solid red line shows the generative process of a probabilistic decoder $p_\theta(x, z) = p_\theta(z)p_\theta(x|z)$. The intractable posterior $p_\theta(z|x)$ of the generative model is approximated by the stochastic encoder $q_\phi(z|x)$.

Generative Adversarial Networks [28] are a type of implicit latent variable model in which a generator $G_{\theta_g}(z)$ is presented with an adversary, the discriminator $D_{\theta_d}(x)$. Both components are part of a two-player mini-max game with the value function

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \in p_{data}(x)}[\log D_{\theta_d}(x)] + \mathbb{E}_{z \in p_z(z)}[\log(1 - D_{\theta_d}(G\theta_g(z)))]. \tag{2.1.8}$$

The discriminator estimates the probability of a sample being real $x \sim p_d(x)$ or generated $x \sim p_{\theta_g}(x)$. This ongoing competition between both components forces the generator to approximate the real data distribution as closely as possible.

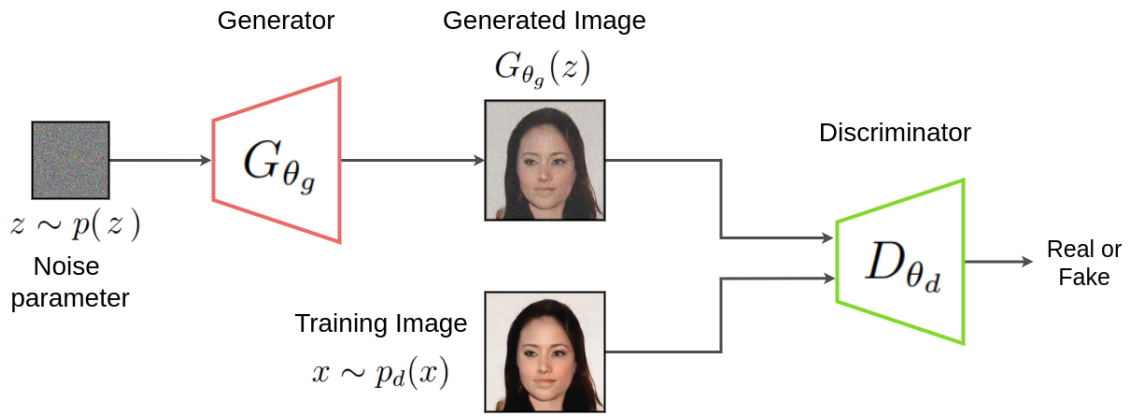


Figure 2.4: Figure adapted from [120]. The generator G_{θ_g} produces samples given an input noise parameter $z \sim p(z)$, while the discriminator D_{θ_d} learns to estimate the probability of a given sample being from the real data distribution or the generator. $D(x)$ represents the probability that the sample came from the real training data $x \sim p_d(x)$ rather than the generator's distribution $x \sim p_{\theta_g}(x)$.

GANs are considered likelihood-free models and do not try to infer latent variables. One key challenge with GANs is that training both the weights of the generator and discriminator at the same time often results in a difficult-to-solve saddle point [85].

2.1.4 Evaluation

Alongside the theoretical background and examples given in the previous section, it is important to discuss how to evaluate generative models, specifically for image generation. The three most common approaches are the Kullback-Leibler (KL) divergence, used during optimization, and the Inception Score (IS), as well as the Fréchet Inception Distance (FID), which are used to evaluate generated samples

Kullback-Leibler Divergence [53] measures the difference between two probability distributions

$$DKL(P||Q) = \sum_x P(x) \log(\frac{P(x)}{Q(x)}). \tag{2.1.9}$$

KL divergence is commonly used because minimizing the empirical KL divergence is equivalent to standard maximum likelihood optimization.

Inception Score [89] utilizes the inception model [104] to evaluate the quality of generated images. The model is pre-trained and generates a conditional label distribution $p(y|x)$ for each image. Two assumptions are made about generated images: meaningful objects in an image result in a conditional label distribution with low entropy and a large variety in an image can be measured by the entropy of the

marginal $\int p(y|x = G(z))dz$ [12]. Combined these two conditions give us the metric

$$exp(\mathbb{E}_x KL(p(y|x)||p(y))). \tag{2.1.10}$$

Fréchet Inception Distance [34] assumes that both real and generated images from a pre-trained Inception network have a Gaussian distribution. With this assumption, we can apply the Fréchet distance and measure the distance between two multivariate Gaussians. The Fréchet distance $d(.,.)$ between the Gaussian with mean (m, C) obtained from $p(x)$ and the Gaussian with mean (m_w, C_w) obtained from $p_\theta(x)$ is given as

$$d^2((m, C), (m_\theta, C_\theta)) = ||m - m_w||_2^2 + Tr(C + C_w - 2(CC_w)^{1/2}). \tag{2.1.11}$$

The FID is consistent with increasing disturbances in human judgement, such as noise, blur, covered image areas or swirls.

Even though it is common practice to evaluate generative models based on the introduced metrics, it is not without critique. The lack of interpretability of the metrics was noted in a study [12] which compared KL divergence, IS, and FID on a synthetic dataset. Researchers noticed that while the metrics show a good correlation, there are issues with volatility, especially with the IS for different models. Furthermore, the use of the Inception network needs to be questioned. The Inception model was trained on ImageNet [84] which prominently features animals and plants and employing it to evaluate generated samples from other domains could pose issues.

Theis et al. [106] also noted that good performance of a generative model in one domain does not necessarily translate to other domains. The authors caution against the assumption that different evaluation measures produce similar results when assessing the same model. Instead, there are trade-offs between different measures, which need to be carefully chosen based on the model's application. Assessing a model based on the log-likelihood of samples can be biased towards overfitting models and high log-likelihood, therefore, does not necessarily result in visually convincing samples.

Furthermore, Nalisnick et al. [66] challenged the notion that generative models are robust against outputting wrong samples with high confidence. They found that VAEs were unable to differentiate images of animals from house numbers and it is not sufficient to compare the likelihoods of generative models to identify input from or similar to the training set. Likelihood attributions produced by a generative model may not align with human expectations of what constitutes an out-of-distribution sample.

2.2 Explainable Artificial Intelligence

With the rise of machine learning applications, especially in high-risk domains such as medicine [27] or socially impactful ones like human resources [51], the need to understand the highly complex ML systems has become more prevalent. XAI as a research field seeks to "produce more explainable models while maintaining a high level of performance, and enable human users to understand, trust and effectively manage the emerging generation of artificially intelligent partners" [29]. Alongside XAI taxonomies from previous surveys [3, 5, 92], we divide explainability methods along three axes: transparency, scope, and specificity.

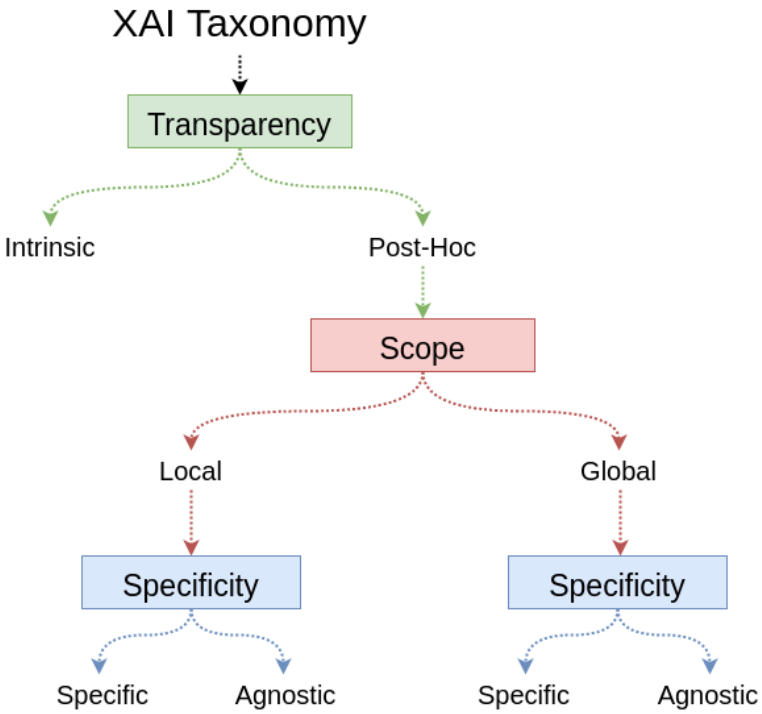


Figure 2.5: Figure adapted from [5]. XAI methods can be distinguished along three axes: transparency, scope, and specificity. Transparency refers to whether a model is intrinsically interpretable or if it is a complex model, also referred to as a black-box model. Complex models can be made interpretable through post-hoc explainability methods [3]. These methods can either work on a global or a local level. Global XAI methods seek to explain the model's entire reasoning, while local methods seek to understand specific examplesThe last axes along which we can differentiate XAI methods is specificity. Specific methods are limited to specific model classes, while agnostic methods are not [3].

2.2.1 Transparent vs. Post-hoc

Transparency in XAI determines whether a model class is intrinsically interpretable or only through the application of post-hoc methods. Transparent models include linear/logistic regression, K-nearest neighbor, and decision trees, as their reasoning is easily understood due to their simplicity. A clear drawback of this simplicity is the trade-off between interpretability and accuracy [3]. "Accuracy generally requires more complex prediction methods and simple and interpretable functions do not make the most accurate predictors" [14].

Models for high-performance tasks such as image classification [123, 125] or natural language processing have up to 175 billion [15] parameters. To interpret such architectures, post-hoc methods are applied after the model has been trained. Post-hoc methods reverse engineer the model's reasoning to trace it back to interpretable decisions without changing the model itself [3]. They are designed to provide easily interpretable explanations and are often provided in the form of visual explanations [5]. Post-hoc methods can be designed to be generally applicable - model-agnostic - or tailored to a specific model class - model-specific (Section 2.2.3).

2.2.2 Global vs. Local

Global interpretability methods seek to understand the model's entire reasoning and internal representations as a whole, which can be hard for models with a large number of parameters [3]. The focus is on analyzing how a decision has been made. Yang et al. [122] proposed using a binary interpretation tree, explicitly representing decision rules followed by a complex model. Other works seek to identify human-understandable concepts within the model's internal state [2, 11, 18, 26, 48].

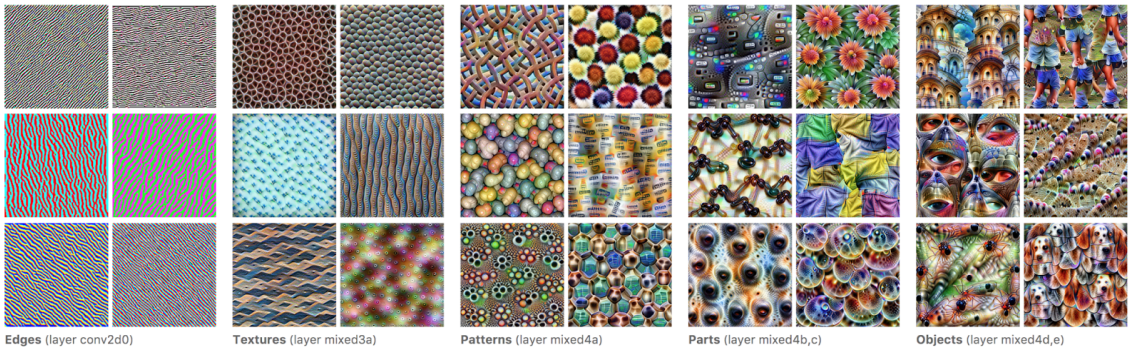


Figure 2.6: Figure and example by Olah et al. [69]. Feature Visualization generates inputs maximizing a units response and thereby visualizing its learned concepts. The example displays the layer-wise concepts such as edges, textures and patterns of GoogleLeNet [102], trained on ImageNet [84].

Feature Visualization [68, 69, 124] aims at identifying prototypical inputs that maximize the activation of individual units within the model. Units can refer to individual neurons, channels, or entire layers of the network. The maximizing inputs

can either be found through optimization or by generating synthetic samples using generative models. By analyzing which types of images activate a unit the most, we can visualize the unit's learned concepts.

Local interpretability, on the other hand, refers to explaining a model's decision for individual or groups of inputs and seeks to understand why a decision was made, rather than how [92]. Attribution heatmaps (Figure 2.7) are often used to visualize how features in the input space impact the model's decision. They can be generated in a model-agnostic fashion through perturbation, see Section 2.2.3, or model-specific through sensitivity analysis [9] or backpropagation [94, 95, 100, 101].

Layer-wise Relevance Propagation (LRP) [8] redistributes a classifier's prediction backward through the network using local redistribution rules assigning a relevance score to each input image pixel. A core property of LRP is the conservation of the relevance, i.e. the received relevance of a neuron has to be equally redistributed to the lower layer.

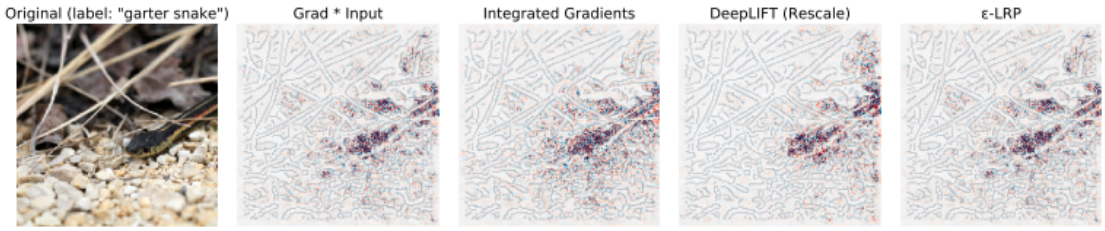


Figure 2.7: Figure by Anonca et al. [4]. Exemplary attribution heatmaps for the classification of an image of a garden snake. The backpropagation-based attribution methods used are *Gradient * Input* [95], *Integrated Gradients* [101], *DeepLift* [94] and ϵ-*LRP* [8].

2.2.3 Specific vs. Agnostic

Model-specific methods are designed for specific model classes, as they often rely on accessing the model's inner workings such as learned weights. Examples are the backpropagation-based attribution maps introduced in the previous subsection.

In contrast, model-agnostic methods apply to any model and only need access to the model's input and output [92]. Two prominent model-agnostic perturbation methods are Local Interpretable Model-agnostic Explanations (LIME) [77] and SHapley Additive exPlanations (SHAP) [63]. LIME learns a simpler surrogate model around a single prediction which is trained by perturbing the input and observing the impact the perturbation has on the prediction.

Counterfactual explanations [112], also called adversarial examples, test the model's prediction boundaries by changing the input features up until the point that the prediction changes. Generative models can be used to obtain counterfactual inputs that still resemble the original input while closely controlling which attributes change [57].

2.3 Explainable Deep Generative Models

The XAI methods introduced in the previous section were largely designed for discriminative models such as image classifiers. The use-case for this thesis, however, are generative models and the high-dimensional output poses an obstacle to explainability methods. The following section covers existing XAI methods for generative models, which largely propose to either modify or extract internal representations to circumvent the problem of dimensionality. This section introduces a taxonomy of existing approaches: ablation methods, attention-extraction, and latent space discovery.

2.3.1 Unit Ablation

Unit ablation methods seek to identify and match semantic structures encoded in the generative model to a specific unit in the model. One of the first frameworks using this approach was GAN Dissect [10], proposed by Bau et al. in 2018. Given a generator $g : z \rightarrow x$, with the latent vector $z \in \mathbb{R}^{|z|}$ and image $x \in \mathbb{R}^{H \times W \times 3}$ the goal is to analyze the internal semantic representations of g. To understand how the representations r of g encode the information to produce a concept class c, r is decomposed into two components

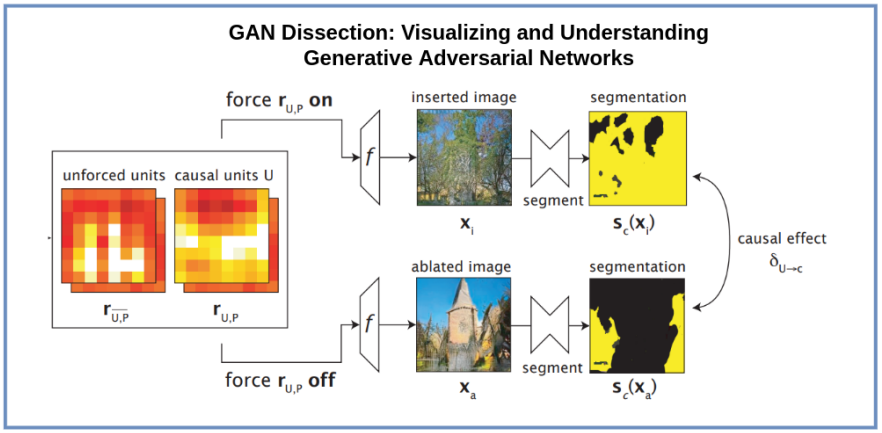
$$r_{\mathbb{U},P} = (r_{U,P}, r_{\overline{U},P}). \tag{2.3.1}$$

$r_{U,P}$ is the subset of units that correspond to the concept class c and $r_{\overline{U},P}$ the units which do not. The units are matched to c by measuring the spatial agreement between the unit's feature map and the concept class segmentation map. A causal relationship can then be established by ablating $r_{U,P}$ and observing the change in the output image x.

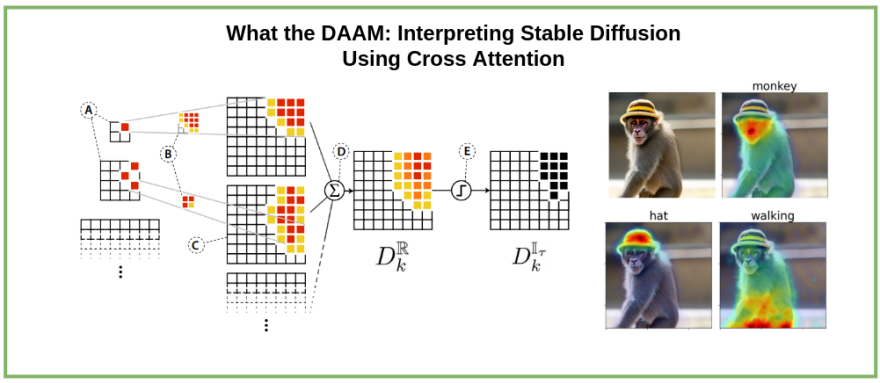
Results show that specific units can indeed be matched to concepts classes. Artifacts can be efficiently removed from samples, allowing for efficient editing of images. However certain concepts are more easily removed from images, whereas others cannot be removed at all. As an example, it was not possible to remove the concept of chairs from an image of a conference room. A hypothesis for this phenomenon is that some concepts are too closely associated in the generators representations to be disentangled.

Another ablation study by Yang et al. [121] aimed at quantifying the relationship between layer-wise activations and semantic objects in the output image. They were able to identify that a layer-wise semantic hierarchy exists within a GAN: "early layers tend to determine spatial layout, middle layers control the categorical objects and later layers details such as color scheme". They also implemented ablation in their experiments on a layer-wise level. By manipulating the input latent code only at layers relevant to attributes, they could modify the semantic output without affecting other attributes.

Unit Ablation



Attention Extraction



Latent Space Discovery

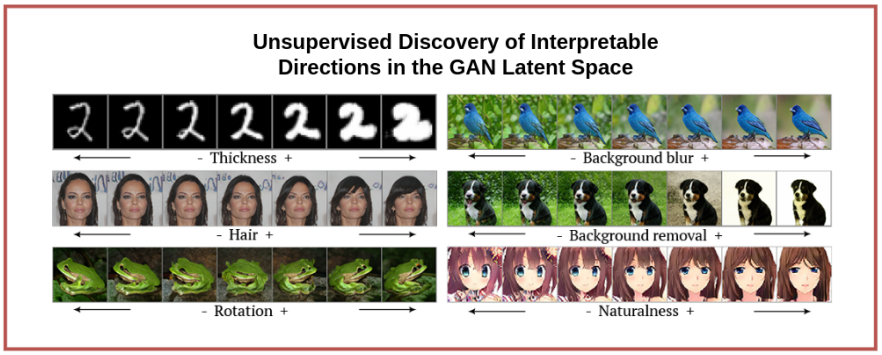


Figure 2.8: Examples of explainability methods for DGM. (Top) Identification and ablation of concept related units in DGMs. Figure by Bau et al. [10]. (Center) Text-to-image attribution maps by extracting cross-attention for diffusion models. Figure by Tang et al. [105], licensed under CC BY 4.0. (Bottom) Interpretable directions in the latent space of GANs. Figure by Voynov and Babenko [111].

2.3.2 Attention Extraction

Attention was popularized in the field of Natural Language Processing (NLP) with the introduction of the transformer architecture [109]. In NLP attention is used to encode long-term dependencies between text sequences and guides the model's attention toward specific input words. In computer vision and generative modeling, attention mechanisms are used to divert attention towards important image regions by weighting features in the input [30]. Self-Attention GAN (SAGAN) [126] is an attention-driven GAN with attention layers complementary to convolutional layers. The attention in SAGAN enables dependencies across multi-level image regions for long ranges. The two methods introduced in this section extract the attention layers from a generative network to visualize how they encode semantic information within the network.

Tang et al. [105] proposed a novel approach for text-to-image attribution analysis of latent diffusion models [79]. The latent diffusion model employs a cross-attention mechanism that pre-processes the input text prompt y by projecting it into an intermediate representation $\tau_\theta(y)$ using an encoder τ_θ. The representation is mapped across the intermediate layers of the neural network using attention defined as

$$\text{Attention}(Q, K, V) = softmax\Big(\frac{QK^T}{\sqrt{d}}\Big) \cdot V \tag{2.3.2}$$

with Q, K, V being

$$\begin{aligned} Q &= W_q^{(i)} \cdot \varphi_i(z_t) \\ K &= W_K^{(i)} \cdot \tau_\theta(y) \\ V &= W_V^{(i)} \cdot \tau_\theta(y). \end{aligned}$$

To extract and visualize the attention attribution map, all attention layers are upsampled to be of the same spatial size. Furthermore, a diffusion process consists of 1000 passes through a network, so the attention layers are additionally aggregated along the temporal axis. Ultimately, both aggregations produce a two-dimensional attribution map across the synthesized image for each word in the input prompt (Figure 2.8).

Liu et al. [58] combined gradient-based and attention-based approaches to generate attribution maps for VAEs. In the proposed method, a latent vector z is obtained from the posterior distribution $q(z|x)$. For each element z_i, the gradients are back-propagated to the last convolutional feature maps $A \in \mathbb{R}^{n \times h \times w}$. This results in an attention map

$$M^i = ReLU(\sum_{k=1}^{n} a_k A_k) \tag{2.3.3}$$

for each z^i, with scalar $\alpha_k = GlobalAveragePooling(\frac{\partial z_i}{\partial A_k})$ and feature map A_k of the k^{th} channel. This is repeated for all elements $z_1, z_2, ..z_D$ in the latent space vector, resulting in $M^1, ..., M^D$ attention maps which are aggregated as $M = \frac{1}{D}\sum_i^D M^i$. The attention map M can be used for anomaly detection and latent space disentanglement.

2.3.3 Latent Space Discovery

A central question for generative models is to understand how the sampled latent $z \sim \mathcal{Z}$ relates to the semantic attributes in the output image and if it is possible to match latents to specific image attributes.

Information Maximizing GAN (INFOGAN) [17] is an earlier method from 2016 that employs an information-theoretic way of disentangling representations in an entirely unsupervised manner. The noise latent z is decomposed into two parts: (i) a source of incompressible noise and (ii) a latent code, targeting salient structured semantic features of the data distribution. The generator G_{θ_g} then receives the incompressible noise z and the latent code c as an input. To avoid a trivial solution, the mutual information between latent code c and generator distribution $G(z, c)$ should be high. INFOGAN can successfully disentangle writing styles from MNIST [22] digits and visual concepts such as hairstyles, presence of eyeglasses, and emotions from the CelebA [60] dataset.

The β-VAE [35] is a deep unsupervised generative model for disentangled factor learning. The framework is an extension of the original VAE (Section 2.1.3) which introduces a hyperparameter β and reformulates the model into a constrained optimization problem. The constraints strongly limit the capacity of the latent space and push the model to learn statistically independent latent factors. Even without prior knowledge of the data, this allows for learning disentangled representations of data generative factors.

A model-agnostic and unsupervised method for latent space discovery was proposed by Voynov and Babenko [111] in 2020. The method aims at learning latent space directions inducing image transformations that are easily distinguishable from each other. They jointly learn a set of directions and train a separate model to distinguish the corresponding image transformations. A matrix $A \in \mathbb{R}^{d \times K}$ is used to track the latent space directions, with d being the latent space dimensionality and K being the number of directions to discover. The reconstructor R reproduces the shift in latent space, obtaining the image pair $G(z), G(z + A(\epsilon e_k))$. $G(z)$ is a generated image from a GAN, whereas the $G(z + A(\epsilon e_k))$ is an image generated with the shifted latent. The method can identify key directions corresponding to rotation, background blur, and background removal (Figure 2.8).

Chapter 3

Diffusion Models

Diffusion models were first proposed in 2015 by Sohl-Dickinstein [97]. They are inspired by annealed importance sampling [67] and Langevin dynamics [54], a method from nonequilibrium thermodynamics to statistically model molecular systems. The core idea is to apply Langevin dynamics in a generative Markov chain, which slowly converts a known distribution, such as a Gaussian, into an unknown target distribution. Langevin dynamics sample from a probability density $p(x)$ using only the score function $\nabla_x \log p(x)$ to recursively compute

$$\tilde{x}_t = \tilde{x}_{t-1} + \frac{\delta}{2}\nabla_x \log p(\tilde{x}_{t-1}) + \sqrt{\delta}\epsilon_t \tag{3.0.1}$$

with step size δ, noise $\epsilon_t \sim \mathcal{N}(0, I)$ and $\tilde{x}_0 \sim \pi$ sampled from a prior distribution π. The distribution of $\tilde{x}_T$ becomes an exact sample from $p(x)$, as $\delta \to 0$ and $T \to \infty$ [98, 115].

Diffusion models are currently the state-of-the-art model for generating synthetic imaging, having outperformed GANs [24]. This breakthrough was initiated by Ho et al. [36] in 2020, who trained diffusion models on a weighted variational bound, an approach also called Denoising Diffusion Probabilistic Models (DDPM). This concept is based on a connection between diffusion probabilistic models [97] and denoising score matching [98].

3.1 Theoretical Background

A diffusion model is a parametrized Markov chain consisting of a forward and a backward process. The forward process gradually adds Gaussian noise to a training image, thereby perturbing the image until its information is destroyed. The reverse process learns the Gaussian transitions starting from the final step to recover the original image in T time steps.

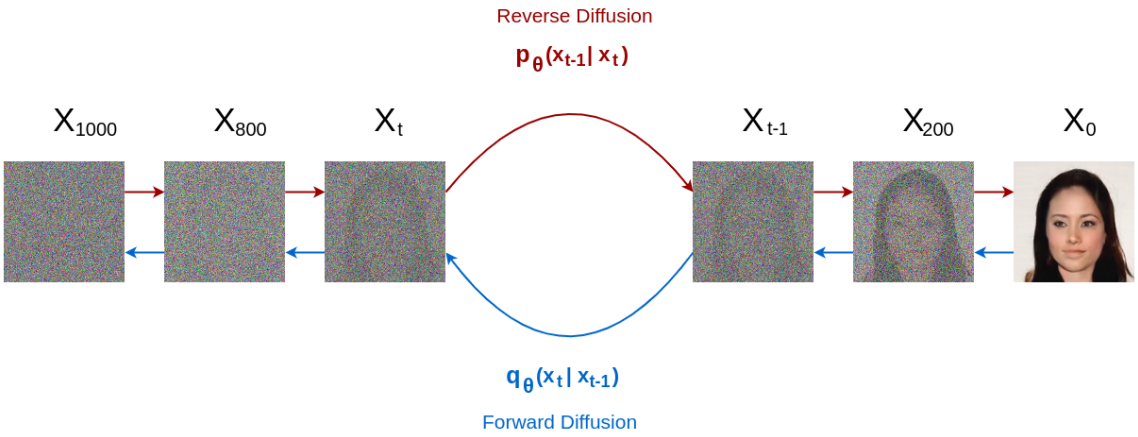


Figure 3.1: A diffusion model consists of two processes. The forward process (blue) is a parametrized Markov chain perturbing the training image x_0 by adding a controlled amount of Gaussian noise $\epsilon \sim \mathcal{N}(0, I)$ over T timesteps. The reverse process (red) recovers the original image from x_T by undoing the corresponding perturbations in the forward process.

The forward process $q(x_{1:T}|x_0)$ of a diffusion model starts with a sample $x_0 \sim q(x)$ from a real data distribution. A fixed Markov chain of perturbation steps gradually adds noise $\epsilon \sim \mathcal{N}(0, I)$ to the data, according to a controlled variance schedule $\beta_1, ..., \beta_T$. The forward process produces a sequence of noisy samples $x_1, ..., x_T$, where the time step t determines the mixture of the original image x_0 and the noise ϵ [24, 36]. The final step x_T is then equivalent to an isotropic Gaussian [117].

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}) \qquad q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{(1-\beta_t)}x_{t-1}, \beta_t I) \tag{3.1.1}$$

An important property of the forward process is that it allows sampling x_t at an arbitrary step t in closed form [36]. Using reparametrization we define $a_t := 1 - \beta_t$ and $\overline{a}_t := \prod_{s=1}^{t} a_s$ such that

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\overline{a}_t}x_0, (1-\overline{a}_t)I). \tag{3.1.2}$$

The reverse process $p_\theta(x_{0:T})$ is a Markov chain of learned Gaussian transitions that undo the corresponding perturbation in the forward process. It starts at $p(x_T) = \mathcal{N}(x_T; 0, I)$ and iteratively samples the conditional probabilities $q(x_{t-1}|x_t)$. As it is generally intractable to sample the conditional probabilities directly, we train a parametrized DNN p_θ to approximate them instead. The variance $\Sigma_\theta(x_t, t)$ is set to time dependent constants $\sigma_t^2 I$, with $\sigma_t^2 = \beta_t$ [36, 117]

$$p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \quad p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)). \tag{3.1.3}$$

3.2 Training

Training a diffusion model is performed by optimizing the variational bound on the negative log likelihood

$$\begin{aligned} \mathbb{E}[-\log p_\theta(x_0)] &\leq \mathbb{E}_q\Big[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\Big] \\ = \mathbb{E}_q\Big[-\log p(x_T) - \sum_{t\geq 1} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}\Big] &=: L. \end{aligned} \tag{3.2.1}$$

However, multiple modifications have been proposed to simplify the training objective. The two major contributions are variance reduction and a re-parametrization of the mean [20].

3.2.1 Variance Reduction

Sohl-Dickstein et al. [97] showed that by conditioning all forward process posteriors on x_0

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I) \tag{3.2.2}$$

it is tractable to re-write Equation 3.2.1 as the KL-divergence between $p_\theta(x_{t-1}|x_t)$ and the forward posteriors

$$\mathrm{E}_q\left[\underbrace{D_{KL}(q(x_T|x_0)||p(x_T))}_{L_T} + \textstyle\sum_{t>1}\underbrace{D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))}_{L_{t-1}} - \underbrace{\log p_\theta(x_0|x_1)}_{L_0}\right]. \tag{3.2.3}$$

Therefore, all KL divergences of term L_{t-1} in Equation 3.2.3 are just comparisons between two Gaussians, which can be computed in closed form, as opposed to high variance Monte Carlo estimates [36].

3.2.2 Re-parametrization of the Mean

Based on the analysis of L_t in Equation 3.2.3, Ho et al. [36] proposed to parametrize the mean $\mu_\theta(x_t, t)$ as

$$L_{t-1} = \mathbb{E}_q\Big[\frac{1}{2\sigma_t^2}||\tilde{\mu}(x_t, x_0) - \mu_\theta(x_t, t)||^2\Big] + C \tag{3.2.4}$$

The most straightforward parametrization of the mean is a neural network which predicts the forward process posterior mean according to

$$\tilde{\mu}(x_t, x_0) := \frac{\sqrt{\overline{\alpha}_{t-1}}\beta}{1-\overline{\alpha}} x_0 + \frac{\sqrt{\alpha_t}(1-\overline{\alpha}_{t-1})}{1-\overline{\alpha}_t} x_t. \tag{3.2.5}$$

However, using the property from Equation 3.1.2, we can write $x_t(x_0, \epsilon) = \sqrt{\overline{\alpha}_t}x_0 + \sqrt{1-\overline{\alpha}_t}\epsilon$ with $\epsilon \sim \mathcal{N}(0, I)$. Applying this to Equation 3.2.5 allows us to reparametrize the network s.t. ϵ_θ becomes a function approximator predicting the noise level ϵ of x_t at timestep t

$$\mu_\theta(x_t, t) = \tilde{\mu}_t\Big(x_t, \frac{1}{\sqrt{\overline{\alpha}_t}}(x_t - \sqrt{1-\overline{\alpha}_t}\epsilon_\theta(x_t))\Big) \tag{3.2.6}$$

This reparametrization has the major advantage that the complete sampling procedure resembles Langevin dynamics. Sampling $x_{t-1} \sim p_\theta(x_{t-1}|x_t)$ is computing

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\Big(x_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}}\epsilon_\theta(x_t, t)\Big) + \sigma_t z, \text{where } z \sim \mathcal{N}(0, I). \tag{3.2.7}$$

3.2.3 Simpler Training Objective

Ho et al. [36] also found the following simplification of the loss function to be most advantageous

$$L_{simple}(\theta) := \mathbb{E}_{t,x_0,\epsilon}\Big[||\epsilon - \epsilon_\theta(\sqrt{\overline{\alpha}_t}x_0 + \sqrt{1-\overline{\alpha}_t}\epsilon, t)||^2\Big] \tag{3.2.8}$$

where t is uniform between 1 and T.

The following are the proposed algorithms for training and sampling, which are used in the implementation of this thesis.

Algorithm 1 Diffusion Training Algorithm [36]

repeat
$x_0 \sim q(x_0)$
$t \sim \text{Uniform}(\{1, ..., T\})$
$\epsilon \sim \mathcal{N}(0, I)$
Take gradient step on: $\nabla_\theta||\epsilon - \epsilon_\theta(\sqrt{\overline{a}_t}x_0 + \sqrt{1-\overline{a}_t}\epsilon, t)||^2$
until converged

The training algorithm corresponds to the forward process in Figure 3.1. We first take an image x_0 from a training dataset. We uniformly sample a time step t and estimate the noisy image at this time step. We then optimize the loss between the actual noise level ϵ and the estimated noise level predicted by our model ϵ_θ.

Algorithm 2 Diffusion Sampling Algorithm [36]

$x_T \sim \mathcal{N}(0, I)$
for $t = T, ..., 1$ **do**
  $z \sim \mathcal{N}(0, I)$ if $t > 1$, else $z = 0$
  $x_{t-1} = \frac{1}{\sqrt{a_t}}(x_t - \frac{1-a_t}{\sqrt{1-\bar{a}_t}}\epsilon_\theta(x_t, t)) + \sigma_t z$
end for
return x_0

The sampling algorithm corresponds to the reverse process in Figure 3.1. To generate data from a trained diffusion model, we first sample x_T as isotropic Gaussian noise of the same shape as the desired sample. For each time step of the diffusion chain starting at T, we then sample the noise estimate from our model $\epsilon_\theta(x, t)$. This noisy term is plugged into the sampling procedure (Equation 3.2.7) to calculate the sample at the following time step. This procedure is repeated until we reach x_0.

A clear drawback of diffusion models might be obvious: this sampling procedure requires all T steps to be executed for just one sample. Therefore, sampling diffusion models is a lot slower in comparison to other architectures such as GANs [24].

3.3 Architecture

The practical implementation of a diffusion model requires the network to be able to intake a noisy sample x_t and output a slightly de-noised version x_{t-1}. As input and output are to be of identical size, a U-Net is chosen to be the network architecture. The U-Net was first introduced by Ronneberger et al. [80] and consists of a contracting and an expanding path. The contracting path is identical to a typical CNN and is made up of the repeated application of downsampling convolutions and residual layers [31]. The expansive part follows with upsampling convolutions and residual layers, upscaling the compressed sample until it is of the same dimension as the original input again. Additionally, the output of the up-convolution is concatenated with the cropped feature map of the corresponding resolution in the contracting path, which is called a *skip connection*. The output of the U-Net is a segmentation map, assigning each pixel to the desired number of classes. In diffusion models, instead of a class segmentation map, the output x_{t-1} is the slightly de-noise sample (Figure 3.3).

In the base implementation of this thesis [78], further additions were made to the core U-Net architecture. Each resolution level consists of two ConvNeXT [59] modules, one attention residual module and either a down- or upsampling convolutional layer. Additionally, each ConvNeXt module receives sinusoidal position embeddings as an input parameter, encoding the current time step t of the diffusion process. Figure 3.2 depicts the exemplary succession of modules and operations performed at one resolutional level of the contracting path. This is relevant for Chapters 4 and 5, where we analyze changes within the modules over the diffusion process.

The following list provides more detail on each of the additions made to the adapted U-Net architecture:

- **Sinusoidal position embeddings** encode the current time step t to inform the network about the noise level it should assume it is working on. The idea originated from the positional encodings of the transformer architecture [109] where the position of a token is injected into the model when forming sentences.

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{3.3.1}$$

  Each positional encoding corresponds to a sinusoid with pos being the position and i the dimension. The embeddings are added to each residual module [78].

- **ConvNeXt module** [59] is an improved version of the standard ConvNet module towards the design of a vision transformer, while retaining the simplicity of a standard ConvNet. Starting with a ResNet-50 [31] model, the researchers studied a series of improved design choices. Among other changes, they inverted the dimensions of the bottleneck, increased convolutional kernel size and grouped convolutions, an idea adopted from ResNeXt [119]. On a micro level of layer design, they replaced ReLU with GeLU [32] and only used a single activation function in each layer. They also reduced the number of normalization layers, replaced Batch Normalization with Layer Normalization [7] and separated the downsampling layers. In this implementation GroupNorm is used instead of LayerNorm.

- **Attention Residual** includes a linear attention [93] module which is a more time and memory-efficient implementation of self-attention [19, 73, 109]. Using attention layers for generative models was first proposed for SAGAN [126]. Attention layers are used to highlight the spatial importance of image regions and are essential for high-frequency and semantic details of the sampled image [41].

- **GroupNorm** [118] is an alternative to batch normalization [44]. Despite the popularity of batch normalization, its error increases rapidly with smaller batch sizes, which is an issue for larger model types such as diffusion models. Group normalization divides the color channels of the input into groups and within each group computes the group mean μ and variance σ. The computation is independent of the batch size and the accuracy is stable with varying batch sizes.

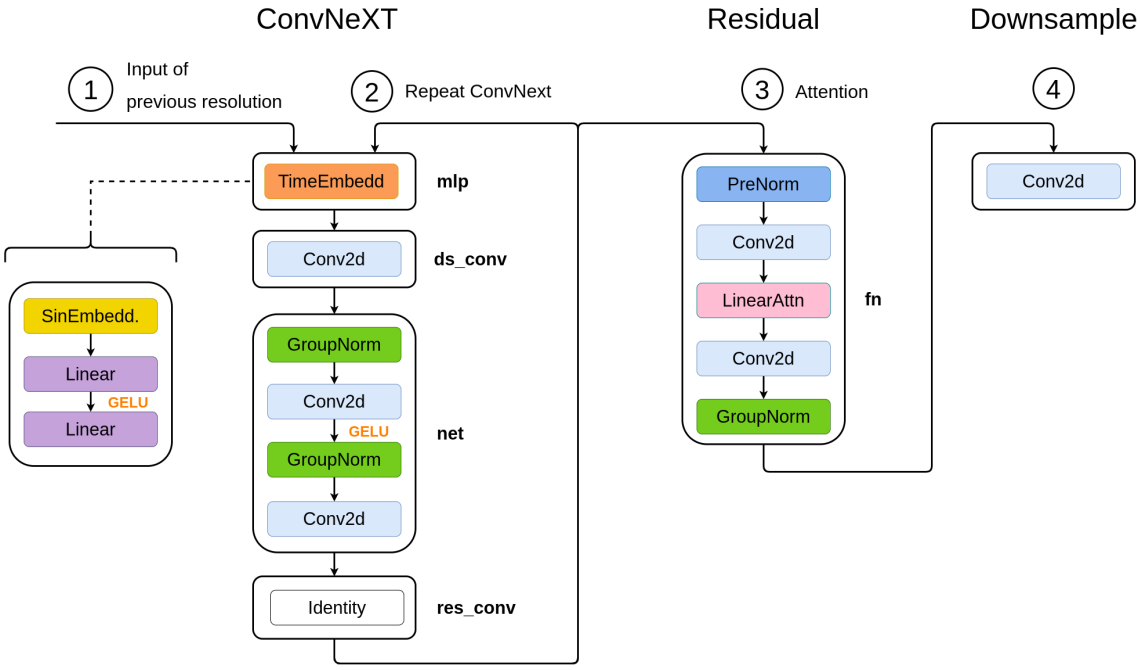


Figure 3.2: This figure depicts the succession of modules at one resolution level of the contracting path. The core three modules of the U-Net for each resolution layer are the CovNeXt modules, the attention residual, and a downsampling or upsampling convolution. As the input, this resolution level receives the output of the previous level and the sinusoidal embedding of the current time step t. The input is forwarded through a downsampling convolution and the ".net" module. After applying a residual convolution, the CovNeXt module's output is passed through a CovNeXt module once again. Afterward, the output passes the attention residual which includes the linear attention layer. Lastly, the output is passed through one final downsampling convolution and passed onto the next resolution level. In the bottleneck of the U-Net, the attention residual is interleaved with the ConvNeXT modules and in the expanding path the downsampling convolution is replaced with an upsampling one.

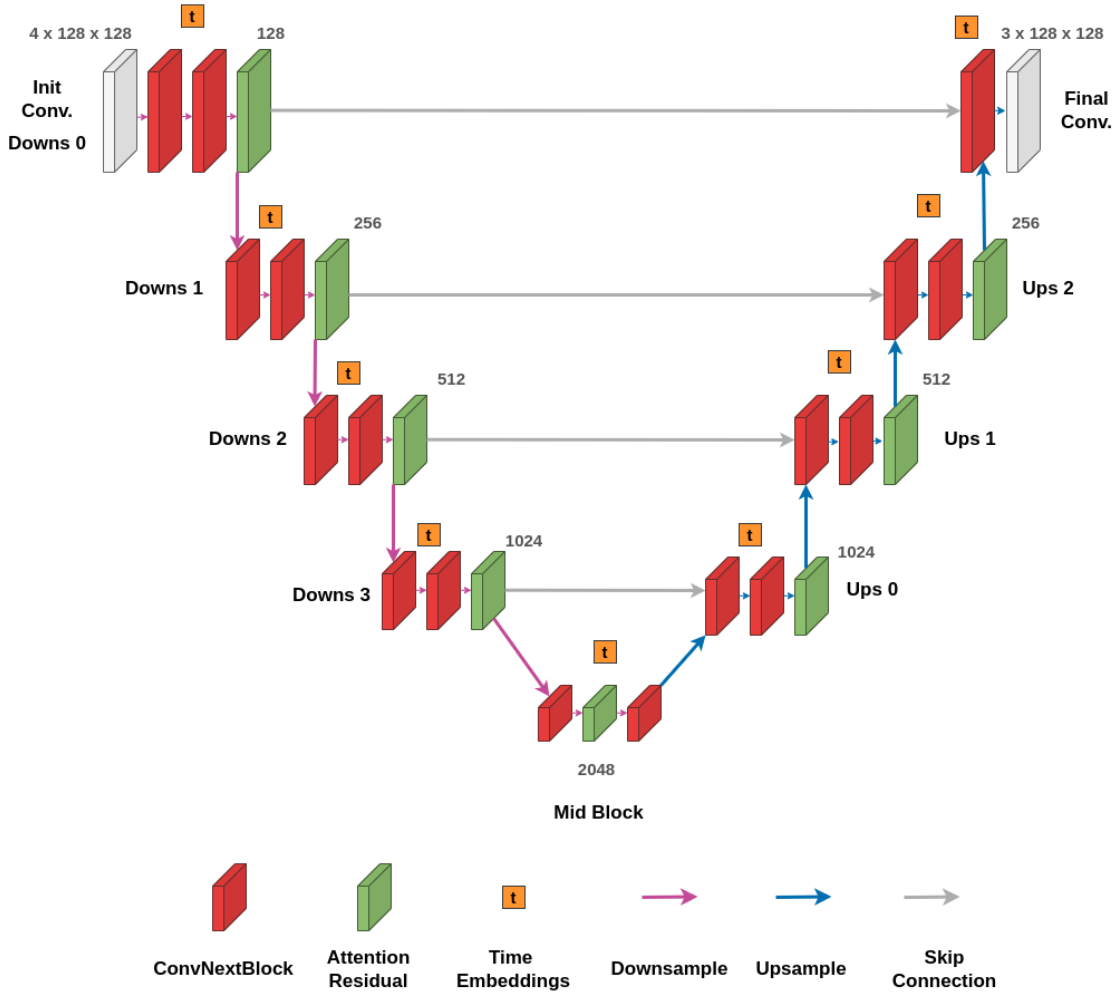


Figure 3.3: The diffusion U-Net architecture for 128 x 128 color images. As an input, the U-Net receives a batch of noisy input images $x_t \in \mathbb{R}^{B\times 3\times H\times W}$, concatenated with their respective class labels $y \in \mathbb{R}^{B\times 1\times H\times W}$, and the current time step t. First, a convolution is applied to the input and the sinusoidal position embeddings are computed for the current time step. In the contracting (downsampling) path, each resolution level is made up of two ConvNeXt modules, a linear attention residual, and a downsampling convolution. The mid-block of the network consists of two ConvNeXt modules with one attention residual between them. The expanding (upsampling) path is identical to the contracting path, except that the downsampling convolution is replaced with an upsampling one.

3.4 Conditional Diffusion

The theory and architecture for diffusion models introduced in the previous sections covered unconditional diffusion, in which we cannot control image features of the output. In this thesis, however, we want to specifically find ways to interpret how image features are formed, hence we need to be able to control them. Currently, there are two approaches how to condition diffusion models with feature labels.

Classifier Guided Diffusion [24] trains a separate classifier $f_\phi(y|x_t, t)$ on a noisy image x_t and uses its gradients $\nabla_{x_t} \log p_\phi(y|x_t, t)$ to guide the diffusion sampling process towards a class label y. Making use of the established connection between diffusion models and score matching [98], a score function can be derived from the model $\epsilon_\theta(x_t)$ predicting the noise added to a sample

$$\nabla_{x_t} \log p_\theta(x_t) = -\frac{1}{\sqrt{1-\overline{\alpha}_t}} \epsilon_\theta(x_t). \tag{3.4.1}$$

By substituting this into the score function for $p(x_t)p(y|x_t)$ we can derive a new noise prediction $\hat{\epsilon}(x_t)$ that corresponds to the score of the joint distribution

$$\hat{\epsilon}(x_t) := \epsilon_\theta(x_t) - \sqrt{1-\overline{\alpha}} \nabla_{x_t} \log p_\phi(y|x_t). \tag{3.4.2}$$

The conditional sampling procedure is identical to the regular one, only with a modified noise predictor.

Classifier-Free Guidance [37] conditions the diffusion model, not using a classifier, but rather by merging an unconditional model $p_\theta(x)$ parameterized by a score estimate $\epsilon_\theta(x_t, t)$ with a conditional model $p_\theta(x|y)$ parameterized by $\epsilon_\theta(x_t, t, y)$. A single neural network is used to parameterize and train both models, where y is treated as a hyperparameter. The unconditional model receives the null token $\varnothing$ instead of y. During training, the mode of the model is randomly set to unconditional with a specified probability, ensuring that the model learns both unconditional and conditional image generation. Sampling is carried out through a linear combination of conditional and unconditional score estimates

$$\tilde{\epsilon_\theta}(x_t, t, y) = (1+w)\epsilon_\theta(x_t, t, y) - w\epsilon_\theta(x_t, t) \tag{3.4.3}$$

where hyperparameter w controls the guidance strength. For the implementation of this thesis, we employ classifier-free guidance, as it simpler than training a separate classifier. To obtain the conditioning labels, we first extract a subset of conditioning attributes from out training dataset. The conditioning attributes are binary - the attribute is either present or not. For each combination of binary labels we assign a decimal number which forms the final conditioning label y. The conditioning label y is reshaped into the dimensions of x_t and concatenated onto x_t as a fourth channel with the color channels (Appendix C.2). This concatenation is repeated for every time step t of the sampling process before x_t is passed through the network $\epsilon_\theta(x_t, t)$.

Algorithm 3 Conditional Sampling

Input: $y \in \mathbb{N}$ - conditioning label

Output: x_0 - conditioned sample

$x_T \sim \mathcal{N}(0, I)$

H, W = height and width of x_T

Extend $y \in \mathbb{N}^{1 \times H \times W}$

$labels$ = Tensor$(1, H, W)$

y = Einstein summation "ijkl,ij →ijkl" between $labels$ and y

for t = T, ..., 0 **do**

Sample x_{t-1} with $x_t = (x_t, y)$ (Algorithm 2)

end for

Chapter 4

An XAI Interface for Diffusion Models

In this chapter, we introduce multiple explainability methods for diffusion models, including their theoretical design and implementation. We then propose the core contribution of this thesis: an XAI interface for diffusion models. The interface allows for an intuitive exploration of the explainable diffusion properties. It is guided by the concept of a *diffusion profile* - a compact object for characterizing the generative process for which secondary analyses can be derived.

4.1 XAI Approaches for Diffusion Models

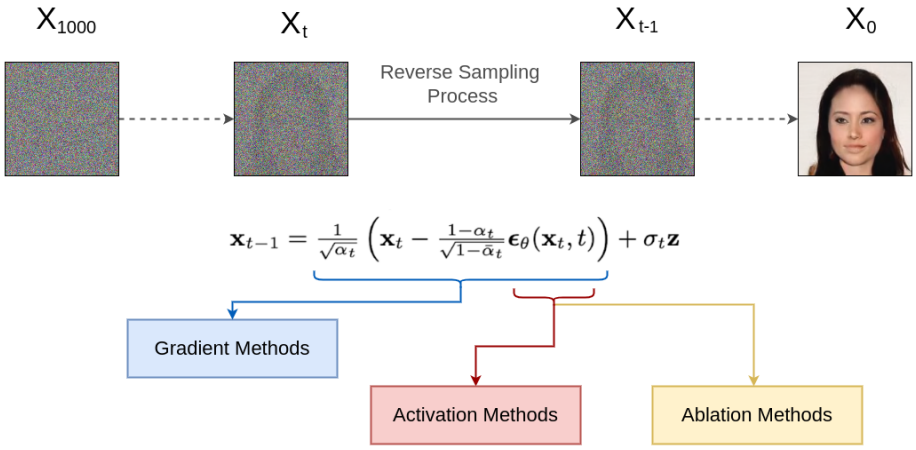


Figure 4.1: The three types of methods from which to approach explainability of diffusion models. Gradient-based methods focus on comparing the new sample x_{t-1} to the previous sample x_t. Activation-based and ablation methods focus on extracting information or intervening in the network $\epsilon(x_t, t)$ only.

4.1.1 Gradient Methods

As seen in Section 2.2.2, a model's gradient is an important object to be explored for explainability methods. It is crucial to understand if and how diffusion models encode human-understandable image concepts and how they take their form over time. When approaching this question for diffusion models, the aforementioned difficulty remains: the immense dimensionality of the sampling procedure. A standard diffusion process consists of 1000 sampling steps and backpropagating the final sample x_0 back to x_{1000} through the network $\epsilon_\theta(x_t, t)$ is computationally infeasible. Furthermore, there is no rule-based approach on how to guide backpropagation and therefore no guarantee of sensible results. Lastly, the re-parametrization of the network by ϵ_θ (Section 3.2.2) results in the network being a noise predictor, which could likely result in learned features being noisy, unlike human-understandable concepts shown in classification models [2]. Therefore, rather than backpropagating through the entire process, we propose a method that observes the gradient-based flow of information between the samples of two consecutive time steps.

Gradient Volume

Let $x_{t+1} \in \mathbb{R}^{3\times H\times W}$ be a generated sample and ϵ_θ the diffusion model, trained to predict the noise level of x_{t+1}. The slightly de-noised sample x_t is calculated using Equation 3.2.7. We would like to know how each pixel value in x_{t+1} has contributed to each pixel value in x_t and therefore define a *pixel-wise gradient* as

$$\nabla x_t^{i,j} = \frac{\partial x_t^{i,j}}{\partial x_{t+1}} = \frac{\partial \epsilon_\theta(x_{t+1}, t+1)^{i,j}}{\partial x_{t+1}} \tag{4.1.1}$$

with i, j being the positional indices of the pixel for which the gradient is calculated. The pixel-wise gradient map $\nabla x_t^{i,j}$ shows how much each pixel in x_{t+1} has impacted the individual pixel $x_t^{i,j}$. This notion is then extended upon by the *pixel-wise gradient volume*

$$PGV_{x_t} = (\nabla x_t^{0,0}, \nabla x_t^{1,0}, ..., \nabla x_t^{i,j}). \tag{4.1.2}$$

The gradient volume $PGV_{x_t} \in \mathbb{R}^{K\times H\times W}$ is the concatenation of each of the individual gradient pixel maps, where $K = H \times W$ is the number of pixels in x. As PGV_{x_t} can be extracted at any time step, we can observe if and how the gradient volume changes during the reverse generative process. Furthermore, PGV_{x_t} does not have to be calculated for all pixels but can be limited to a specified image area. Thus, we can analyze how an image area in the final output x_0 forms over time. Ideally, this would allow us to trace the formation of semantically meaningful image areas.

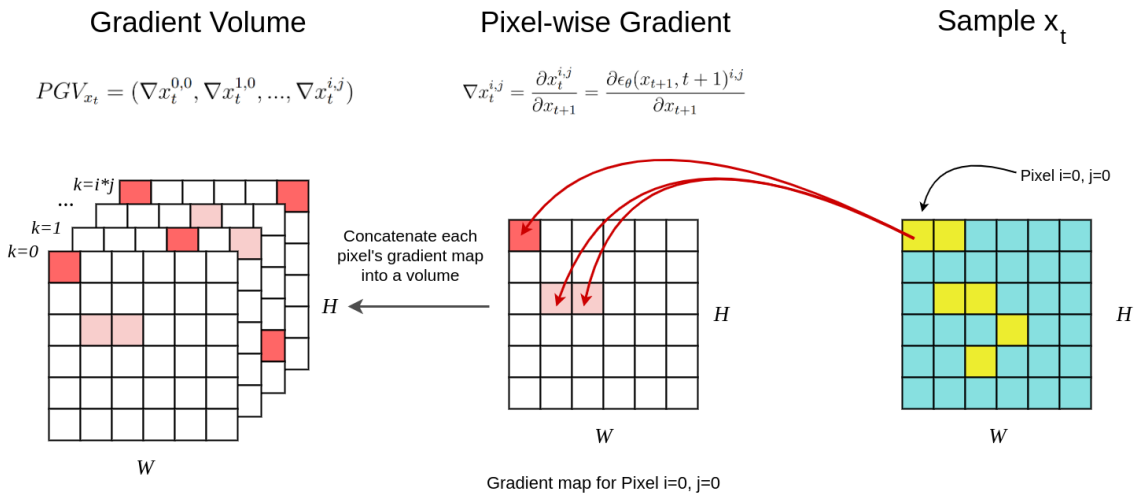


Figure 4.2: To calculate PGV_{x_t} we first calculate the pixel-wise gradient $\nabla x_t^{i,j}$, resulting in a gradient map indicating how each pixel in the previous sample x_{t+1} contributed to the selected pixel $x_t^{i,j}$. We repeat the calculation for all selected pixels and receive the pixel-wise gradient volume in $\mathbb{R}^{H \times W \times K}$, with K being the number of selected pixels.

Implementation

To calculate PGV_{x_t}, we need to gain access to the model's gradients. To do this, we modify the sampling procedure (Algorithm 2) of the base implementation [78]. We extend the original sampling procedure by two parameters: a sampling time step s at which PGV_{x_t} should be extracted, and a binary image mask $M \in \{0,1\}^{H \times W}$. Let $C = \{(c_1, c_2) | c_1 \leq H, c_2 \leq W\}$ be the set of tuples representing the selected pixels for which the gradient should be computed. Based on C, we set the values of M to

$$M_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in C \\ 0 & \text{else} \end{cases} \tag{4.1.3}$$

In the adapted sampling algorithm, we first sample x_t according to the original implementation. We then mask x_t by multiplying it with the binary image mask M and store the output in a separate variable $\overline{x}_t$. To reduce the dimensionality of PGV_{x_t}, we flatten the masked sample $\overline{x}_t$ such that the pixel indices i, j are of one dimension with size $K = H \times W$. Instead of returning the new sample immediately, we calculate ∇v^k for every non-masked element in $\overline{x}_t$ and concatenate the individual maps into PGV_{x_t}. As we are sampling RGB images, we technically receive a gradient map for every color channel. However, to further reduce the size of the gradient volume, the channels are added into one value per pixel. The following Algorithm 4 summarizes the gradient volume calculation for one sample time step s.

Algorithm 4 Pixel-wise Gradient Volume

Input: s - sampling step , M - image mask

Output: x_0 - final sample, PGV_{x_t} - gradient volume

```
for t = 1000, ..., 0 do
    Sample x_t from ε_θ(x_{t+1}, t+1) (Algorithm 2)
    if t is equal to s then
        C, H, W ← Shape of x_t
        K ← H × W
        Initialize PGV_{x_t} as Tensor(K, C, H, W)
        x̄_t ← x_t × M
        Flatten x̄_t into shape (K,C)
        for index k, pixel-value v in x̄_t do
            if v is masked then skip
            ∇v^k = ∂v^k / ∂x_{t+1}
            Sum ∇v^k along channel axis C
            PGV_{x_t}[k] ←∇v^k
        end for
    end if
end for
return x_0 and PGV_{x_t}
```

As the computation of the gradient volume already requires a backward pass through the entire network, we can additionally extract layer-wise gradient maps. Using the PyTorch module *hooks*, we attach a function extracting the gradient of an individual layer. This function is triggered with each backward pass and stores the gradient output in a separate dictionary, identifiable by layer id and time step. In addition to the gradient volume, we therefore have access to the layer-wise gradient maps $g_{l_t} \in \mathbb{R}^{C \times H \times W}$, where C denotes the number of feature maps in the layer, not to be confused with the color channel of the samples. For the PyTorch implementation of the hook and channel ablation, see Appendix C.3.

4.1.2 Activation Methods

Inspired by unit ablation (Section 2.3.1) and feature visualization (Section 2.2.2) methods, we would like to explore if it is possible to match units to semantic concepts or image attributes. We therefore extract and visualize layer-wise activations a_{l_t} at specified time steps and search for semantically meaningful structures.

Activation Maps

The previously mentioned PyTorch module *hooks* can also be used to extract a layer's output in the forward pass through the network. The focus is on analyzing the channel-wise activation maps of individual layers and trying to match highly activating patterns to the image output.

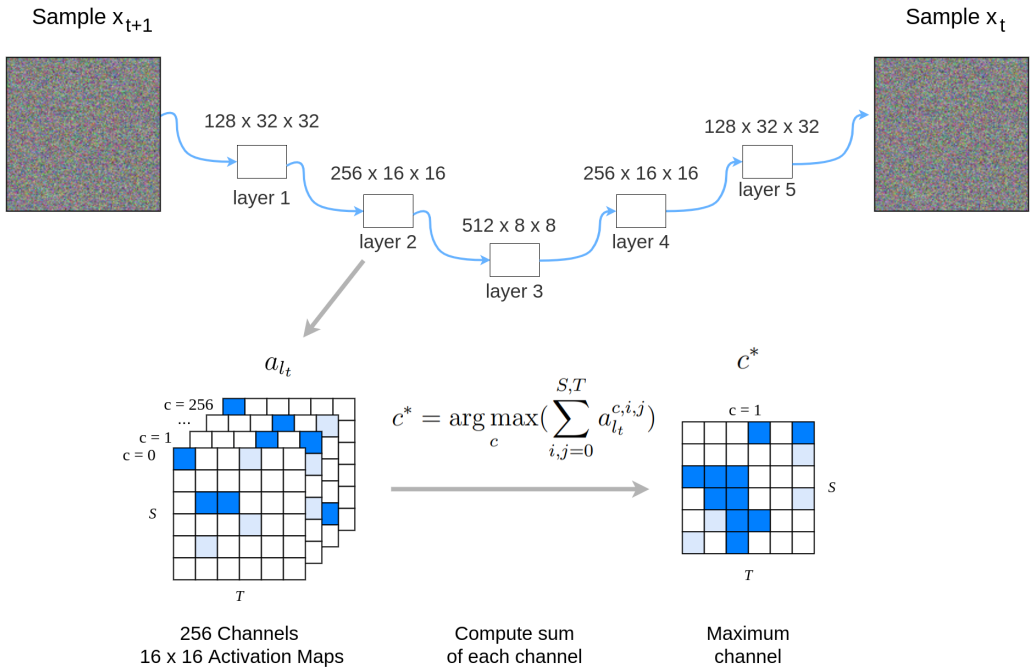


Figure 4.3: To calculate the maximum channel of a layer $a_{l_t}^{c^*}$, we first extract the target layer using the PyTorch module *hooks*. For every channel in the layer, we compute the sum of all activation values and return c^* as the channel with the maximum sum.

To avoid "trying to find a needle in a haystack", the starting point is the maximum channel c^* of a selected layer's activations $a_{l_t} \in \mathbb{R}^{C \times S \times T}$, with S and T being the spatial size of the activation map. The maximum channel c^* is the channel with the maximum sum of elements

$$c^* = \arg\max_c (\sum_{i,j=0}^{S,T} a_{l_t}^{c,i,j}). \tag{4.1.4}$$

It is worth noting that defining c^* as the maximum channel-wise sum can result in an aggregation problem. For example, a channel could have highly activating areas and negatively activating areas, which cancel out when calculating the sum. We might overlook channels with structures relevant to only a subset of the final sample. To counteract this effect, we enable the calculation of c^* wrt. a specified image area of x_t. We apply a binary mask M to a_{l_t}, similar to the masking of the gradient volume (Section 4.1.1). The key difference is that we are working with

different resolution levels and cannot simply apply the binary image mask. Instead the image mask $M \in \{0,1\}^{H \times W}$ is defined on the final output $x_0 \in \mathbb{R}^{C \times H \times W}$. We then re-sample the activations, and bilinearly interpolate M to the spatial size of $S \times T$ before calculating c^*.

Extracting the activations throughout the diffusion process gives us multiple interpretation angles. We can analyze the impact of highly activating channels by ablating them (Section 4.1.3) and track if c^* is constant or changes during the generative process. Visualizing individual activation channels could help to understand if there is a switch in the functionality of the diffusion process. Furthermore, we can apply the maximum channel calculation to the layer-wise gradient maps g_{lt} of the previous section, to find $g_{l_t}^{c^*}$.

Attention Map

Similarly to the previously introduced methods of attention extraction (Section 2.3.2), a special focus is given to the attention layers in the network.

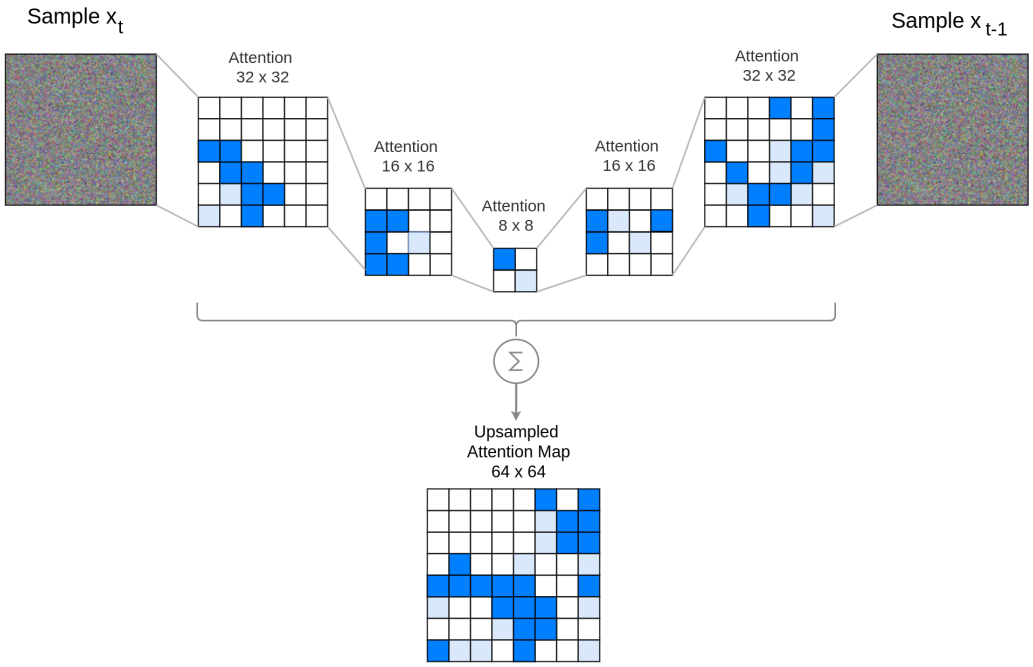


Figure 4.4: This figure depicts the approach of calculating an attention map $A_t\mathbb{R}^{1 \times 64 \times 64}$ at time step t. The attention layers of all resolution levels are extracted and upsampled to the same width and height as $x_t \in \mathbb{R}^{C \times H \times W}$. The upsampled attention layers are then added into a singular map $A_t \in \mathbb{R}^{1 \times H \times W}$, highlighting which areas of x_t are focused on by the attention layers.

The attention layers guide the diffusion process towards salient image regions and play an important role in the formation of structures. This provides a good starting point for visualizing the diffusion process. The goal is to extract a single attention

map for a specified time step t. The extraction algorithm is identical to extracting layer-wise activations (Algorithm 5). However, as we extract attention layers at different resolution levels, we upsample each extracted layers through bilinear interpolation and add them into a single attention map A_t.

Implementation

We further extend the sampling procedure and attach a forward hook at all selected layers. During the forward pass through the network, if the current time step t is equal to the sampling time step s, the hook extracts the layer-wise activations a_{l_t}.

Algorithm 5 Extracting Layer-wise Activations

Input: *layers* - layer id's , s - sampling time step, M - binary image mask

Output: *acts* - layer-wise activations

```
for t = 1000, ..., 0 do
    if t is equal to s then
        Attach hook h to all layers
        Sample x_t from ε_θ(x_{t+1}, t + 1) (Algorithm 2)
        for l in layers do
            acts[t][l] = h[l]
            Detach hook h from all layers
        end for
    end if
end for
return acts
```

4.1.3 Ablation Methods

Algorithm 6 Channel Ablation

Input: *layer* - Layer id, *channel* - channel index

Output: x_0 - final sample with ablated channel

```
S, T = height and width of layer
Output hook h[channel] = Zero Tensor(1, S, T)
Attach hook h to layer
Sample until x_0 (Algorithm 2)
```

Channel ablation is implemented to investigate the impact of an individual channel c on the final output. We attach a forward hook, similar to the activations hook

(Subsection 4.1.2), to the selected layer. The hook modifies the layer's output s.t. the output is identical except for channel c, which is set to zero. The channel ablation is active for the entire sampling process. After sampling with the ablated channel, we analyze the final sample x_0, activation maps a_{l_t} and gradient maps g_{l_t} for changes. Furthermore, multiple channels can be masked at the same time, by attaching multiple hooks. For the PyTorch implementation of the hook and channel ablation, see Appendix C.1.

4.1.4 Measures

In addition to extracting and visualizing the model's internals throughout the generative process, we employ quantitative measures for our analysis. To assess image quality and changes between samples, we use the Structural Similarity Index Measure (SSIM) [113]. Furthermore, we use the Euclidean distance to trace changes in activations and gradients.

Structural Similarity Index Measure

The Mean-Square Error (MSE) and the Peak-Signal-to-Noise Ratio (PSNR) are two common metrics to evaluate the quality of images. A drawback of these metrics, however, is that they are not normalized and do not render absolute errors. Furthermore, they are not designed to take the functionality of human perception into account. A "good" result for MSE does not necessarily translate into a visually convincing sample, as saliency-based errors are not assessed, limiting the interpretability of both methods [90].

The SSIM [113] is a measure developed for assessing perceptual image quality by quantifying image quality degradation. It is a difference measure, meaning it cannot assess individual images, but requires a reference image. The SSIM aims to mimic the human visual system which is highly adapted to extract structural information and consider dependencies between pixels. Structural information are attributes of an image that represent the structure of objects independent from the luminance or contrast. As such, SSIM separately measures luminance l, contrast c, structure s and constants C_1, C_2, C_3

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{4.1.5}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_1} \tag{4.1.6}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x^2\sigma_y^2 + C_3} \tag{4.1.7}$$

with μ being the mean intensity of the image and σ the standard deviation of the mean. Additionally, each image is normalized to have unit standard deviation

before comparison.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 C_2)} \tag{4.1.8}$$

The SSIM is symmetric $S(x,y) = S(y,x)$, bounded $S(x,y) \leq 1$ and has a maximum $S(x,y) = 1$ that is only attained if x and y are identical [113].

To assess the generative process of a single sample, we measure SSIM(x_t, x_{t-1}) for every $t \in [0, 999]$ and track how much the image has changed in comparison to the previous time step. This can help us analyze if the diffusion process is linear or shows a different trajectory.

Euclidean Distance

For a matrix $A \in \mathbb{R}^{m \times n}$ the Euclidean norm, also called Frobenius norm [114], is defined by

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{i,j}|^2}. \tag{4.1.9}$$

To observe the change in layer-wise activations between two timesteps, we calculate the two layers Euclidean distance along channel axis C, height S, and width T. Let $a1 = a_{l_t}$ and $a2 = a_{l_{t+1}}$:

$$LayerDist(a1, a2) = |a1 - a2|_F = \sqrt{\sum_{k=1}^{C}\sum_{i=1}^{S}\sum_{j=1}^{T} |a1_{k,i,j} - a2_{k,i,j}|^2} \tag{4.1.10}$$

4.2 Diffusion Interface

In the previous section, we explored multiple methods from which to approach explainability for diffusion models. We now introduce the *diffusion profile*, combining all methods into a novel, unified primitive object. The diffusion profile is unique for every sample x and allows for inter- and intra-object comparison of its attributes. Using it, we can analyze the diffusion model's mechanism along the profile's four axes: time, objects of interest, measures, and conditioning.

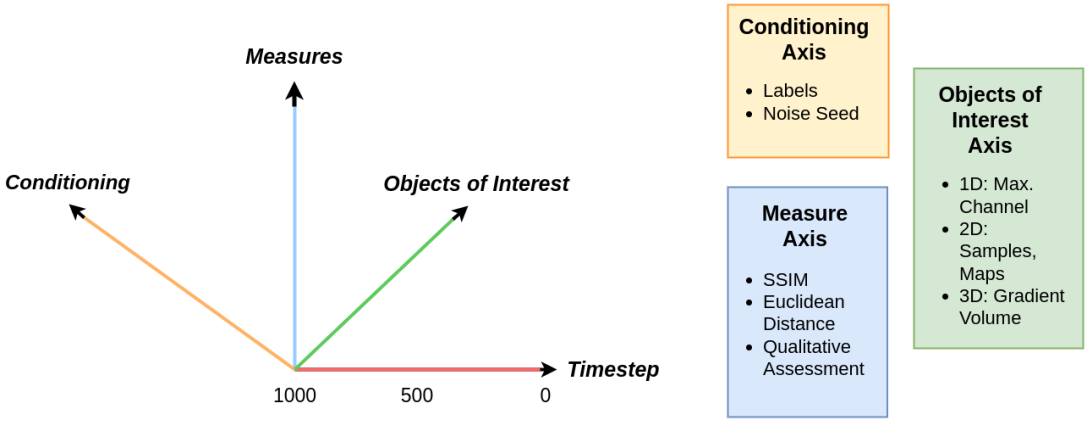


Figure 4.5: The diffusion profile, combines the multiple explainability methods, conditioning attributes, and measures into a unified primitive for each time step of the generative process.

The time axis of the diffusion profile mirrors the time axis of the reverse generative process. Each time step t generates a unique set of objects of interest, mirrored in the second axis, which we can extract using the methods introduced in the previous section. This includes layer-wise activations a_{l_t}, their maximum channels c^*, the gradient volume PGV_{x_t} and the sample x_t itself (Section 4.1.1 - 4.1.3). The third axis comprises the measures used to quantify the objects of interest, namely SSIM and Euclidean distance (Section 4.1.4). We also include the qualitative assessment of the samples, such as the presence of noise, artifacts, and fidelity to the conditioning label. A key interest is on how conditioning (Section 3.4) affects the objects of interest, represented as the fourth axis. Ultimately, this results in a high-dimensional, yet accessible tensor characterizing the generative process for every time step.

4.2.1 Interface Concept

To make the diffusion profile accessible, we build an interface allowing sampling from a pre-trained model and inspection of the sample-wise diffusion profile. As a first step, we create a conceptual design of the interface. To avoid overwhelming a user, we separate the interface into two parts: the general settings, required to sample the diffusion profile, and the results, sub-divided into the different objects of interest.

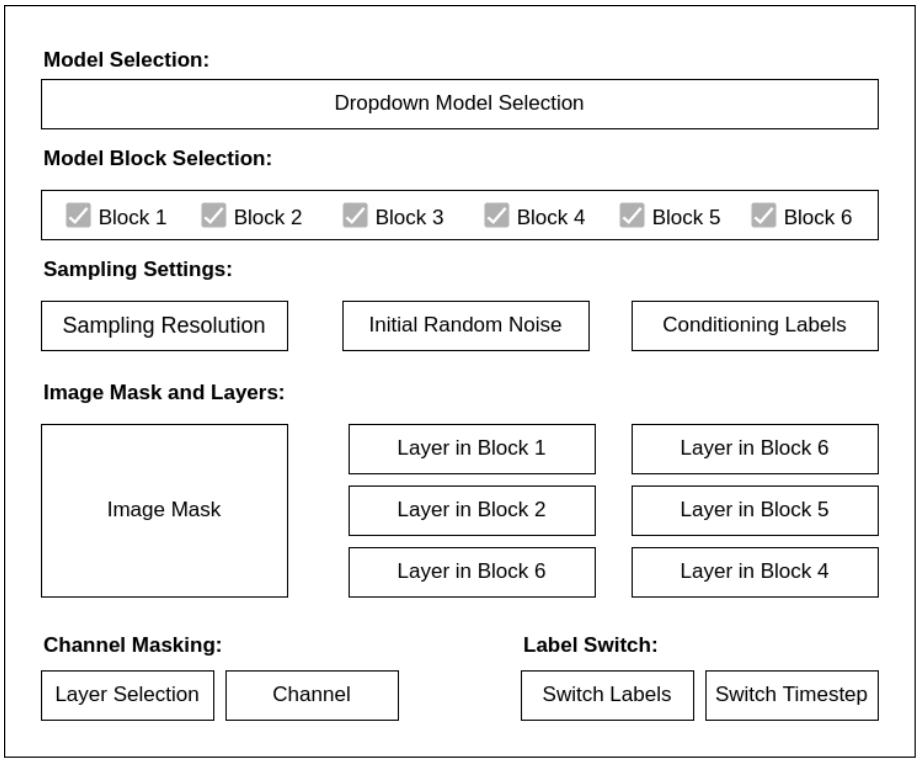


Figure 4.6: Wireframe of the interface settings section. This includes model, block, and layer selection. Furthermore, the sampling resolution, noise seed ϵ, conditioning label y, and image mask M are configurable. The section also includes the settings for two experiments: channel ablation and label switch.

The outline for the general settings can be seen in Figure 4.6, for which we require a variety of inputs and parameters:

- **Model Selection** allows the user to choose which pre-trained model to load into the interface. This involves setting the appropriate diffusion parameters and loading the model's weights.

- **Model Block Selection** allows the user to choose at which resolution levels, here referred to as blocks, to extract the diffusion profile's objects from. There is an equal amount of blocks in both the contracting and expanding path of the U-Net, as well as one block in the bottleneck (Section 3.3). The user can select at most six blocks for visualization at a time.

- **Sampling Settings** includes parameters necessary for sampling and conditioning. The sampling resolution defines the interval of sampling steps at which the objects of interest are drawn from the diffusion model. The initial noise seed $\epsilon \sim \mathcal{N}(0, I)$ can be manually set to allow re-sampling of the same sample with different parameters. Furthermore, the conditioning labels (Section 3.4) are to be selected in this section.

- **Image Mask** M is used to mask the sampled activations a_{l_t} or gradient volume PGV_{x_t}. The mask is defined by drawing on a previously sampled image (Section 4.1.1).
- **Layer Selection** is made based on the previously selected blocks. Ultimately one layer per block is visualized in the interface.
- **Channel Masking** allows the user to ablate a specified channel by layer id and channel index (Section 4.1.3).
- **Label Switch** receives a time step and a new label to switch to conditioning during the sampling process. This is part of the experiments, more detail in Section 5.3.2.

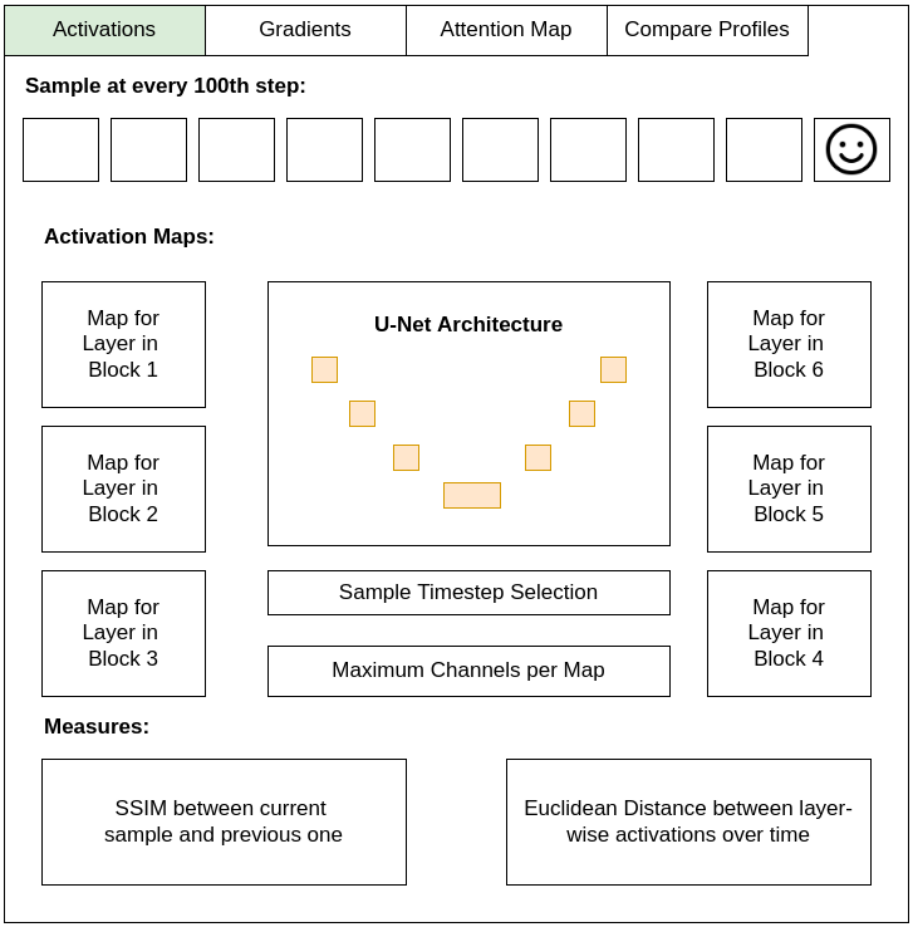


Figure 4.7: Wireframe of the interface activation-based results. This includes the sample, activation maps a_{l_t} for all selected layers, their maximum channels, the SSIM trajectory and the Euclidean distance between sampling steps s.

The second section of the interface is meant to visualize the extracted objects of interest and measures. It is once more subdivided into different sections. For the

intra-comparison of the diffusion profile, we include separate sections for activation, attention, and gradient-based results. A fourth section implements the inter-comparison between diffusion profiles of different conditioning such as noise seed ϵ and conditioning label y.

The wireframe in Figure 4.7 outlines how the activation-based results are visualized. All sections follow the similar structure, in which the samples are visualized alongside objects of interest and measures relevant for the selected method.

- **Sample** shows x_t at every 100th time step during the generative process. We can see how the final sample forms over time and adjust the choice of noise seed and conditioning labels to generate a new sample.
- **Activation Maps** displays the layer-wise activation maps $a_{l_t}^c$ based on the selected block and layer in the settings section. Using a separate panel we can select the sampled time step for which to visualize $a_{l_t}^c$. Furthermore, the maximum channels c^* are displayed for every layer.
- **Measures** visualize the change of SSIM between x_t and x_{t-1} for every time step in the generative process. The Euclidean distance is calculated between layer-wise activations according to the sampling resolution at a specified interval between two consecutive steps. The results are visualized for every layer type separately.

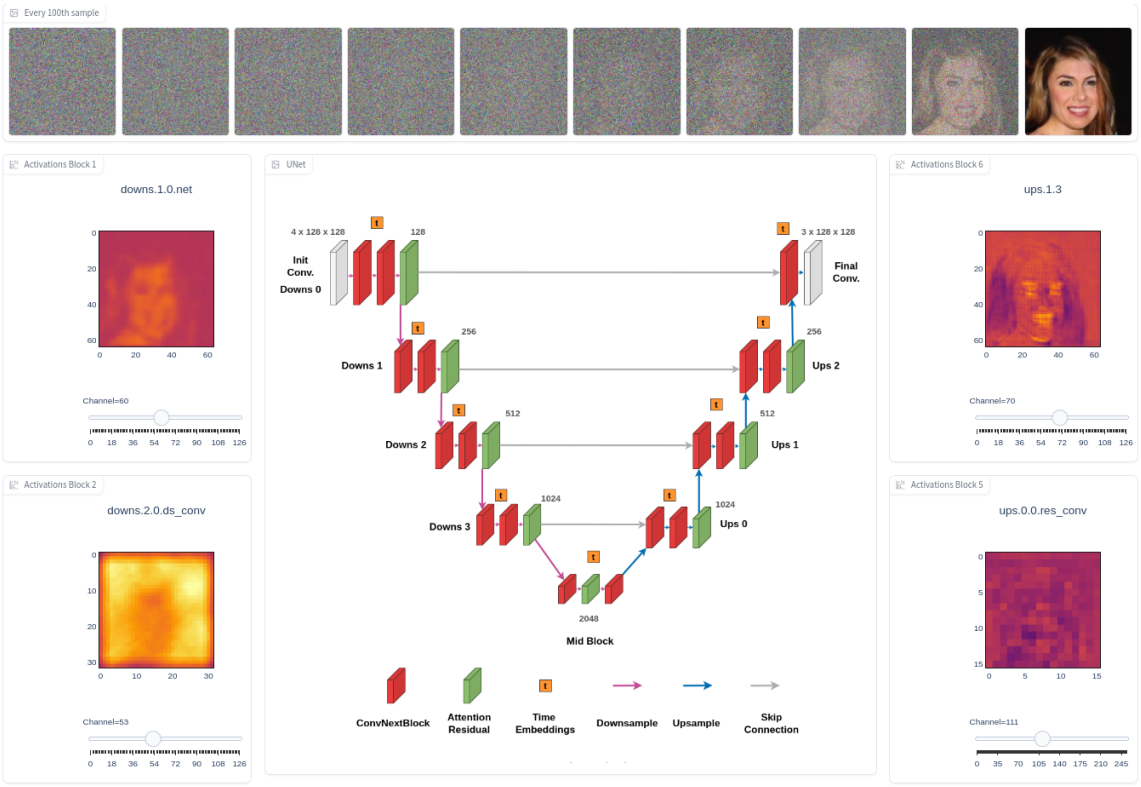


Figure 4.8: In the finished interface, the layer-wise activation maps are shown alongside the sample and a schematic overview of the U-Net architecture.

Figure 4.8 shows the activation-based section of the finished interface, including a sample and its extracted activation maps. The remainder of the finished interface can be seen in Appendix A.

Chapter 5

Experiments

In the penultimate chapter of this thesis, we showcase the capabilities of the diffusion interface in action through a series of experiments. First, we detail the dataset used to train two diffusion models. Afterward, we demonstrate the extraction of objects of interest and measures using the interface. We conclude the chapter with two higher-level analyses that allow for a novel characterization of the generative diffusion process.

5.1 Dataset

CelebFaces Attributes Dataset (CelebA) [60] is a large-scale dataset containing over 200.000 images of celebrity faces. Each image is annotated with 40 attributes, allowing for detailed filtering of facial characteristics such as skin tone, hair color, mouth shape or even worn accessories. The images also vary in posing and background, offering large diversity with over 10.000 different persons depicted. Since its release in 2015, the dataset has been popular for training various types of generative models [24, 45, 47, 108]. A clear advantage of generating synthetic human faces is that the results can be assessed intuitively, as opposed to data requiring domain experts for interpretation.

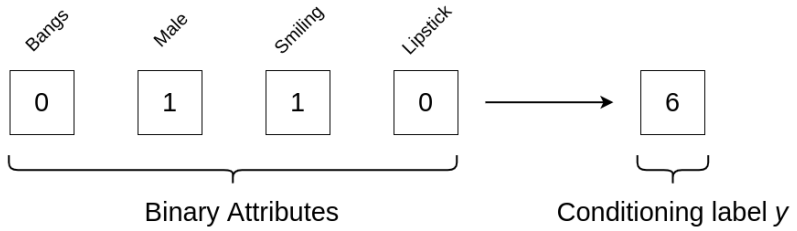
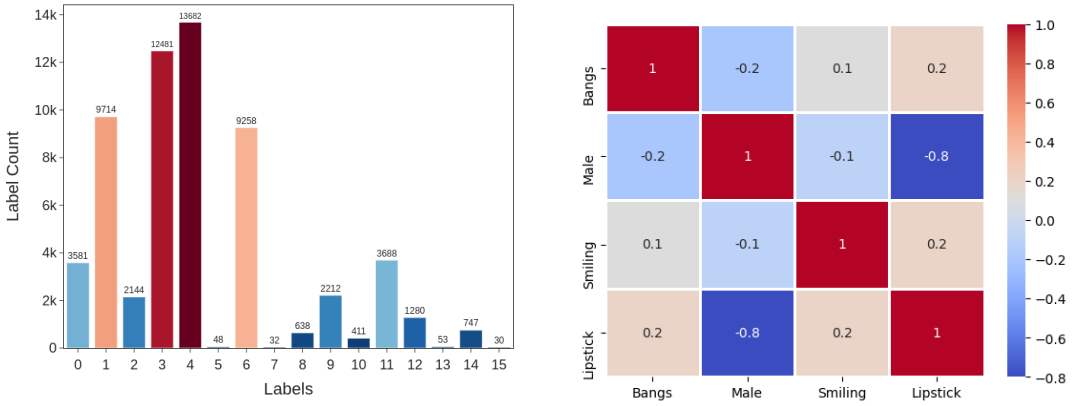


Figure 5.1: For each image, the facial annotations for bangs, male, smile, and lipstick are acquired. Each label then corresponds to a four-bit string. If the attribute is present in the image, the digit is set to 1, if absent, it is set to zero. The binary label is then converted to decimal for training and conditional sampling (Algorithm 3).

We trained on a subset of 60.000 CelebA images and to condition the diffusion model, we had to select the training labels from CelebA . With over 40 different attributes, the binary combination of all attributes would have resulted in 2^{40} different labels, evidently infeasible to train with. Instead, the controlled attributes were narrowed down to four binary labels: bangs, male, smile, and lipstick, resulting in 16 different training label combinations (Figure 5.1).



Training label distribution. Attribute correlations.

Figure 5.2: (Left) The label distribution of the 60.000 training images used for our experiments. There is a stark label imbalance for any label which combines the attributes "man" and "lipstick". (Right) The correlations between the four selected attributes from the CelebA dataset. The strongest (negative) correlation is between "man" and "lipstick", but "bangs" also has a stronger association with "woman" or the presence of "lipstick".

One issue in working with facial attributes is that some are more strongly associated than others (Figure 5.2). Generating binary labels results in a stark class imbalance for less common attribute combinations (see Figure 5.2). Label four, equivalent to a man with no bangs, no lipstick and not smiling, is a combination of attributes that occurs the most with nearly 14.000 occurrences. The strong negative correlation between "Male" and "Lipstick" results in a very low amount of training images for any labels which combine these attributes, namely five (0101), seven (0111), thirteen (1101), and fifteen (1111). While biased training data poses clear drawbacks, it allows for qualitatively inspecting the differences between class labels or how the generative process of diffusion models is affected by biased data.

For the following experiments, two diffusion models were trained (Appendix B): a lower resolution model with an image size 64x64, helpful for demonstrating the pixel-wise gradient volume, and a model with a larger 128x128 image size.

5.2 Interface for Individual Samples

In this section, we demonstrate the capabilities of the interface to extract and visualize the objects of interest for individual samples. Based on the extracted information, we characterize the emergence of structure during the generative process, as well as showing how localized the information is. Lastly, we analyze the importance of individual channels through measuring the impact of ablation.

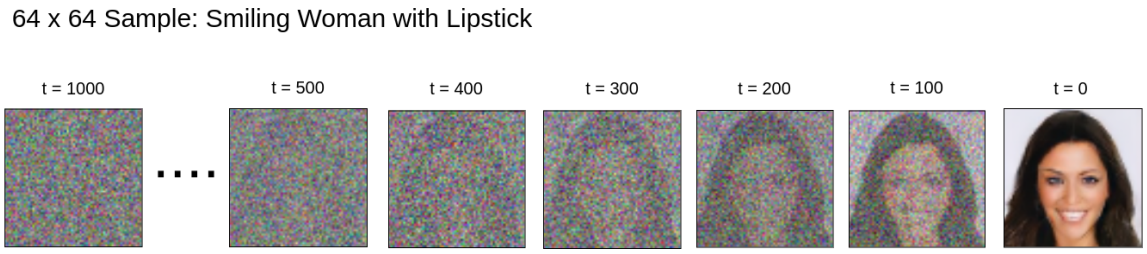


Figure 5.3: Exemplary diffusion process for the 64 model with noise seed 321 and label 0011. The samples from the diffusion process are drawn every 100 steps. Just looking at the samples, the final image emerges around 400 steps and details are distinguishable from the noise around step 100. The slight pixelation of the image is caused by the low resolution and not the presence of noise or artifacts.

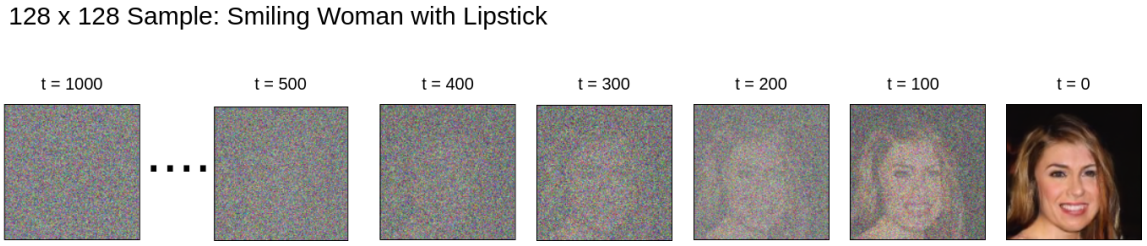


Figure 5.4: Exemplary diffusion process for the 128 model with noise seed 432 and label 0011. For approximately the first 700 steps no shape can be detected through mere visual inspection. The first facial outline emerges around step 300 and details emerge around step 100, similarly to the sample from the 64 model.

For gradient-based methods, we use a sample from the 64 model (Figure 5.3) to reduce computational overhead. Other analysis methods are demonstrated using the sample from the 128 model (Figure 5.4). The samples represent a good result for the label 0011 - a woman without bangs, who is smiling and wearing lipstick.

Both the 64 and the 128 models have the same core architecture in terms of depth and amount of layers, they only differ in resolution. In total, they consist of 10 different blocks: the initial and final convolutional block, four downsampling, three upsampling, and one bottleneck block. As mentioned in Section 3.3, we have

four types of modules: the CovNeXt (in the network denoted as ".net"), downsampling/upsampling (".ds_conv"), residual (".res_conv") and attention (".fn") modules. Furthermore, we can extract the last convolutional layer and output of the block (".3").

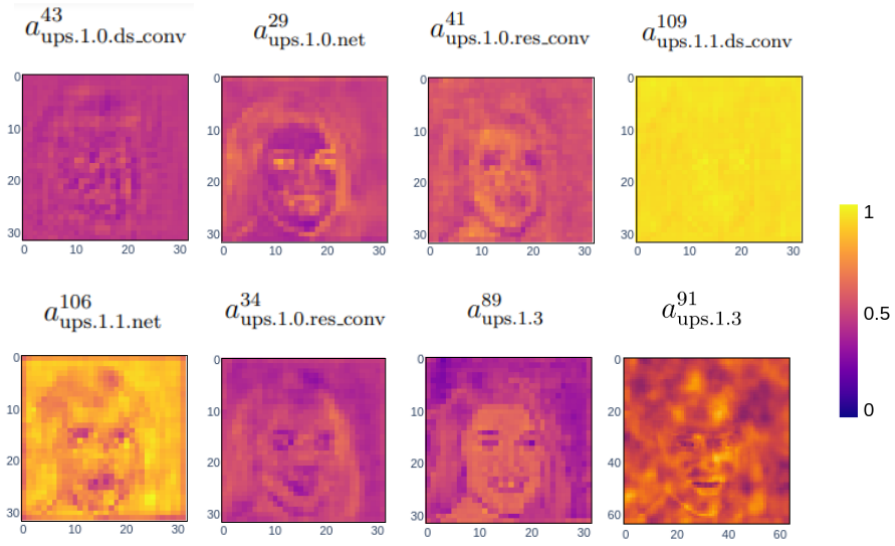


Figure 5.5: Different types of layers within the second upsampling block at time step $t = 200$. We can see that the upsampling layers (".ds_conv") activate more evenly across the activation map, with the structure being less visible in comparison to the other layers. The CovNeXt layers (".net") show structures and activate for edges and eyes in the first CovNeXt layer (".0.net") while the second (".1.net") activates more highly for background, skin tone, and hair, as opposed to detailed features such as eyes. The attention layer (".fn") is responsible for encoding important image regions and producing activations which closely match the outlines of the final sample, including the presence of a smile and the shape of the hair.

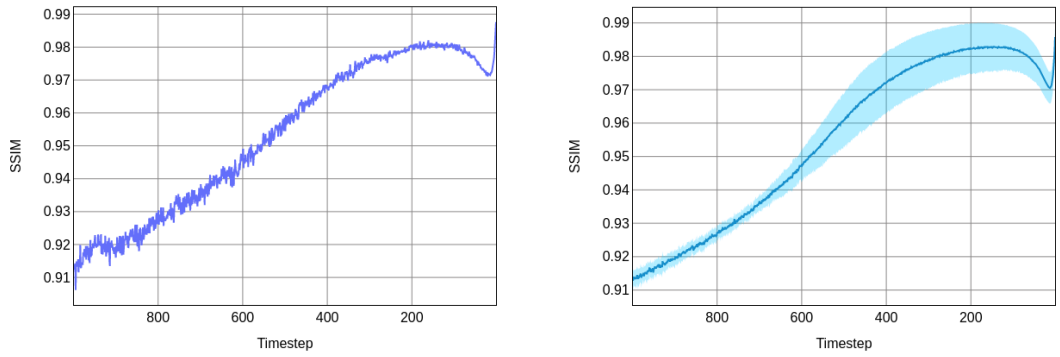
As the focus of the experimental section is placed on extracting visually interpretable information, the lower-resolution blocks are disregarded. In Figure 5.5, we visualize $a_l^{c^*}$ for every layer l in the second upsampling block at $t = 200$. We can see that some layers activate more broadly, while others contain structures such as the facial outline. Based on this observation, we focus on visualizing the attention (".fn") module and the last convolutional layer (".3") of selected block.

5.2.1 The Emergence of Structure

As we have seen in Figure 5.5, we can extract and visualize network internals which contain structure matching the final sample. However, this is not particularly surprising, especially at a fairly late time step. The core question we focus on in this

section is to identify how early we can see structures emerge using the diffusion interface. Looking just at the samples themselves, the emergence seems to occur around step 300 to 200 (Figures 5.3 and 5.4). Firstly, we apply our measures to quantify change during the generative process and define time steps of interest. Secondly, we extract objects of interest at these time steps and inspect the localization of information. We uncover a unique trajectory of the generative process and propose that objects of interest show structures earlier than in the noisy samples.

Using Measures to Identify Points of Interest



SSIM $\epsilon = 432$, $y = 0011$ SSIM 30 random noise seeds ϵ

Figure 5.6: (Left) The SSIM trajectory for the 128 sample (Figure 5.4). For the first 750 time steps of the generative process, the SSIM progressively increases - meaning the change between samples decreases, until it plateaus around 250 time steps. This plateau indicates a steady level of similarity between samples. Interestingly, in roughly the last 10% of the generative process, the similarity sharply decreases and then increases again until the SSIM reaches the value of 1 at the final step. (Right) To test if this trajectory is unique to the individual sample or a general pattern for the reverse process, we calculate the SSIM trajectory for 30 random samples. All samples are of label 0011 and from the 128 model. The depicted mean and variance bound show that the trajectories are following a similar pattern as the individual one.

Deja et al. [21] proposed a characterization of the diffusion process which distinguishes between two functionalities of the model: the generator, generating a corrupted sample from noise, and a denoiser, which reconstructs the corrupted sample into the final sample x_0. They found that roughly the first 10% of the forward process constitute the denoiser which fluidly transitions into a generator. Similarly, we are interested in uncovering the progression of the reverse generative process. Using the SSIM (Section 4.1.4), we measure the similarity between a sample x_t and the following, slightly de-noised sample x_{t-1} for every time step in the reverse process. The results (Figure 5.6) depict a non-linear progression for the 128 sample. To verify

that the trajectory is not unique to the specific sample, we compute it for 30 random samples with the same label and find a consistent pattern: in the first approx. 750 steps of the generative process similarity between samples increases, meaning the change between time steps decreases. However, the trajectory plateaus around $t = 250$ until a distinct decrease and increase of similarity in the last 100 time steps. This trajectory pattern gives us multiple time steps of interest to sample the diffusion profile.

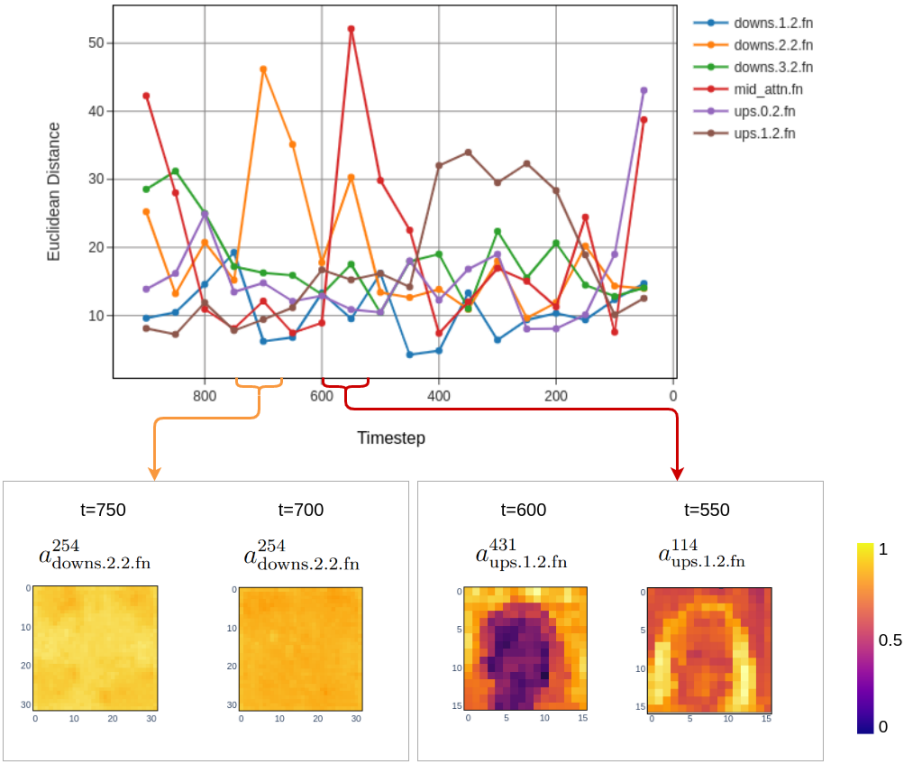


Figure 5.7: $LayerDist(a_{lt}, a_{lt-50})$ for a subset of attention layers (Subsection 4.1.4). Visualizing two time steps, we see the change in $a^{c^*}_{\text{ups.1.2.fn}}$ around time step 600. $a^{431}_{\text{ups.1.2.fn}}$ activates highly for the background while $a^{114}_{\text{ups.1.2.fn}}$ activates for the hair. $a^{c^*}_{\text{downs.2.2.fn}}$ stays constant between time step 750 and 700 and activates broadly across the activation map.

Furthermore, we use the Euclidean distance (Section 4.1.4) to find time steps of interest. In the example shown in Figure 5.7, we extracted a_l for six attention layers at every 50 time steps during the reverse process. We then calculate the distance between a_{l_t} and $a_{l_{t-50}}$, quantifying the change between layer-wise activations. Our examples demonstrates that a change in Euclidean distance does not necessarily mean we see an interpretable change in the activation maps. We do however notice, that layer "mid_attn.fn" already contains the outline of the final sample (Figure 5.4), while the corresponding sample x_{550} is still noisy without structure. This is an indication that structures emerge earlier than anticipated.

Localized Information

Using the pixel-wise gradient volume PGV_{x_t}, we now inspect where each pixel in the sample x_t draws its information from the preceding sample x_{t+1}. We sample at four time steps from the 64 sample (Figure 5.3). The time steps are chosen such that they follow the points of interest of the SSIM trajectory. Furthermore, we define an image mask M, limiting the gradient calculation to the area of the eyes. It is important to mention that the gradient volume is thresholded and only values above the gradient volume mean are shown. Beyond gradient volume, we also track the layer-wise gradient maps $g_l^{c^*}$ for three attention layers (Figure 5.9).

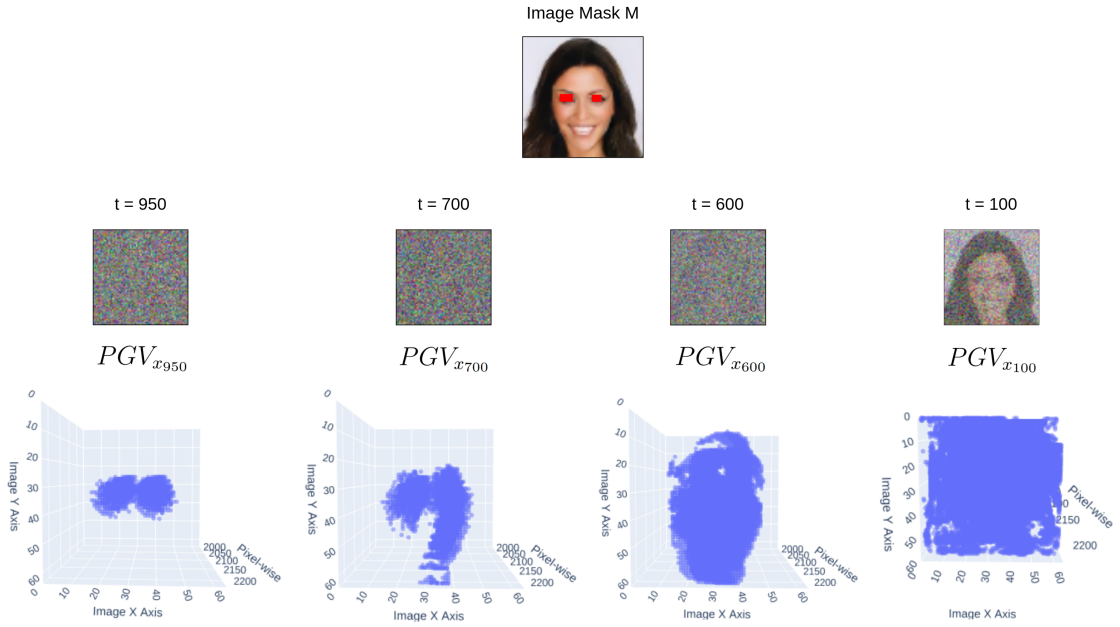


Figure 5.8: Gradient volumes $PGV_{x_{950}}, PGV_{x_{700}}, PGV_{x_{600}}$ and $PGV_{x_{100}}$ wrt. an image mask M covering the eyes of the sample. In $PGV_{x_{950}}$ we can see that the relevant pixels from the previous samples match the area of the eyes also. In other words, the information is drawn from the same image area. $PGV_{x_{700}}$ has already expanded beyond the eyes and covers the right side of the face. At $PGV_{x_{600}}$ we can then see that the gradient volume has expanded to an area roughly matching the entire face. Finally, at $PGV_{x_{100}}$, the gradient volume has expanded significantly, meaning nearly all pixels in the previous sample contribute to the area of image mask M.

We see a similar pattern in both the gradient volume and the gradient maps. In the earlier time steps, the relevant pixels for the gradient volume and maps are restricted to the image area of M. At $t = 600$, we notice that $PGV_{x_{600}}$ matches the general outline of the face in the final sample. This means all pixels within the facial area of sample x_{601} are relevant for the area of the eyes in sample x_{600}. This could suggest that at some point in the generative process, the model draws from conceptual knowledge such as where the eyes of a person belong in a face.

However, Hong et al. [41] found that self-attention in diffusion models significantly overlaps with high-frequency details of the image, such as texture and edges. $PGV_{x_{600}}$ matching the facial shape could therefore also be traced to image details such as eyes, skin tone, and texture activating together instead of a conceptual understanding existing within the diffusion model.

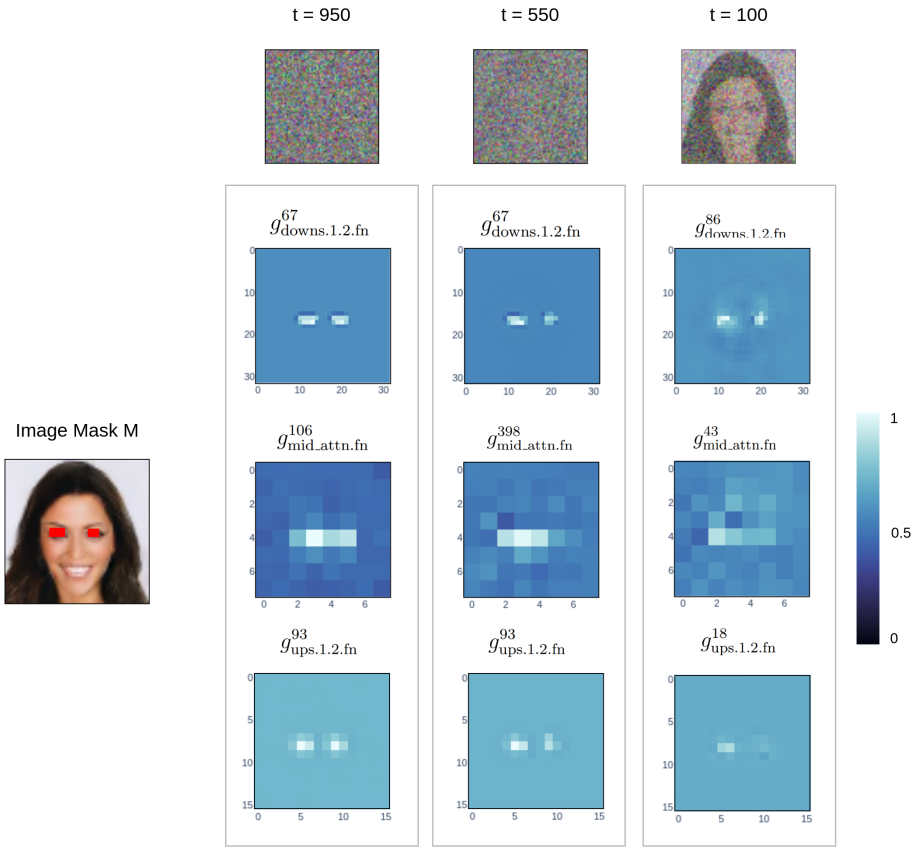


Figure 5.9: Maximum channels of the gradient maps $g^{c^*}_{\text{downs.1.2.fn}}$, $g^{c^*}_{\text{mid\_attn.fn}}$ and $g^{c^*}_{\text{ups.1.2.fn}}$ wrt. an image mask M covering the eyes of the sample 5.3 for three different time steps. We see that the most relevant pixels of the previous sample correspond to the selected image area. While the general localization of the highest contribution is constant, the gradient map diffuses in the later time step. This is especially visible for $g^{86}_{\text{downs.1.2.fn}}$, where we can see the facial outline in the gradient map. This result is consistent with the gradient volume, which also diffuses the localization of the gradient during the later time steps.

In the gradient maps, we also see high values being restricted to the image mask M for $t = 950$ and $t = 550$. Yet, for both the gradient volume and maps we see the relevant image area disperse at $t = 100$. The gradient map $g^{86}_{\text{downs.1.2.fn}}$ disperses slightly and we can see a faint facial outline in the gradient map. For $PGV_{x_{100}}$ this is even more prominent, as the gradient volume covers almost the entirety of the image area. This means almost all pixels in the previous sample are relevant for the

eyes of sample x_{100}. A hypothesis is that during the denoiser mode of the diffusion model [21], all pixels are relevant for the image reconstruction.

5.2.2 Matching Concepts to Channels

Rather than extracting interpretable objects as done in the previous section, we now modify the diffusion model and observe changes in the final sample x_0. As introduced in unit ablation (Section 4.1.3), we individually select $a_l^{c^*}$ and set its output to zero, thereby ablating the channel.

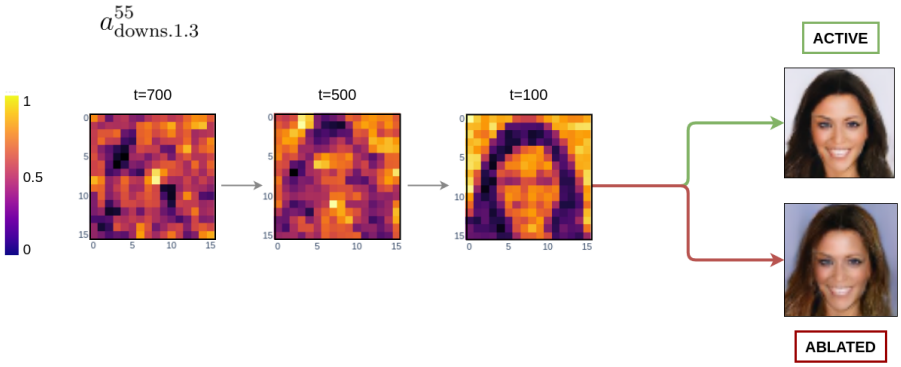


Figure 5.10: The figure depicts the maximum channel $a^{55}_{downs.1.3.}$ for activations in the second downsampling block, at time steps 700, 500, and 100. When masking $a^{55}_{downs.1.3.}$ the sample changes in skin tone, background, and hair color.

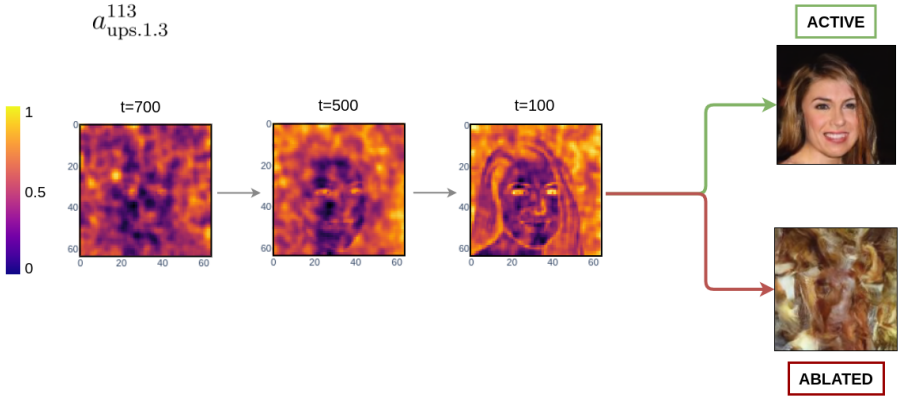


Figure 5.11: The figure depicts the maximum channel $a^{113}_{ups.1.3}$ for activations in the second upsampling block, at time steps 700, 500, and 100. When masking $a^{113}_{ups.1.3}$, the entire structure of the sample is destroyed.

We can match $a^{55}_{\text{downs.1.3}}$ to image details such as background, skin, and hair color. Rather than identifying channels that only activate highly for concepts such as eyes or the mouth, we found that generally, channels activate for large image

areas such as the background or the entire face. As we were not able to find channels with separable concepts, it means we cannot ablate singular concepts. This is seen when ablating $a^{113}_{\text{ups.1.3}}$, which activates highly for the eyes and the background also. Ablating the channel results in destroying the entire structure of the image, as human-understandable concepts are too intertwined in the diffusion model's representations.

Layer l	Max Channel c^*	SSIM
"downs.0.3"	41	**0.122**
"downs.1.3"	21	0.433
"downs.2.3"	27	0.63
"downs.3.3"	333	0.928
"mid_block2.res_conv"	333	0.987
"ups.0.3"	61	0.994
"ups.1.3"	91	0.879
"ups.2.3"	5	0.993

Table 5.1: We ablate $a_l^{c^*}$ for every final layer in the network and measure the change using the SSIM. c^* was calculated using a binary image mask M for the facial area of the sample at $t = 200$.

Layer l	Max Channel c^*	SSIM
"downs.0.2.fn"	12	**0.92**
"downs.1.2.fn"	47	0.985
"downs.2.2.fn"	140	0.998
"downs.3.2.fn"	149	0.998
"mid_attn.fn"	26	0.997
"ups.0.2.fn"	6	0.998
"ups.1.2.fn"	106	0.998
"ups.2.2.fn"	1	0.957

Table 5.2: We ablate $a_l^{c^*}$ for every attention layer in the network and measure the change using the SSIM. c^* was calculated using a binary image mask M for the facial area of the sample at $t = 200$.

While we were not successful at matching structural concepts to individual channels, we can observe the impact of channel ablation depending on resolution and layer

type. The results in Tables 5.1 and 5.2 show that ablating channels in the output layer impact the final sample more than ablating in attention layers. Furthermore, ablating in the contracting path and at a higher resolution causes a larger change in SSIM than ablating in the bottleneck.

5.3 Higher-Level Analysis

In this section, we demonstrate two use cases for higher-level analysis: determining the origin of artifacts and analyzing the predetermination of the diffusion process. For the second use-case, we compare two different diffusion profiles, unlike in the previous sections, where we extracted all information from a single profile.

5.3.1 Identifying the Origin of Artifacts

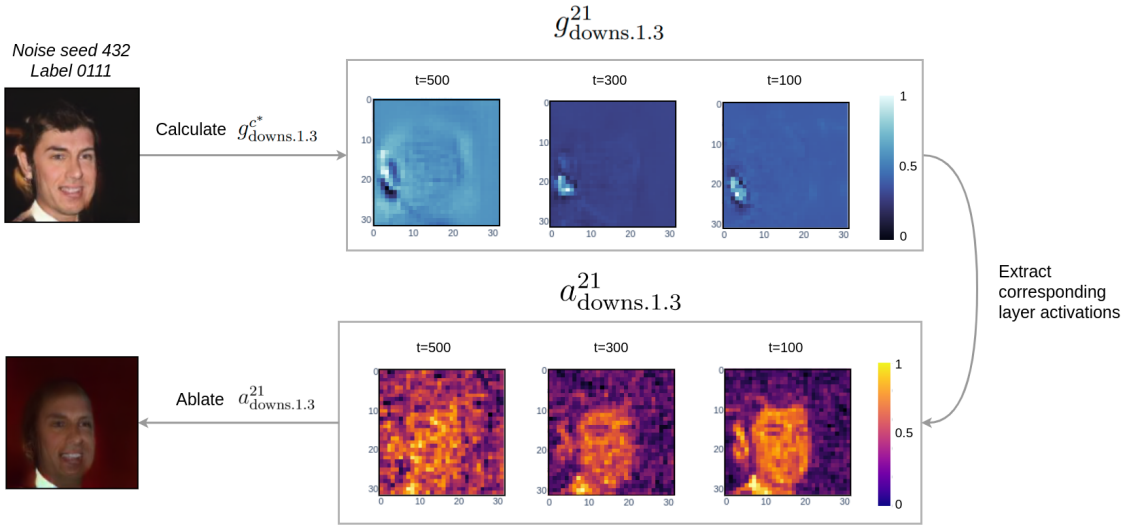


Figure 5.12: In this example, we try to localize a channel that highly contributes to the artifact, ablate the channel and observe the change in the output. We first calculate $g^{c^*}_{\text{downs.1.3}}$ wrt. the artifact area close to the man's ear, resulting in $c^* = 21$. Inspecting the gradient maps $g^{21}_{\text{downs.1.3}}$ in the second half of the diffusion process, we indeed see that the channel highly contributes to the artifact area. However, in the activation maps for the channel $a^{21}_{\text{downs.1.3}}$, we do see that the channel highly activates for the facial area also. As we are not able to separate the artifact from other image elements, the ablated image shows a significant loss in detail and color together with removing the artifact.

An issue with image generative models is the emergence of artifacts or unwanted structures. Using our diffusion interface, we try to detect the origin of an artifact using the example in Figure 5.12. We sample from the 128 model with noise seed $\epsilon = 432$ and label 0111 - a man with no bangs, but smiling and wearing lipstick. The

resulting image is fairly accurate, but we can see an artifact close to the man's ear. To localize where the artifact might have emerged, we calculate $g^{c^*}_{\text{downs.1.3}}$ with M covering the artifact image area at $t = 200$, resulting in $c = 21$. The layer is chosen as it typically has a significant impact on the output (Table 5.1).

We then ablate the corresponding activations $a^{21}_{\text{downs.1.3}}$ to inspect the channels impact on the final sample and observe a significant change. Indeed, we were successful at removing the artifact by ablating the channel, yet it also caused the removal of details and a darkening of the skin tone. This relationship is also visible in $a^{21}_{\text{downs.1.3}}$, which activates highly for the areas matching the artifact and the face together. Similarly to matching concepts to channels (Section 5.2.2), we are not successful at removing specific elements of the image by ablating a single channel. We are, however, able to easily identify highly contributing channels to specified parts of the image.

5.3.2 Label Divergence and Predetermination

In this final part of the experiments, we want to determine if there is a predetermination to the generative process. We have seen that structures emerge earlier in the layers of the network than in the samples, but we do not know how much they contribute to the final sample x_0. To test how early the final image is predetermined, we compare the diffusion profiles of two samples with the same noise seed $\epsilon = 4363$ but different labels $y = 0110$ and $y = 0111$ (See Figure 5.13). We then measure the point of divergence in different objects of interest.

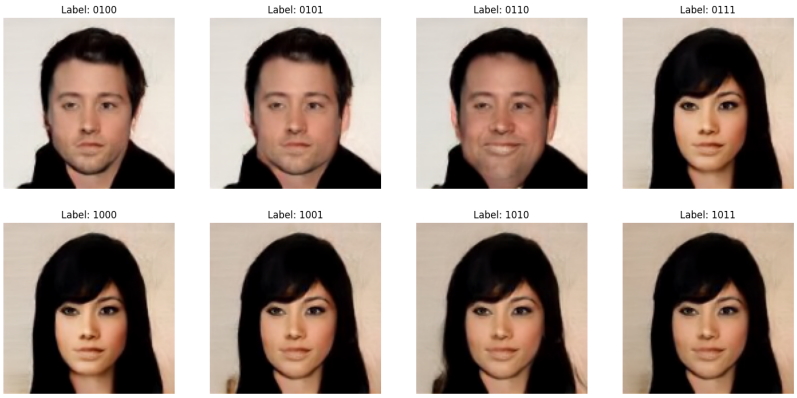


Figure 5.13: A subset of samples with noise seed $\epsilon = 4363$ from the 128 model. We notice that the output for label 0111 is wrong. Instead of depicting a smiling man with lipstick, the sample depicts a woman with bangs, almost identical to label 1000. This is not a general model error, as we can generate samples with label 0111 that are correct for different noise seeds (Figure 5.12).

Firstly, we analyse how the SSIM between both samples changes over time. This gives a first intuition for when the samples start to diverge. Indeed, as seen in Figure

5.14, the samples are almost identical for the first 200 steps of the generative process. The similarity between the samples decreases until a plateau around $t = 400$. For the last 100 steps, there is a quick decrease in similarity. We want to ensure this pattern generally holds for the inter-comparison of SSIM trajectories. Therefore, we repeatedly calculate the SSIM trajectory between samples with label 0110 and 0111 but with random noise seeds ϵ.

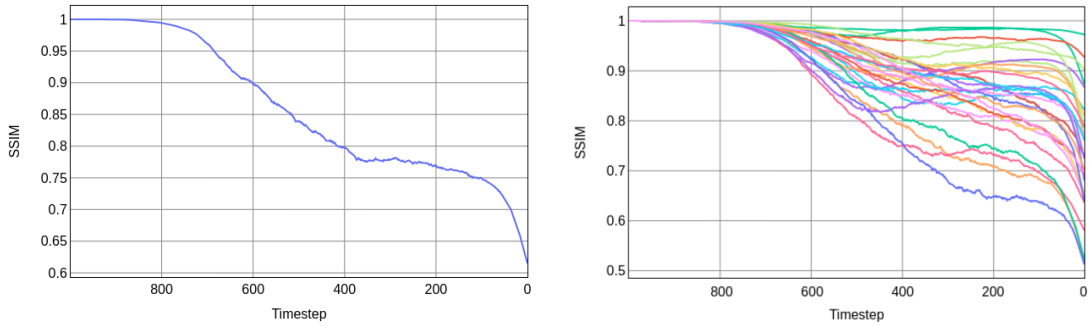


Figure 5.14: (Left) Individual SSIM trajectory between labels 0110 and 0111. The trajectory shows that the samples are identical up until $t = 800$ and then gradually diverge until a plateau at $t = 400$ which lasts for approximately 300 steps. For the last 100 steps in the diffusion process, the similarity between both samples decreases fast. (Right) The same pattern can be seen when calculating the SSIM for 30 random noise seeds with the same labels. Even though the final amount of similarity differs, the divergence for all trajectories beings around $t = 800$ and includes a sudden decrease in similarity around $t = 100$, relative to the final SSIM.

While the final SSIM value varies between the random trajectories, the pattern is the same. The divergence initiates around time step 800. The trajectory then either reaches a plateau or evenly decreases until the last 100 steps, after which the SSIM drops off. This matches the findings in Section 5.2.1 and the diffusion characterization by Deja et al. [21], which hypothesized that the last steps constitute the denoiser. The final emergence of the sample is initiated by the removal of the remaining noise, thereby decreasing the similarity between different labels. Additionally, we calculate the SSIM trajectory for a different label combination, namely $y = 0000$ and $y = 01000$. The results (Figure D.1) follow the same pattern, although the final SSIM value tends to be lower.

To further inspect when the samples start to diverge, we extract the attention maps A_t (Section 4.1.2) for both samples during the generative process. As seen in Figure 5.15, the attention maps for both labels are fairly similar until $t = 800$. The Euclidean distance between the attention maps increases significantly at $t = 700$, around when we can also see the distinct outlines of the final samples.

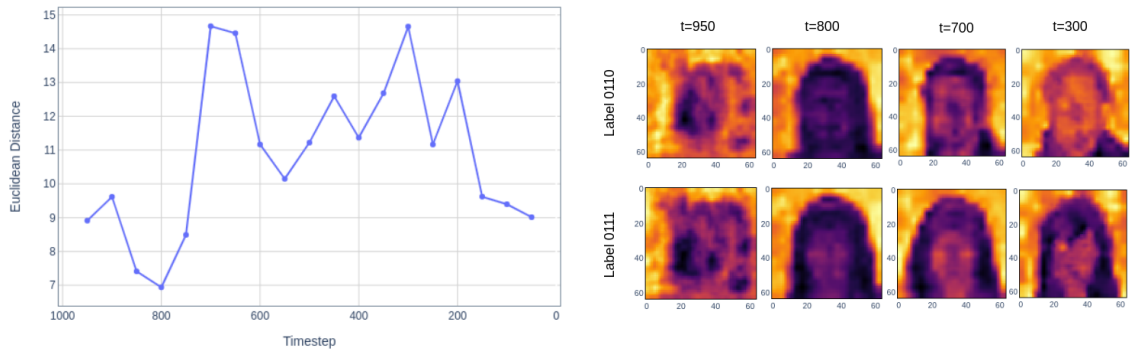


Figure 5.15: (Left) The Euclidean distance between A_t for labels 0110 and 0111 is sampled at every 50th step of the reverse process. (Right) Attention maps A_t for both labels at four time steps of interest. A_{950} and A_{800} are visually fairly similar for both labels. At $t = 700$ we already see the difference in hairstyles; short hair for label 0110 and long hair for label 0111. The outline remains at $t = 300$.

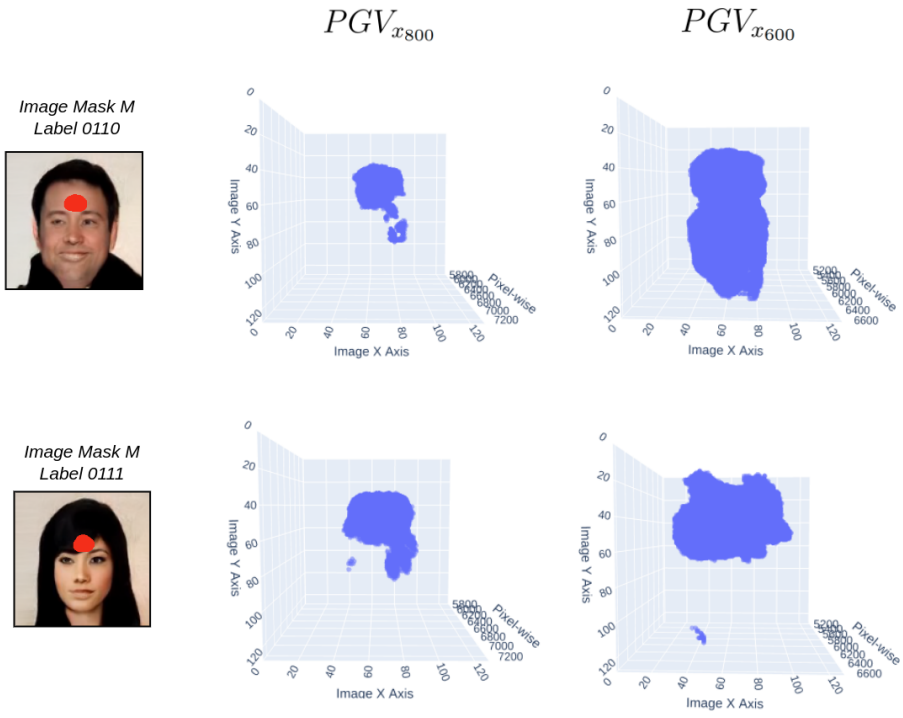


Figure 5.16: For both labels $PGV_{x_{800}}$ is localized in the neighborhood of the mask M. As we have seen in Figure 5.8, the gradient volumes disperse during the generative process. The same can be seen in $PGV_{x_{600}}$, where for label 0110 the volume covers the central image area, matching the man's face. For label 0111, however, the gradient volume is restricted to the upper half of the image area, matching the larger forehead of the woman.

The last object of interest we use to inspect the divergence between labels is the gradient volume PGV_{x_t}. We calculate $PGV_{x_{800}}$ and $PGV_{x_{600}}$ for both labels wrt. the forehead area (Figure 5.16). Both gradient volumes $PGV_{x_{800}}$ are fairly localized, matching the wider location of the image mask M. At $t = 600$, however, we see the gradient volumes disperse. $PGV_{x_{600}}$ for label 0110 takes the shape of the face, while the gradient volume for label 0111 matches the larger forehead area. A hypothesis for this change is again that pixel values of similar colors overlap (Section 5.2.1). Therefore, the forehead of the man would draw relevant information from the other areas of the image with the same skin tone and the woman's forehead from the are of the bangs, which are also black hair.

We have seen that the divergence between objects of the diffusion profile starts as early as $t = 800$. This begs the question of what functionality the first 200 steps pose for the generative process. Based on the intuition that the first 200-300 steps of the generative process could already significantly determine the final sample, we propose our final experiment called label switching.

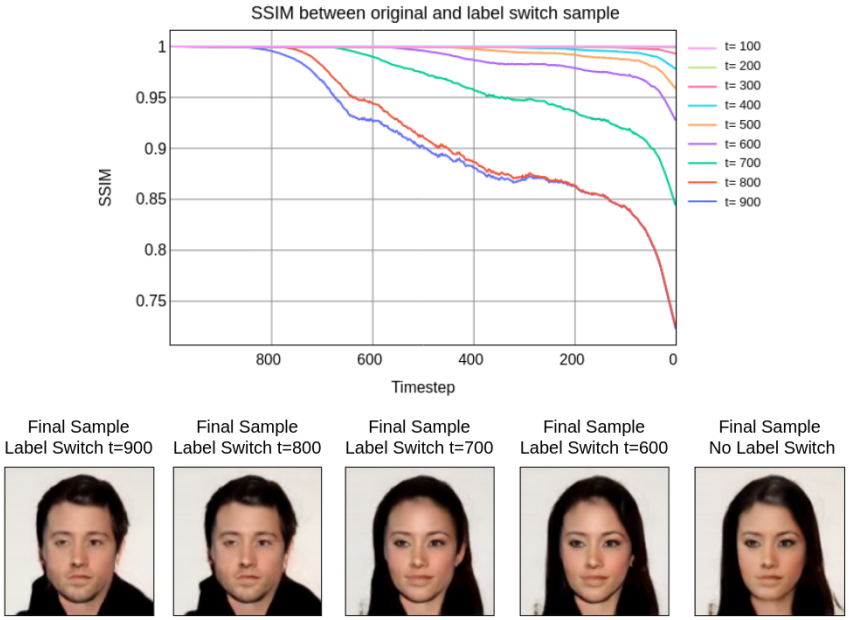


Figure 5.17: Switching from original label 0000 to label 0100 at different time steps. Switching conditioning labels within the first 200 time steps results in the final sample more closely resembling label 0100. With $t = 700$ as the swichting time step, the final sample has more similar facial features and hairstyle to the original sample, but the SSIM of 0.85 still indicates less clearly visible dissimilarities. Switching the labels after 400 time steps of the generative process results in at least 0.9 SSIM similarity, with progressively more similarity the later the switch occurs.

The generative process starts with a label y_1 and at different time steps throughout the process, we swap the conditioning to a new label y_2. We then measure

the difference between the original sample and the label-switch sample using the SSIM. In the example in Figure 5.17, we perform label switching for $y_1 = 0000$ and $y_2 = 0100$. We see that switching labels within the first 300 time steps causes the sample to match label y_2 more closely than y_1. Label switching in the last 100 time steps barely affects the final sample.

Repeating label switching with different labels $y_1 = 0100$ and $y_2 = 1000$ (Appendix D.2) shows that the portion of time steps which predetermine the final sample is fluid and can be later than 300 steps. Yet the observation remains, that after about half of the generative process, changing the conditioning label has less impact on the final sample. This further indicates that in the later portion of the diffusion process the generative functionality is mostly completed (Section 5.2.1).

Chapter 6

Conclusion

The goal of this thesis was to develop explainability methods for diffusion models and build an interface to demonstrate their applicability. The following contributions were made:

1. We identified and implemented three types of explainability methods for diffusion models: Firstly, gradient-based methods such as the novel concept of a gradient volume PGV_{x_t} and gradient maps g_{l_t}. Secondly, activation-based methods such as activation a_{l_t} and attention maps A_t. And lastly, we implemented channel ablation for diffusion models, including the identification of a layer's maximum channel c^*.

2. We combined the proposed explainability methods, measures, conditioning, and the time steps of the diffusion process into a compact primitive object for standardized inter- and intra-object comparison. This object, called the diffusion profile, is the foundation for the final interface, which was newly designed and implemented, extending the base implementation of the diffusion model.

3. Using the interface we observed a distinct emergence of structure, shared among all samples of the diffusion model. The unique trajectory was uncovered using the SSIM between samples over the generative process.

4. We inspected the gradient volume PGV_{x_t} for a masked sample during the generative process. In the beginning of the generative process, we found it to be highly localized to the image mask M. Yet, after 300 time steps the gradient volume spread out to cover first the facial area of the sample and then the entire image region during the last 100 time steps. This suggests that later during the generative process all pixels of a previous sample are relevant, even for just a subsection of the current sample (Figure 5.8).

5. We were not able to match semantic concepts to individual channels of the diffusion models. However, using channel ablation we could show the interplay between high-frequency details such as hair color or skin tone. We can correctly

identify the channels which highly contribute to an artifact, even though the artifact itself is too entangled with the rest of the image to be removed by channel ablation only (Figure 5.12).

6. Based on the SSIM trajectory and the visualization of attention maps, we hypothesized that the structure of the final output emerges earlier than in the samples. We found further evidence for this hypothesis by testing the predetermination of the generated samples through label switching at different time steps during the generative process.

For future work, we would like to extend the applicability of the XAI interface beyond the diffusion architecture used by Ho et al. [36]. Since its release in 2020, numerous proposals have been made to improve the core architecture, most notably latent diffusion [79] in summer 2022. Latent diffusion applies diffusion models to the latent space of pre-trained autoencoders, increasing visual fidelity, incorporating multi-modal conditioning and reducing complexity. It would be desirable to extend the XAI interface to include latent diffusion models. Incorporating their multi-modal conditioning could enable a less restricted analysis of diffusion models than the label pre-selection the current interface is relies on. Furthermore, as we are already operating in latent space, it would be a sensible starting point to explore latent space discovery with diffusion models (Section 2.3.3) and extending the diffusion profile by another explainability method.

Another improvement in the future could be including methods to reduce the number of time steps needed during sampling. Progressive distillation [88] distills a slow teacher diffusion model into a faster student model with half the step size. By repeating the distillation and halving the time steps every time, it is possible to sample images with as few as four time steps without losing much perceptual quality. Using a distilled diffusion model would allow us to extract the diffusion profile for every time step, instead of hand-picking a subset of time steps. The predetermination and conditioning of diffusion models (Section 5.3.2) could further be explored to compress and shorten the generative process.

Bibliography

[1] Abubakar Abid et al. *Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild*. 2019. arXiv: 1906.02569 [cs.LG].

[2] Reduan Achtibat et al. *From "Where" to "What": Towards Human-Understandable Explanations through Concept Relevance Propagation*. 2022. arXiv: 2206.03208 [cs.LG].

[3] Amina Adadi and Mohammed Berrada. "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)". In: *IEEE Access* 6 (2018), pp. 52138–52160. DOI: 10.1109/ACCESS.2018.2870052.

[4] Marco Ancona et al. *Towards better understanding of gradient-based attribution methods for Deep Neural Networks*. 2018. arXiv: 1711.06104 [cs.LG].

[5] Alejandro Barredo Arrieta et al. *Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI*. 2019. arXiv: 1910.10045 [cs.AI].

[6] Omri Avrahami, Dani Lischinski, and Ohad Fried. *Blended Diffusion for Text-driven Editing of Natural Images*. 2022. arXiv: 2111.14818 [cs.CV].

[7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[8] Sebastian Bach et al. "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation". In: *PLOS ONE* 10.7 (July 2015), pp. 1–46. DOI: 10.1371/journal.pone.0130140.

[9] David Baehrens et al. *How to Explain Individual Classification Decisions*. 2009. arXiv: 0912.1128 [stat.ML].

[10] David Bau et al. *GAN Dissection: Visualizing and Understanding Generative Adversarial Networks*. 2018. arXiv: 1811.10597 [cs.CV].

[11] David Bau et al. "Understanding the role of individual units in a deep neural network". In: *Proceedings of the National Academy of Sciences* 117.48 (Sept. 2020), pp. 30071–30078. DOI: 10.1073/pnas.1907375117.

[12] Eyal Betzalel et al. *A Study on the Evaluation of Generative Models*. 2022. arXiv: 2206.10935 [cs.LG].

[13] Jackson Borchardt and Saul Kato. *Implicitization of Biquadratic Bézier Triangle and Quadrilateral Surfaces*. 2023. arXiv: 2301.01318 [cs.CG].

[14] Leo Breiman. "Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author)". In: *Statistical Science* 16 (Aug. 2001). DOI: 10.1214/ss/1009213726.

[15] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[16] Alberto Cavallo. *On Bennequin type inequalities for links in tight contact 3-manifolds*. 2020. arXiv: 1801.00614 [math.GT].

[17] Xi Chen et al. *InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets*. 2016. arXiv: 1606.03657 [cs.LG].

[18] Zhi Chen, Yijie Bei, and Cynthia Rudin. "Concept whitening for interpretable image recognition". In: *Nature Machine Intelligence* 2.12 (Dec. 2020), pp. 772–782. DOI: 10.1038/s42256-020-00265-z.

[19] Jianpeng Cheng, Li Dong, and Mirella Lapata. *Long Short-Term Memory-Networks for Machine Reading*. 2016. arXiv: 1601.06733 [cs.CL].

[20] Ayan Das. *An introduction to Diffusion Probabilistic Models*. Dec. 2021. URL: https://ayandas.me/blog-tut/2021/12/04/diffusion-prob-models.html (visited on 02/24/2023).

[21] Kamil Deja et al. *On Analyzing Generative and Denoising Capabilities of Diffusion-based Deep Generative Models*. 2022. arXiv: 2206.00070 [cs.LG].

[22] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[23] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[24] Prafulla Dhariwal and Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. 2021. arXiv: 2105.05233 [cs.LG].

[25] Yilun Du and Igor Mordatch. *Implicit Generation and Generalization in Energy-Based Models*. 2020. arXiv: 1903.08689 [cs.LG].

[26] Amirata Ghorbani et al. *Towards Automatic Concept-based Explanations*. 2019. arXiv: 1902.03129 [stat.ML].

[27] Judy Wawira Gichoya et al. "AI recognition of patient race in medical imaging: a modelling study". In: *The Lancet Digital Health* 4.6 (June 2022), e406–e414. DOI: 10.1016/s2589-7500(22)00063-2.

[28] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[29] David Gunning and David Aha. "DARPA's Explainable Artificial Intelligence (XAI) Program". In: *AI Magazine* 40.2 (June 2019), pp. 44–58. DOI: 10.1609/aimag.v40i2.2850.

[30] Meng-Hao Guo et al. "Attention mechanisms in computer vision: A survey". In: *Computational Visual Media* 8.3 (Mar. 2022), pp. 331–368. DOI: 10.1007/s41095-022-0271-y.

[31] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[32] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2020. arXiv: 1606.08415 [cs.LG].

[33] Matteo Hessel et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017. arXiv: 1710.02298 [cs.AI].

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

[35] Irina Higgins et al. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning Representations*. 2016.

[36] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG].

[37] Jonathan Ho and Tim Salimans. *Classifier-Free Diffusion Guidance*. 2022. arXiv: 2207.12598 [cs.LG].

[38] Jonathan Ho et al. *Cascaded Diffusion Models for High Fidelity Image Generation*. 2021. arXiv: 2106.15282 [cs.CV].

[39] Jonathan Ho et al. *Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design*. 2019. arXiv: 1902.00275 [cs.LG].

[40] Jonathan Ho et al. *Video Diffusion Models*. 2022. arXiv: 2204.03458 [cs.CV].

[41] Susung Hong et al. *Improving Sample Quality of Diffusion Models Using Self-Attention Guidance*. 2023. arXiv: 2210.00939 [cs.CV].

[42] D. Hubel and T. Wiesel. "Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex". In: *Journal of Physiology* 160 (1962), pp. 106–154.

[43] Plotly Technologies Inc. *Collaborative data science*. 2015. URL: https://plot.ly.

[44] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[45] Yifan Jiang, Shiyu Chang, and Zhangyang Wang. *TransGAN: Two Pure Transformers Can Make One Strong GAN, and That Can Scale Up*. 2021. arXiv: 2102.07074 [cs.CV].

[46] Andrej Karpathy et al. *Generative Models*. 2016. URL: https://openai.com/blog/generative-models/ (visited on 02/16/2023).

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

[48] Been Kim et al. *Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV)*. 2018. arXiv: 1711.11279 [stat.ML].

[49] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. arXiv: 1312.6114 [stat.ML].

[50] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trends in Machine Learning* 12.4 (2019), pp. 307–392. DOI: 10.1561/2200000056.

[51] Alina Köchling and Marius Claus Wehner. "Discriminated by an algorithm: a systematic review of discrimination and fairness by algorithmic decision-making in the context of HR recruitment and HR development". In: *Business Research* 13.3 (Nov. 2020), pp. 795–848. ISSN: 2198-2627. DOI: 10.1007/s40685-020-00134-w.

[52] B. Kröse and P. Van der Smagt. *An Introduction to Neural Networks*. Amsterdam, Netherlands: University of Amsterdam, 1996.

[53] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1 (Mar. 1951), pp. 79–86. DOI: 10.1214/aoms/1177729694.

[54] P Langevin. *Sur la theorie du mouvement brownien.* CR Acad. Sci. Paris, 1908, 146(530–533).

[55] Yann Lecun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86 (Dec. 1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[56] Seppo Linnainmaa. *Taylor Expansion of the Accumulated Rounding Error*. Vol. 16. 2. USA: BIT Computer Science and Numerical Mathematics, June 1976, pp. 146–160. DOI: 10.1007/BF01931367.

[57] Shusen Liu et al. *Generative Counterfactual Introspection for Explainable Deep Learning*. 2019. arXiv: 1907.03077 [cs.LG].

[58] Wenqian Liu et al. *Towards Visually Explaining Variational Autoencoders*. 2020. arXiv: 1911.07389 [cs.CV].

[59] Zhuang Liu et al. *A ConvNet for the 2020s*. 2022. arXiv: 2201.03545 [cs.CV].

[60] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)* (Dec. 2015).

[61] Andreas Lugmayr et al. *RePaint: Inpainting using Denoising Diffusion Probabilistic Models*. 2022. arXiv: 2201.09865 [cs.CV].

[62] Troy Luhman and Eric Luhman. *Improving Diffusion Model Efficiency Through Patching*. 2022. arXiv: 2207.04316 [cs.LG].

[63] Scott Lundberg and Su-In Lee. *A Unified Approach to Interpreting Model Predictions*. 2017. arXiv: 1705.07874 [cs.AI].

[64] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[65] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[66] Eric Nalisnick et al. *Do Deep Generative Models Know What They Don't Know?* 2019. arXiv: 1810.09136 [stat.ML].

[67] Radford M. Neal. *Annealed Importance Sampling*. 1998. arXiv: physics/9803008 [physics.comp-ph].

[68] Anh Nguyen, Jason Yosinski, and Jeff Clune. *Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks*. 2016. arXiv: 1602.03616 [cs.NE].

[69] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. "Feature Visualization". In: *Distill* (2017). https://distill.pub/2017/feature-visualization. DOI: 10.23915/distill.00007.

[70] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. *Pixel Recurrent Neural Networks*. 2016. arXiv: 1601.06759 [cs.CV].

[71] Aaron van den Oord et al. *Conditional Image Generation with PixelCNN Decoders*. 2016. arXiv: 1606.05328 [cs.CV].

[72] Aaron van den Oord et al. *WaveNet: A Generative Model for Raw Audio*. 2016. arXiv: 1609.03499 [cs.SD].

[73] Ankur P. Parikh et al. *A Decomposable Attention Model for Natural Language Inference*. 2016. arXiv: 1606.01933 [cs.CL].

[74] Walter H. L. Pinaya et al. *Brain Imaging Generation with Latent Diffusion Models*. 2022. arXiv: 2209.07162 [eess.IV].

[75] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[76] Aditya Ramesh et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. arXiv: 2204.06125 [cs.CV].

[77] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. 2016. arXiv: 1602.04938 [cs.LG].

[78] Niels Rogge and Kashif Rasul. *The Annotated Diffusion Model*. 2022. URL: https://huggingface.co/blog/annotated-diffusion (visited on 10/05/2022).

[79] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. arXiv: 2112.10752 [cs.CV].

[80] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[81] Kevin Roose. *An A.I.-Generated Picture Won an Art Prize. Artists Aren't Happy*. Sept. 2, 2022. URL: https://www.nytimes.com/2022/09/02/technology/ai-artificial-intelligence-artists.html (visited on 03/17/2023).

[82] F. Rosenblatt. *The perceptron - A perceiving and recognizing automaton*. 85-460-1. Ithaca, New York: Cornell Aeronautical Laboratory, Jan. 1957.

[83] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Neurocomputing: Foundations of Research* (1988), pp. 696–699.

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

[85] Lars Ruthotto and Eldad Haber. *An Introduction to Deep Generative Modeling*. 2021. arXiv: 2103.05180 [cs.LG].

[86] Chitwan Saharia et al. *Image Super-Resolution via Iterative Refinement*. 2021. arXiv: 2104.07636 [eess.IV].

[87] Chitwan Saharia et al. *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*. 2022. arXiv: 2205.11487 [cs.CV].

[88] Tim Salimans and Jonathan Ho. *Progressive Distillation for Fast Sampling of Diffusion Models*. 2022. arXiv: 2202.00512 [cs.LG].

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

[90] Umme Sara, Morium Akter, and Mohammad Shorif Uddin. "Image Quality Assessment through FSIM, SSIM, MSE and PSNR—A Comparative Study". en. In: *J. Comput. Commun.* 07.03 (Mar. 2019), pp. 8–18.

[91] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

[92] Gesina Schwalbe and Bettina Finzel. "A comprehensive taxonomy for explainable artificial intelligence: a systematic survey of surveys on methods and concepts". In: *Data Mining and Knowledge Discovery* (Jan. 2023). DOI: 10.1007/s10618-022-00867-8. URL: https://doi.org/10.10072Fs10618-022-00867-8.

[93] Zhuoran Shen et al. *Efficient Attention: Attention with Linear Complexities*. 2020. arXiv: 1812.01243 [cs.CV].

[94] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. *Learning Important Features Through Propagating Activation Differences*. 2019. arXiv: 1704.02685 [cs.CV].

[95] Avanti Shrikumar et al. *Not Just a Black Box: Learning Important Features Through Propagating Activation Differences*. 2017. arXiv: 1605.01713 [cs.LG].

[96] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[97] Jascha Sohl-Dickstein et al. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015. arXiv: 1503.03585 [cs.LG].

[98] Yang Song and Stefano Ermon. *Generative Modeling by Estimating Gradients of the Data Distribution*. 2020. arXiv: 1907.05600 [cs.LG].

[99] Yang Song et al. *Solving Inverse Problems in Medical Imaging with Score-Based Generative Models*. 2022. arXiv: 2111.08005 [eess.IV].

[100] Jost Tobias Springenberg et al. *Striving for Simplicity: The All Convolutional Net*. 2015. arXiv: 1412.6806 [cs.LG].

[101] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. *Axiomatic Attribution for Deep Networks*. 2017. arXiv: 1703.01365 [cs.LG].

[102] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[103] Christian Szegedy et al. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016. arXiv: 1602.07261 [cs.CV].

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

[105] Raphael Tang et al. *What the DAAM: Interpreting Stable Diffusion Using Cross Attention*. 2022. arXiv: 2210.04885 [cs.CV].

[106] Lucas Theis, Aäron van den Oord, and Matthias Bethge. *A note on the evaluation of generative models*. 2016. arXiv: 1511.01844 [stat.ML].

[107] J.M. Tomczak. *Deep Generative Modeling*. Springer International Publishing, 2022. ISBN: 9783030931575.

[108] Arash Vahdat and Jan Kautz. *NVAE: A Deep Hierarchical Variational Autoencoder*. 2021. arXiv: 2007.03898 [stat.ML].

[109] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[110] James Vincent. *Getty Images is suing the creators of AI art tool Stable Diffusion for scraping its content*. Jan. 17, 2023. URL: https://www.theverge.com/2023/1/17/23558516/ai-art-copyright-stable-diffusion-getty-images-lawsuit (visited on 03/17/2023).

[111] Andrey Voynov and Artem Babenko. *Unsupervised Discovery of Interpretable Directions in the GAN Latent Space*. 2020. arXiv: 2002.03754 [cs.LG].

[112] Sandra Wachter, Brent Mittelstadt, and Chris Russell. *Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR*. 2018. arXiv: 1711.00399 [cs.AI].

[113] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Trans. Image Process* 13.4 (Apr. 2004), pp. 600–612.

[114] Eric W. Weisstein. *Frobenius Norm*. URL: https://mathworld.wolfram.com/FrobeniusNorm.html (visited on 02/28/2023).

[115] Max Welling and Yee Whye Teh. "Bayesian Learning via Stochastic Gradient Langevin Dynamics". In: *Omnipress*. ICML'11 (2011), pp. 681–688.

[116] Lilian Weng. *From Autoencoder to Beta-VAE*. 2018. URL: https://lilianweng.github.io/posts/2018-08-12-vae/ (visited on 10/25/2022).

[117] Lilian Weng. *What are diffusion models?* 2021. URL: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/ (visited on 10/05/2022).

[118] Yuxin Wu and Kaiming He. *Group Normalization*. 2018. arXiv: 1803.08494 [cs.CV].

[119] Saining Xie et al. *Aggregated Residual Transformations for Deep Neural Networks*. 2017. arXiv: 1611.05431 [cs.CV].

[120] Sunil Yadav. *Generative Adversarial Networks*. 2022. URL: https://data-science-blog.com/blog/2022/05/20/generative-adversarial-networks/ (visited on 02/20/2023).

[121] Ceyuan Yang, Yujun Shen, and Bolei Zhou. *Semantic Hierarchy Emerges in Deep Generative Representations for Scene Synthesis*. 2020. arXiv: 1911.09267 [cs.CV].

[122] Chengliang Yang, Anand Rangarajan, and Sanjay Ranka. *Global Model Interpretation via Recursive Partitioning*. 2018. arXiv: 1802.04253 [cs.LG].

[123] Jiahui Yu et al. *CoCa: Contrastive Captioners are Image-Text Foundation Models*. 2022. arXiv: 2205.01917 [cs.CV].

[124] Matthew D Zeiler and Rob Fergus. *Visualizing and Understanding Convolutional Networks*. 2013. arXiv: 1311.2901 [cs.CV].

[125] Xiaohua Zhai et al. *Scaling Vision Transformers*. 2022. arXiv: 2106.04560 [cs.CV].

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

[127] Zhixing Zhang et al. *SINE: SINgle Image Editing with Text-to-Image Diffusion Models*. 2022. arXiv: 2212.04489 [cs.CV].

Appendix A

Interface Implementation

The interface and its components are implemented using two tools: Gradio and Plotly. Gradio [1] is an open-source library for building web applications, mainly for machine learning demonstrations. It provides the general structure and interactive elements needed for the interface. Generally, Gradio offers two ways of building web applications, through an interface class or a low-level API called "Blocks". The "Blocks" API is used in this thesis as it allows for more customizability and full control over a larger interface. The core user interface structure is based around the defined components (Section 4.2.1). Using Gradio, the components be arranged in rows and columns, grouped and toggled. This simplifies the user interface to avoid having all functionalities displayed at once. For each component, the necessary interactive elements are implemented. Gradio defines a variety of elements, but for this use case mainly the following are used:

- **gradio.Dropdown** boxes where the user can select model, blocks, and layers to analyze.
- **gradio.Number** for the numeric input/output field required for the sampling resolution, random noise seed, and masked channel.
- **gradio.Checkbox** allowing the selection of the binary conditioning labels.
- **gradio.Image** to display images and enable a drawing pad to define the image mask, required for activation/gradient masking.
- **gradio.Gallery** to display the generated samples along the 1000 time steps as a scrollable list. To reduce complexity, every 100th sample is rendered.
- **gradio.Plot** to render more complicated visualizations made using Plotly.

Plotly [43] is an interactive, open-source Python library to generate browser-based plots and charts. Plotly is used for elements of the interface that exceed the visualization capabilities of Gradio. Activation and gradient maps for one layer can have up to 512 different channels and are visualized using a slider along the channel

axis. The measures SSIM and Euclidean distance are visualized as line charts and the gradient volume as a 3D scatter plot.

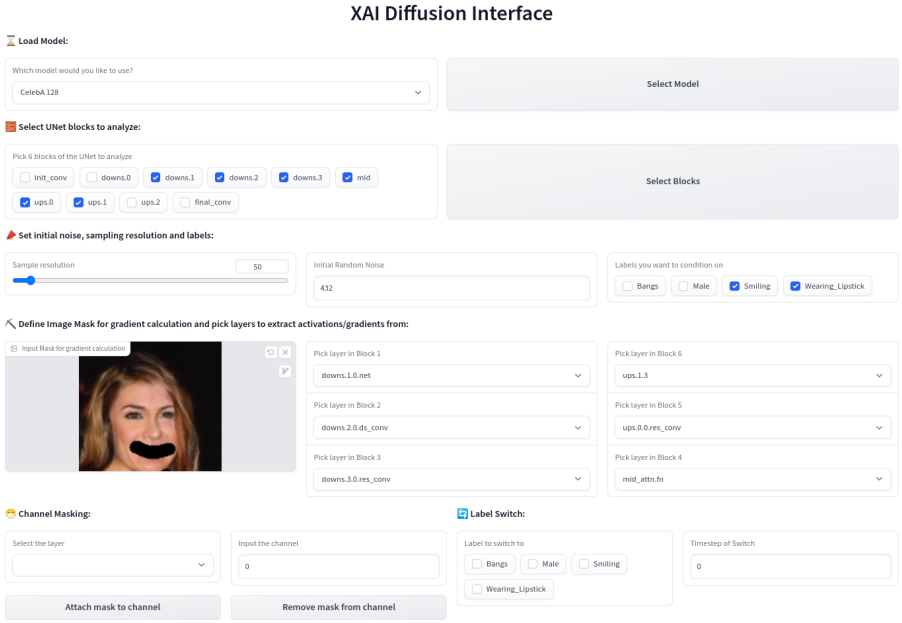


Figure A.1: The settings section of the finished interface allows the user to set parameters for sampling the diffusion profile.

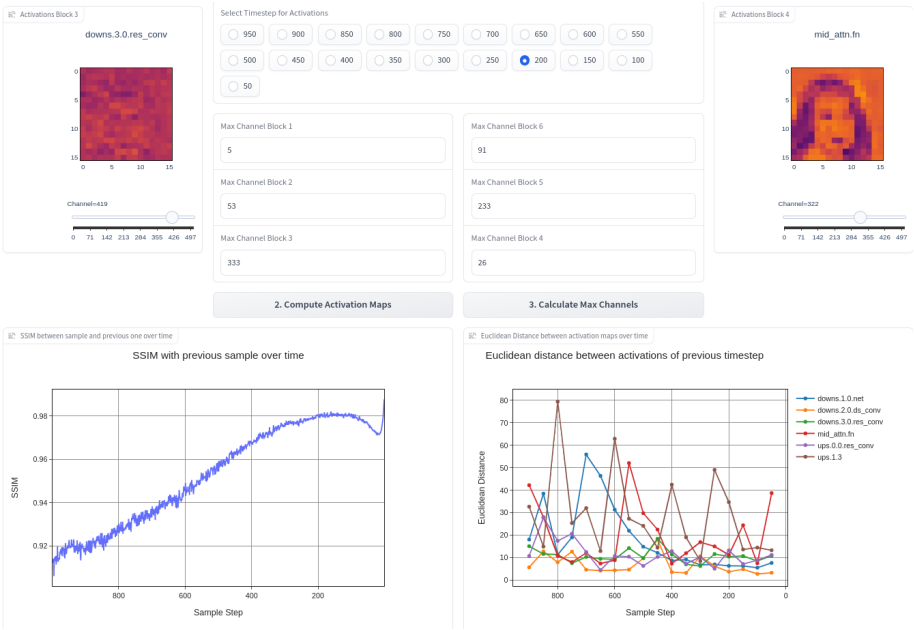


Figure A.2: The activation-based results of the finished interface, including activation maps, maximum channels, the SSIM trajectory and the layer-wise Euclidean distance.

Appendix B

Model Training

The following hyperparameters were used to train both diffusion models used for the experiments:

- The U-Net for both models is of dimension 64 for the initial convolution. The input for the lower resolution is $x_t \in \mathbb{R}^{4x64x64}$ and $x_t \in \mathbb{R}^{4x128x128}$ for the higher resolution. The network has four input channels, namely the three color channels of the image and one conditioning channel y. The output x_{t-1} only has three channels. The dimension multiplications are set to (1,2,4,8), defining the resolutional depth of the U-Net.

- For pre-processing, the training images were filtered for faulty data, center cropped and resized to the appropriate resolution of either 64x46 or 128x128.

- The 64 model was trained with a batch size of 32 for 51 epochs with approximately 94.000 steps in total. The learning rate was fixed at $2e^{-4}$ throughout training.

- The 128 model was trained with a batch size of 16 for 32 epochs with approximately 120.000 steps in total. The learning rate was fixed at $2e^{-4}$ throughout training.

- The variance β_t was determined by a linear beta schedule [36]. The error at the start of the reverse process should approximately be zero $L_T \approx 0$, with $\beta_T = 0.02$ and $\beta_1 = 0.0001$.

- The training progress was evaluated based on three metrics: L1 loss, SSIM, and PSNR (Figures B.1 - B.2). The L1 loss should be minimized, while high values for SSIM and PSNR indicate better image quality.

- We trained on an Amazon Web Services instance of type *g4dn.xlarge* with 1 GPU, 4 vCPU, 16 GB RAM and 125 GB SSD.

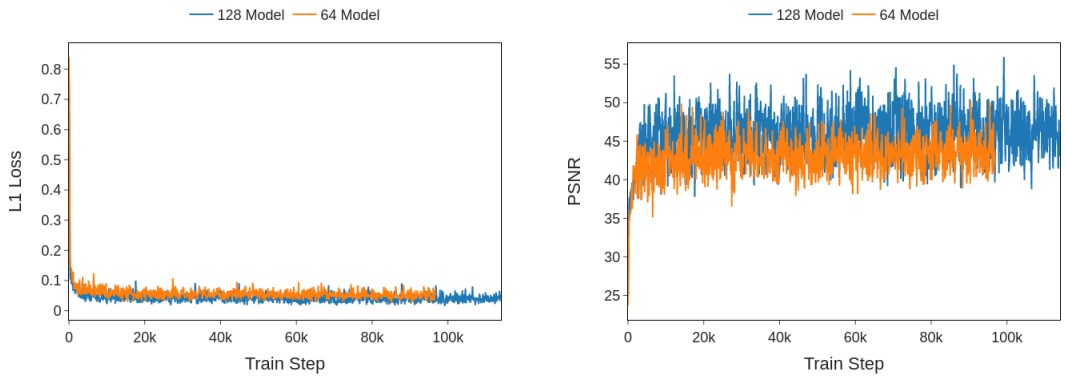


Figure B.1: L1 Loss and PSNR training performance for both the 64 and the 128 diffusion model.

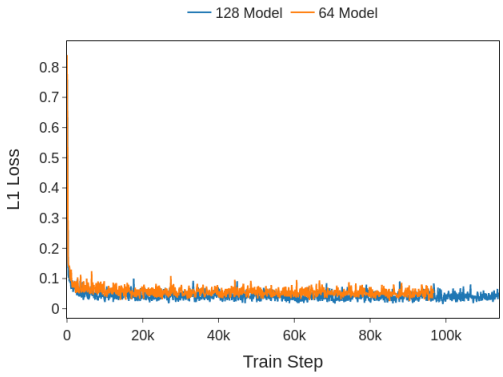


Figure B.2: SSIM between training image and produced sample during training for both the 64 and the 128 diffusion model.

Appendix C

PyTorch Implementation

At the time of writing the thesis, the code for the adapted diffusion model and the interface is currently being prepared to be made publicly available. The following are three relevant parts of the implementation for a demonstration:

```
1
2  class AblateChannelHook():
3      def __init__(self, layer, channel = 0):
4          self.channel = channel
5          self.hook = layer.register_forward_hook(self.hook_fn)
6
7      def hook_fn(self, layer, input, output):
8          #set output of channel to zero
9          output[:,self.channel,:] = torch.zeros_like(output[:,self.
       channel,:])
10         return output
11
12     def close(self):
13         self.hook.remove()
14
15 def set_msk_ch(layer_id, channel):
16     layer = dict([*model.named_modules()])[layer_id]
17     hooks[layer_id + str(channel)] =  AblateChannelHook(layer, int(
       channel))
18
19 def remove_msk_ch(layer_id, channel):
20     hooks[layer_id + str(channel)].close()
```

Listing C.1: PyTorch Implementation Channel Ablation Hook

```

def label_reshaping(self, y, b, h, w, device):
        y = y[:,None] if y.ndim == 1 else y
        assert y.ndim == 2, f'conditions shape {y.shape} should be (batch, n_conditions)'
        assert torch.is_tensor(y), 'labels array should be a pytorch tensor'
        n_conditions = y.shape[-1]
        labels = torch.ones((b,n_conditions,h,w)).to(device)
        return torch.einsum('ijkl,ij->ijkl', labels, y)

#conditioning
if self.mode == 'c':
    if len(y.shape) == 1: # Labels 1D
        y = self.label_reshaping(y, b, self.im_size, self.im_size, self.device)
    img = self.label_concatenate(img,y)
```

Listing C.2: PyTorch Implementation Label Conditioning

```
def gradient_sample(self, x, t, t_index, mask = None):

        noise = torch.randn_like(x[:,:self.channels])

        betas_t = self.extract(self.betas, t, x.shape)

        input = x.detach()
        input.requires_grad = True

        prediction =  self.sqrt_recip_alphas_t * (
            input[:,:self.channels] - betas_t * self.model(input, t) / self.sqrt_one_minus_alphas_cumprod_t
        )

        posterior_variance_t = self.extract(self.posterior_variance, t, x[:,:self.channels].shape)

        b,c,h,w = x.shape
        gradient_volume = numpy.empty(shape = (h*w,1,h,w))

        pred = prediction if mask is None else prediction * mask

        #If prediction has more than one color channels, sum them up
        if self.channels > 1:
```

```
            pred = torch.sum(pred[0],0)

        for k, f_val in enumerate(pred.flatten()):
            #if value has been masked, continue
            if  f_val == 0:
                continue

            input.grad.data.zero_()

            f_val.backward(retain_graph=True)

            #Calculate pixel-wise gradient map for pixel at index k
            pwg = input.grad.data.cpu().numpy()
            pwg = pwg.reshape(c,h,w)
            gradient_volume[k] = numpy.sum(pwg, axis=0)

        #return sample xt-1 and gradient volume PGV
        return (model_mean + torch.sqrt(posterior_variance_t) *
noise), gradient_volume
```

Listing C.3: PyTorch Implementation Gradient Volume

Appendix D

Additional Results

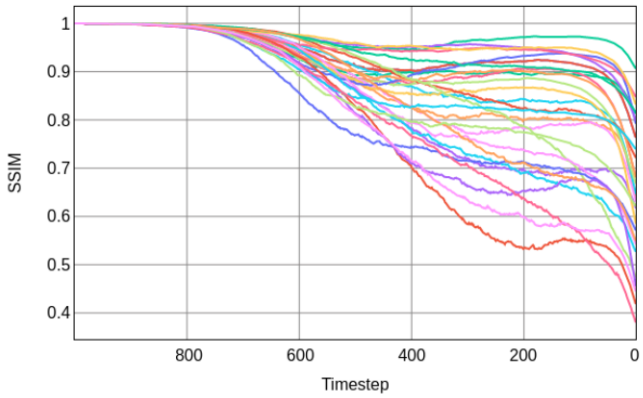


Figure D.1: The SSIM trajectory between labels 0000 and 0100 for 30 random noise seeds. Even though overall the final samples have less similarity than compared to Figure 5.14, the pattern of the trajectory remains.

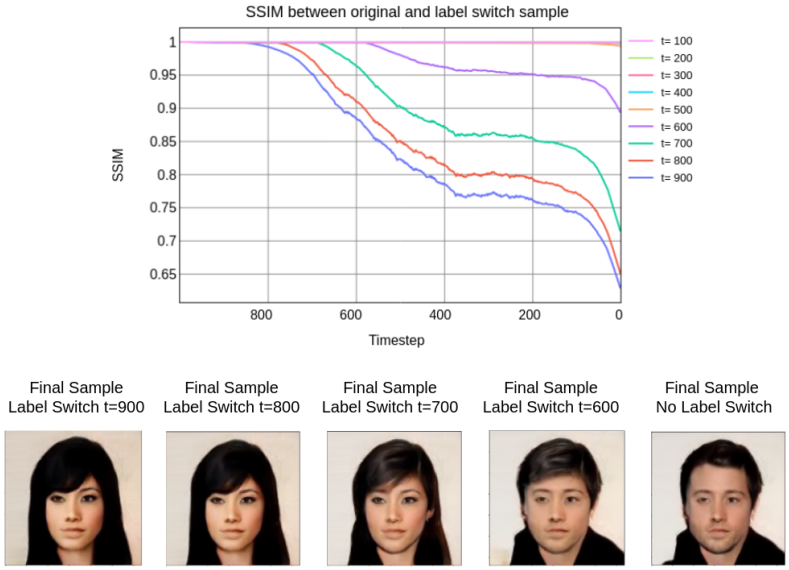


Figure D.2: Label switch between original label 0100 and label 1000.